DUALITÄT VON SUCHSTRATEGIEN AUF PLANAREN GRAPHEN

Diplomarbeit
von
Vanessa Kääb

Fakultät für Mathematik und Informatik

1999

Hiermit versichere ich, daß ich die vorliegende Arbeit selbstständig verfaßt und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Konstanz, den 27. Juli 1999 Vanessa Kääb

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundbegriffe der Graphentheorie **4**
2.1 Graphen und Untergraphen 4
2.2 Wege, Kreise und Zusammenhang 6
2.3 Bäume und Wälder 7
2.4 Adjazenzmatrix und Adjazenzlisten 8
2.5 Planare Graphen 9
2.6 Der Dualgraph 10

3 Suchstrategien **12**
3.1 Graphensuche 12
3.2 Planare Graphensuche 24

4 Dualität von Tiefen- und Breitensuche **40**
4.1 Der Schachtelungsbaum 40
4.2 E-DFS$_{\circlearrowleft}$ und BFS$_{\circlearrowright}$ sind nicht dual 42
4.3 E-DFS$_{\circlearrowleft}$ und die duale Breitensuche 46
4.4 BFS$_{\circlearrowright}$ und die duale Kanten-Tiefensuche 54

5 Dualität des Tiefensuchbaumes **64**
5.1 Die Right-First-Knoten-Tiefensuche 64
5.2 Dualitätseigenschaft des Tiefensuchbaumes 65

6 Schlußbemerkungen **78**

Kapitel 1

Einleitung

Die Graphentheorie ist heute ein wichtiges Teilgebiet der Mathematik. Vor über 200 Jahren verfaßte Euler mit [Eul36] die historisch erste graphentheoretische Arbeit. Seitdem stießen Wissenschaftler aus verschiedenen Bereichen immer wieder auf graphentheoretische Strukturen. Das Werk von König [Kö36] aus dem Jahre 1936 faßt die meisten wesentlichen Ergebnisse der Graphentheorie zusammen und hat entscheident dazu beigetragen, daß die Graphentheorie sich als wissenschaftlich anerkannte Theorie manifestieren konnte.
Zwei der klassischen Fragestellungen in der Graphentheorie sind das traveling salesman problem und das Vierfarbenproblem. Beim traveling salesman problem hat ein Handlungsreisender die Aufgabe zum Beispiel Kunden in bestimmten Städten aufzusuchen. Das Problem besteht nun darin eine Rundfahrt durch alle diese Städte zu finden, deren Gesamtstrecke so kurz wie möglich ist. Das Vierfarbenproblem dagegen fragt danach, ob die Länder einer Landkarte mit höchstens vier Farben so eingefärbt werden können, daß je zwei benachbarte Länder unterschiedliche Farben haben. In beiden Aufgaben können die gegebenen Daten als Graph modeliert werden. Auf abstrakten Graphen lassen sich dann Algorithmen beschreiben, die das jeweilige Problem lösen. Ein Graph besteht aus einer Menge von Knoten und einer Menge von Kanten. Die Kanten sind Linien, die je zwei Knoten verbinden. Im Fall des traveling salesman problem bilden die aufzusuchenden Städte die Menge der Knoten des Graphen und die gegebenen Straßenverbindungen werden durch die Kanten symbolisiert. Man kann jeder Kante noch zusätzlich die Länge der Straße zuweisen, welche sie repräsentiert. Auf einem solchen Graphen, der die reinen Verbindungseigenschaften und deren Längen wiederspiegelt, läßt sich das Problem mittels eines entsprechenden Algorithmus lösen. Das Vierfarbenproblem kann ebenfalls als Graphenproblem formuliert werden. In diesem Fall stellen die Länder der Landkarte die Knoten des Graphen dar und jede Grenze, die zwei Länder voneinander trennt, wird durch eine Kante repräsentiert.
Durch die Darstellung der gegebenen Daten in Form eines Graphen werden die Informationen auf die für die Lösung des Problems kritischen Fakten reduziert. Aspekte wie der genaue Straßenverlauf, die Größe eines Landes oder einer Stadt oder die Länge der Grenze zwischen zwei Ländern sind für die Lösung nicht von Bedeutung. Man kann sich nun

sicherlich vorstellen, daß sich Aufgaben aus den unterschiedlichsten Bereichen als graphentheoretisches Problem formulieren lassen.
Ein wichtiger Teil der Graphentheorie ist die Untersuchung von Strategien einen Graphen zu durchlaufen. Eine Durchlaufstrategie für einen Graphen bezeichnet man auch als Graphsuche. Wir werden uns in dieser Arbeit vor allem mit zwei der bekanntesten Durchlaufstrategien durch Graphen beschäftigen, der Breiten- und der Tiefensuche. Die Breitensuche ist eine Strategie, welche den Graphen schichtweise absucht. Sie geht vom Startknoten aus zuerst alle Knoten durch, zu welchen eine Kante existiert, bevor sie von einem anderen besuchten Knoten aus weitermacht. Die Tiefensuche dagegen geht so tief in den Graphen hinein wie möglich, indem sie immer von dem zuletzt besuchten Knoten aus zu einem neuen Knoten weiterläuft. Diese Arbeit wird sich mit speziellen Formen dieser beiden Graphsuchen beschäftigen.

Kapitel 2 enthält eine Einführung in einige wichtige Grundbegriffe der Graphentheorie. Wir haben uns bei den Bezeichnungen weitgehend an die angegebenen Bücher gehalten. Natürlich gibt es unzählige Werke, die einen umfassenden Einblick in die Graphentheorie geben, die verwendeten Bezeichnungen sind dabei aber sehr unterschiedlich. Wir haben uns bemüht in dem Kapitel die Grundbegriffe zu Graphen soweit einzuführen, daß ein zusätzliches Studium der Literatur zum Verständnis der Arbeit nicht notwendig ist. Diese Einführung beschränkt sich dabei aber im wesentlichen auf die benötigten Definitionen und kann keinen tieferen Einblick in die Graphtheorie bieten.
In Kapitel 3 werden Graphsuche allgemein und die schon erwähnten Tiefen- und Breitensuche definiert. Einige Anwendungsbeispiele sollen aufzeigen, daß Suchstrategien in vielen Lösungsalgorithmen eine zentrale Rolle übernehmen. Der zweite Teil des Kapitels führt den Begriff der dualen Suche ein und gibt einen ersten Ausblick auf die weiteren Forschungen.
Die Kapitel 4 und 5 umfassen die zentralen Aussagen der Arbeit. Zu Begin des 4. Kapitels werden wir die Struktur eines planaren Graphen betrachten. Ein planarer Graph ist ein Graph, der sich auf ein Blatt Papier derart zeichnen läßt, daß sich keine zwei Kanten überschneiden. Zu einem planar gezeichneten Graphen ist der Dualgraph definiert als der Graph, dessen Knotenmenge aus den Gebieten des Originalgraphen besteht (man vergleiche hierzu die Modelierung des Graphen zum Vierfarbenproblem). Zu jeder Kante des Originalgraphen existiert im Dualgraph eine Dualkante, welche die durch die Kante getrennten Gebiete verbindet. Das 4. Kapitel beschäftigt sich nun mit der Dualität von Tiefen- und Breitensuche, wenn die eine Suche auf dem Graphen ausgeführt wird und die andere auf dem Dualgraphen. Zwei Suchen sind dual, wenn die Suche auf dem Dualgraphen die Dualkanten in der gleichen Reihenfolge aufnimmt, wie die andere Suche die Kanten des Originalgraphen. Es wird eine modifizierte Breitensuche vorgestellt, welche dual zur Tiefensuche arbeitet. Ebenso führen wir eine modifizierte Tiefensuche ein, die dual ist zur Breitensuche.
Das 5. Kapitel beschäftigt sich mit einer speziellen Tiefensuche. Man kann jedem Knoten eines Graphen die Kante zuordnen, über welche er zum ersten Mal besucht wurde. Diese Kanten bilden einen Baum im Graphen, das heißt einen Untergraphen, der keinen Kreis enthält. Dieser Baum wird als Suchbaum bezeichnet. Alle anderen Kanten heißen

Nichtbaumkanten. Wir werden zeigen, daß die Dualkanten der Nichtbaumkanten im Dualgraphen wiederum einen Tiefensuchbaum bilden. Außerdem läßt sich die Kantenaufnahmereihenfolge der Tiefensuche, die diesen Tiefensuchbaum im Dualgraphen erzeugt aus dem Tiefensuchbaum im Originalgraphen rekonstruieren.
Die Ergebnisse der Arbeit werden in Kapitel 6 noch einmal kurz zusammengefaßt.

Kapitel 2

Grundbegriffe der Graphentheorie

Als Euler 1736 [Eul36] das bekannte Königsberger Brückenproblem löste, legte er damit den Grundstein zur Graphentheorie. Heutzutage finden Graphen in vielen Bereichen des täglichen Lebens Anwendung. Ein Beispiel für einen Graphen ist der Plan eines Untergrundbahnnetzes, dessen Linien nicht den tatsächlichen Verlauf der Schienen angeben, sondern nur die Verbindungen zwischen den verschiedenen Bahnhöfen wiederspiegeln. Wir wollen uns im folgenden etwas genauer mit einigen Problemen der Graphentheorie auseinander setzen. Dazu werden in diesem Kapitel grundlegende sowie im weiteren häufig verwendete Begriffe definiert. Für einen tieferen Einblick siehe auch [Har69] oder [Jun94].

2.1 Graphen und Untergraphen

Ein (ungerichteter) *Graph* $G = (V, E)$ besteht aus einer endlichen Menge $V \neq \emptyset$ und einer Menge E, welche eine Teilmenge der zweielementigen Teilmengen von V ist. Die Elemente $v \in V$ heißen *Knoten* (engl. vertices oder nodes), die Elemente $e \in E$ heißen *Kanten* (engl. edges). Sind die Kanten von G keine zweielementigen Teilmengen von V sondern geordnete Paare $(v, w) \in V \times V$, so ist der Graph *gerichtet*. Kanten der Form $\{v, v\}$ werden als *Schlingen* oder *Schleifen* (engl. loops) bezeichnet, und mehrere Kanten $\{v_1, w_1\}$, $\{v_2, w_2\}$ mit $v_1 = v_2$ und $w_1 = w_2$ heißen *Mehrfachkanten* (engl. multiple edges).
Die *Ordnung* eines Graphen G ist die Anzahl seiner Knoten, die wir mit $n := n(G) := |V|$ bezeichnen. Die Anzahl seiner Kanten notieren wir mit $m := m(G) := |E|$.
In einem ungerichteten Graphen werden die Knoten v und w einer Kante $e = \{v, w\}$ *Endknoten* von e genannt. Ein Knoten $v \in V$ und eine Kante $e \in E$ heißen *inzident*, falls $v \in e$, ebenso heißen zwei Kanten zueinander inzident, falls sie einen gemeinsamen Endknoten besitzen. Ist $\{v, w\} \in E$, so sind die zwei Knoten v und w *adjazent*. Die Menge $N(v) := \{u \in V \mid \{v, u\} \in E\}$ der *Nachbarn* von v bezeichnen wir als die *Nachbarschaft* des Knotens $v \in V$. Die Anzahl der Kanten, die zu einem Knoten $v \in V$ inzident sind,

bestimmen den *Grad* (endl. degree) $d(v) := |N(v)|$ des Knotens.
Zwei wichtige Beispiele von Graphen sind der *vollständige Graph* (engl. complete graph) und der *vollständig bipartite Graph*. Der vollständige Graph $G = (V, E)$ auf n Knoten hat als Kanten alle zweielementigen Teilmengen von V und wird mit K_n notiert. Die Knotenmenge V des vollständig bipartiten Graphen K_{n_1,n_2} ist die Vereinigung zweier disjunkter Knotenmengen V_1 und V_2 mit $|V_1| = n_1$ und $|V_2| = n_2$. Die Kantenmenge besteht aus genau den Kanten $\{v, w\}$ für die $v \in V_1$ und $w \in V_2$. In Abbildung 2.1 sind die vollständigen Graphen K_1 bis K_5 und der vollständig bipartite Graph $K_{3,3}$ dargestellt.

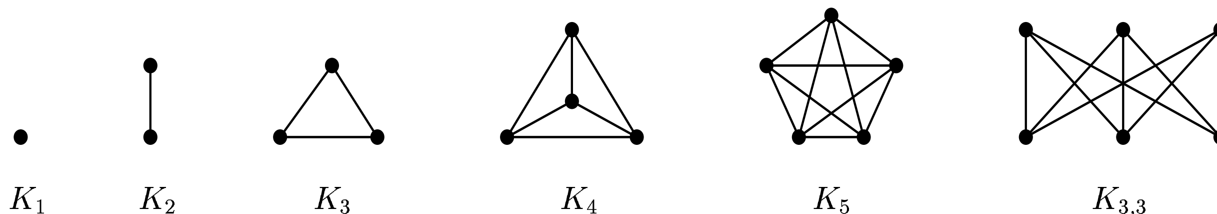


Abbildung 2.1: Beispiele für vollständige und vollständig bipartite Graphen

In einem gerichteten Graphen $D = (V, \vec{E})$ (engl. directed graph), auch Digraph genannt, heißt v der *Anfangs-* oder auch der *Auslaufknoten* und w der *End-* oder *Einlaufknoten* der Kante $\vec{e} = (v, w)$. Die Knoten v und w sind *inzident* zu $\vec{e}$, v und w sind wiederum *adjazent*. Die *Nachbarschaft* $N(v) := \{u \in V \mid (v, u) \in \vec{E}\}$ eines Knotens $v \in V$ ist definiert als die Menge der Endknoten aller aus v auslaufenden Kanten. Die Anzahl der Nachbarn geben den *Grad* $d(v) := d_{out}(v) := |N(v)|$ an, auch *Auslaufgrad* (engl. outdegree) genannt. Die Anzahl der Kanten, die v als Endknoten haben, bestimmen den *Einlaufgrad* (engl. indegree) $d_{in}(v) := |\{w \in V \mid (w, v) \in \vec{E}\}|$.

Man kann aus einem ungerichteten Graphen $G = (V, E)$ einen gerichteten Graphen D machen, indem man jede Kante $e \in E$ durch zwei entgegengesetzt gerichtete Kanten ersetzt. Ebenso kann man aus einem gerichteten Graphen $D = (V, \vec{E})$ einen ungerichteten Graphen erzeugen, indem jede gerichtete Kante $\vec{e} \in \vec{E}$ durch eine ungerichtete ersetzt wird und danach alle Mehrfachkanten bis auf eine Version gelöscht werden. Natürlich erhält man bei Hintereinanderausführung diese Aktionen nicht notwendigerweise wieder den selben Graphen, da die Richtung einer Kante in D bei der Konstruktion von G verloren geht.

Für einen Graphen $G = (V, E)$ sei eine Teilmenge V' von V gegeben. Die Menge aller Kanten e, die beide Endknoten in V' haben werde mit $E(V')$ bezeichnet. Der Graph $G' = (V', E(V'))$ heißt der durch V' *(knoten-) induzierte Untergraph* (engl. induced subgraph) von G und wird mit $G(V')$ bezeichnet. Für jede Menge $V' \subseteq V$ und $E' \subseteq E(V')$ heißt ein Graph $G' = (V', E')$ ein *Untergraph* oder *Teilgraph* von G. Ist $V' = V$ und $E' \subseteq E$, so ist $G' = (V, E')$ ein *aufspannender* Untergraph von G. Für eine Menge $V' \subseteq V$ bezeichnen wir mit $G \backslash V'$ den durch $V \backslash V'$ induzierten Subgraphen von G. Ist $V' = \{v\}$ einelementig, so schreiben wir auch $G \backslash v$.

In einem Graphen $G = (V, E)$ sei eine Teilmenge E' von E gegeben. Die Menge $V(E')$ enthalte alle Knoten $v \in V$, die Endknoten einer Kante $e \in E'$ sind. Dann heißt der Graph $G' = (V(E'), E')$ der durch E' *(kanten-) induzierte Untergraph* von G und wird entsprechend mit $G(E')$ bezeichnet.

2.2 Wege, Kreise und Zusammenhang

Es sei $v_0, v_1, \ldots, v_l$ eine Folge von (nicht notwendigerweise verschiedenen) Knoten eines Graphen $G = (V, E)$. Falls für alle $i = 1, \ldots, l$ gilt, daß $\{v_{i-1}, v_i\} \in E$, so handelt es sich bei der Folge um einen *Weg* (engl. walk) in G. Sind die v_i für alle $i = 0, \ldots, l$ paarweise verschieden, so heißt der Weg *Pfad* (engl. path). Die *Länge* eines Weges $W = (v_0, v_1, \ldots, v_l)$ ist die Anzahl l der Kanten, die er enthält. Ein Pfad auf n Knoten wird mit $P_n = (v_0, v_1, \ldots, v_n)$ bezeichnet und hat die Länge $n-1$. Dabei ist v_0 sein *Anfangsknoten* und v_n sein *Endknoten*. Alle anderen Knoten heißen *innere Knoten*. Ein Pfad von v nach w wird auch als v-w-*Pfad* bezeichnet.

Ein Weg $W = (v_0, v_1, \ldots, v_l)$ heißt *geschlossen* oder *Zykel* (engl. cycle), falls Anfangs- und Endknoten identisch sind. Sind die Knoten $v_1, v_2, \ldots, v_l$ alle paarweise verschieden, so handelt es sich um einen *Kreis*, für den wir auch C_l schreiben. Liegt ein Kreis in der Ebene, so kann man das *Innere* und das *Äußere* dieses Kreises unterscheiden. Liegt das Innere des Kreises rechts des gerichteten Weges $v_0, v_1, \ldots, v_l$, so handelt es sich um einen *Rechtskreis*, liegt das Innere links des Weges, so bezeichnen wir den Kreis als *Linkskreis*.

Zwei Knoten $v, w \in V$ eines Graphen G heißen *verbindbar* (engl. connected), falls es einen v-w-Pfad in G gibt. Die Länge eines kürzesten v-w-Pfades definiert den *Abstand*[1] (engl. distance) $dist(v, w)$ zwischen v und w. Existiert kein solcher Pfad, so sei $dist(v, w) := \infty$. Ein Graph G heißt *zusammenhängend*, falls für je zwei Knoten $v, w \in V$ ein v-w-Pfad existiert. Jeder Knoten $v \in V$ ist mit sich selbst verbindbar, den zugehörigen Pfad $P_1 = (v)$ nennen wir *trivial*. Die zusammenhängenden Teilgraphen von G, welche bezüglich ihrer Knotenmenge inklusionsmaximal sind, werden *(Zusammenhangs-) Komponenten* genannt. Eine Menge $V_S \subset V$ wird als *Separator* bezeichnet, falls der Untergraph $G(V \backslash V_S)$ des zusammenhängenden Graphen $G = (V, E)$ in wenigstens zwei Zusammenhangskomponenten zerfällt. Falls $V_S = \{v\}$ einelementig, so wird der Knoten v als Separator bezeichnet.
Eine Kante $e \in E$ eines zusammenhängenden Graphen G heißt *Brücke*, falls das Entfernen von e den Zusammenhang von G zerstört. Ein *Schnitt* $E_S \subset E$ ist eine Menge von Kanten, durch deren Entfernung aus E der zusammenhängende Graph G in wenigstens

[1] Die Abbildung $dist : V^2 \longrightarrow \mathbb{R}^+$ genügt den folgenden Bedingungen und ist demnach eine Metrik:

1. $dist(x, y) \geq 0$ und $dist(x, y) = 0$ genau dann wenn $x = y$
2. $dist(x, y) = dist(y, x)$
3. $dist(x, z) \leq dist(x, y) + dist(y, z)$ (Dreiecksungleichung)

zwei Zusammenhangskomponenten zerlegt wird. Ein Schnitt wird auch häufig mit S oder $(S, V\backslash S)$ notiert. Dabei sind S und $V\backslash S$ die Knotenmengen der durch E_S getrennten Zusammenhangskomponenten von G. Das heißt $E_S = \{\, \{v,w\} \in E \,|\, v \in S, w \in V\backslash S\,\}$, und für alle Kanten $\{v,w\} \in E\backslash E_S$ gilt, daß $v,w \in S$ oder $v,w \in V\backslash S$. Eine Brücke e ist also ein einelementiger Schnitt $E_S = \{e\}$.
Ein gerichteter Graph $D = (V, \vec{E})$ heißt *stark zusammenhängend*, falls zu je zwei Knoten $v,w \in V$ (gerichtete) Wege $P_1 = (v,\ldots,w)$ und $P_2 = (w,\ldots,v)$ existieren. D wird als *(schwach) zusammenhängend* bezeichnet, falls die ungerichetet Version G von D zusammenhängend ist.

2.3 Bäume und Wälder

Ein zusammenhängender Graph $T = (V,E)$, der keinen Kreis enthält, heißt *Baum* (engl. tree). Ein Baum erfüllt folgende Eigenschaften:

(T1) Er enthält genau $n-1$ viele Kanten.

(T2) Zwischen je zwei Knoten $v,w \in V$ existiert genau ein v-w-Pfad.

(T3) Ein Baum ist *minimal zusammenhängend*, das heißt, es kann keine Kante entfernt werden ohne den Zusammenhang zu zerstören.

(T4) Er ist *maximal kreisfrei*, das heißt, jede Kante $e \notin E$, die zwischen zwei Knoten $v,w \in V$ eingefügt wird, schließt einen Kreis.

Ein Graph F wird *Wald* (engl. forest) genannt, wenn alle seine Zusammenhangskomponenten Bäume sind. Die Knoten $v \in V$ eines Waldes F mit $d(v) = 1$ heißen *Blätter* (engl. leafs), alle anderen Knoten werden als *innere Knoten* bezeichnet.
Ein Baum T, der Teilgraph eines Graphen G ist mit $V(G) = V(T)$, heißt *aufspannender Baum* von G.

Ein *Wurzelbaum* (engl. rooted tree) ist ein Baum $T = (V,E)$ zusammen mit einem ausgezeichneten Knoten $r \in V$. Dieser Knoten r wird als *Wurzel* von T bezeichnet und im allgemeinen nicht als Blatt aufgefaßt. Ein Knoten u heißt *Nachfolger* eines Knoten v, falls $v \in \{r,u\}$ oder der r-u-Pfad in T v als inneren Knoten enthält. Knoten v ist dann *Vorgänger* von u. Ein Knoten v wird also auch als sein eigenener Vorgänger beziehungsweise Nachfolger angesehen. Ist $\{v,u\} \in E$, so bezeichnen wir u als *direkten* Nachfolger von v und v entsprechend als *direkten* Vorgänger von u. Die Knoten $v \in V$ mit $dist(r,v) = i$ bilden die *i-te Schicht* (engl. level) des Wurzelbaumes $T = (V,E)$.

2.4 Adjazenzmatrix und Adjazenzlisten

Für die Darstellung eines Graphen $G = (V, E)$ im Computer benötigen wir eine Datenstruktur, die alle notwendigen Informationen enthält, wenig Platz einnimmt und einen schnellen Zugriff auf die einzelnen Informationen gewährleistet. In der Komplexitätstheorie wird die Effizienz einer Datenstruktur oder eines Algorithmus in Abhängigkeit von der "Größe" der Eingabe betrachtet. Die Komplexität eines Algorithmus A ist also eine Funktion f, deren Wert $f(n)$ die Schrittzahl, bzw. den Speicherplatzt angibt, den A bei einer Eingabe der Länge n zur Lösung benötigt. Betrachtet man die Schrittzahl eines Algorithmus, so spricht man von seiner *Zeitkomplexität* (engl. time complexity), betrachtet man den benötigten Speicherplatzt, so spricht man von der *Raumkomplexität* (engl. space complexity). Meist kann man nur eine *obere Schranke* oder eine *Wachstumsrate* für $f(n)$ angeben. Hierzu werden im allgemeinen die folgenden Notationen verwendet. Seien h und g Abbildungen von $\mathbb{N}$ nach $\mathbb{R}^+$, dann werden die Mengen $\mathcal{O}$, Ω und Θ von Funktionen wie folgt definiert:

$$\begin{aligned}
\mathcal{O}(g(n)) &:= \{ h : \mathbb{N} \longrightarrow \mathbb{R}^+ \mid \exists\, c > 0, n_0 \;:\; h(n) \leq c \cdot g(n) \;\; \forall n \geq n_0 \,\} \\
\Omega(g(n)) &:= \{ h : \mathbb{N} \longrightarrow \mathbb{R}^+ \mid \exists\, c > 0, n_0 \;:\; h(n) \geq c \cdot g(n) \;\; \forall n \geq n_0 \,\} \\
\Theta(g(n)) &:= \{ h : \mathbb{N} \longrightarrow \mathbb{R}^+ \mid \exists\, c_1 > 0, c_2 > 0, n_0 \;:\; c_1 \cdot g(n) \leq h(n) \leq c_2 \cdot g(n) \\
&\qquad\qquad \forall n \geq n_0 \,\}
\end{aligned}$$

Gilt für einen Algorithmus, daß $f(n) \in \Theta(g(n))$, so ist das meist die schärfste Aussage die man über die Laufzeit eines Algorithmus machen kann, denn

$$f(n) \in \Theta(g(n)) \iff f(n) \in \mathcal{O}(g(n)) \,\wedge\, f(n) \in \Omega(g(n)).$$

Im allgemeinen ist die Raumkomplexität höchstens so groß wie die Zeitkomplexität, da in jedem Schritt des Algorihtmus höchstens ein Datum angefaßt wird und alle Daten eingelesen werden müssen. Liegen Zeit- und Raumkomplexität in $\mathcal{O}(g(n))$, so sagt man, der Algorithmus hat Komplexität $\mathcal{O}(g(n))$.

Bezeichne $n = |V|$ wie gewohnt die Anzahl der Knoten des Graphen und $m = |E|$ die Anzahl der Kanten. Die am häufigsten verwendeten Datenstrukturen sind die *Adjazenzmatrix* und die *Adjazenzlisten*. Die Adjazenzmatrix ist eine $n \times n$ -Matrix AM mit

$$AM(i,j) := \begin{cases} 1 & \text{, falls } \{i,j\} \in E \text{ bzw. } (i,j) \in \vec{E} \text{ im gerichteten Fall,} \\ 0 & \text{, sonst.} \end{cases}$$

Bei einem ungerichteten Graphen ist die Adjazenzmatrix symmetrisch. Eine Adjazenzmatrix benötigt $\mathcal{O}(n^2)$ viel Speicherplatzt, bietet aber den Vorteil, daß ein Algorithmus in $\mathcal{O}(1)$, also in einem Schritt auf jede Kante zugreifen kann.
Die Adjazenzlisten sind ein Array AL der Länge n von Listen. Für jeden Knoten $v \in V$

enthält die Liste $AL[v]$ alle zu v adjazenten Knoten. Im allgemeinen ist $m \in \mathcal{O}(n^2)$ und somit auch die Raumkomplexität der Adjazenzlisten in $\mathcal{O}(n^2)$. Leider gewähren die Adjazenzlisten keinen direkten Zugriff auf eine einzelne Kante, da man erst die Adjazenzliste des Anfangsknotens durchgehen muß, bis man den Endknoten gefunden hat. Das heißt, die Zugriffszeit auf eine Kante über Adjazenzlisten ist im schlechtesten Fall in $\mathcal{O}(n^2)$.

2.5 Planare Graphen

Ordnet man allen Knoten eines Graphen $G = (V, E)$ paarweise verschiedene Punkte $p = (x, y)$ in der Ebene zu und stellt die Kanten als deren Endpunkte verbindende Linien dar, so erhält man eine graphische Darstellung von G. Eine solche Darstellung eines Graphen bezeichnet man als seine *Einbettung in die Ebene*. Besitzt ein Graph G eine Einbettung, bei der sich je zwei Liniensegmente zu Kanten, die keinen gemeinsamen Endknoten besitzen, nicht schneiden, so heißt der Graph sowie seine Einbettung *planar*. Die Gebiete, in welche die Ebene durch eine planare Einbettung eines (planaren) Graphen G unterteilt wird, werden *Facetten* genannt. Jede Kante e besitzt entweder eine angrenzende Facette, falls e eine Brücke ist, oder zwei angrenzende Facetten. Bei gerichteten Kanten spricht man auch von der Facette rechts und der Facette links von der Kante. Für die Anzahl der Facetten f gilt (siehe auch [Jun94]):

$$n - m + f = 2 \qquad \text{(Eulersche Polyederformel)}$$

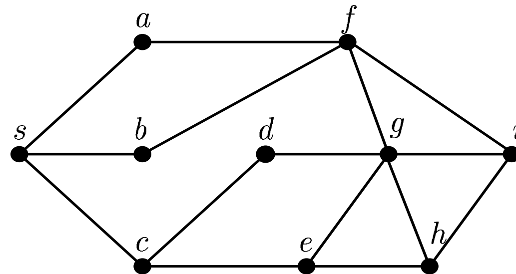
Daraus ergibt sich eine obere Schranke für die Kantenzahl m eines planaren Graphen (ebenfalls nachzulesen in [Jun94]):

$$m \leq 3n - 6 \;\; \text{(für } n \geq 3\text{)}$$

Das heißt $m \in \mathcal{O}(n)$ und daraus ergibt sich für die Zugriffszeit auf eine Kante über Adjazenzlisten eine Komplexität in $\mathcal{O}(m) = \mathcal{O}(n)$.
Als Beispiele für planare Graphen sei auf die in Abbildung 2.1 vorgestellten vollständigen Graphen K_1 bis K_4 mit höchstens 4 Knoten verwiesen. Nicht planar sind dagegen der K_5 und der vollständig bipartite Graph $K_{3,3}$. Ebenfalls planar sind Wälder.
Eine planare Einbettung eines Graphen G ist nicht eindeutig, das heißt ein planarer Graph hat unter Umständen ganz unterschiedliche planare Einbettungen. Wenn wir in Zukunft von einem planaren Graphen sprechen, so gehen wir davon aus, daß dieser in einer festen planaren Einbettung vorliegt.

Gegeben sei ein planar eingebetteter Graph $G = (V, E)$, dann wird durch diese feste planare Einbettung den zu einem Knoten v inzidenten Kanten eine Reihefolge, in der sie an v angrenzen, zugeordnet. Sind die Adjazenzlisten der Knoten entsprechend dieser Reihenfolge sortiert, so spricht man von der *kombinatorischen Einbettung* von G. Eine solche kombinatorische Einbettung läßt sich in Linearzeit berechnen, siehe hierzu [HT74]. In Abbildung 2.2 sind ein planarer Graph und seine gemäß der kombinatorischen Einbettung

sortierten Adjazenzlisten dargestellt. Die Adjazenzliste eines Knotens $v \in V$ enthält die zu v adjazenten Knoten entsprechend ihres Auftretens im Gegenuhrzeigersinn.
Ist ein planarer Graph G gegeben, so gehen wir in Zukunft davon aus, daß die Adjazenzlisten von G gemäß der planaren Einbettung sortiert sind.



$$
\begin{aligned}
AL[s] &= \{c, b, a\} \\
AL[a] &= \{s, f\} \\
AL[b] &= \{s, f\} \\
AL[c] &= \{e, d, s\} \\
AL[d] &= \{c, g\} \\
AL[e] &= \{h, g, c\} \\
AL[f] &= \{a, b, g, i\} \\
AL[g] &= \{d, e, h, i, f\} \\
AL[h] &= \{i, g, e\} \\
AL[i] &= \{f, g, h\}
\end{aligned}
$$

Abbildung 2.2: Planarer Graph und seine entsprechend sortierten Adjazenzlisten

Gegeben ein Wurzelbaum T in einer festen kombinatorischen Einbettung. Sei u direkter Vorgänger von v in T. Ein direkter Nachfolger w von v heißt *rechtester (linkester) Nachfolger* von v, falls w erster Knoten nach (letzter Knoten vor) u in der im Gegenuhrzeigersinn geordneten Adjazenzliste von v ist. Ein Blatt w ist *rechtestes Blatt*, falls w und alle seine Vorgänger, außer der Wurzel, jeweils rechteste Nachfolger ihrer direkten Vorgänger sind. Auf dem gerichteten Pfad von der Wurzel zu dem rechtesten Blatt gibt es nach dieser Definition keine Kante, die rechts des Pfades liegt.

2.6 Der Dualgraph

Es sei ein planarer Graph $G = (V, E)$ in einer festen planaren Einbettung gegeben. Ordnet man jeder Facette f von G einen Knoten zu und jeder Kante $e \in E$ eine *duale* Kante e^*, die die (nicht notwendigerweise verschiedenen) Knoten verbindet, welche die an die Kante angrenzenden Facetten repräsentieren, so erhält man den sogenannten *Dualgraph* $G^* = (V^*, E^*)$ zu G. In einem gerichteten Graphen D sind die Kanten $\vec{e}^* \in \vec{E}^*$ des Dualgraphen $D^* = (V^*, \vec{E}^*)$ von rechts nach links gerichtet. In Abbildung 2.3 ist ein ungerichteter und ein gerichteter Graph jeweils mit dem zugehörigen Dualgraphen dargestellt.

Der Dualgraph zu G^* ist wieder der Ausgangsgraph, das heißt $G^{**} = G$. Im gerichteten Fall ist das nicht richtig, da die Kantenorientierung von D^{**} gerade umgekehrt zu der von D ist, es gilt $D^{****} = D$. Entsprechend der Konstruktion hat der Dualgraph G^* gerade soviele Knoten wie G Facetten besitzt und soviele Facetten wie G Knoten. Die Anzahl der Kanten von G und G^* ist gleich. die Dualkante zu einer Brücke in G ist eine Schleife in G^*. Da sich planare Graphen unterschiedlich in die Ebene einbetten lassen, ergeben sich

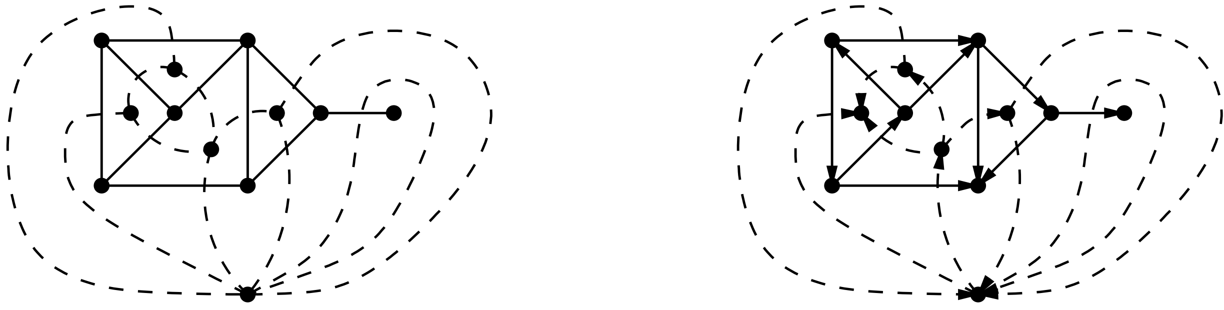


Abbildung 2.3: Graph und Dualgraph (gestrichelt) im ungerichteten und im gerichteten Fall

aus diesen Einbettungen auch unterschiedliche Dualgraphen. Wir betrachten deshalb den Dualgraphen immer bezüglich einer festen Einbettung von G.

Kapitel 3

Suchstrategien

In diesem Kapitel wollen wir uns damit beschäftigen, wie man einen Graphen "geschickt" durchläuft. Dabei kann man auf sehr unterschiedliche Weise vorgehen. Wir werden zuerst Graphensuche definieren und danach die zwei bekanntesten Beispiele vorstellen. Im zweiten Teil werden wir unser Augenmerk auf die planaren Graphen richten. Man kann auf planaren Graphen die Suchen noch weiter spezifizieren, indem man die feste kombinatorische Einbettung des Graphen ausnutzt. Außerdem werden wir die zu einer Suche duale Suche definieren und ein erstes Beispiel kennenlernen.

3.1 Graphensuche

In diesem Abschnitt gehen wird davon aus, daß ein gegebener Graph $G = (V, E)$ ungerichtet und zusammenhängend ist.

Definition 3.1.1 (Graphensuche)
Eine *Graphensuche* (engl. graph traversal) auf G von einem gegebenen Startknoten $v_s \in V$ aus ist eine Abbildung $\sigma : \{1, 2, \ldots, m\} \longrightarrow \vec{E}$ mit folgenden Eigenschaften:

(S1) $\sigma(t_1) = \sigma(t_2) \implies t_1 = t_2$ (Injektivität)

(S2) $(v, w) \in \vec{E}_m \implies (w, v) \notin \vec{E}_m$ (Antisymmetrie)

(S3) Sei $G_t := G(\vec{E}_t) = (V_t, \vec{E}_t)$ für alle $t = 1, \ldots, m$, dann existiert für jeden Knoten $v \in V_t$ ein gerichteter v_s-v-Pfad in G_t,

wobei $\vec{E}_t := \sigma(\{1, 2, \ldots, t\})$ und $V_t := V(\vec{E}_t)$, für $1 \leq t \leq m$.

V_0 ist die einelementige Menge, die nur den Startknoten v_s enthält. Zu jedem Zeitpunkt $t \in \{1, 2, \ldots, m\}$ enthält die Menge $V_t = V(\vec{E}_t)$ alle Knoten, die inzident zu einer Kante aus $\vec{E}_t = \{\vec{e}_{\sigma(1)}, \vec{e}_{\sigma(2)}, \ldots, \vec{e}_{\sigma(t)}\}$ sind. Eine Kante $\{u, v\} \in E$ heißt *besucht*, falls $(u, v) \in \vec{E}_t$ oder $(v, u) \in \vec{E}_t$, ansonsten ist sie zum Zeitpunkt t noch *unbesucht*. Aufgrund der Injektivität von σ läßt sich jeder Kante $\vec{e} \in \vec{E}_m$ in eindeutiger Weise der Zeitpunkt t zuordnen, zu welchem sie besucht wurde. Wir bezeichnen diesen Zeitpunkt als das *Alter* (engl. age) der Kante $\vec{e}$:

$$age(\vec{e}) := t, \text{ falls } \sigma(t) = \vec{e}$$

Ist ein Knoten v aus $V \backslash V_t$, so ist v zum Zeitpunkt t noch *unbesucht*. Die Menge V_t der bereits besuchten Knoten unterteilt sich in zwei disjunkte Mengen. Die Knoten, welche inzident zu einer noch unbesuchten Kante sind, nennen wir *aktiv*, die anderen vollständig *abgearbeitet*. Die Menge der aktiven Knoten wird mit $V_a \subset V_t$ notiert.

Eine Graphensuche läßt sich in zwei Schritte aufteilen. Solange noch nicht alle Kanten besucht sind, wird aus der Menge der aktiven Knoten zunächst ein Knoten $v \in V_a$ ausgewählt. Diese Knotenauswahl läßt sich als Abbildung $\nu : \mathbb{N} \longrightarrow V$ formulieren. Ist $\nu(t) = v$, so bezeichnen wir den Knoten v als den zum Zeitpunkt t *aktuellen* Knoten. Danach wird aus der Menge der zu v inzidenten unbesuchten Kanten eine Kante $\{v, w\}$ ausgewählt und aus v auslaufend orientiert. Dieser Schritt wird durch die Abbildung $\alpha : \mathbb{N} \longrightarrow \vec{E}$ mit $\alpha(t) = (\nu(t), w) \in \vec{E}$ beschrieben, und $(\nu(t), w)$ heißt die *aktuelle* Kante zum Zeitpunkt t. Die Projektion der Kantenauswahl auf den Endknoten der gewählten Kante werde mit $\tilde{\nu}(t)$ bezeichnet, das heißt $\tilde{\nu}(t) = w$ genau dann, wenn $\alpha(t) = (\nu(t), w) \in \vec{E}$.
Um die unterschiedlichen Suchstrategien einheitlich darzustellen, werden wir im folgenden immer auf Mengen von Kanten arbeiten, die in einem *Kantencontainer* M gespeichert werden. Die Kante $\vec{e} = (u, v) \in M$, über welche ein Knoten v von der Knotenauswahlregel zu einem Zeitpunkt t gewählt wird, nennen wir *Referenzkante* von v. Zu einem Knoten v können mehrere Referenzkanten in M gespeichert sein. Knotenauswahl sowie Kantenauswahl können auf die in M gespeicherten Kanten zugreifen und M gegebenenfalls aktualisieren.
Die Realisierung einer Suche besteht demnach aus drei Teilen,

- einem *(Kanten-) Container M* und eventuell anderen Datenstrukturen zur Speicherung von Zusatzinformationen,
- einer *Knotenauswahlregel* $\nu : t \longmapsto v \in V_a$ und
- einer *Kantenauswahlregel* $\alpha : t \longmapsto (\nu(t), w) \in \vec{E}$.

Ein entsprechender Suchalgorithmus hat dann die folgende einfache Form, welche in der darauffolgenden Skizze 3.1 noch einmal graphisch dargestellt wird:

(1) Initialisierung

(2) solange Kantencontainer M nicht leer

(3) $\quad v :=$ Knotenauswahl $\nu(t)$

(4) $\quad (v, w) :=$ Kantenauswahl $\alpha(t)$

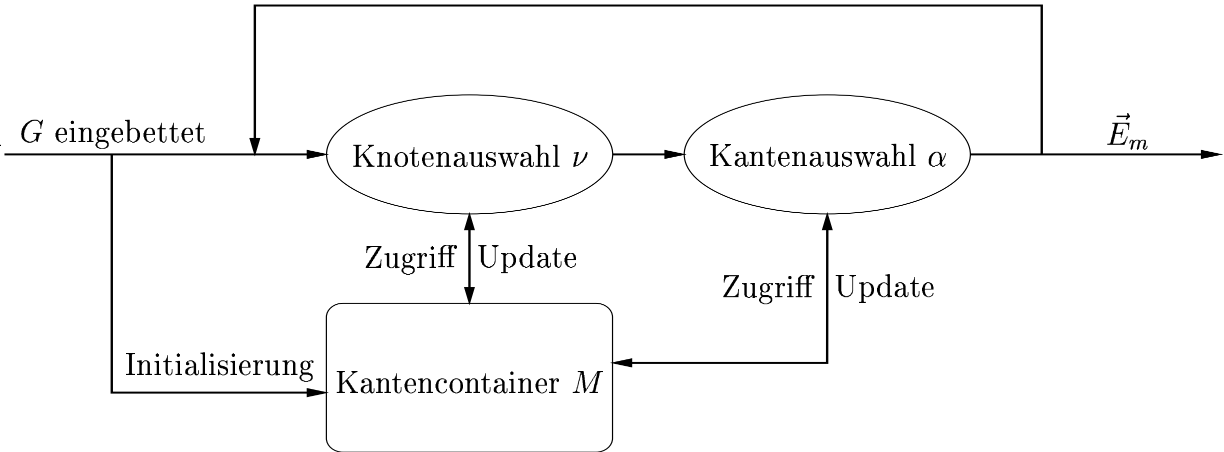


Abbildung 3.1: Schematische Darstellung einer Graphensuche

Die Kanten, durch welche ein Knoten bei einer Suche aktiv wird, ergeben den sogenannten *Graphensuchbaum*. Alle anderen Kanten werden *Nichtbaumkanten* oder *Rückwärtskanten* genannt.

3.1.1 Breitensuche

Eine der bekanntesten Suchstrategien ist die Breitensuche. Mit ihrer Hilfe lassen sich verschiedene graphentheoretische Probleme lösen. Gegeben sei ein ungerichteter, zusammenhängender Graph $G = (V, E)$ und ein Startknoten $v_s \in V$. Die Breitensuche arbeitet von diesem Knoten aus alle Nachbarn $N(v_s)$ ab und danach alle Nachbarn dieser Knoten, solange, bis alle Kanten besucht sind.

Viele Suchstrategien werden allein durch ihre Knotenauswahlregel bestimmt. Dabei erhalten die aktiven Knoten eine Rangfolge, die von den Zeitpunkten abhängt, zu denen sie besucht wurden. Wir führen deshalb für die Knoten einen Zeitstempel ein. Eine Suche startet zum Zeitpunkt $t = 1$, wobei $\tilde{\nu}(0) := v_s$. Sei

$$\tau(v, t) := \{t' < t \mid \tilde{\nu}(t') = v\},$$

also die Menge der Zeitpunkte t' vor t, zu denen die Kantenauswahl $(\nu(t'), v)$ gewählt hat. Bei einer Breitensuche erhält jeder Knoten einen fixen Zeitstempel, den Zeitpunkt, zu dem er zum ersten Mal besucht wurde :

$$\tau_{min}(v, t) := min\{t' \in \tau(v, t)\}$$

Die Breitensuche wird vollständig durch ihre Knotenauswahlregel festgelegt. Wir können sie nun mit den gegebenen Hilfsmitteln definieren. Die Breitensuche wählt aus der Menge der aktiven Knoten denjenigen Knoten aus, der den kleinsten Zeitstempel besitzt.

Definition 3.1.2 (Breitensuche)
Eine Graphsuche heißt *Breitensuche*, falls

$$\nu(t) = v \text{ mit } \tau_{min}(v,t) = min_{w \in V_a}\{\tau_{min}(w,t)\}$$

Die Breitensuche, englisch Breadth-First-Search, wird häufig mit BFS abgekürzt. Aus der Definition ergeben sich für die Breitensuche folgende Eigenschaften. Zu jedem Zeitpunkt $t \in \{1, 2, \ldots, m\}$ ist für jeden Knoten $v \in V_t$ der kürzeste gerichtete v_s-v-Weg in G_t auch ein kürzester v_s-v-Weg in G. Kanten aus $\vec{E}_t$ verlaufen nur zwischen Knoten aus V_t, deren Abstände zu v_s in G_t sich um höchstens 1 unterscheiden.
Entsprechend dieser Eigenschaften ist der Breitensuchbaum ein Baum kürzester Wege von v_s zu allen anderen Knoten in G. Der Abstand eines Knotens zur Wurzel v_s wird als sein *Level* bezeichnet. Die Menge aller Knoten v mit $level(v) = i$ bildet die *i-te Schicht* des Breitensuchbaumes. Die Breitensuche läßt sich mit Hilfe unseres Suchschemas folgendermaßen realisieren.

Algorithmus 3.1.3 (Breitensuche (BFS))
Eingabe: Ungerichteter, zusammenhängender Graph $G = (V, E)$ und ein ausgezeichneter Knoten $v_s \in V$.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$.

Knotenauswahl

(1) Endknoten v der Kante (u, v), die am längsten in M liegt

Kantenauswahl($v; (u, v)$)

(1) falls v aktiv
(2) wähle eine unbesuchte zu v inzidente Kante $\{v, w\}$ und orientiere sie: (v, w)
(3) falls w unbesucht
(4) lege (v, w) in M ab
(5) gib (v, w) aus
(6) ansonsten
(7) entferne (u, v) aus M

Erläuterung: Die Breitensuche beginnt mit Startknoten v_s und behält diesen solange als aktuellen Knoten bei, wie er noch aktiv ist. Die Breitensuche betrachtet zuerst alle Nachbarn von v_s und legt diese, beziehungsweise die Kanten über welche die Knoten zuerst entdeckt wurden, in M ab. Sind alle Nachbarn von v_s besucht, ist v_s nicht mehr aktiv

und (x, v_s) wird aus M entfernt. Die Endknoten der zu diesem Zeitpunkt in M liegenden Kanten bilden die 1. Schicht. Nun besucht die Breitensuche alle noch nicht gefundenen Nachbarn dieser Knoten, welche Schicht 2 bilden. Da die Referenzkanten aller Knoten aus Schicht 1 vor den Referenzkanten der Knoten aus Schicht 2 in M abgelegt wurden, wird der erste Knoten der 2. Schicht erst bearbeitet, wenn alle Knoten aus Schicht 1 abgearbeitet sind und ihre Referenzkanten aus M gelöscht wurden. Die Breitensuche arbeitet demnach schichtweise den ganzen Graphen ab.

Abbildung 3.2 zeigt eine Breitensuche angewendet auf einen Graphen G und den entstandenen Breitensuchbaum. Die dünn gestrichelten Pfeile zwischen Knoten des Baumes sind die Nichtbaumkanten. Wie man sieht verlaufen Nichtbaumkanten nur zwischen Knoten derselben Schicht oder von einem Knoten aus einer Schicht i zu einem Knoten aus Schicht $i+1$.

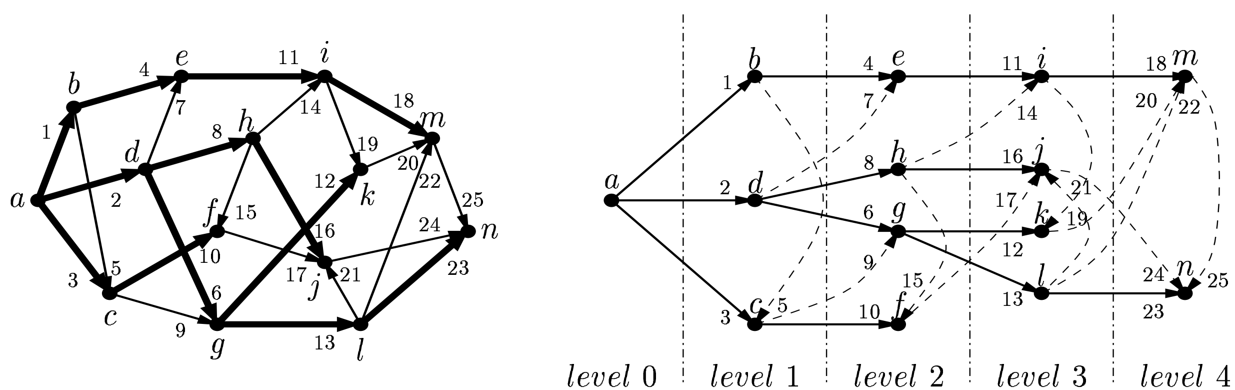


Abbildung 3.2: Breitensuche angewendet auf Graphen G und der zugehöriger Breitensuchbaum T. Die Pfeile in G kennzeichnen die durch die BFS festgesetzte Orientierung der Kanten. Der Breitensuchbaum wurde in G durch dicke Pfeile markiert. Die Zahl an einem Pfeil gibt das Alter der Kante an, also den Zeitpunkt t, zu dem sie ausgewählt wurde.

Laufzeitanalyse: Jede Kante wird im Laufe des Algorithmus einmal gefunden und dann nicht wieder betrachtet. Pro Kante werden eine konstante Anzahl einfacher Operationen ausgeführt, deren Laufzeit jeweils in $\mathcal{O}(1)$ ist. Die Gesamtlaufzeit der Breitensuche liegt somit in $\mathcal{O}(m)$, ist also linear in der Anzahl der Kanten von G. Der Speicherbedarf liegt in $\mathcal{O}(n)$, da für jeden Knoten genau eine Referenzkante, jene über die er zum ersten Mal gefunden wird, in M abgelegt wird. Danach ist er besucht und es wird keine weitere in ihn einlaufende Kante gespeichert. Damit hat die Breitensuche eine Komplexität in $\mathcal{O}(m)$.

An dem noch folgenden Beispiel werden wir sehen, in welcher Form die Breitensuche in der Praxis Anwendung findet. Vorab noch einige Definitionen:
Ein *Netzwerk* (engl. network) ist ein Paar (G, w), bestehend aus einem Graphen oder Digraphen $G = (V, E)$ und einer (Kanten-) Gewichtsfunktion[1] $w : E \longrightarrow \mathbb{R}^+$. Die Kan-

[1] Meist wird die Gewichtsfunktion mit $w : E \longrightarrow \mathbb{R}$ definiert. Wir werden aber nur Netzwerke mit positiven Kantengewichten betrachten und können uns somit ersparen, auf die Bedeutung der Existenz von negativen Gewichten einzugehen.

tengewichtsfunktion weist jeder Kante $e \in E$ ein Gewicht (engl. weight) $w(e)$ zu. In einem Straßennetz läßt sich dieses Gewicht zum Beispiel als Länge der Strecke interpretieren. Kommt es eher auf die Reisegeschwindigkeit an, so kann $w(e)$ aber auch ein Maß für die Zeit sein, die zum Befahren einer Strecke nötig ist. In einem Strom- oder Wassernetz ist wiederum die Kapazität einer Leitung von großer Bedeutung. Im folgenden bleiben wir bei der Interpretation von $w(e)$ als Länge der Strecke e.

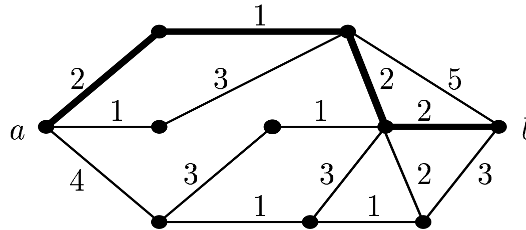


Abbildung 3.3: Ein Netzwerk in einem ungerichteten Graphen

Mit Hilfe der Gewichtsfunktion w kann jedem Weg $W = (v_0, v_1, \ldots, v_l)$ in G eine Länge $w(W) = w(\{v_0, v_1\}) + w(\{v_1, v_2\}) + \cdots + w(\{v_{l-1}, v_l\})$ zugeordnet werden. Sind zwei Knoten u und v in G verbindbar, so ist der *Abstand* bezüglich w $dist_w(u, v)$ definiert als das Minimum der Längen aller Wege, die u und v verbinden. Liegen u und v in unterschiedlichen Zusammenhangskomponenten des Graphen, so sei $dist_w(u, v) := \infty$. In Abbildung 3.3 ist $dist_w(a, b) = 7$ der Abstand von a und b auf dem dick markierten Weg. Nun aber zu dem angekündigten Anwendungsbeispiel.

Kürzeste Wege

Problem: Gegeben ein ungerichteter Graph $G = (V, E)$, eine Kantengewichtsfunktion $w : E \longrightarrow \mathbb{R}^+$ und ein Startknoten $v_s \in V$. Bestimme die kürzesten Wege von v_s zu jedem anderen Knoten $v \in V \backslash \{v_s\}$.

Zur Bestimmung der kürzesten Wege von $v_s \in V$ aus und deren Längen benötigen wir zwei zusätzliche Datenstrukturen:

- Knoten-Array VOR, in dem der direkte Vorgänger jedes Knotens auf einem kürzesten Weg gespeichert wird. Zu Beginn des Algorithmus ist $VOR[v]$ undefiniert für alle $v \in V$.
- Knoten-Array $DIST$, in welchem die Länge eines kürzesten Weges gespeichert wird. Zu Beginn des Algorithmus ist $DIST[v_s] := 0$ und $DIST[v] := \infty$ für alle $v \in V \backslash \{v_s\}$.

Für die Implementation eines Algorithmus zur Berechnung der kürzesten Wege können wir die Breitensuche in leicht modifizierter Form verwenden.

Algorithmus 3.1.4 (Kürzeste Wege)
Eingabe: Ungerichteter Graph $G = (V, E)$ und ein ausgezeichneter Knoten $v_s \in V$; Kantengewichtsfunktion $w : E \longrightarrow \mathbb{R}^+$.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$;
$DIST[v_s] := 0$ und $DIST[v] := \infty \ \forall v \in V \backslash \{v_s\}$.

Knotenauswahl

(1) Endknoten v einer Kante $(u, v) \in M$ mit $DIST[v] = min_{(x,y)\in M}\{DIST[y]\}$

Kantenauswahl(v; (u, v))

(1) falls v aktiv
(2) wähle eine unbesuchte zu v inzidente Kante $\{v, w\}$ und orientiere sie: (v, w)
(3) falls $DIST[v] + w(\{v, w\}) < DIST[w]$
(4) falls w unbesucht
(5) lege (v, w) in M ab
(6) $DIST[w] := DIST[v] + w(\{v, w\})$
(7) $VOR[w] := v$
(8) ansonsten
(9) entferne (u, v) aus M

Erläuterung: Im Gegensatz zur BFS wählt der Algorithmus 'Kürzeste Wege' immer denjenigen Knoten v aus der Menge der aktiven Knoten aus, der den kleinsten Abstand zu v_s hat. Von diesem Knoten aus werden alle Nachbarn besucht. Dabei wird für jeden Knoten $w \in N(v)$ nachgesehen, ob die Länge eines Weges über den aktuellen Knoten v kürzer ist als die Länge $DIST[w]$, welche bis zu diesem Zeitpunkt als die kürzeste betrachtet wurde. Daraus folgt für alle besuchten Kanten $\vec{e} = (v, w)$ die Eigenschaft, daß $DIST[v] \leq DIST[w]$. Aus dieser Eigenschaft ergibt sich unmittelbar, daß der Algorithmus 'Kürzeste Wege' für jeden Knoten $v \in V \backslash \{v_s\}$, der von v_s aus erreichbar ist, die Länge eines kürzesten v_s-v-Weges berechnet. Diese Länge ist am Ende des Algorithmus in $DIST[v]$ gespeichert, wobei ein Eintrag $DIST[u] = \infty$ bedeutet, daß u und v_s nicht verbindbar sind. Ein Weg W von v_s nach v mit $DIST[v] < \infty$, läßt sich über den Knoten-Array VOR von v aus rückwärts rekonstruieren:

$$W := (v_s = VOR^i[v], VOR^{i-1}[v], \ldots, VOR[v], v),$$

wobei $VOR^i[v] = \underbrace{VOR[VOR[\ldots VOR[v}_{i\text{-mal}}] \ldots]]$.

Die Korrektheit dieser Aussage ergibt sich aus der folgenden Eigenschaft von kürzesten Wegen.

Lemma: Gegeben sei ein Netzwerk (G, w). Ist $W = (v_0, v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_k)$ ein kürzester v_0-v_k-Weg, dann ist für jeden inneren Knoten v_i $(1 \leq i \leq k-1)$ $W_1 :=$

$(v_0, v_1, \ldots, v_{i-1}, v_i)$ ein kürzester v_0-v_i-Weg und $W_2 := (v_i, v_{i+1}, \ldots, v_k)$ ein kürzester v_i-v_k-Weg.

Beweis: Angenommen W_1 sei kein kürzester v_0-v_i-Weg, dann existiert ein Weg $W_3 := (u_0 = v_0, u_1, \ldots, u_j = v_i)$ mit $dist_w(W_3) < dist_w(W_1)$ und außerdem $i \neq j$ oder $v_l \neq u_l$ für ein $l \in \{1, \ldots, i = j\}$. Sei $W_4 := (u_0, u_1, \ldots, u_j = v_i, v_{i+1}, \ldots, v_k)$, dann ist $dist_w(W_4) = dist_w(W_3) + dist_w(W_2) < dist_w(W_1) + dist_w(W_2) = dist_w(W)$ im Widerspruch dazu, daß W ein kürzester v_0-v_k-Weg ist. Damit ist die Behauptung beweisen. □

3.1.2 Knoten-Tiefensuche

Die klassische Umkehrung des Vorgehens der Breitensuche kommt in der Tiefensuche zum Ausdruck. Bei der Tiefensuche wird solange vom zuletzt gesehenen Knoten aus weitergesucht, bis die Suche einen schon bearbeiteten Knoten findet. Bei der Tiefensuche kann man zwei Ausprägungen unterscheiden. Die Knoten-Tiefensuche macht nur dann von dem Endknoten der gewählten Kante aus weiter, wenn der Knoten noch unbesucht ist, die Kanten-Tiefensuche dagegen geht immer vom Endknoten der gewählten Kante aus weiter. Die Knoten-Tiefensuche richtet sich also nach dem Status des Knotens, die Kanten-Tiefensuche dagegen nach dem Status der Kante.
Wir wollen uns in diesem Abschnitt die Knoten-Tiefensuche und eine ihrer Anwendungen anschauen. Sei $\tau(v,t)$ wie oben definiert. Bei einer Knoten-Tiefensuche erhält jeder Knoten, genau wie bei der Breitensuche, einen fixen Zeitstempel, den Zeitpunkt $\tau_{min}(v,t) = min\{t' \in \tau(v,t)\}$, zu dem er zum ersten Mal besucht wurde.

Definition 3.1.5 (Knoten-Tiefensuche)
Eine Graphsuche heißt *Knoten-Tiefensuche*, falls

$$\nu(t) = v \ \text{ mit } \ \tau_{min}(v,t) = max_{w \in V_a}\{\tau_{min}(w,t)\}$$

Im allgemeinen wird die Knoten-Tiefensuche mit V-DFS bezeichnet, eine Abkürzung des englischen Vertex-Depth-First-Search. Der folgende Algorithmus implementiert eine Knoten-Tiefensuche gemäß unseres Suchschemas.

Algorithmus 3.1.6 (Knoten-Tiefensuche (V-DFS))
Eingabe: Ungerichteter, zusammenhängender Graph $G = (V, E)$ und ein ausgezeichneter Knoten $v_s \in V$.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$.

Knotenauswahl

(1) Endknoten v der Kante (u,v), die zuletzt in M eingefügt wurde

Kantenauswahl($v; (u,v)$)

(1) falls v aktiv
(2) wähle eine unbesuchte zu v inzidente Kante $\{v, w\}$ und orientiere sie: (v, w)
(3) falls w unbesucht
(4) lege (v, w) in M ab
(5) gib (v, w) aus
(6) ansonsten
(7) entferne (u, v) aus M

Erläuterung: Die Knoten-Tiefensuche arbeitet, im Gegensatz zur Breitensuche, immer von dem zuletzt neu gefundenen Knoten aus weiter. Mit dem Startknoten beginnend geht sie über eine unbesuchte Kante zum nächsten Knoten. Ist dieser noch unbesucht, so geht sie im nächsten Schritt von diesem Knoten aus weiter. Ist der Knoten besucht, so wird die Kante nicht in M abgelegt und der aktuelle Knoten bleibt aktueller Knoten für den nächsten Schritt. Ist der aktuelle Knoten nicht mehr aktiv, so führt die Tiefensuche einen *Backtrack* Schritt durch, das heißt, sie entfernt die Referenzkante des aktuellen Knoten aus M und macht von deren Anfangsknoten aus weiter. Auf ihrem Weg durch den Graphen macht die V-DFS von einem Knoten, der vor dem aktuellen Knoten gefunden wurde, nur dann weiter, wenn sie durch entsprechend viele Backtrack Schritte zu der Referenzkante dieses Knotens in M zurückkehrt.

Laufzeitananlyse: Genau wie die BFS besucht auch die V-DFS alle Kanten und führt pro Kante eine konstante Anzahl $\mathcal{O}(1)$-Operationen durch. Damit ergibt sich für die Knoten-Tiefensuche ebenfalls eine Laufzeit in $\mathcal{O}(m)$, Speicherbedarf in $\mathcal{O}(n)$ und folglich eine Gesamtkomplexität in $\mathcal{O}(m)$.

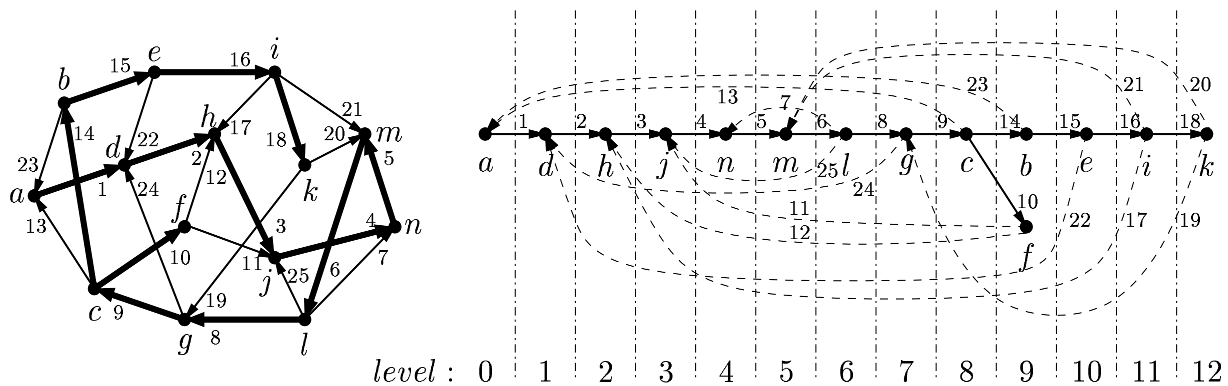


Abbildung 3.4: Knoten-Tiefensuche auf Graph und zugehöriger Tiefensuchbaum

Abbildung 3.4 zeigt eine Knoten-Tiefensuche angewendet auf einen Graphen G und den dadurch entstandenen Tiefensuchbaum. Die dünn gestrichelten Kanten stellen die Nichtbaumkanten dar. Nichtbaumkanten verlaufen immer von einem Knoten v zu einem Vorgänger

w von v im Baum. Sie verlaufen niemals zwischen Knoten aus unterschiedlichen Ästen des Baumes wie bei der Breitensuche. Jede Nichtbaumkante $\vec{e} = (u, v)$ schließt einen Kreis bestehend aus dem v-u-Pfad in dem Tiefensuchbaum und der Rückwärtskante (u, v).

Der Tiefensuchbaum T zusammen mit den Rückwärtskanten bilden eine gerichtete Version von G. In diesem Graphen wurde jede Kante $e \in E$ durch eine gerichtete Kante $\vec{e} \in \vec{E}$ ersetzt. Diesen gerichteten Graphen wollen wir mit $G^{V\text{-DFS}} = (V, \vec{E})$ bezeichnen. Eine Kante $\vec{e} = (u, v)$ aus der Menge der Baumkanten $\vec{E}_T \subset \vec{E}$ werde durch $(u, v)^{\rightarrow}$ gekennzeichnet, eine Kante aus der Menge der Nichtbaumkanten $\vec{E}_N = \vec{E} \backslash \vec{E}_T$ durch $(u, v)^{\leftarrow}$. Damit ist $T = (V, \vec{E}_T)$.
Das folgende Beispiel nach Tarjan beschreibt eine wichtige Anwendung der Tiefensuche in der Graphentheorie. Wir werden hier nur die wesentlichen Resultate nennen, die jeweiligen Beweise sind in [Tar72] nachzulesen.

2-facher Zusammenhang (nach Tarjan [Tar72])

Definition: Gegeben ein ungerichteter Graph $G = (V, E)$. Falls für je drei paarweise verschiedene Knoten v, w und $a \in V$ ein v-w-Pfad P existiert, so daß a nicht auf P liegt, dann ist G *2-fach zusammenhängend*.

Lemma: Sei $G = (V, E)$ ein ungerichteter Graph. Wir definieren auf der Menge der Kanten E folgende Äquivalenzrelation:
Zwei Kanten sind äquivalent genau dann, wenn sie auf demselben Zykel liegen.
Bezeichnen $E_i, 1 \leq i \leq n$, die verschiedenen Äquivalenzklassen dieser Relation und $G_i = (V_i, E_i)$, wobei $V_i = V(E_i)$, dann gilt:

(i) G_i ist 2-fach zusammenhängend, für alle $1 \leq i \leq n$.
(ii) Keiner der G_i ist ein echter Untergraph eines 2-fach zusammenhängenden Untergraphen von G.
(iii) Jeder Separator $u \in V$ von G kommt in mehr als einem der $V_i, 1 \leq i \leq n$ vor, jeder Knoten $v \in V$, der kein Separator ist, liegt in genau einer der Mengen $V_i, 1 \leq i \leq n$.
(iv) Jede Schnittmenge $V_i \cap V_j$ enthält mindestens einen Knoten $u \in V$, für alle $1 \leq i, j \leq n$. Ein solcher Schnittpunkt u ist ein Separator von G.

Die Untergraphen G_i von G sind die *2-fachen Zusammenhangskomponenten* von G.

Problem: Gegeben ein Graph $G = (V, E)$, bestimme die 2-fachen Zusammenhangskomponenten von G.

Shirey's Algorithmus [Shi69] testet jeden Knoten daraufhin ab, ob er ein Separator ist oder nicht. Die Laufzeit dieses Verfahrens liegt in $\mathcal{O}(|V| \cdot |E|)$. Ein effizienterer Algorithmus, den wir hier vorstellen, benutzt die V-DFS.
Sei $G = (V, E)$ und der durch eine Knoten-Tiefensuche entstandene gerichtete Graph $G^{V\text{-DFS}}$ gegeben. Die Knoten $v \in V$ seien in der Reihenfolge durchnumeriert, wie sie von

der Suche gefunden wurden, das heißt, jeder Knoten erhält eine Nummer $NUM(v) := \tau_{min}(v,t)$, die ihn identifiziert. Ist v ein Vorgänger von $w \neq v$ in T, so gilt $NUM(v) < NUM(w)$. Für jeden Knoten $v \in V$ wird ein Tiefpunkt definiert durch

$$LOWPT(v) := min\{\, NUM(v)\,,\ min\{\, NUM(w) \mid \exists u \in V, u \text{ Nachfolger von } v \text{ in } T \text{ und } (u,w)^{\leftarrow} \in \vec{E}_N \,\}\,\}$$

$LOWPT(v)$ ist also die Nummer des Knotens mit kleinster Nummer, der von v aus über einen v-u-Pfad (u nicht notwendigerweise verschieden von v) gefolgt von höchstens einer Nichtbaumkante $(u,w)^{\leftarrow}$ erreichbar ist.

Lemma: Gegeben ein zusammenhängender ungerichteter Graph $G = (V,E)$, $G^{V\text{-DFS}}$ die durch eine V-DFS entstandene gerichtete Version von G und T der Tiefensuchbaum in $G^{V\text{-DFS}}$. Seien a, v und $w \in V$ drei paarweise verschiedene Knoten mit $(a,v)^{\rightarrow} \in \vec{E}_T$ und w kein Nachfolger von v in T. Falls $LOWPT(v) \geq NUM(a)$, dann ist a ein Separator von G und das Entfernen von a trennt v und w. Umgekehrt gilt, falls a ein Separator von G ist, dann existieren Knoten $v, w \in V$, welche die obigen Eigenschaften erfüllen.
Es gilt außerdem, daß sich die Tiefpunkte wie folgt berechnen lassen:

$$LOWPT(v) = min(\ \{NUM(v)\} \cup \{LOWPT(w) | (v,w)^{\rightarrow} \in \vec{E}_T\} \cup \{NUM(w) | (v,w)^{\leftarrow} \in \vec{E}_N\}\ )$$

Der folgende Algorithmus ist eine erweiterte Knoten-Tiefensuche, die über diese Gleichung die Tiefpunkte berechnet und mit diesen die 2-fachen Zusammenhangskomponenten des Eingabegraphen G bestimmt.

Algorithmus: (2-fache Zusammenhangskomponenten)
Eingabe: Ungerichteter, zusammenhängender Graph $G = (V,E)$ und Startknoten $v_s \in V$.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$.
Zähler $i := 0$, Knoten-Arrays LOW und NUM, $LOW[v_s] := NUM[v_s] := 0$.
Kanten-Menge $\bar{M} := \emptyset$.

Knotenauswahl

(1) Endknoten v der Kante (u,v), die zuletzt in M eingefügt wurde

Kantenauswahl$(v; (u,v))$

(1) falls v aktiv
(2) wähle eine unbesuchte zu v inzidente Kante $\{v,w\}$ und orientiere sie: (v,w)
(3) falls w noch nicht numeriert (w noch unbesucht)
(4) lege (v,w) in M und in $\bar{M}$ ab

(5) $i := i + 1$
(6) $LOW[w] := NUM[w] := i$
(7) ansonsten
(8) lege (v, w) in $\bar{M}$ ab
(9) $LOW[v] := min\{\ LOW[v], NUM[w]\ \}$
(10) gib (v, w) aus
(11) ansonsten (v nicht mehr aktiv)
(12) entferne (u, v) aus M
(13) $LOW[u] := min\{\ LOW[u], LOW[v]\ \}$
(14) falls $LOW[v] \geq NUM[u]$
(15) lege neue 2-fache Zusammenhangskomponente K_j an
(16) sei $\vec{e} = (x, y)$ Kante, die zuletzt in $\bar{M}$ eingefügt wurde
(17) solange $NUM[x] \geq NUM[v]$
(18) entferne (x, y) aus $\bar{M}$ und füge sie in K_j ein
(19) entferne (u, v) aus $\bar{M}$ und füge sie in K_j ein

Erläuterung: Der Zähler i merkt sich die Nummer des zuletzt neu gefundenen Knotens. In $NUM[v]$ wird die Nummer des Knotens v gespeichert und in $LOW[v]$ ein Wert kleiner gleich $NUM[v]$ und größer gleich des tatsächlichen $LOWPT(v)$. Im Laufe des Algorithmus wird der Tiefpunkt eines Knotens v bestimmt und in $LOW[v]$ vermerkt. Die Menge $\bar{M}$ speichert alle Kanten, die auch in M abgelegt werden und zusätzlich alle Nichtbaumkanten. Würden die Nichtbaumkanten ebenfalls in M abgelegt werden, so würde die Knotenauswahl den Endknoten einer Nichtbaumkante im nächsten Schritt auswählen entgegen der Definition von Knoten-Tiefensuche. Der Algorithmus '2-fache Zusammenhangskomponenten' führt in erster Linie eine Knoten-Tiefensuche durch. In $\bar{M}$ werden dabei alle Kanten, die auf dem Weg gefunden werden abgelegt. Je nachdem, welchen Status der aktuelle Knoten beziehungsweise der Endknoten der gewählten Kante hat führt die Suche eine von drei unterschiedliche Aktionen durch. Sei v der aktuelle Knoten und (u, v) seine Referenzkante in M.

a) v ist aktiv und die Suche wählt eine Kante (v, w) mit noch unbesuchtem Endknoten w. Dann wird diese Kante in M und $\bar{M}$ aufgenommen und
$LOW[w] := NUM[w]$ gesetzt (Schritt (4) und (6)).

b) v ist aktiv und die Suche wählt eine Kante (v, w) mit besuchtem Endknoten w, also $NUM[w] < NUM[v]$. Nichtbaumkante $(v, w)^{\leftarrow}$ wird in $\bar{M}$ aufgenommen und
$LOW[v] := min\{LOW[v], NUM[w]\}$ gesetzt (Schritt (8) und (9)).

c) v ist abgearbeitet. Das bedeutet insbesondere, daß auch alle Knoten, die Nachfolger von v im Tiefensuchbaum sind, abgearbeitet sind. Die Kante $(u, v)^{\rightarrow}$ wird aus M entfernt und $LOW[u] := min\{LOW[v], LOW[u]\}$ gesetzt (Schritt (12) und (13)).

In Fall a) geht die Suche einen Schritt weiter im Tiefensuchbaum und in Fall b) bearbeitet sie eine Nichtbaumkante $(v,w)^{\leftarrow} \in \vec{E}_N$. In Fall c) geht die Suche im Tiefensuchbaum eine Kante zurück und für den Anfangsknoten dieser Kante gilt die folgende Aussage.
Beh: Entfernt die Tiefensuche zum Zeitpunkt t die Referenzkante (u,v) von v aus M, dann ist $LOW[v] = LOWPT(v)$ korrekt berechnet.
Bew: Ist v ein Blatt des Tiefensuchbaumes, dann ist die Menge $\{w|(v,w)^{\rightarrow} \in \vec{E}_T\}$ leer und somit $LOWPT(v) = min(\ \{NUM(v)\} \cup \{NUM(w)|(v,w)^{\leftarrow} \in \vec{E}_N\}\ )$. Da v abgearbeitet ist wurden alle Kanten $(v,w)^{\leftarrow} \in \vec{E}_N$ durch eine Aktion b) bearbeitet. Damit ist $LOW[v] = LOWPT(v)$ bestimmt. Sei v nun kein Blatt, aber alle $LOW[x] = LOWPT(x)$ für $x \in \{w|(v,w)^{\rightarrow} \in \vec{E}_T\}$ korrekt berechnet. Da v inaktiv, wurden alle Kanten $(v,w)^{\leftarrow} \in \vec{E}_N$ durch eine Aktion b) bearbeitet und außerdem alle Kanten $(u,v)^{\rightarrow} \in \vec{E}_T$ durch einen Schritt c) aus M entfernt. Somit ist auch für einen Knoten v, der kein Blatt ist, beim Entfernen seiner Referenzkante (u,v) $LOW[v] = LOWPT(v)$ richtig berechnet. □
Schließlich überprüft der Algorithmus in Schritt (14), ob der Anfangsknoten u der aus M entfernten Kante (u,v) ein Separator ist. Nach dem 2. Lemma muß hierzu gelten, daß $LOWPT(v) \geq NUM(u)$. Ist dies der Fall, so trennt der Separator u die Kante (u,v) und alle Kanten, die nach (u,v) gefunden wurden, vom Rest des Graphen. Diese Kanten sind nach Konstruktion genau die Kanten, die nach (u,v) in $\bar{M}$ abgelegt wurden. Diese werden nun in Schritt (15) bis (19) zusammen mit (u,v) aus $\bar{M}$ entfernt und in einer 2-fachen Zusammenhangskomponente gespeichert.

Laufzeitanalyse: Jede Kante wird im Laufe des Algorithmus einmal besucht. Pro Kante werden einige Operationen wie Zuweisungen und Minimumsbildung von zwei Zahlen ausgeführt. Diese Operationen lassen sich in konstanter Zeit, das heißt in $\mathcal{O}(1)$ durchführen. Die Laufzeit für Algorithmus '2-fache Zusammenhangskomponenten' liegt demnach in $\mathcal{O}(m)$. Im Gegensatz zur Knoten-Tiefensuche benötigt dieser Algorithmus Speicherplatzt in $\mathcal{O}(m)$, da jede Kante in $\bar{M}$ abgelegt wird. Die zusätzlichen Datenstrukturen NUM und LOW benötigen $\mathcal{O}(n)$ Platz, da sie Informationen über Knoten speichern. Die Gesamtlaufzeit liegt demzufolge wieder in $\mathcal{O}(m)$.

Tarjan stellt in seinem Artikel [Tar72] auch eine Anwendung der Knoten-Tiefensuche auf gerichteten Graphen vor. Mit Hilfe der Tiefensuche läßt sich ein gerichteter Graph auf starken Zusammenhang prüfen. Die Vorgehensweise ist der zum Test auf 2-fachen Zusammenhang sehr ähnlich.

3.2 Planare Graphensuche

In diesem Abschnitt wollen wir uns Suchen auf planaren Graphen widmen. Dabei sei ein planarer Graph, wie schon früher erwähnt, immer in einer festen kombinatorischen Einbettung gegeben. Suchen auf planaren Graphen nutzen diese zusätzlichen Informationen über die Einbettung des Graphen aus, um eine gezieltere Kantenauswahl zu treffen. Man

spricht von einer *Left-First-Suche*, wenn die inzidenten Kanten von einer Referenzkante aus im Uhrzeigersinn abgearbeitet werden. Entsprechend ist eine *Right-First-Suche* eine Suche, bei welcher der rechteste adjazente Knoten zuerst bearbeitet wird. Die Adjazenzlisten werden dann im Gegenuhrzeigersinn durchlaufen. Benutzt eine Suche auf einem planaren Graphen die Left-First- (Right-First-) Auswahlregel, so wollen wir die Suche mit einem $\circlearrowright$ ($\circlearrowleft$) kennzeichnen.
Wir werden zu Begin dieses Abschnittes eine planare Suche und ein Anwendungsbeispiel vorstellen. Im zweiten Teil werden wir die zu einer Suche duale Suche definieren. Das darauffolgende einfache Beispiel soll die Idee und einige der auftretenden Probleme verdeutlichen.

3.2.1 Right-First-Kanten-Tiefensuche

Gegeben ein planarer Graph $G = (V, E)$ in einer festen planaren Einbettung. Wir wollen uns die schon in Abschnitt 3.1.2 erwähnte Kanten-Tiefensuche anschauen. Die Kanten-Tiefensuche arbeitet immer von dem Endknoten der gewählten Kante aus weiter, egal, ob dieser bereits zu einem früheren Zeitpunkt besucht wurde oder nicht. Sei $\tau(v,t)$ wieder wie in Abschnitt 3.1.1 definiert. Bei einer Kanten-Tiefensuche erhält jeder Knoten $v \in V$, im Gegensatz zur Knoten-Tiefensuche, einen variablen Zeitstempel $\tau_{max}(v,t)$, den Zeitpunkt, zu dem er zum letzten Mal gefunden wurde:

$$\tau_{max}(v,t) := max\{t' \in \tau(v,t)\}$$

Definition 3.2.1 (Kanten-Tiefensuche)
Eine Graphsuche heißt *Kanten-Tiefensuche*, falls

$$\nu(t) = v \;\text{ mit }\; \tau_{max}(v,t) = max_{w \in V_a}\{\tau_{max}(w,t)\}$$

Der folgende Algorithmus implementiert eine Right-First-Kanten-Tiefensuche, die wir im folgenden mit E-DFS$_{\circlearrowleft}$ notieren, für Edge-Depth-First-Search mit Right-First-Auswahlregel.

Algorithmus 3.2.2 (Right-First-Kanten-Tiefensuche (E-DFS$_{\circlearrowleft}$))
Eingabe: Ungerichteter planar eingebetteter Graph $G = (V, E)$ und ein ausgezeichneter Knoten $v_s \in V$ auf dem Rand der äußeren Facette.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$ in der äußeren Facette.

Knotenauswahl

(1) Endknoten v der Kante (u, v), die zuletzt in M eingefügt wurde

Kantenauswahl($v; (u, v)$)

(1) falls v aktiv

(2) wähle die im Gegenuhrzeigersinn von (u, v) aus nächste unbesuchte zu v inzidente Kante $\{v, w\}$ und orientiere sie: (v, w)

(3) lege (v, w) in M ab

(4) gib (v, w) aus

(5) ansonsten

(6) entferne (u, v) aus M

Erläuterung: Die Right-First-Kanten-Tiefensuche geht von der Startkante aus auf dem Rand der äußeren Facette entlang. Sie nimmt an jedem Knoten v die von der aktuellen Kante aus nächst rechte Kante $\{v, w\}$ auf. Solange durch Entfernen des bisher besuchten Weges der Graph zusammenhängend bleibt, arbeitet sich die Kanten-Tiefensuche demnach spiralförmig in den Graphen vor. Zerfällt der Graph durch diesen Prozeß, so bleibt auch die Kanten-Tiefensuche, wie die Knoten-Tiefensuche, in einer Zusammenhangskomponente hängen und arbeitet diese vollständig ab, um danach durch entsprechend viele Backtrack Schritte zu noch unbesuchten Teilen des Graphen zurückzukehren.

Laufzeitanalyse: Wie schon bei den anderen beiden Suchen ist auch die Zeitkomplexität der Kanten-Tiefensuche in $\mathcal{O}(m)$, da jede Kante einmal gefunden, abgespeichert und später wieder gelöscht wird. Die Laufzeit ist demnach linear in der Anzahl der Kanten des Eingabegraphs. Im Gegensatz zu den beiden anderen Suchen speichert die E-DFS$_{\circlearrowleft}$ aber alle Kanten ab, hat also Speicherbedarf in $\mathcal{O}(m)$. In Abschnitt 2.5 haben wir gesehen, daß die Anzahl der Kanten eines planaren Graphen durch die Anzahl der Knoten beschränkt wird, das heißt $m \in \mathcal{O}(n)$. Damit ergibt sich für die Kanten-Tiefensuche auf planaren Graphen eine Laufzeit, Speicherbedarf und Gesamtkomplexität in $\mathcal{O}(n)$.

Wir werden später noch von den besonderen Eigenschaften der E-DFS$_{\circlearrowleft}$ Gebrauch machen. Deshalb wollen wir hier eine wichtige Eigenschaft als Lemma festhalten.

Lemma 3.2.3
Die durch den Algorithmus E-DFS$_{\circlearrowleft}$ geschlossenen Kreise sind Linkskreise.

Beweis:
Der Startknoten der E-DFS$_{\circlearrowleft}$ liegt nach Definition auf dem Rand der äußeren Facette. Die eingefügte Dummykante liegt ebenfalls in der äußeren Facette und bildet den Anfang des Suchweges. Trifft die Tiefensuche nun auf einen Knoten v, den sie bereits vorher besucht hat, so trifft sie auf den durch v laufenden Suchweg von links.
Angenommen das trifft nicht zu. In Abbildung 3.5 ist ein Suchweg dargestellt, bei dem die Tiefensuche auf den bisherigen Suchweg von rechts trifft. Da die Startkante immer außerhalb des geschlossenen Kreises liegt, ist die in der Abbildung dargestellte Situation die einzige Möglichkeit einen Rechtskreis zu schließen. Die Tiefensuche hat in der Abbildung den Weg von x nach v genommen und ist von dort aus über w und u gegangen. Die letzte aufgenommene Kante ist (u, v), welche zu diesem Zeitpunkt und somit auch zu jedem

Zeitpunkt davor noch unbesucht war. Bei der Wahl der Kante (v, w) von v aus hat die Tiefensuche demnach die Right-First-Auswahlregel verletzt. Daraus folgt, daß die durch die E-DFS$_{\circlearrowleft}$ geschlossenen Kreise Linkskreise sind. $\square$

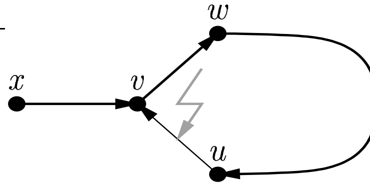


Abbildung 3.5: Kann die E-DFS$_{\circlearrowleft}$ einen Rechtskreis schließen, so wurde zu einem früheren Zeitpunkt die Right-First-Auswahlregel verletzt.

Wir wollen uns nun ein Beispiel für die Anwendung der E-DFS$_{\circlearrowleft}$ anschauen. Gegeben sei ein planarer Graph $G = (V, E)$ und zwei ausgezeichnete Knoten $s, t \in V$, $s \neq t$. G heißt *s-t-planar*, falls s und t auf dem Rand derselben Facette liegen. Ohne Einschränkung sei die planare Einbettung des s-t-planaren Graphen G derart, daß s und t auf dem Rand der äußeren Facette liegen.

Kantendisjunktes Menger Problem auf s-t-planaren Graphen

Problem: Gegeben ein s-t-planarer Graph $G = (V, E)$ und $s, t \in V$ auf dem Rand der äußeren Facette. Finde so viele kantendisjunkte s-t-Wege in G wie möglich.

Dieses Problem läßt sich mit Hilfe der Right-First-Kanten-Tiefensuche sehr einfach lösen: Die Suche startet mit Knoten s und nimmt die rechteste zu s inzidente Kante auf. Sobald eine neue unbesuchte Kante (v, w) von der Kantenauswahl gefunden wurde, wird abgefragt, ob der Endknoten w gleich s oder t ist. Ist dies der Fall, so werden alle Kanten $\vec{e} \in M$, außer der Dummykante, aus M entfernt. Ist $w = t$, so bilden diese Kanten in der Reihenfolge, in der sie in M aufgenommen wurden, einen s-t-Pfad, ist dagegen $w = s$, so bildet der gefundene Pfad einen s-Cycle. Danach ist die einzige in M verbliebene Kante die Dummykante und es wird demnach erneut von s aus nach dem nächsten Weg gesucht. Wir wollen diese modifizierte Tiefensuche s-t-Tiefensuche nennen.
In Abbildung 3.6 ist ein s-t-planarer Graph G dargestellt. Die farblich abgesetzten Wege kennzeichnen die vier Lösungswege, die von einer s-t-Tiefensuche gefunden werden.

Laufzeitanalyse: Die s-t-Tiefensuche unterscheidet sich von der E-DFS$_{\circlearrowleft}$ nur dadurch, daß sie, sobald sie s oder t gefunden hat, wieder bei s anfängt. Ihre Laufzeit liegt demnach ebenfalls in $\mathcal{O}(n)$.

Das kantendisjunkte Menger Problem in s-t-planaren Graphen ist ein Spezialfall aus einer ganzen Klasse von graphentheoretischen Problemen. Diese werden im allgemeinen unter *disjunkte Wege in planaren Graphen* zusammengefaßt. Wer sich einen Überblick über verschiedene Aufgabenstellungen und deren Lösungen verschaffen will, sei auf [RLWW95]

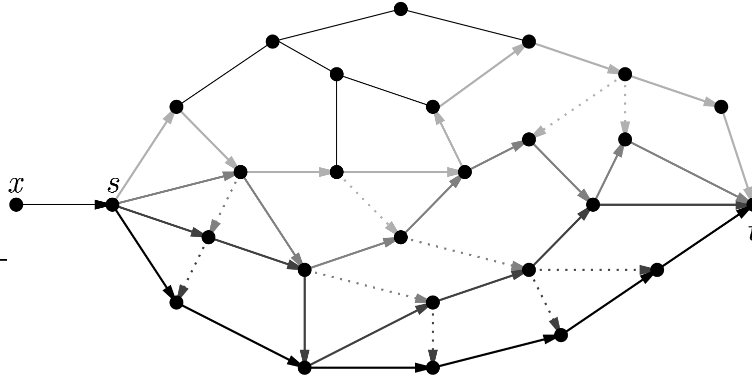


Abbildung 3.6: s-t-Tiefensuche angewendet auf s-t-planaren Graphen. Gepunktete Pfeile kennzeichnen Backtrack-Kanten, die dünnen schwarzen Linien ohne Pfeile entsprechen Kanten, die nicht gefunden werden.

verwiesen. In diesem Artikel wird unter anderem auch das knotendisjunkte Menger Problem in s-t-planaren Graphen betrachtet. Es ist leicht einzusehen, daß es sich mit einer Right-First-Knoten-Tiefensuche auf planaren Graphen lösen läßt. K.Weihe präsentiert in [Wei94] einen Linearzeitalgorithmus für das kantendisjunkte Menger Problem in planaren Graphen. Dieser verwendet ebenfalls die Right-First-Kanten-Tiefensuche, arbeitet aber auf einer modifizierten gerichteten Version des Graphen. Coupry [Cou97] vereinfachte den Algorithmus von Weihe, indem er die Referenzkante eines Knoten auf die Kante, über die er zum erstem Mal gefunden wurde, festsetzt. In [BW97] wird ein linearer Algorithmus zur Lösung des kantendisjunkte Menger Problems in gerichteten planaren Graphen vorgestellt, und in [RLWW93] präsentieren die Autoren einen Algorithmus mit Komplexität $\mathcal{O}(n)$ für das knotendisjunkte Menger Problem in planaren Graphen.

3.2.2 Duale Graphensuche

Eine Graphensuche ist im wesentlichen eine Aufzählung der Kanten. Wie wir in Abschnitt 2.6 gesehen haben, existiert eine Bijektion zwischen den Kanten eines Graphen $G = (V, E)$ und den Kanten des Dualgraphen $G^* = (V^*, E^*)$. Es liegt also nahe, zu einer Suche σ auf G eine duale Suche σ^* auf G^* zu definieren.

Definition 3.2.4 (duale Suche)
Gegeben eine Suchstrategie σ auf einem planar eingebetteten Graphen $G = (V, E)$. Ist

$$\sigma^*(i) := [\sigma(i)]^*$$

eine Suche in G^*, dann heißt σ^* *duale Suche* zu σ.

Es stellt sich die Frage, wann zu einer Suche eine duale Suche existiert und wie sie sich formulieren läßt. Die erste Frage wird durch das folgende Lemma geklärt. Die zweite Frage

ist weit schwieriger zu beantworten und wird uns in den folgenden Kapiteln noch intensiv beschäftigen.

Lemma 3.2.5
Sei σ eine Suche auf einem planaren Graphen $G = (V, E)$. Es existiert eine duale Suche σ^* auf $G^* = (V^*, E^*)$ genau dann, wenn die Untergraphen von G^*, die den $G_1, G_2, \ldots, G_m$ entsprechen, zusammenhängend sind.

In dem folgenden Unterabschnitt werden wir uns mit einer anderen Definition des Dualgraphen beschäftigen. Diese wird uns Aufschluß über einige grundlegende Beziehungen zwischen Graph und Dualgraph liefern. Für einen ausführlichen Einblick sei auf den genannten Artikel verwiesen. Wir werden uns hier auf die wesentlichen Definitionen und Resultate beschränken, da uns eine genauere Ausführung der von Guibas und Stolfi entwickelten Theorie zu weit von unserem Thema abbrächte.

Dualität nach Guibas und Stolfi [GS85]

In [GS85] formulieren die beiden Autoren eine völlig neue Darstellung für Graphen. Die Ebene wird von Guibas und Stolfi als zweidimensionale Mannigfaltigkeit angesehen und ein Graph ist eine feste Unterteilung dieser Mannigfaltigkeit in drei Teile, Knoten, Kanten und Gebiete. Dabei wird jede Kante eines Graphen durch zwei Felder charakterisiert, die ihre Richtung und ihre Orientierung beschreiben. Die Orientierung gibt die Seite an, also oben oder unten, von der aus die Kante betrachtet wird. Gebiete werden von einer alternierenden zusammenhängenden Folge von Knoten und Kanten umgeben, entsprechend wird ein Knoten von einer alternierenden zusammenhängenden Folge von Gebieten und Kanten umgeben. Für jede orientierte, gerichtete Kante kann eindeutig ihr Anfangsknoten $e\,Org$, ihr Endknoten $e\,Dest$, die Facette links $e\,Left$ und die Facette rechts $e\,Right$ definiert werden. Außerdem sei $e\,Flip$ als dieselbe unorientierte Kante definiert mit gleicher Richtung aber entgegengesetzter Orientierung, und $e\,Sym$ als dieselbe ungerichtete Kante mit gleicher Orientierung aber entgegengesetzter Richtung. Durch diese Darstellung ist die im Gegenuhrzeigersinn nächste Kante mit demselben Anfangsknoten $e\,Onext$ und die im Gegenuhrzeigersinn nächste Kante mit derselben linken Facette $e\,Lnext$ eindeutig bestimmt. Die fogende Definition des Dualgraphen durch Guibas und Stolfi unterscheidet sich deutlich von der in Abschnitt 2.6 vorgestellten.

Definition: Zwei Unterteilungen S und S^* sind dual zueinander, falls für jede gerichtete und orientierte Kante e einer Unterteilung eine Kante $e\,Dual$ in der anderen Unterteilung mit folgenden Eigenschaften existiert:

(D1) $(\,e\,Dual\,)\,Dual = e$
(D2) $(\,e\,Sym\,)\,Dual = (\,e\,Dual\,)\,Sym$
(D3) $(\,e\,Flip\,)\,Dual = (\,e\,Dual\,)\,Flip\,Sym$
(D4) $(\,e\,Lnext\,)\,Dual = (\,e\,Dual\,)\,Onext^{-1}$.

Gleichung D4 besagt, daß das Umrunden der Facette links von e im Gegenuhrzeigersinn in der Unterteilung S das gleiche ist, wie das Durchgehen im Uhrzeigersinn durch die Adjazenzliste des Anfangsknotens von *e Dual* in der dualen Unterteilung S^*. Dies wird klarer, wenn man sich die Folge von Kanten auf dem Rand der Facette links von e im Gegenuhrzeigersinn anschaut. Dies ist der Pfad

$$(\, e\ Lnext, e\ Lnext^2, \ldots, e\ Lnext^k = e\,),$$

für ein $k \geq 1$ welcher durch Gleichung D4 in die Folge

$$(\,(\, e\ Dual\,)\ Onext^{-1}, (\, e\ Dual\,)\ Onext^{-2}, \ldots, (\, e\ Dual\,)\ Onext^{-k} = e\ Dual\,)$$

übergeht. Das sind gerade alle Kanten im Uhrzeigersinn, die aus $v = (\, e\ Dual\,)\ Org$ von S^* auslaufen. Zum besseren Verständnis betrachte man die linke Skizze in Abbildung 3.7. Die Umkehrung des Umlaufsinns ergibt sich aus der entgegengesetzten Orientierung von *e Dual*.

Diese Definition des Dualgraphen stimmt nicht mit der Definition in Abschnitt 2.6 überein. Um eine einfache Darstellung für die Dualkante, wie sie im allgemeinen definiert wird, zu erhalten, führen die Autoren von [GS85] die Funktion

$$e\ Rot = e\ Flip\ Dual = e\ Dual\ Flip\ Sym$$

ein. Die Kante *e Rot* entspricht der von uns definierten Dualkante e^*, ist also von rechts nach links gerichtet und genauso orientiert, wie die Kante e. Daraus folgt, daß das Umrunden der Facette rechts von e im Gegenuhrzeigersinn dem Ablaufen der Adjazenzliste des Anfangsknotens von *e Rot* im Gegenuhrzeigersinn entspricht.

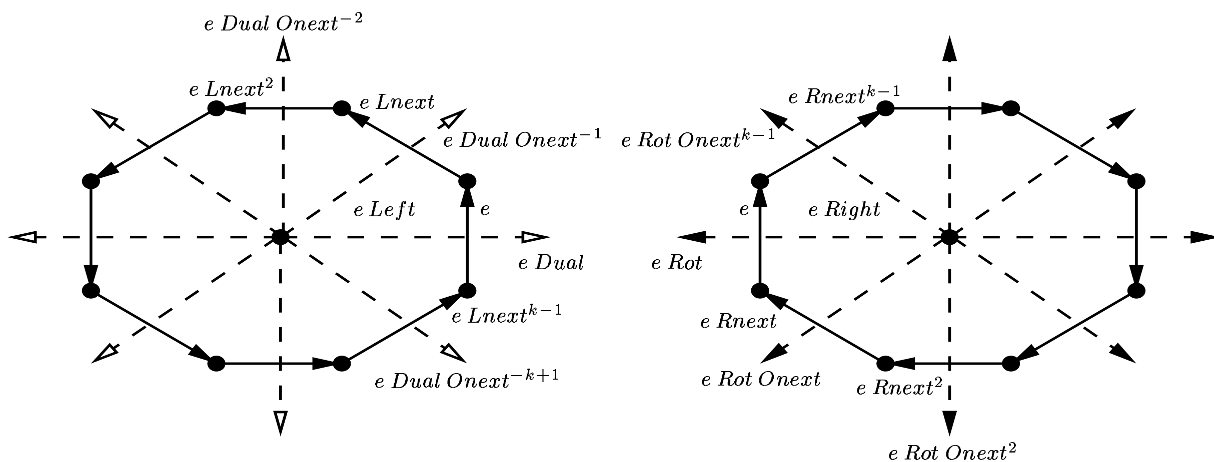


Abbildung 3.7: Das linke Bild zeigt die Facette links von e und die sie umrundenden Kanten $(\, e\ Lnext, e\ Lnext^2, \ldots, e\ Lnext^k = e\,)$ zusammen mit den dualen Kanten $(\,(\, e\ Dual\,)\ Onext^{-1}, (\, e\ Dual\,)\ Onext^{-2}, \ldots, (\, e\ Dual\,)\ Onext^{-k} = e\ Dual\,)$ (gestrichelt). Die nicht ausgefüllten Pfeile an den Dualkanten kennzeichnen die umgekehrte Orientierung der Kanten. Das rechte Bild zeigt die Facette rechts von e und die sie umrundenden Kanten $(e\ Rnext^{k-1}, \ldots, e\ Rnext, e)$. Außerdem sind die rotierten Dualkanten (gestrichelt) $(e\ Rot\ Onext^{k-1}, \ldots, e\ Rot\ Onext, e\ Rot)$ dargestellt.

Man mache sich die Unterschiede zwischen *e Dual* und *e Rot* mit Hilfe der Abbildung 3.7 klar.
Betrachten wir nun einmal den Graphen C_k bestehend aus einem einfachen Kreis. Startet man die E-DFS$_{\circlearrowleft}$ auf einem Knoten dieses Kreises, so nimmt sie die Kanten von C_k im Gegenuhrzeigersinn auf (man beachte, daß die Dummykante in der äußeren Facette liegt). Der resultierende Graph ist ein Linkskreis, die äußere Facette liegt demnach rechts des Weges und wird im Uhrzeigersinn umrundet. Startet man eine Breitensuche auf dem die äußere Facette repräsentierenden Knoten f_a des Dualgraphen C_k^* und läßt diese die zu f_a inzidenten Kanten im Uhrzeigersinn aufnehmen, so arbeiten diese beiden Suchen offensichtlich dual zueinander. Auf einem einfachen Kreis ist folglich die zur E-DFS$_{\circlearrowleft}$ duale Suche eine Breitensuche mit Left-First-Auswahlregel. Wir wollen nun untersuchen, ob diese Dualität für allgemeine Graphen gilt.

3.2.3 Left-First-Breitensuche

In diesem Abschnitt werden wir uns die Left-First-Breitensuche auf planaren Graphen ansehen. Entsprechend unserer zu Beginn des Kapitels 3.2 eingeführten Konvention werden wir sie als BFS$_{\circlearrowleft}$ bezeichnen. Der folgende Algorithmus implementiert die gesuchte Breitensuche.

Algorithmus 3.2.6 (Left-First-Breitensuche (BFS$_{\circlearrowleft}$))
Eingabe: Ungerichteter, planar eingebetteter Graph $G = (V, E)$ und ein ausgezeichneter Knoten $v_s \in V$ auf dem Rand der äußeren Facette.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$ in der äußeren Facette.

Knotenauswahl

(1) Endknoten v der Kante (u, v), die am längsten in M liegt

Kantenauswahl($v; (u, v)$)

(1) falls v aktiv
(2)   wähle die im Uhrzeigersinn von (u, v) aus nächste unbesuchte zu v inzidente Kante $\{v, w\}$ und orientiere sie: (v, w)
(3)   falls w unbesucht
(4)   lege (v, w) in M ab
(5)   gib (v, w) aus
(6) ansonsten
(7)   entferne (u, v) aus M

Erläuterung und Laufzeitanalyse: Genau wie jede andere Breitensuche auch arbeitet die BFS$_{\circlearrowright}$ die Knoten des Graphen Schicht für Schicht ab. Dabei wählt sie die zu dem aktuellen Knoten inzidenten Kanten allerdings nicht in beliebiger Reihenfolge, sondern gemäß der planaren Einbettung von links nach rechts. Da wir von einer kombinatorischen Einbettung des Graphen ausgehen, sind die Adjazenzlisten der Knoten bereits entsprechend sortiert. Dadurch erfolgt die Wahl der nächsten Kante im Uhrzeigersinn in $\mathcal{O}(1)$, indem einfach in der Adjazenzliste eine Kante weiter gegangen wird. Die Komplexität der BFS$_{\circlearrowright}$ liegt wieder in $\mathcal{O}(n)$, da die Anzahl m der Kanten eines planaren Graphen linear in n ist.

Wir werden uns nun damit beschäftigen, in welcher Reihenfolge die Kanten, die inzident zu Knoten aus einer Schicht $i > 0$ sind, bearbeitet werden. Hierzu einige Bezeichnungen: Gegeben ein planar eingebetteter Graph $G = (V, E)$. B_0 sei die Menge der Kanten auf dem Rand der äußeren Facette von $G_0 = G$. $G_1 = G(E \backslash B_0)$ bezeichne den durch $E \backslash B_0$ kanteninduzierten Untergraphen von G und B_1 die Kanten auf dem Rand der äußeren Facette von G_1. Entsprechend sind $G_0^* = G^*$ und G_1^* die Dualgraphen von G und G_1. V_i^* bezeichne die Knoten von G^*, welche die i-te Schicht bilden. Demnach besteht V_0^* also nur aus dem die äußere Facette von G repräsentierenden Knoten f_a. Zunächst zeigen wir folgendes Lemma.

Lemma 3.2.7
Gegeben ein planar eingebetteter Graph G. Eine BFS$_{\circlearrowright}$ S_0 auf G_0^* bearbeitet die Kanten aus B_1^* in derselben Reihenfolge wie eine BFS$_{\circlearrowright}$ S_1 auf G_1^* mit entsprechender Startkante.

Beweis:
Sei $\vec{e}_{0_1}^*$ die von der Startkante $\vec{e}_0$ aus im Uhrzeigersinn erste zu f_a inzidente Kante aus B_0^*. Ohne Einschränkung sei $f_{1_1} \neq f_a$ Endknoten von $\vec{e}_{0_1}^*$. Wir setzen f_{1_1} als Startknoten der Suche S_1 und initialisieren M_{S_1} mit $\vec{e}_{0_1}^*$. Die Suche S_0 habe alle zu f_a inzidenten Kanten bearbeitet, dann bezeichne $\vec{e}_{0_1}^*$ bis $\vec{e}_{0_i}^*$ die in M_{S_0} gespeicherten Referenzkanten der Knoten aus V_1^*.
Beim Entfernen des Randes der äußeren Facette verändern die Kanten aus B_1 ihre Lage zueinander nicht. Daraus folgt sofort, daß auch die Reihenfolge der Dualkanten der Kanten aus B_1 erhalten bleibt. Durch das Entfernen der Kanten aus B_0 werden die Endknoten der Kanten aus B_1^*, also die Knoten aus V_1^* zusammengezogen zu einem Knoten. Dabei werden Kanten aus B_1^*, die beide Endknoten in V_1^* haben zu Schleifen.
Die BFS$_{\circlearrowright}$ S_1 auf G_1 arbeitet die zur äußeren Facette f_{1_1} von G_1 inzidenten Kanten im Uhrzeigersinn ab. Dabei beginnt sie mit der nach $\vec{e}_{0_1}^*$ im Uhrzeigersinn ersten Kante in der Adjazenzliste von f_{1_1}.
Die BFS$_{\circlearrowright}$ S_0 ihrerseits nimmt f_{1_1} als aktuellen Knoten und wählt als erste Kante die im Uhrzeigersinn von $\vec{e}_{0_1}^*$ aus nächste Kante in der Adjazenzliste von f_{1_1}. Dies ist durch unsere Wahl der Startkante von S_1 dieselbe Kante wie die erste Kante von S_1. Danach geht S_0 die zu f_{1_1} adjazenten Kanten im Uhrzeigersinn durch. Ist f_{1_1} abgearbeitet, so nimmt S_0 f_{1_2} als aktuellen Knoten und bearbeitet dessen Adjazenzliste. Da die Knoten aus V_1^* in M im Uhrzeigersinn angeordnet sind und die zu ihnen inzidenten Kanten durch die Kantenauswahl

im Uhrzeigersinn besucht werden, ergibt sich insgesamt dieselbe Bearbeitungsreihenfolge wie bei der Suche S_1. In Abbildung 3.8 ist die beschriebene Situation an einem einfachen Beispiel dargestellt. □

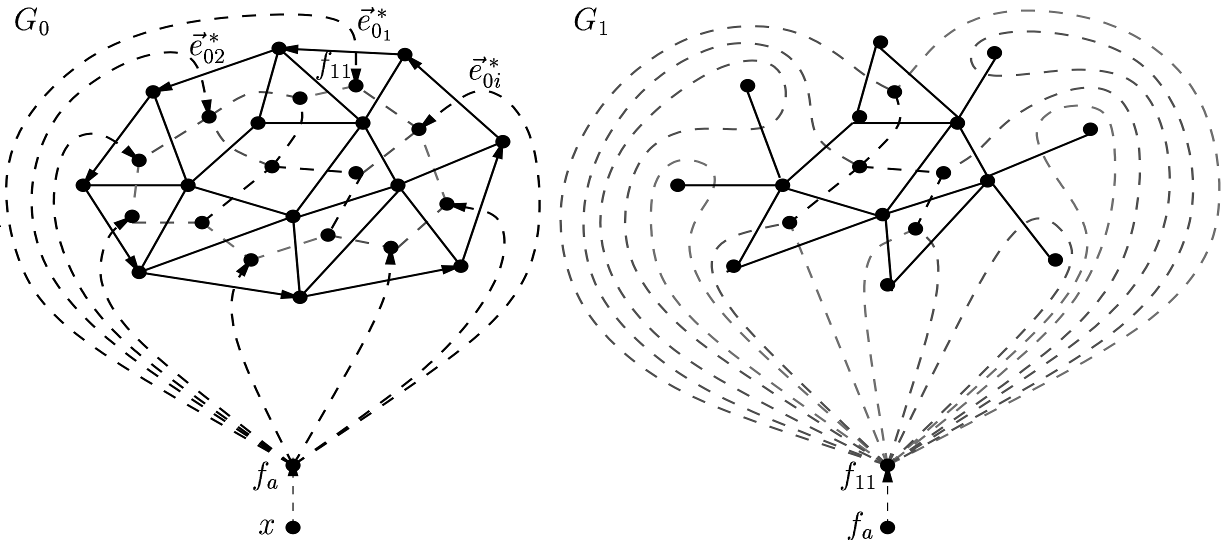


Abbildung 3.8: Graph und Dualgraph vor und nach dem Entfernen der Kanten auf dem Rand der äußeren Facette

Dieses Lemma wird uns in den folgenden Sätzen sehr nützlich sein, da wir viele Beweise durch Induktion führen. Dabei können wir nun davon ausgehen, daß eine Breitensuche die Kanten, die inzident zu Knoten einer tieferen Schicht sind, in derselben Reihenfolge bearbeitet, wie eine auf dem Dualgraphen des durch die unbesuchten Kanten induzierten Teilgraphen gestartete Breitensuche.

Kantendisjunktes Menger Problem auf dem Dualgraphen

Betrachten wir nun noch einmal das kantendisjunkte Menger Problem. Es stellt sich die Frage, ob die Aufgabenstellung auch mit Hilfe der Breitensuche auf dem Dualgraphen zu lösen ist. Hierzu schauen wir uns noch einmal den s-t-planaren Graphen aus Abschnitt 3.2.1 zusammen mit seinem Dualgraphen an. Der Dualgraph sei in diesem Fall nicht der gewöhnlich Dualgraph, sondern einer, der die äußere Facette durch zwei Knoten f_s und f_t repräsentiert. Dabei seien bei einer Orientierung des Randes der äußere Facette im Gegenuhrzeigersinn alle Dualkanten zu Kanten, die auf diesem gerichteten Kreis nach s und vor t liegen zu f_s inzident alle übrigen zu f_t. Graph und modifizierter Dualgraph sind in Abbildung 3.9 dargestellt.

Offensichtlich nimmt eine Left-First-Breitensuche auf G^* mit Startknoten f_s in dem Beispiel die Kanten in derselben Reihenfolge auf wie die s-t-Tiefensuche. Allerdings besucht sie auch die Kanten, hier durch dünne schwarze Linien ohne Pfeile gekennzeichnet, welche

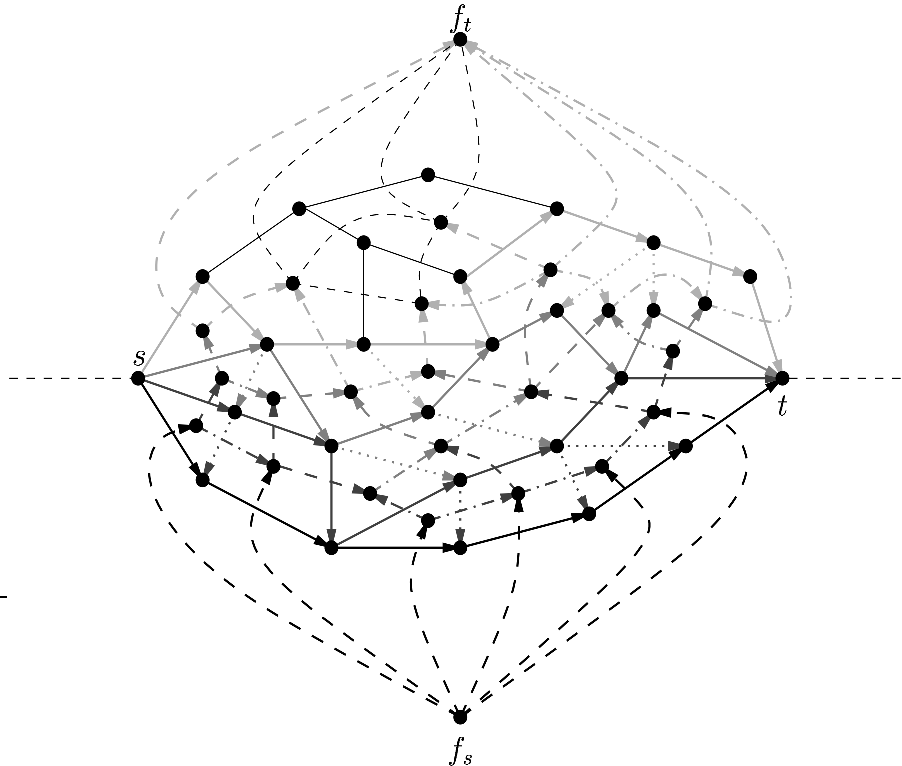


Abbildung 3.9: E-DFS$_{\circlearrowleft}$ angewendet auf s-t-planaren Graphen. Gepunktete Pfeile kennzeichnen Backtrack-Kanten. Der Dualgraph ist gestrichelt und die Nichtbaumkanten im Dualgraphen sind Punkt-Strich-Pfeile.

die Tiefensuche nicht findet. Um eine zur s-t-Tiefensuche duale Suche zu erhalten, muß die Breitensuche modifiziert werden.
Wir werden später duale Suchen zur Tiefen- und zur Breitensuche angeben und uns in diesem Beispiel darauf beschränken, eine Breitensuche zu beschreiben, welche die Dualkanten der s-t-Wege und der s-Cycle bestimmt, also dasselbe Ergebnis liefert wie die s-t-Tiefensuche. Dabei werden Kanten, die nicht zu einem der s-t-Pfade oder s-Cycle gehören, von der Tiefen- und der Breitensuche teilweise unterschiedlich behandelt.
Der Beispielgraph in Abbildung 3.9 legt die Vermutung nahe, daß die Kanten, deren Anfangsknoten in Schicht i und Endknoten in Schicht $i+1$ liegen, gerade alle Dualkanten (in der richtigen Reihenfolge) des i-ten s-t-Pfades sind. Wir werden sehen, daß diese Vermutung im allgemeinen falsch ist. Die Dualkanten der Kanten, deren Endknoten in derselben Schicht liegen, sind entweder Backtrack-Kanten der s-t-Tiefensuche, oder Kanten, welche die Tiefensuche nicht gefunden hat, da sie links eines Pfades liegen und keine Verbindung zur rechten Seite des nächsten Pfades besitzen.
Der folgende Algorithmus ist eine modifizierte Breitensuche, welche die Dualkanten der s-t-Wege und s-Cycle in G ausgibt. Die Kanten eines jeden Weges und Cycles werden dabei in der gleichen Reihenfolge ausgegeben wie durch die s-t-Tiefensuche.

Algorithmus: (s-t-Breitensuche)
Eingabe: Ungerichteter, planar eingebetteter Graph $G = (V, E)$ und zwei ausgezeichnete Knoten $v_s, v_t \in V$.
Initialisierung: $M := (x, v_s)$ mit Dummyknoten $x \notin V$ in der äußeren Facette.
Knoten-Array $LEVEL$, $LEVEL[v_s] := 0$, $LEVEL_{max} := LEVEL[v_s] + 1$.
Knoten-Array $LAST$, $LAST[v_s] := (v_s, x)$.

Knotenauswahl

(1) Endknoten v der Kante (u, v), die am längsten in M liegt
(2) falls $LEVEL[v] = LEVEL_{max}$
(3)   beende Prozedur

Kantenauswahl($v; (u, v)$)

(1) falls die von $LAST[v]$ aus im Uhrzeigersinn nächste zu v inzidente Kante e noch unbesucht
(2)   wähle diese Kante $e = \{v, w\}$ und orientiere sie: (v, w)
(3)   falls $v = w$
(4) $LAST[v] := (w, v)$
(5)   ansonsten
(6) $LAST[v] := (v, w)$
(7)   falls w noch unbesucht
(8) $LAST[w] := (w, v)$
(9) $LEVEL[w] := LEVEL[v] + 1$
(10)   lege (v, w) in M ab
(11)   falls $w = f_t$
(12) $LEVEL_{max} := LEVEL[w]$
(13)   ansonsten
(14) $LEVEL_{max} := LEVEL[w] + 1$
(15)   ansonsten falls $LEVEL[w] = LEVEL[v] + 1$
(16)   sei (x, y) Kante, die zuletzt in M eingefügt wurde
(17)   solange $y \neq w$
(18)   entferne (x, y) aus M
(19)   falls $LEVEL[w] = LEVEL[v] + 1$
(20)   lege (v, w) in Pfad $P_{LEVEL[w]}$ ab

(21) ansonsten falls die von $LAST[v]$ aus im Uhrzeigersinn nächste zu v inzidente Kante $e = (w, v)$ besucht und $LEVEL[w] = LEVEL[v]$

(22) $\quad LAST[v] := (v, w)$

(23) ansonsten

(24) $\quad$ entferne (u, v) aus M

Erläuterung: In $LEVEL[v]$ wird die Schicht eines besuchten Knotens v gespeichert und in $LEVEL_{max}$ die Schicht von v_t, sobald v_t gefunden wurde. Außerdem wird in $LAST[v]$ die zuletzt von v aus gefundene Kante gespeichert. Die s-t-Breitensuche arbeitet im wesentlichen wie die BFS$_{\circlearrowright}$ auf einem planaren Graphen mit drei Modifikationen.
Mod. 1) Vom aktuellen Knoten aus werden nicht mehr unbedingt alle inzidenten Kanten bearbeitet. Schritt (1) der Kantenauswahl wurde folgendermaßen abgeändert. Sei v der aktuelle Knoten und (u, v) seine Referenzkante, also die Kante, über die er zum ersten Mal besucht wurde. Sei weiterhin $LEVEL[u] = i$ und somit $LEVEL[v] = i + 1$. Bevor v als aktueller Knoten bearbeitet wird ist $LAST[v] = (v, u)$. Ist die im Uhrzeigersinn von $LAST[v]$ aus nächste zu v inzidente Kante $e = \{v, w\}$ unbesucht, dann wird sie weiter bearbeitet, ist sie jedoch besucht, also $e = (w, v)$, dann sind zwei Fälle zu unterscheiden.
Fall a) $LEVEL[w] = LEVEL[v]$, dann ist e eine Nichtbaumkante der Breitensuche und verläuft zwischen Knoten derselben Schicht. Kante e wird in den Schritten (21) und (22) einfach übergangen.
Fall b) $LEVEL[w] = LEVEL[v] - 1$, dann ist e eine Kante, über die v von einem Knoten aus Schicht i gefunden wurde ($u = w$ möglich). Tritt dieser Fall bei der Bearbeitung von v zum erstem Mal ein, so ist $e = (w, v)$ die letzte Kante, über die v von einem Knoten der Schicht i gefunden wurde. Die Bearbeitung von v als aktuellen Knoten wird durch das Entfernen seiner Referenzkante aus M in den Schritten (23) und (24) beendet.
Diese Abänderung der Breitensuche bewirkt, daß nur noch die Kanten im Uhrzeigersinn zwischen der Referenzkante, also der Kante, über die v zum ersten Mal von einem Knoten aus Schicht i gefunden wurde, und der Kante, über die v zum letzten Mal von einem Knoten aus Schicht i gefunden wurde, von der s-t-Breitensuche bearbeitet werden.
Mod. 2) Die zweite Änderung betrifft die Schritte (13) bis (16) der s-t-Breitensuche. Findet die Suche einen Knoten v aus Schicht $i + 1$ zum zweiten Mal von einem Knoten aus Schicht i, so löscht sie alle in M gespeicherten Kanten, die nach der Kante eingefügt wurden, über die v zum ersten Mal gefunden wurde. Sei (u, v) die in M gespeicherte Referenzkante von v und w der aktuelle Knoten. Es ist $LEVEL[u] = LEVEL[w] = i$ und $LEVEL[v] = i + 1$.
Modifikation 1 und 2 bewirken zusammen, daß Teile des Graphen, die durch die Dualkanten des i-ten s-t-Weges und zwei Kanten (u, v), (w, v) vom Rest des Graphen abgetrennt werden, von der s-t-Breitensuche weder von "unten" besucht werden, das heißt über die Endknoten der Dualkanten des i-ten s-t-Weges, noch von "oben", also von v aus, bei Abarbeitung der Knoten aus Schicht $i + 1$.
Mod. 3) Findet die s-t-Breitensuche an dem aktuellen Knoten v eine Schleife, dann ist im Dualgraphen das Innere dieser Schleife nur über die Dualkante der Schleife mit dem Rest

des Graphen verbunden. Man erinnere sich daran, daß die Dualkante einer Schleife eine Brücke ist. Die innerhalb der Schleife liegenden Kanten können demnach nicht zu einem s-t-Weg gehören. Deshalb überspringt die s-t-Breitensuche in Schritt (4) durch entsprechendes Setzen von $LAST$ alle Kanten innerhalb der Schleife und fährt mit der Bearbeitung von v fort.

Laufzeitanalyse: Die s-t-Breitensuche ist im wesentlichen eine Breitensuche. Die Modifikationen bewirken nur, daß Kanten erst gar nicht bearbeitet werden. Im Gegensatz zur Left-First-Breitensuche werden manche Kanten zweimal angeschaut. Ist dies der Fall, so werden sie allerdings sofort übergangen und oft zusätzlich andere noch unbesuchte Kanten ebenfalls (siehe Modifikation 1 und 2). Da die s-t-Breitensuche auf planaren Graphen arbeitet, ist ihre Komplexität wieder in $\mathcal{O}(n)$, wie auch die der s-t-Tiefensuche.

Lemma 3.2.8
Die s-t-Breitensuche angewendet auf den oben beschriebenen Dualgraphen von G liefert die Dualkanten der s-t-Pfade, welche die s-t-Tiefensuche in G findet. Eventuell vorhandene s-Cycle werden durch die s-t-Breitensuche dem nächsten s-t-Weg vorangestellt.

Beweis:
Der Beweis erfolgt in zwei Schritten. Zuerst wird gezeigt, daß der erste Pfad von beiden Suchen korrekt gefunden wird. Danach zeigen wir, daß für den zweiten Pfad dieselbe Situation vorliegt, wie für den ersten Pfad.
1. Pfad: Sei $\vec{e}_{1_1}$ rechteste zu v_s inzidente Kante. Wir wissen bereits, daß das Umrunden im Uhrzeigersinn der Facette rechts von $\vec{e}_{1_1}$ dasselbe ist, wie das Durchlaufen der Adjazenzliste im Uhrzeigersinn von $\vec{e}_{1_1}$ $Rot\, Org = f_s$. Da v_t auf dem Rand der äußeren Facette liegt, welche umrundet wird, finden die beiden Suchen die Kanten zwischen v_s und v_t in derselben Reihenfolge. In v_t stoppt die Tiefensuche nach Voraussetzung. In G^* sind genau die Dualkanten der Kanten zwischen v_s und v_t inzident zu f_s. Es gibt also keine weiteren Kanten von Knoten aus Schicht 0 nach Knoten aus Schicht 1 außer den schon besuchten. Findet die Tiefensuche auf diesem Weg wieder s, so gehören diese Kanten zu einem s-Cycle und die s-t-Tiefensuche startet wieder bei s um den ersten s-t-Pfad zu finden. Die Breitensuche kann nicht unterscheiden, ob der Weg zu s zurück führt oder in t endet. Deshalb werden die Kanten eines s-Cycles durch die Breitensuche vor den Kanten des nächsten s-t-Weges ausgegeben. Sei nun der 1. s-t-Pfad gefunden.
Da die s-t-Tiefensuche eine Right-First-Tiefensuche ist, sind nach Auffinden des 1. Pfades nur die Pfadkanten selbst und Kanten, die rechts des Weges liegen, besucht. Für die Kanten, die rechts des Weges liegen, gilt die folgende Aussage.
Beh: Die Zusammenhangskomponenten des Teilgraphen von G, der besucht ist, aber keine Wegkante enthält, werden jeweils durch eine Brücke vom Weg getrennt.
Bew: Angenommen dies ist nicht der Fall, so existieren für eine dieser Zusammenhangskomponenten zwei Kanten, die jeweils genau einen Endknoten mit dem s-t-Pfad gemeinsam haben. Da die Zusammenhangskomponente aber rechts des Pfades liegt, geht die s-t-Tiefensuche zuerst über eine der beiden Kanten in die Zusammenhangskomponente und

später über die andere der beiden Kanten wieder auf den Weg zurück, von wo aus sie über das restliche Teilstück des Pfades v_t findet, im Wiederspruch zur Voraussetzung. □
Die Dualkante einer Brücke ist eine Schleife in G^*. Findet die s-t-Breitensuche diese Schleife (u, v), so gilt $u = v$ und durch Modifikation 3 werden alle Kanten in der Schleife übersprungen. Dadurch werden weder die Schleife noch die Kanten, die innerhalb der Schleife liegen, als Pfadkanten abgespeichert. Man betrachte die Schleife $\vec{e}_2^*$ in Abbildung 3.10 und den durch sie eingeschlossenen Untergraphen von G. Dieser wird von der Tiefensuche durchsucht, dann aber wieder durch Backtrack Schritte aus M entfernt. Die Breitensuche "weiß", daß innerhalb einer Schleife kein Weg nach v_t führt und kann deshalb den durch sie abgetrennten Teilgraphen überspringen.
Ausgangslage für den 2. Pfad: Betrachten wir zuerst Graph G nach Auffinden von Pfad P_1. Sei $G_0 = G$ und G_1 die s und t enthaltende Zusammenhangskomponente, die entsteht, indem man P_1 aus G_0 entfernt. Kanten und Knoten, die beim Entfernen von P_1 nicht mehr von s und t aus erreichbar sind, fallen bei diesem Prozeß weg. Dies sind Kanten, die links an den 1. Pfad angrenzen und nicht mehr über unbesuchte Kanten mit s oder t verbunden sind. Als Beispiel betrachte man die dünnen Kanten in Abbildung 3.10. Teilgraph G_1 ist zusammenhängend und enthält genau einen s-t-Pfad weniger als G_0. Eine s-t-Tiefensuche gestartet auf G_1 findet exakt die gleichen Kanten und demnach auch den gleichen Pfad, wie die s-t-Tiefensuche auf G_0 bei ihrem zweiten Durchgang, da G_1 dieselben Kanten und Knoten enthält, wie G_0 noch unbesuchte Kanten, bis auf Kanten, die durch den Pfad 1 vom Rest des Graphen G_0 abgetrennt werden und somit im 2. Durchlauf von der Tiefensuche ohnehin nicht mehr gefunden werden können.
Betrachten wir nun die Situation in G^* nach Bearbeitung des ersten Pfades: Alle Facetten, die links an Kanten des Pfades P_1 angrenzen, sind besucht. Das Entfernen von P_1 in G bewirkt in G^* ein Zusammenziehen der schon besuchten Facetten zu einer Facette. Wir nennen sie wieder f_s. Was passiert nun mit den Kanten und Knoten, die durch f_s vom Rest des Graphen und somit auch von f_t getrennt werden. Diese Kanten entsprechen den Kanten in G, die durch Entfernen des Pfades P_1 von der s und t enthaltenden Zusammenhangskomponente abgetrennt werden. Im Gegensatz zur Tiefensuche kann die Breitensuche die Kanten von f_s aus noch finden, da G^* durch das Entfernen von P_1 aus G nicht zerfällt. Man überlege sich, daß Kanten dann abgetrennt werden, wenn die s-t-Breitensuche in G^* einen Knoten aus Schicht 1 zum 2. Mal findet. Existieren Mehrfachkanten in G^* von f_s aus zu einer Facette f aus Schicht 1, so bilden diese einen Schnitt der Größe 2 und sie trennen alles innerhalb von f_t ab. Zum besseren Verständnis betrachte man Abbildung 3.10. Die Kanten $\vec{e}_1^*$ und $\vec{e}_6^*$ inzident zu f_s ganz links und ganz rechts haben denselben Endknoten f und bilden einen Schnitt der Größe 2 in G^*. Hier kommen die Modifikationen 1 und 2 zum Tragen. Diese bewirken, wie schon erwähnt, daß der Teilgraph innerhalb des Schnittes weder von f_s aus noch von f aus weiter bearbeitet wird. Durch Modifikation 2 wird in unserem Beispiel die Kante $\vec{e}_3^*$ aus M entfernt, die zu einer Facette innerhalb des Schnittes inzident ist. Die Kanten $\vec{e}_4^*$ und $\vec{e}_5^*$ sind natürlich nicht in M, da sie denselben Endknoten wie $\vec{e}_3^*$ haben.
Die 3. Modifikation sorgt dafür, daß alle Untergraphen rechts eines Weges, die nur über eine Brücke mit dem Weg verbunden sind, von der Breitensuche erst gar nicht abgesucht

werden. Aus Lemma 3.2.7 folgt, daß die s-t-Breitensuche den 2. Pfad genauso abarbeitet, wie eine s-t-Breitensuche gestartet auf dem Dualgraph von G_1. Die korrekte Ausgabe aller anderen Pfade ergibt sich per Induktion.

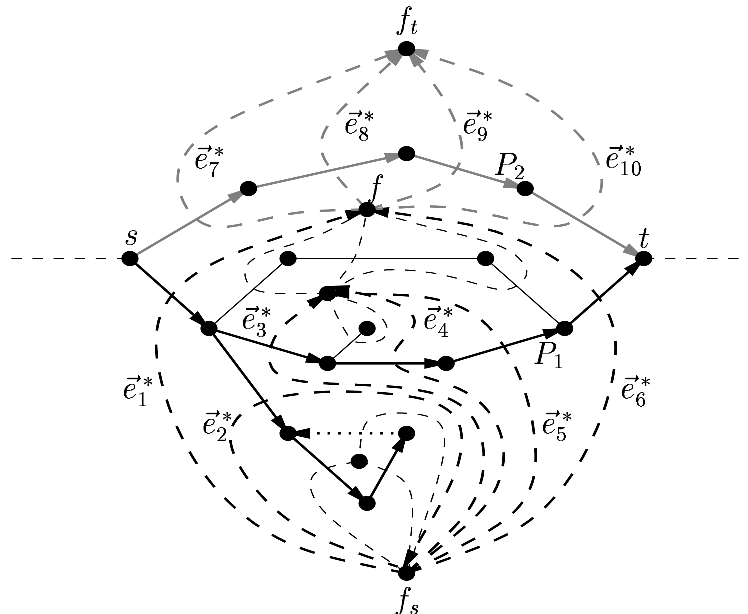


Abbildung 3.10: Kanten links des s-t-Pfades P_1, die nicht von Pfad P_2 aus erreicht werden können und Kanten rechts des Weges, die nicht zu einem s-t-Pfad gehören.

Ende des Algorithmus: Solange es noch s-t-Pfade gibt, werden diese von den beiden Suchen gefunden und aufgelistet. Sind alle Pfade gefunden, aber noch unbesuchte Kanten inzident zu v_s, so fährt die s-t-Tiefensuche solange in G fort, bis alle noch unbesuchten zu v_s inzidenten Kanten bearbeitet sind. Dabei wird kein weiterer Pfad ausgegeben, da v_t nicht mehr erreicht werden kann. Ist v_s nicht mehr aktiv, endet die s-t-Tiefensuche.
Die Breitensuche dagegen "weiß", wann alle Pfade gefunden wurden. Genau dann, wenn f_t zum ersten Mal erreicht wird, hat die Suche einen kürzesten f_s-f_t-Weg bestimmt. Die Dualkanten eines kürzester f_s-f_t-Weges bilden in G eine minimale s und t trennende Kantenmenge. Nach dem Satz von Menger, siehe zum Beispiel [Jun94], ist die Maximalzahl kantendisjunkter s-t-Wege gleich der Kardinalität einer minimalen s und t trennenden Kantenmenge. Demnach können keine weiteren s-t-Pfade existieren, sobald die Breitensuche f_t gefunden hat. $LEVEL_{max}$ wird dann auf $LEVEL[f_t]$ gesetzt, und die Knotenauswahlregel beendet die Prozedur sobald von einem Knoten aus $LEVEL_{max}$ weitergesucht werden soll, was impliziert, daß alle Knoten aus der Schicht $LEVEL_{max} - 1$ vollständig bearbeitet sind. Wie man sieht, stoppen die beiden Suchen im allgemeinen nicht zur selben Zeit. Sie liefern aber dieselben s-t-Pfade und s-Cycle. Die Kanten eines jeden Weges und Cycles werden außerdem in derselben Reihenfolge aufgelistet. □

Kapitel 4

Dualität von Tiefen- und Breitensuche

In diesem Kapitel wollen wir uns intensiv mit der Dualität der Tiefen- und Breitensuche auf ungerichteten, planaren Graphen in einer festen planaren Einbettung auseinandersetzen. Es wird sich zeigen, daß im allgemeinen Tiefen- und Breitensuche nicht dual zueinander sind. Wir werden deshalb darlegen, unter welchen Voraussetzungen an den Graphen und seine planare Einbettung Dualität vorliegt. Außerdem wollen wir eine modifizierte Breitensuche vorstellen, die dual zur Kanten-Tiefensuche ist und eine modifizierte Kanten-Tiefensuche, die ihrerseits dual zur Breitensuche ist.
Zuerst einmal aber wollen wir die Struktur eines planar eingebetteten Graphen betrachten. Da wir uns in diesem Kapitel ausschließlich mit planaren Graphen beschäftigen werden, gehen wir in Zukunft davon aus, daß ein gegebener Graph $G = (V, E)$, sofern nicht explizit anders bezeichnet, ungerichtet, zusammenhängend und planar eingebettet ist.

4.1 Der Schachtelungsbaum

Definition 4.1.1 (Rand)
Gegeben ein Graph $G = (V, E)$. Die Menge der Kanten $B \subset E$, die den Rand der äußeren Facette bilden, heißt *Rand* (engl. border) des Graphen.

Eine Kante $e \in E$, durch deren Wegnahme der Graph in zwei Komponenten zerfällt, die jeweils wenigstens einen Kreis enthalten, heißt *wesentliche Brücke*. Ein Rand, dessen Brücken alle wesentliche sind, heißt *einfach*. Der durch die nicht wesentlichen Brücken induzierte Untergraph F von G ist ein Wald. Den Knoten $r \in V$, den eine Zusammenhangskomponente T von F mit dem einfachen Rand gemeinsam hat, fassen wir wieder als Wurzel des Baumes T auf.

Definition 4.1.2 (Antenne)
Sei T eine Zusammenhangskomponente von F und r die Wurzel von T. Die durch die Wurzel r voneinander getrennten Zusammenhangskomponenten von T werden als *Antennen* bezeichnet.

Jede Antenne ist also wieder ein Wurzelbaum. Die Wurzel r einer Antenne ist der Knoten, den sie mit dem einfachen Rand gemeinsam hat. Zwei Antenne können denselben Knoten als Wurzel haben, die Wurzel einer Antenne hat aber immer nur genau einen Nachfolger in der Antenne. Ein Rand besteht also aus dem einfachen Rand und den Antennen. Zum besseren Verständnis der Definitionen schaue man sich Abbildung 4.1 an.

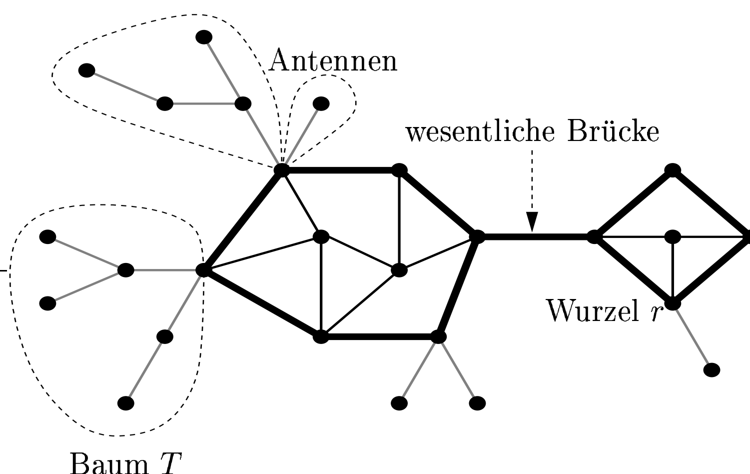


Abbildung 4.1: Planarer Graph G mit einer wesentlichen Brücke. Die grauen Kanten induzieren den Wald F der nicht wesentlichen Brücken. Der einfache Rand von G ist durch dicke Kanten markiert.

Definition 4.1.3 (Schachtelungsbaum)
Gegeben ein Graph $G = (V, E)$. Der *Schachtelungsbaum* von G ist ein Wurzelbaum T^G, dessen Knoten Ränder von Graphen repräsentieren. Der Rand B von G bildet die Wurzel r von T^G. Die Schachtelungsbäume der Zusammenhangskomponenten des durch $E \backslash B$ kanteninduzierten Untergraphen $G(E \backslash B)$ von G bilden die direkten Nachfolger von r.

Diese Definition des Schachtelungsbaumes ist induktiv. Wenn wir in Zukunft von den Rändern eines Graphen G reden, so meinen wir damit seinen Rand und die Ränder der Untergraphen von G, die durch die Knoten des Schachtelungsbaumes T^G repräsentiert werden. Abbildung 4.2 zeigt die Unterteilung eines Graphen in seine Ränder und den zugehörigen Schachtelungsbaum. Diese strukturelle Zerlegung der planaren Einbettung eines Graphen wird uns die später folgenden Analysen und Beweise vereinfachen.

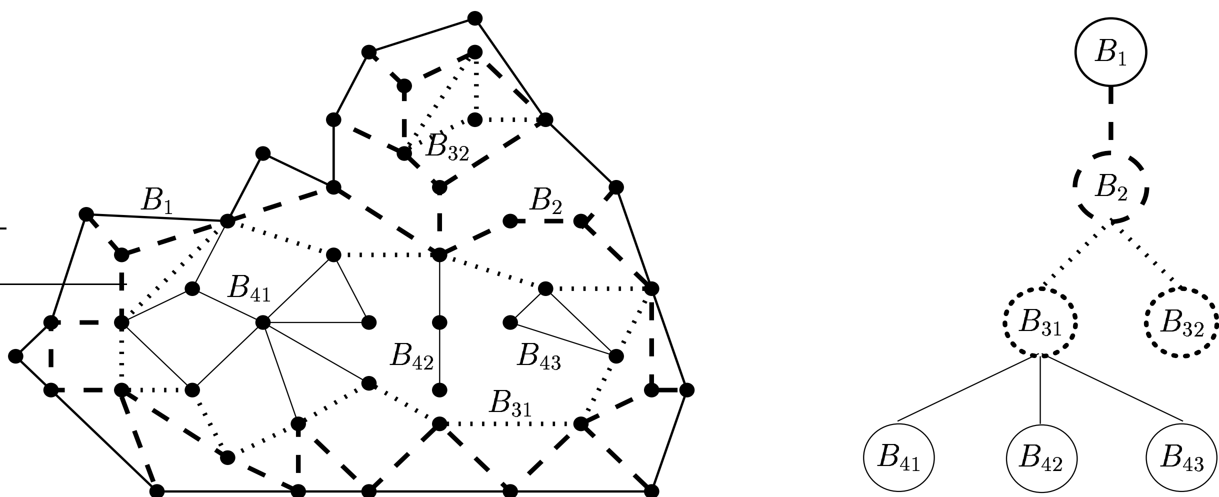


Abbildung 4.2: Planar eingebetteter Graph und sein Schachtelungsbaum T^G.

4.2 E-DFS$_\circlearrowleft$ und BFS$_\circlearrowright$ sind nicht dual

In diesem Abschnitt werden wir zeigen, daß die Right-First-Kanten-Tiefensuche auf beliebig planar eingebetteten Graphen nicht dual ist zur Left-First-Breitensuche. Wir werden außerdem sehen, daß auf Graphen, deren planare Einbettung bestimmte Eigenschaften erfüllt, die beiden Suchen dual sind.

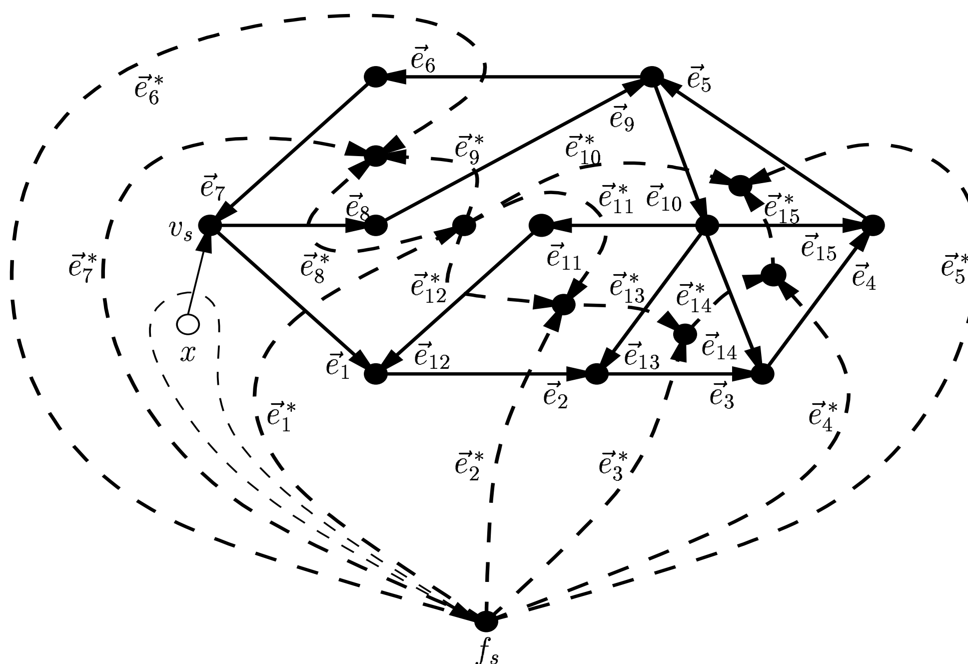


Abbildung 4.3: E-DFS$_\circlearrowleft$ auf Graph G und BFS$_\circlearrowright$ auf dem Dualgraphen (gestrichelt)

Abbildung 4.3 zeigt einen planar eingebetteten, ungerichteten, zusammenhängenden Graphen $G = (V, E)$ und seinen Dualgraphen $G^* = (V^*, E^*)$. Der Startknoten aus V ist mit v_s markiert. Die Zahlen und Pfeile an den Kanten des Graphen G entsprechen dem Alter und der Orientierung der Kanten gemäß einer E-DFS$_{\circlearrowleft}$ mit Startknoten v_s. Die Zahlen und Pfeile an den (gestrichelten) Kanten des Dualgraphen G^* entsprechen dem Alter und der Orientierung der Kanten gemäß einer BFS$_{\circlearrowright}$ mit Startknoten f_s. Die Dummykante ist jeweils dünn eingezeichnet. In diesem einfachen Beispiel produzieren die beiden Suchen die gleiche Sortierung der Kanten.

Lemma 4.2.1
E-DFS$_{\circlearrowleft}$ und BFS$_{\circlearrowright}$ sind nicht dual.

In Abbildung 4.4 ist ein Graph abgebildet, bei dem sich die Aufnahmereihenfolgen der Kanten bei der Tiefen- und Breitensuche unterscheiden. Die Kanten, die von den Suchen zu unterschiedlichen Zeitpunkten besucht werden, sind dick gezeichnet. Auf beliebigen Graphen arbeiten E-DFS$_{\circlearrowleft}$ und BFS$_{\circlearrowright}$ demnach nicht dual.

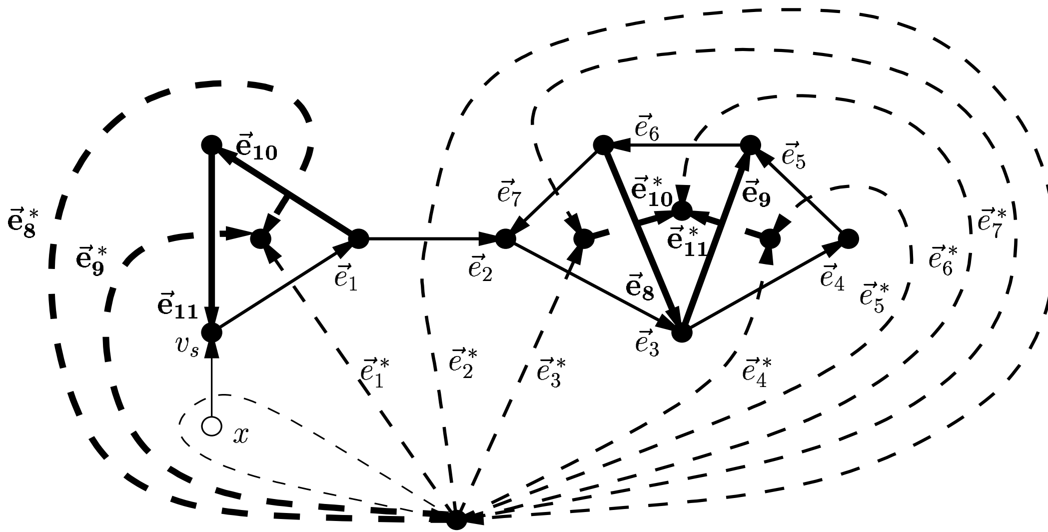


Abbildung 4.4: E-DFS$_{\circlearrowleft}$ auf Graph und BFS$_{\circlearrowright}$ auf dem Dualgraphen (gestrichelt)

Offensichtlich ist in dem Graphen aus Abbildung 4.4 die Brücke e_2 dafür verantwortlich, daß sich die Suchen zum Zeitpunkt $t = 8$ für zwei verschiedene Kanten entscheiden. Seien $Komp_1$ und $Komp_2$ die beiden durch die Brücke e_2 getrennten Zusammenhangskomponenten von G. Sobald e_2 von der Tiefensuche bearbeitet wird, gibt es keinen unbesuchten Weg mehr von der einen Komponente in die andere. Befindet sich die Tiefensuche nach Bearbeitung der Brücke in $Komp_2$, dann wird sie erst diese Komponente vollständig bearbeiten, bevor sie durch entsprechend viele Backtrack Schritte wieder zu einer Kante aus $Komp_1$ zurückkehrt. Die Breitensuche dagegen durchsucht den Graphen schichtweise. Sie arbeitet sich also von der äußeren Facette aus gleichmäßig in den Graphen hinein. Dabei werden die Komponenten $Komp_1$ und $Komp_2$ immer im Wechsel bearbeitet.

Sei $G_i^{\complement} := G(E \backslash \vec{E}_i)$ der kanteninduzierte Untergraph von G, der entsteht, wenn man die bis zum Zeitpunkt i gefundenen Kanten $\vec{E}_i$ aus E entfernt. Offensichtlich arbeiten Tiefen- und Breitensuche nicht dual, wenn der Graph $G_i^{\complement}$ in mehrere Zusammenhangskomponenten zerfällt. Es stellt sich die Frage, ob man die Graphen, auf denen sich Tiefen- und Breitensuche dual zueinander verhalten, charakterisieren kann.
Hierzu werden wir uns den Schachtelungsbaum des Graphen $G = (V, E)$ anschauen. Vorab noch eine Bezeichnung. Sei $v_s \in V$ (auf dem Rand der äußeren Facette) Startknoten der E-DFS$_{\circlearrowleft}$ in G. Wir wollen den Startknoten des Randes B_i eines Graphen definieren als denjenigen Knoten v_{s_i}, von dem aus die Tiefensuche zum ersten Mal eine Kante $e \in B_i$ des Randes wählt. Folglich ist $v_s = v_{s_0}$ Startknoten des Randes $B = B_0$ von G. Für die Ränder der Zusammenhangskomponenten des kanteninduzierten Untergraphen $G(E \backslash B)$ hängen die Startknoten von v_s ab und sind eindeutig bestimmt, da die E-DFS$_{\circlearrowleft}$ durch die Right-First-Auswahlregel nach der Wahl der Startkante deterministisch verläuft.

Satz 4.2.2
Gegeben ein Graph $G = (V, E)$. Bildet der Schachtelungsbaum von G einen einfachen Weg und erfüllen die Startknoten der Ränder von G die folgende Bedingung:

> Liegt der Startknoten eines Randes auf einer Antenne, so ist er das rechteste Blatt dieses Wurzelbaumes.

Dann sind E-DFS$_{\circlearrowleft}$ und BFS$_{\circlearrowleft}$ dual auf G.

Beweis:
In dem Satz werden zwei Forderungen an die Einbettung des Graphen $G = (V, E)$ gestellt. Erstens muß der Schachtelungsbaum einen einfachen Weg bilden, das heißt, er darf sich nicht verzweigen. Zweitens müssen die Startknoten aller Ränder des Graphen eine bestimmte Voraussetzung erfüllen.
Bezeichne $G_0 = (V_0, E_0) = G$ den Ausgangsgraphen und B_0 seinen Rand. Für $i = 1, \ldots, k$ sei $E_i := E_{i-1} \backslash B_{i-1}$ und $V_i := V(E_i)$. Ist der Schachtelungsbaum eines Graphen unverzweigt, so sind alle Untergraphen $G_i = (V_i, E_i) = G(E_{i-1} \backslash B_{i-1})$ jeweils mit Rand B_i, $i = 1, \ldots, k$ zusammenhängend. Wir haben deshalb zwei Dinge zu zeigen:
a) Die Vorraussetzung an die Startknoten der Ränder ist notwendig und hinreichend.
b) Die Aufnahmereihenfolge der Kanten des Randes eines zusammenhängenden Graphen ist bei den beiden Suchen gleich. Außerdem ist die Ausgangslage für den nächsten Rand die gleiche wie für den gerade bearbeiteten.
Teil a) Wir werden uns zuerst einmal die Voraussetzung an den Startknoten eines Randes anschauen. Liegt der Startknoten v_{s_i} des i. Randes, $0 \leq i \leq k$, auf einer Antenne, so ist er das rechteste Blatt dieses Wurzelbaumes. Nach Voraussetzung existiert auf dem gerichteten r-v_{s_i}-Pfad von der Wurzel r_i der Antenne zu dem Startknoten des Randes keine Kante, die rechts des Weges liegt. Betrachtet man den Pfad von v_{s_i} nach r_i, so liegt keine Kante links des gerichteten Weges. Alle Unterbäume, die Nachfolger von Knoten auf dem Weg von v_{s_i} nach r_i sind, hängen demnach rechts des Weges. Wenn die E-DFS$_{\circlearrowleft}$ vom Startknoten v_{s_i} aus ihre Suche beginnt, wird sie durch die Right-First-Auswahlregel alle

Kanten der Antenne bearbeiten. Hat die Tiefensuche die Wurzel der Antenne erreicht, so ist der durch die zu diesem Zeitpunkt noch unbesuchten Kanten induzierte Untergraph $G_k^{\complement}$ zusammenhängend. Die Bedingung ist somit hinreichend. Im folgenden werden wir sehen, daß sie auch notwendig ist.

Angenommen der Startknoten v_{s_i} eines Randes B_i liegt auf einer Antenne und ist nicht das rechteste Blatt von dieser. Dann existiert eine Kante e, die links des Weges von v_{s_i} zur Wurzel r_i der Antenne liegt. Diese Kante wird von der Tiefensuche auf dem Weg zur Wurzel r_i nicht gefunden und kann ab da erst wieder durch Backtrack Schritte nach Bearbeitung des Randes und allem, was innerhalb liegt, erreicht werden. Die Breitensuche dagegen findet e bei der Bearbeitung der Adjazenzliste der Facette rechts des v_{s_i}-r_i-Pfades. Diese Facette ist die letzte Facette außerhalb des Randes B_i und wird bearbeitet bevor die Breitensuche innerhalb von B_i weitersucht. Somit verhalten sich die beiden Suchen nicht dual.

Teil b) Betrachten wir nun die Bearbeitung des Randes B_0 des Graphen G. Die E-DFS$_{\circlearrowleft}$ läuft den Rand der äußeren Facette im Uhrzeigersinn ab. Wir definieren $\hat{v}_{s_i} := r_i$, wobei r_i die Wurzel der Antenne ist, auf der v_{s_i} liegt, falls v_{s_i} auf einer Antenne liegt und $\hat{v}_{s_i} := v_{s_i}$, sonst. Sind alle Kanten aus B_0 von der Tiefensuche abgearbeitet worden, dann ist ohne Einschränkung $\hat{v}_{s_0}$ aktueller Knoten. Gehören die letzten unbesuchten Kanten zu einer Antenne, so gilt der Rand erst dann als vollständig bearbeitet, wenn alle diese Baumkanten durch Backtrack Schritte wieder aus M_{DFS} entfernt wurden. Ist B_0 abgearbeitet, dann ist $\hat{v}_{s_0}$ aktueller Knoten und die Kanten des einfachen Randes $\hat{B}_0$ sind in M_{DFS} gespeichert. Die Kanten der Antennen sind besucht und durch Backtrack Schritte aus M_{DFS} entfernt worden.

Die BFS$_{\circlearrowleft}$ geht entsprechend die zur äußeren Facette f_s inzidenten Kanten im Uhrzeigersinn durch. Wir wissen bereits aus Abschnitt 3.2.2, daß die Kantenreihenfolge dabei die gleiche ist wie beim Umrunden des Randes der Facette im Uhrzeigersinn. Sind alle zu f_s inzidenten Kanten besucht, dann sind die Endknoten der Dualkanten der Kanten aus $\hat{B}_0$ in M_{BFS} abgelegt. Die Dualkanten der Kanten der Antennen sind Schleifen und wurden somit nicht in M_{BFS} gespeichert. Insgesamt ergibt sich, daß die Tiefen- und die Breitensuche auf B_0 dual arbeiten.

Wir haben nun noch zu zeigen, daß für die Bearbeitung des Randes B_1 dieselbe Ausgangslage vorliegt wie für B_0. Dazu betrachten wir zuerst den Übergang von $\hat{v}_{s_0}$ zum Startknoten beziehungsweise zur ersten Kante von B_1.

Bezeichne $\vec{e}_{0_1} \in \hat{B}_0$ die erste besuchte Kante des einfachen Randes $\hat{B}_0$. Ist $\hat{v}_{s_0}$ aktiv, dann nimmt die E-DFS$_{\circlearrowleft}$ die rechteste der zu $\hat{v}_{s_0}$ inzidenten Kanten, das heißt die im Gegenuhrzeigersinn nächste Kante nach $\vec{e}_{0_1}$, auf. Der Knoten $\hat{v}_{s_0}$ ist Startknoten des nächsten Randes, das heißt $\hat{v}_{s_0} = v_{s_1}$. Ist $\hat{v}_{s_0}$ abgearbeitet, dann führt die Tiefensuche so lange Backtrack Schritte durch, bis der aktuelle Knoten v aktiv und somit Startknoten von B_1 ist. Sei dabei die letzte aus M_{DFS} entfernte Kante $\vec{e}_{0_j}$. Dann nimmt die Tiefensuche die im Gegenuhrzeigersinn nächste Kante nach $\vec{e}_{0_j}$ auf. In beiden Fällen wählt die Tiefensuche die im Uhrzeigersinn nächste Kante $\vec{e}_{1_1}$ auf dem Rand der Facette f_1 links von $\vec{e}_{0_1}$.

Die BFS$_{\circlearrowleft}$ ihrerseits nimmt f_1 als aktuellen Knoten, da dieser der Endknoten von $\vec{e}_{0_1}^{\,*}$ ist, der ersten Kante, die in M_{BFS} abgelegt wurde. Die im Uhrzeigersinn erste unbesuchte Kante in der Adjazenzliste von f_1 ist gerade $\vec{e}_{1_1}^{\,*}$. Alle eventuell vorhandenen Kanten zwischen

$\vec{e}^*_{0_1}$ und $\vec{e}^*_{1_1}$ sind inzident zu f_s und somit besucht. In Abbildung 4.5 ist die Situation an einem abstrakten Beispiel dargestellt.

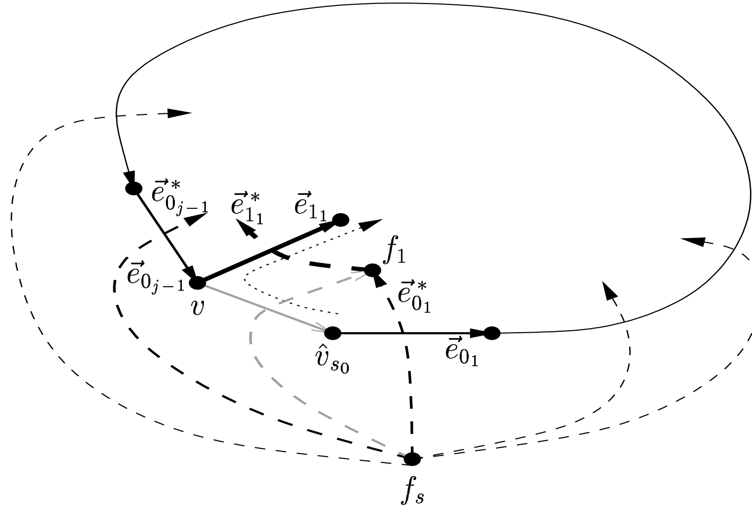


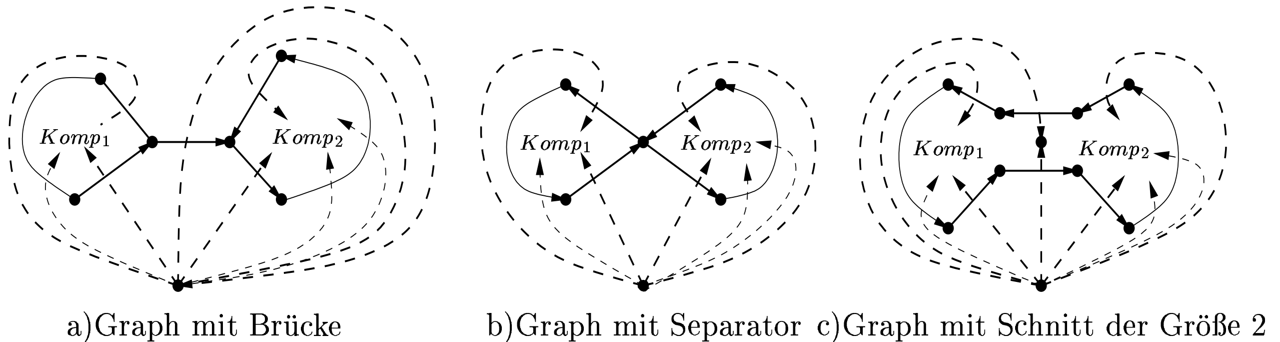
Abbildung 4.5: Wahl der Startkante des nächsten Randes durch die Tiefen- und Breitensuche. Die hellgrauen Kanten kennzeichnen die Kanten, über welche die Tiefensuche durch Backtrack Schritte zum letzten aktiven Knoten v zurückkehrt.

Tiefen- und Breitensuche wählen dieselbe Kante als erste Kante des nächsten Randes. Wir müssen uns nun noch überlegen, ob die Ausgangslage auf dem Rand B_1 tatsächlich für beide Suchen dieselbe ist wie für B_0. Für die E-DFS$_\circlearrowleft$ gilt das offensichtlich, da sie immer vom zuletzt besuchten Knoten aus weiterarbeitet und die besuchten Kanten von B_0 für die Suche nicht mehr existieren. Für die Breitensuche folgt dies aus Lemma 3.2.7, das wir in Abschnitt 3.2.3 gezeigt haben.
Die Ausgangslage für die Bearbeitung des Randes B_1 ist also die gleiche wie die für B_0 und der Beweis folgt somit per Induktion über die Knoten des Schachtelungsbaumes. □

4.3 E-DFS$_\circlearrowleft$ und die duale Breitensuche

Wir haben im vorigen Abschnitt gesehen, daß Kanten-Tiefensuche und Breitensuche auf bestimmten Graphklassen dual zueinander sind. In diesem Abschnitt wollen wir die Breitensuche leicht abändern, so daß sie auf jedem Graphen dual zur Kanten-Tiefensuche ist.
Überlegen wir uns also noch einmal, wie die E-DFS$_\circlearrowleft$ vorgeht. Die Tiefensuche arbeitet den Rand der äußeren Facette vollständig ab, falls dieser keine Brücke enthält. Zerfällt der Graph nach dem Entfernen des Rands in mehrere Zusammenhangskomponenten, so wird die Tiefensuche zuerst die Komponente bearbeiten, in der sie sich befindet, bevor sie durch Backtrack Schritte, also das Entfernen abgearbeiteter Kanten aus der Menge M, in eine noch nicht abgearbeitete Komponente gelangt. Besitzt der Rand eine Brücke, so wird dieser nicht vollständig von der Tiefensuche abgearbeitet, da sie wie in dem Fall davor in einer

Komponente steckenbleibt. Die Breitensuche hingegen arbeitet sich Schicht für Schicht vor. Sie muß also erkennen, wann eine Komponente für die Tiefensuche nicht mehr erreichbar ist. Dabei sind prinzipiell drei Fälle zu unterscheiden, welche in Abbildung 4.6 dargestellt sind.



a)Graph mit Brücke b)Graph mit Separator c)Graph mit Schnitt der Größe 2

Abbildung 4.6: Graph G, dessen Rand im Schachtelungsbaum wenigstens zwei direkte Nachfolger besitzt

Der folgende Algorithmus implementiert eine modifizierte Left-First-Breitensuche. Der Eingabegraph G sei in Form seiner gemäß einer festen kombinatorischen Einbettung sortierten Adjazenzlisten AL gegeben. Wenn die im Uhrzeigersinn nächste Kante in $AL[v]$ angesprochen wird, so ist immer die von der aktuellen Kante $\vec{e}_{act}$ aus im Uhrzeigersinn nächste Kante gemeint.

Algorithmus 4.3.1 (Jump-BFS$_{\circlearrowright}$)
Eingabe: Planar eingebetteter Graph $G = (V, E)$ und ein Startknoten $v_s \in V$ auf dem Rand der äußeren Facette.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$ in der äußeren Facette.
Kanten-Array AGE, Zeitpunkt $AGE_{act} := AGE[(x, v_s)] := 0$ und $AGE[e] := -1 \; \forall e \in E$
Menge $A := \emptyset$, Kante $\vec{e}_{act} := (x, v_s)$.
Kante $\vec{e}_{last} := (x, v_s)$ und Zeiger $PREV$ von jeder neu aufgenommenen Kante auf die direkt vor ihr aufgenommene Kante.

Knotenauswahl

(1) Endknoten v der aktuellen Kante $\vec{e}_{act}$

Kantenauswahl(v)

(1) wähle aus $AL[v]$ die im Uhrzeigersinn nächste Kante $e_n := \{v, w\}$

(2) entferne e_n aus $AL[v]$

(3) falls e_n unbesucht

(4) orientiere $\vec{e}_n := (v, w)$ und gib $\vec{e}_n$ aus

(5) $AGE_{act} := AGE_{act} + 1$, $AGE[\vec{e}_n] := AGE_{act}$

(6) $PREV[\vec{e}_n] := \vec{e}_{last}$
(7) $\vec{e}_{last} := \vec{e}_n$
(8) falls w unbesucht
(9) lege $\vec{e}_n$ in M ab
(10) ansonsten falls $\vec{e}_n = (w, v) \neq \vec{e}_{act}$
(11) falls $\vec{e}_n = \vec{e}_{last}$
(12) setze $\vec{e}_{last} := PREV[\vec{e}_n]$
(13) ansonsten, sei $t := AGE[\vec{e}_n]$
(14) füge $AGE[\vec{e}_{act}]$ in A ein und
(15) setze $\vec{e}_{act}$ auf die Kante $\vec{e}_k$ mit $AGE[\vec{e}_k] = min_{\vec{e} \in M}\{\, AGE[\vec{e}\,] \mid AGE[\vec{e}\,] > t \,\}$
(16) setze $PREV[\vec{e}_{act}] := PREV[\vec{e}_n]$
(17) ansonsten ($\vec{e}_n = \vec{e}_{act}$)
(18) sei $t := AGE[\vec{e}_{act}]$ und $\vec{e}_t := \vec{e}_{act}$, entferne $\vec{e}_{act}$ aus M
(19) falls eine Kante $\vec{e}_k \in M$ mit $AGE[\vec{e}_k] > t$ existiert
(20) setze $\vec{e}_{act}$ auf die Kante $\vec{e}_k$ mit $AGE[\vec{e}_k] = min_{\vec{e} \in M}\{\, AGE[\vec{e}\,] \mid AGE[\vec{e}\,] > t \,\}$
(21) setze $PREV[\vec{e}_{act}] := PREV[\vec{e}_t]$
(22) ansonsten falls A nicht leer, dann sei s der zuletzt in A eingefügte Zeitpunkt
(23) setze $\vec{e}_{act}$ auf die Kante $\vec{e}_k$ mit $AGE[\vec{e}_k] = min_{\vec{e} \in M}\{\, AGE[\vec{e}\,] \mid AGE[\vec{e}\,] \geq s \,\}$
(24) lösche s aus A
(25) setze $\vec{e}_{last} := PREV[\vec{e}_{last}]$
(26) ansonsten (A leer) beende Procedur

Erläuterung: Das Kanten-Array AGE speichert den Zeitpunkt, zu dem eine Kante zum ersten Mal besucht wurde. In AGE_{act} ist der Zeitpunkt der zuletzt neu gefundenen Kante vermerkt. In der Menge A werden die Zeitpunkte der aktuellen Kante gespeichert, wenn in Schritt (13) bis (15) zu einer anderen Kante gesprungen wird. $\vec{e}_{last}$ enthält die zuletzt neu gefundene und noch nicht von ihrem Endknoten aus bearbeitete Kante. $PREV[\vec{e}_k]$ deutet auf die Kante $\vec{e}_j$, die vor $\vec{e}_k$ gefunden und noch nicht von ihrem Endknoten aus besucht wurde. Diese Kanten, die besucht aber noch nicht aus der Adjazenzliste ihres Endknotens entfernt wurden, werden wir als *aktive* Kanten bezeichnen. Die Zeiger $PREV$ bilden also eine nach dem AGE-Wert absteigend sortierte Kette der aktiven Kanten, wobei $\vec{e}_{last}$ immer die erste Kante dieser Kette enthält.
Das Vorgehen der Jump-BFS$_\circlearrowleft$ entspricht im wesentlichen dem der Left-First-Breitensuche. Wird eine neue Kante gefunden, so wird diese, sofern ihr Endknoten noch nicht besucht ist, in M abgelegt und außerdem verschiedene Variablen angepaßt. Allerdings zählt eine Kante nur von dem Knoten aus als besucht, von dem aus sie gefunden wurde. Erst wenn sie zum

zweiten Mal gewählt wurde, ist eine Kante vollständig abgearbeitet. Die Jump-BFS$_{\circlearrowright}$ geht also im Gegensatz zur Left-First-Breitensuche alle Kanten zweimal durch und zwar von jedem ihrer Endknoten aus genau einmal. Dabei wird jede Kante aus der Adjazenzliste des Knotens entfernt, von dem aus sie betrachtet wurde.
Findet die Breitensuche vom aktuellen Knoten aus eine neue Kante e_n, so wird diese in M abgelegt und die Variablen entsprechend gesetzt. Ist die durch den Kanteniterator gewählte Kante $\vec{e}_n$ schon besucht, so sind zwei Fälle zu unterscheiden.
1. Fall: $\vec{e}_n = \vec{e}_{last}$, das heißt, die gewählte Kante ist gleich der zuletzt neu gefundenen aktiven Kante. In diesem Fall wird diese Kante aus M und der Adjazenzliste ihres Endknotens entfernt, also einfach "übergangen". Kante $\vec{e}_{last}$ wird entsprechend neu gesetzt. (Schritt (11) und (12))
2. Fall: $\vec{e}_n \neq \vec{e}_{last}$. Seit dem Zeitpunkt $s = AGE[\vec{e}_n]$, zu dem $\vec{e}_n$ zum ersten Mal besucht wurde und dem jetzigen Zeitpunkt t, da $\vec{e}_n$ zum zweiten Mal gewählt wird, wurden neue Kanten gefunden und in M abgelegt. Diese bilden die Menge $M_s := \{\vec{e}|AGE[\vec{e}] > s\} \subset M$. Die Jump-BFS$_{\circlearrowright}$ führt nun einen Sprung aus (Schritt (13) bis (16)). Dabei merkt sich die Breitensuche $AGE[\vec{e}_{act}]$ in A (Schritt (14)) und macht mit den Kanten aus M_s weiter. Dabei wird $\vec{e}_{act}$ auf die Kante aus M_s gesetzt, die den kleinsten AGE-Wert hat (Schritt (15)). Dadurch können von nun an nur noch Kanten gefunden werden, die zu Kanten aus M_s inzident sind, da $\vec{e}_{act}$ die Kante mit dem kleinsten AGE-Wert ist und nur Kanten aus M, deren Wert größer als $AGE[\vec{e}_{act}]$ ist, als künftige $\vec{e}_{act}$ genommen werden (vergleiche Schritt (19) bis (21)). Erst wenn diese alle abgearbeitet und folglich aus M_s, beziehungsweise aus M entfernt sind, kehrt die Breitensuche über das in A gespeicherte Alter zu der Kante zurück, die zum Zeitpunkt des Sprunges aktuelle Kante war (Schritt (22) bis (25)).
Die Jump-BFS$_{\circlearrowright}$ führt genau dann einen Sprung aus, wenn sie auf eine einlaufende Kante $\vec{e}_n$ trifft und in M Kanten $\vec{e}$ gespeichert sind, für die $AGE[\vec{e}] > AGE[\vec{e}_n]$ gilt. Existieren keine solchen Kanten, so ist $\vec{e}_n = \vec{e}_{last}$, also selbst die Kante aus M, die den größten AGE-Wert hat. Es tritt Fall 1 ein und die Kante wird übergangen.

Nachdem wir uns klargemacht haben, wie die Jump-BFS$_{\circlearrowright}$ arbeitet werden wir nun den zentralen Satz dieses Abschnitts beweisen.

Satz 4.3.2
Gegeben ein planar eingebetteter Graph $G = (V, E)$ und ein ausgezeichneter Knoten $v_s \in V$ auf dem Rand der äußeren Facette. Die zur E-DFS$_{\circlearrowright}$ duale Suche ist eine Jump-BFS$_{\circlearrowright}$.

Beweis:
Aus Satz 4.2.2 wissen wir bereits, daß sich die beiden Suchen dual verhalten, wenn der Schachtelungsbaum unverzweigt ist und die Startknoten der Ränder bestimmte Vorraussetzungen erfüllen. Wir haben folglich zwei Dinge zu zeigen: Erstens, daß die Startknoten der Ränder beliebig liegen können und zweitens, daß die Suchen bei einer Verzweigung des Schachtelungsbaumes gleich verfahren.
Die äußere Facette f_s von G sei Startknoten der Breitensuche auf G^* und die Dualkante der Startkante der Tiefensuche sei die Initialkante der Breitensuche. Damit starten die Suchen ihren Weg mit derselben Kante.

Teil 1)
Der Startknoten v_{s_i} eines Randes B_i liege auf einer Antenne und bilde nicht das rechteste Blatt. Das heißt auf dem Right-First-Tiefensuchweg von dem Startknoten v_{s_i} zur Wurzel $\hat{v}_{si}$ der Antenne bleibt wenigstens eine Kante e_h links des Weges unbesucht. In Abbildung 4.7 ist die Situation dargestellt.

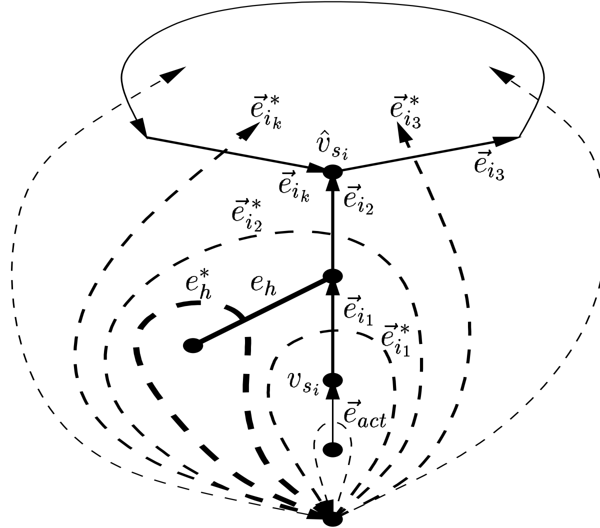


Abbildung 4.7: Der Startknoten v_{s_i} des Randes liegt auf einer Antenne und bildet nicht das rechteste Blatt. Bei Bearbeitung des Randes durch die E-DFS$_\circlearrowleft$ bleibt die Kante e_h unbesucht.

Die Tiefensuche umrundet den Rand und kommt bei $\vec{e}_{i_k}$ als letzter Kante des einfachen Randes an. Von dort aus geht sie weiter zum nächsten darunterliegenden Rand. Die BFS$_\circlearrowleft$ würde e^*_h als nächste Kante nach $\vec{e}^*_{i_k}$ wählen. Die Jump-BFS$_\circlearrowleft$ jedoch wählt als nächstes die schon besuchte Kante $\vec{e}^*_{i_2} \neq \vec{e}_{last}$, die Schleife, deren Dualkante $\hat{v}_{si}$ als Endknoten hat. Somit macht die Kantenauswahl mit Schritt (13) weiter und springt zu der Kante $\vec{e}^*_{i_3}$, die direkt nach $\vec{e}^*_{i_2}$ aufgenommen wurde. Da nach vollständiger Bearbeitung eines Randes nur noch die Kanten des einfachen Randes in M gespeichert sind, ist $\vec{e}^*_{i_3}$ gerade die erste Kante des einfachen Randes und somit genau diejenige Kante, von der aus die Breitensuche in den nächsten Rand übergeht. Das Alter der aktuellen Kante $\vec{e}^*_{act}$ wird in A gespeichert.
Es muß noch gezeigt werden, daß e^*_h später von der Breitensuche zum gleichen Zeitpunkt abgearbeitet wird wie e_h von der Tiefensuche. Dies ergibt sich aus den Eigenschaften der Suchen. Die Tiefensuche wird erst nachdem alles innerhalb des Randes vollständig abgearbeitet ist wieder durch Backtrack Schritte zu $\vec{e}_{i_1}$ zurückkehren und von dort aus e_h finden. Durch den Sprung der Jump-BFS$_\circlearrowleft$ wird das ebenso erreicht. Die Breitensuche übergeht durch den Sprung die Kanten, die dem Alter nach zwischen $\vec{e}^*_{act}$ und der Kante liegen, zu der gesprungen wird, in unserem Beispiel $\vec{e}^*_{i_3}$. Von da an wird zuerst alles bearbeitet, was zu Kanten inzident ist, deren AGE-Wert größer ist als $AGE[\vec{e}^*_{i_3}]$. Erst wenn alle diese Kanten, die Kanten innerhalb des einfachen Randes, bearbeitet sind, kehrt die Jump-BFS$_\circlearrowleft$ über den in A gespeicherten AGE-Wert zur aktuellen Kante zurück und findet von dort aus e^*_h.

Tiefen- und Breitensuche bearbeiten demnach alle Kanten innerhalb des Randes, bevor sie die übergangene Kante e_h finden. Die Lage des Startknotens eines Randes kann demnach beliebig sein.

Teil 2)

Gehen wir nun davon aus, daß der Rand B_i im Schachtelungsbaum wenigstens zwei direkte Nachfolger besitzt. Es tritt also einer der drei Fälle aus Abbildung 4.6 auf. Wir gehen davon aus, daß die beiden Suchen jeweils in Komponente $Komp_1$ starten. Betrachten wir die drei Fälle im Einzelnen. Bezeichne $B_i|_{Komp_j} := B_i \cap E(Komp_j)$, wobei wesentliche Brücken und Schnittkanten zu keiner Komponente gezählt werden. $B_i|_{Komp_j}$ ist also der Teil des Randes B_i, der zu Komponente $Komp_j$ gehört. Entsprechend sei $d(v)|_{Komp_j} := |\{e = \{v, w\} \in E(Komp_j)\}|$ der Grad von v eingeschränkt auf den Untergraphen $Komp_j$.

Fall a) Der Rand B_i enthält eine wesentliche Brücke e_b.

Die Tiefensuche geht über diese Brücke nach $Komp_2$, bearbeitet den Rand von $Komp_2$, $B_i|_{Komp_2}$, und kommt dann nicht wieder nach $Komp_1$ zurück, um den Rest von B_i zu besuchen. Die Tiefensuche verhält sich auf $Komp_2$ so, wie wenn $Komp_2$ ein eigenener Graph wäre und e_b die Startkante. Ist $B_i|_{Komp_2}$ vollständig bearbeitet, so läuft die Suche in die Komponente hinein und besucht den darunter liegenden Rand. Erst wenn die ganze Komponente abgearbeitet ist, kommt die Tiefensuche durch Backtrack Schritte zu der letzten Kante aus $Komp_1$ vor e_b zurück und bearbeitet nun $Komp_1$ wie wenn diese Komponente der Eingabegraph gewesen wäre.

Die Dualkante e_b^* der Brücke ist eine Schleife. Die Jump-BFS$_\circlearrowleft$ besucht die Kanten von B_i in derselben Reihenfolge wie die Tiefensuche. Nachdem alle Dualkanten der Kanten aus $B_i|_{Komp_2}$ gefunden wurden, besucht die Breitensuche die Schleife e_b^* zum zweiten Mal, diesmal als einlaufende Kante. Die Jump-BFS$_\circlearrowleft$ springt demzufolge zu der Kante, die nach e_b^* aufgenommen wurde. Dies ist von den Dualkanten aus $B_i|_{Komp_2}$ die erste Kante, die aufgenommen wurde. Die Jump-BFS$_\circlearrowleft$ betrachtet von nun an nur noch Kanten, die inzident sind zu den Dualkanten aus $B_i|_{Komp_2}$. Erst wenn die gesamte Komponente abgearbeitet ist, springt sie über den in A gespeicherten AGE-Wert zu den Kanten aus $B_i|_{Komp_1}$ zurück.

Man mache sich klar, daß bei Schleifen, deren Dualkanten zu einer Antenne gehören, kein Sprung ausgeführt wird. Dies ergibt sich daraus, daß die Jump-BFS$_\circlearrowleft$ beim Auffinden der Kante, die zuletzt davor aufgenommen wurde, und noch nicht abgearbeitet ist, diese in Schritt (12) und (13) übergeht. Das liegt an der Klammerstruktur der Dualkanten zu Baumkanten und der entsprechenden Verwaltung der $PREV$-Kette mit Anfang $\vec{e}_{last}$.

Fall b) Der Rand B_i enthält einen Separator v_S mit $d(v_S)|_{Komp_1} = 2$ oder $d(v_S)|_{Komp_2} = 2$.

Die beiden auftretenden Fälle sind in Abbildung 4.8 dargestellt. Zur besseren Lesbarkeit wurden nur die Kanten von G beschriftet. Die Dualkante einer Kante $\vec{e}_{i_k}$ wird wie gewohnt mit $\vec{e}^{\,*}_{i_k}$ bezeichnet.

i) $d(v_S)|_{Komp_1} = 2$.

Nachdem die Tiefensuche den Rand B_i vollständig durchsucht hat, kommt sie bei Bearbeitung des darunterliegenden Randes B_{i+1} zum Rand der Facette f_1 in $Komp_1$ links von v_S. Sie läuft den Rand dieser Facette rechts herum ab über die Kanten $\vec{e}_{j_m}$, $\vec{e}_{j_{m+1}}$ und $\vec{e}_{j_{m+2}}$. Beim Endknoten von $\vec{e}_{i_{l+1}}$ angekommen, nimmt sie den rechtesten Weg, der von dort aus

weiterführt, in dem Beispiel nach einem Backtrack Schritt über Kante $\vec{e}_{j_{m+3}}$.

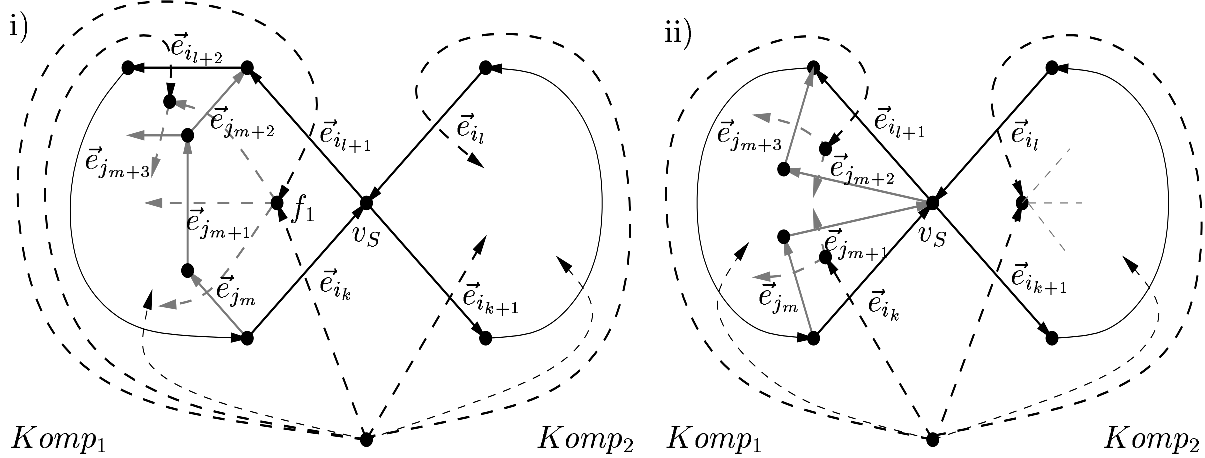


Abbildung 4.8: Die zwei möglichen Ausprägungen eines Randes B_i (schwarz) mit Separator, der im Schachtelungsbaum wenigstens zwei direkte Nachfolger besitzt. Die grauen Kanten gehören zum Rand B_{i+1}.

Die Jump-BFS$_{\circlearrowleft}$ bearbeitet die Dualkanten zu B_i entsprechend und geht danach ebenfalls zur Bearbeitung von B_{i+1} über. Dabei wählt sie irgendwann $\vec{e}^*_{i_k}$ als aktuelle Kante, durchsucht also die zu f_1 inzidenten Kanten. Nachdem sie $\vec{e}^*_{j_m}$ bis $\vec{e}^*_{j_{m+2}}$ besucht hat, findet sie die einlaufend gerichtete Kante $\vec{e}^*_{i_{l+1}} \neq \vec{e}_{last}$. Die Jump-BFS$_{\circlearrowleft}$ führt einen Sprung zu $\vec{e}^*_{i_{l+2}}$ durch, der Kante, die nach $\vec{e}^*_{i_{l+1}}$ in M aufgenommen wurde. Das Alter von $\vec{e}^*_{i_k}$ wird als Rücksprungaddresse in A gespeichert. Alle Dualkanten aus $B_i|_{Komp_2}$ sind vor $\vec{e}^*_{i_{l+1}}$ gefunden worden, haben also einen niedrigeren AGE-Wert als $\vec{e}^*_{i_{l+1}}$. Damit kann Komponente $Komp_2$ erst dann bearbeitet werden, wenn $Komp_1$ vollständig abgearbeitet und alle ihre Kanten wieder aus M entfernt sind.
Bleibt zu überlegen, ob Tiefen- und Breitensuche die Bearbeitung von $Komp_2$ mit derselben Kante starten. Dies ist nach den bisherigen Überlegungen offensichtlich. Die Tiefensuche startet die Bearbeitung von $Komp_2$ mit der zuletzt aufgenommenen Kante $\vec{e}_{i_l} \in B_i|_{Komp_2}$. Somit umrundet sie als nächstes die Facette links von $\vec{e}_{i_{k+1}}$. Die Jump-BFS$_{\circlearrowleft}$ ihrerseits kehrt über den in A gespeicherten AGE-Wert zu $\vec{e}^*_{i_k}$ zurück. Ist f_1 abgearbeitet, so wird die nächst ältere Kante aus M, hier $\vec{e}^*_{i_{k+1}}$, als aktuelle Kante gewählt. Ist f_1 noch nicht abgearbeitet, so sind die noch inzidenten Kanten alle einlaufend und haben entweder einen größeren AGE-Wert als die Kanten aus $B_i|_{Komp_2}$ oder einen kleineren. Haben sie einen größeren AGE-Wert, so bilden sie den Anfang der $PREV$-Kette und werden nacheinander in Schritt (12) und (13) übergangen. Ist ihr AGE-Wert kleiner, so wird wieder ein Sprung ausgeführt, und zwar zu der Kante aus M, die den kleinsten AGE-Wert nach $\vec{e}^*_{i_k}$ hat. Dies ist ebenso Kante $\vec{e}^*_{i_{k+1}}$. Es ergibt sich also, daß beide Suchen nach Bearbeitung von $Komp_1$ die Suche in Komponente $Komp_2$ mit dem Rand der Facette links von $\vec{e}_{i_{k+1}}$ starten.
ii) $d(v_S)|_{Komp_2} = 2$ und $d(v_S)|_{Komp_1} > 2$
Nach vollständiger Abarbeitung des Randes B_i läuft die E-DFS$_{\circlearrowleft}$ auf dem darunter lie-

genden Rand B_{i+1} weiter. Dabei durchsucht sie den Rand der Facetten links von $\vec{e}_{i_k}$ und $\vec{e}_{i_{l+1}}$. In unserem Beispiel durchläuft die Tiefensuche dabei die Kanten $\vec{e}_{j_m}$ bis $\vec{e}_{j_{m+3}}$.
Die Jump-BFS$_\circlearrowleft$ ihrerseits wählt zur Bearbeitung der Kanten aus B^*_{i+1} als aktuelle Kante $\vec{e}^*_{i_k}$. Ist deren Endknoten abgearbeitet, wird $\vec{e}^*_{i_{k+1}}$ aktuelle Kante. Die erste zu ihrem Endknoten inzidente Kante ist die schon besuchte Kante $\vec{e}^*_{i_l} \neq \vec{e}_{last}$. Die Breitensuche führt einen Sprung zu $\vec{e}^*_{i_{l+1}}$ durch, der Kante aus M, die direkt nach $\vec{e}^*_{i_l}$ gefunden wurde. Die Breitensuche bearbeitet also als nächstes ebenfalls die Kanten auf dem Rand der Facette links von $\vec{e}_{i_{l+1}}$ und findet dabei die Kanten $\vec{e}^*_{j_{m+2}}$ und $\vec{e}^*_{j_{m+3}}$. Beim Sprung wird der AGE-Wert von $\vec{e}^*_{i_{k+1}}$ in A abgelegt.
Ist Komponente $Komp_1$ von beiden Suchen vollständig abgearbeitet, kehrt die Tiefensuche durch Backtrack Schritte zu Kante $\vec{e}_{i_l}$ zurück. Vor dort aus muß sie noch solange Backtrack Schritte durchführen, bis sie eine unbesuchte Kante auf dem Rand der Facette links von $\vec{e}_{i_{k+1}}$ findet. Dann umrundet sie den Rand dieser Facette. Die Jump-BFS$_\circlearrowleft$ wählt durch den in A gespeicherten AGE-Wert Kante $\vec{e}^*_{i_{k+1}}$ als aktuelle Kante und arbeitet die Dualkanten der Kanten auf dem Rand der Facette links von $\vec{e}_{i_{k+1}}$ ab. Trifft sie dabei zuerst auf schon besuchte Kanten, also Kanten, über deren Dualkanten die Tiefensuche durch Backtrack Schritte zurück geht, so bilden diese wie schon in i) den Anfang der $PREV$-Kette, da sie die aktiven Kanten mit größtem AGE-Wert sind. Sie werden von der Breitensuche in Schritt (12) und (13) übergangen.
Fall c) Der Rand B_i besitzt einen Schnitt der Größe 2.
Wir wollen die Schnittkanten mit $\vec{e}_{S1}$ und $\vec{e}_{S2}$ bezeichnen, wobei $AGE[\vec{e}_{S1}] < AGE[\vec{e}_{S2}]$ gelte. Die Facette, die links von $\vec{e}_{S1}$ und $\vec{e}_{S2}$ liegt bezeichnen wir mit f_S. Nach Bearbeitung des Randes B_i zerfällt der Graph in die Komponenten $Komp_1$ und $Komp_2$. Die Tiefensuche arbeitet entsprechend $Komp_1$ zuerst vollständig ab und kehrt danach durch Backtrack Schritte zu der Kante aus $B_i|_{Komp_2}$ zurück, die den ältesten AGE-Wert hat und keine Schnittkante ist. Von dort aus durchläuft sie den noch unbesuchten Rand der Facette f_S.
Die Jump-BFS$_\circlearrowleft$ wählt zur Bearbeitung der Kanten aus B^*_{i+1} Kante $\vec{e}^*_{S1}$ als aktuelle Kante. In der Adjazenzliste von f_S findet sie die einlaufende Kante $\vec{e}^*_{S2}$ und führt einen Sprung zu der nach $\vec{e}^*_{S2}$ aufgenommenen Kante aus. Die Jump-BFS$_\circlearrowleft$ bleibt also in $Komp_1$ stecken und kehrt über den in A gespeicherten AGE-Wert nach Abarbeitung von $Komp_1$ zu $\vec{e}^*_{S1}$ zurück. Von dort aus durchsucht sie die restlichen Kanten in $AL[f_S]$, also die Dualkanten der Kanten auf dem Rand der Facette f_S, die zu $Komp_2$ gehören.
Wir wissen bereits , daß sich Jump-BFS$_\circlearrowleft$ und DFS$_\circlearrowleft$ auf einem einzelnen Rand gleich verhalten. In Teil 1 haben wir außerdem gezeigt, daß der Startknoten eines Randes beliebig liegen darf. In Teil 2 des Beweises haben wir das Verhalten der beiden Suchen in einem Graphen mit verzweigtem Schachtelungsbaum untersucht. Dabei haben wir gesehen, daß die Jump-BFS$_\circlearrowleft$ auf dem Schachtelungsbaum eine Tiefensuche ausführt, das heißt, sie bleibt in einer Komponente stecken, genau wie die Tiefensuche. Daraus ergibt sich insgesamt, daß die Jump-BFS$_\circlearrowleft$ die Kanten des Dualgraphen in der gleichen Reihenfolge aufnimmt, wie die DFS$_\circlearrowleft$ auf dem Orginalgraph, die Suchen sind also dual zueinander. □

4.4 BFS$_\circlearrowleft$ und die duale Kanten-Tiefensuche

In dem vorangegangenen Abschnitt haben wir eine zur E-DFS$_\circlearrowleft$ duale Breitensuche konstruiert. Wir wollen uns nun damit beschäftigen, die Tiefensuche so zu modifizieren, daß sie dual zur BFS$_\circlearrowleft$ ist. Die Hauptarbeit besteht darin, der Tiefensuche die Information zugänglich zu machen, wann sie eine Komponente verlassen und in eine andere Komponente, die sie normalerweise erst nach Bearbeitung der aktuellen Komponente erreichen würde, springen muß.
Der Eingabegraph $G = (V, E)$ sei wie schon im letzten Abschnitt durch seine gemäß einer festen kombinatorischen Einbettung sortierten Adjazenzlisten AL gegeben. Im Gegensatz zur Jump-BFS$_\circlearrowleft$ wird die modifizierte Tiefensuche nicht mehr mit einer Menge M zur Speicherung der Kanten auskommen. Wir werden deshalb die Mengen mit unterschiedlichen Indizes versehen. Jede Menge M_i wird durch die in ihr enthaltenen Kanten identifiziert, dementsprechend bezeichne $M(\vec{e}_k)$ die Menge M_j mit $\vec{e}_k \in M_j$. Diese Menge M_j ist eindeutig bestimmt, da keine Kante in zwei unterschiedlichen Mengen abgelegt wird. Der Wert $AGE[e]$ speichere für jede Kante $e \in E$ den Zeitpunkt t, zu dem die Kante zum ersten Mal gefunden wurde. Da die Tiefensuche immer die Kante aus der Menge M_j wählt, die zuletzt gefunden wurde, werden wir für eine Menge M_i diese Kante mit $\vec{e}_{i_{max}}$ bezeichnen. $\vec{e}_{i_{max}}$ ist demnach jene Kante aus M_i, für die gilt $AGE[\vec{e}_{i_{max}}] = max_{\vec{e} \in M_i}\{AGE[\vec{e}]\}$. Ist eine Menge M_i leer, so sei $\vec{e}_{i_{max}} := NIL$ definiert.

Algorithmus 4.4.1 (Jump-E-DFS$_\circlearrowleft$)
Eingabe: Planar eingebetteter Graph $G = (V, E)$ und ein Startknoten $v_s \in V$ auf dem Rand der äußeren Facette.
Initialisierung: $M_0 := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$ in der äußeren Facette.
Kante $\vec{e}_{act} := (x, v_s)$.
Kanten-Array AGE, Zeitpunkt $AGE_{act} := AGE[(x, v_s)] := 0$, $AGE[e] := -1 \; \forall e \in E$.
Kanten-Array $SUCC$, $SUCC[e] := NIL \; \forall e \in E$.

Knotenauswahl

(1) Endknoten v von $\vec{e}_{act}$

Kantenauswahl(v)

(1) wähle aus $AL[v]$ die im Gegenuhrzeigersinn nächste Kante $e_n := \{v, w\}$
(2) entferne e_n aus $AL[v]$
(3) falls e_n unbesucht
(4) orientiere $\vec{e}_n := (v, w)$ und füge $\vec{e}_n$ in $M(\vec{e}_{act})$ ein
(5) $AGE_{act} := AGE_{act} + 1$, $AGE[\vec{e}_n] := AGE_{act}$
(6) $\vec{e}_{act} := \vec{e}_n$
(7) ansonsten falls $\vec{e}_n = (w, v) \neq \vec{e}_{act}$

(8) teile $M(\vec{e}_{act})$ in M_i und M_j mit
$M_i := \{ \vec{e} \in M(\vec{e}_{act}) \mid AGE[\vec{e}] < AGE[\vec{e}_n] \}$ und
$M_j := \{ \vec{e} \in M(\vec{e}_{act}) \mid AGE[\vec{e}] > AGE[\vec{e}_n] \}$,

(9) falls M_i nicht leer

(10) $\vec{e}_{act} := \vec{e}_{i_{max}}$

(11) falls $SUCC[\vec{e}_n] = NIL$

(12) $SUCC[\text{LastSucc}(\vec{e}_{act})] := \vec{e}_{j_{max}}$

(13) ansonsten $(SUCC[\vec{e}_n] \neq NIL)$

(14) $SUCC[\text{LastSucc}(\vec{e}_{act})] := SUCC[\vec{e}_n]$

(15) $SUCC[\text{LastSucc}(SUCC[\vec{e}_n])] := \vec{e}_{j_{max}}$

(16) ansonsten $(M_i = \emptyset)$

(17) falls $SUCC[\vec{e}_n] \neq NIL$

(18) $\vec{e}_{act} := SUCC[\vec{e}_n]$

(19) $SUCC[\text{LastSucc}(\vec{e}_{act})] := \vec{e}_{j_{max}}$

(20) ansonsten $(\vec{e}_n = \vec{e}_{act})$

(21) merke $\vec{e}_s := SUCC[\vec{e}_{act}]$

(22) entferne $\vec{e}_{act}$ aus $M_k := M(\vec{e}_{act})$

(23) falls M_k nicht leer

(24) $\vec{e}_{act} := \vec{e}_{k_{max}}$

(25) falls $\vec{e}_s \neq NIL$

(26) $SUCC[\text{LastSucc}(\vec{e}_{act})] := \vec{e}_s$

(27) ansonsten falls $\vec{e}_s \neq NIL$

(28) $\vec{e}_{act} := \vec{e}_s$

(29) ansonsten beende Prozedur

$\vec{e}_l \longleftarrow$ **LastSucc**$(\vec{e})$

(1) $\vec{e}_l := \vec{e}$

(2) solange $SUCC[\vec{e}_l] \neq NIL$

(3) $\vec{e}_l := SUCC[\vec{e}_l]$

(4) gib $\vec{e}_l$ aus

Erläuterung: Das Kanten-Array AGE speichert für jede Kante $e \in E$ den Zeitpunkt, zu der sie zum ersten Mal gefunden wurde. Die Variable AGE_{act} enthält den Zeitpunkt der zuletzt gefundenen Kante, das heißt $AGE_{act} := max\{AGE[\vec{e}]\}$. Das Kanten-Array $SUCC$ verwaltet Nachfolger von Kanten, welche Kanten aus verschiedenen Mengen M_k miteinander verbinden. Die in $SUCC$ enthaltenen Kanten können ganze Ketten von Nachfolgern bilden, wenn die Kante, die Nachfolger einer anderen Kante ist, wiederum einen Nachfolger besitzt. Die Routine LastSucc($\vec{e}$) bestimmt die letzte Kante einer solchen Kette mit Anfang $\vec{e}$.
Um die Erklärung des Algorithmus zu vereinfachen, wollen wir vorher noch einige Konventionen einführen. $M := \cup M_k$ sei die Vereinigung aller durch den Algorithmus erzeugten nichtleeren Mengen M_k und enthält somit alle aktiven Kanten. Wir werden die einzelnen Mengen M_k als Stacks ansehen. Ein *Stack* ist ein Stapel, der die in ihm gespeicherten Elemente nach dem LIFO Prinzip verwaltet. *LIFO* bedeutet Last-In-First-Out, die Elemente werden also oben auf den Stapel gelegt und auch immer von oben heruntergenommen. Realisieren wir unsere Mengen M_k als Stacks, so wird dadurch die Sortierung der Kanten nach ihrem AGE-Wert erhalten, wobei das oberste Element einer Menge M_k gerade $\vec{e}_{k_{max}}$ ist, die Kante mit maximalem AGE-Wert unter allen Kanten der Menge.

Genau wie die Jump-BFS$_{\circlearrowleft}$ durchsucht auch die Jump-E-DFS$_{\circlearrowleft}$ jede Kante zweimal und zwar von jedem ihrer Endknoten aus genau einmal. Findet die Jump-E-DFS$_{\circlearrowleft}$ bei ihrer Suche eine ungerichtete Kante, so verfährt sie wie die Right-First-Kanten-Tiefensuche (Schritt (3) bis (6)). Ist die von der Kantenauswahl gefundenen Kante $\vec{e}$ dagegen bereits besucht und nicht die aktuelle Kante (Schritt (7)), so führt die Jump-E-DFS$_{\circlearrowleft}$ einen Sprung durch. Bei einem Sprung wird die aktuelle Menge $M(\vec{e}_{act})$ in zwei Mengen M_i und M_j unterteilt, wobei M_i alle Kanten aus $M(\vec{e}_{act})$ enthält, die vor $\vec{e}$ gefunden wurden, und M_j all jene, die nach $\vec{e}$ gefunden wurden (Schritt (8)). Aus Lemma 3.2.3 wissen wir bereits, daß alle geschlossenen Kreise Linkskreise sind. Vor dem Sprung wurde mit der aktuellen Kante ein solcher Linkskreis geschlossen. Alle von außen an diesen Kreis angrenzenden Kanten sind besucht, da die Right-First-Auswahlregel alles rechts des Weges abgesucht hat und vom aktuellen Knoten die nächst rechte Kante $\vec{e}$ ist, also die Kante, von der aus der Kreis begonnen wurde. M_j enthält demnach genau die Kanten des Linkskreises, welche das Innere des Kreises vom Rest des Graphen trennen. M_i dagegen enthält aktive Kanten, die vor dem Kreis gefunden wurden. Von diesen Kanten aus sucht die Jump-E-DFS$_{\circlearrowleft}$ nun weiter, wobei $SUCC$ der Kante, von der aus die erste neue Kante gefunden wird, auf die Kante $\vec{e}_{j_{max}}$ gesetzt wird (Schritt (11) bis (15)). Durch $SUCC$ wird die Verbindung zwischen den einzelnen Mengen M_k hergestellt.
Betrachten wir noch einmal die Situation der Kanten der Menge M_i direkt nach einem Sprung. Ist die Menge M_i leer, so wird direkt zur Menge M_j oder dem Nachfolger der Sprungkante zurück gekehrt (Schritt (16) bis (19)). Die Sprungkante $\vec{e}$ war in diesem Fall die Kante aus $M(\vec{e}_{act})$ die den kleinsten AGE-Wert hatte. Ist die Menge M_i nicht leer, so können die in ihr enthaltenen Kanten noch inzident sein zu unbesuchten Kanten oder nicht. Sind die Kanten nicht mehr inzident zu noch unbesuchten Kanten, so bilden sie den Abschnitt des Randes einer Facette f, der zu dem darüber liegenden Rand gehört. Die

Kanten aus M_i werden nacheinander bearbeitet und aus der Menge M_i entfernt. Danach ist M_i leer und es wird zu einem Nachfolger oder M_j zurückgekehrt. Man überlege sich, daß durch den Sprung mit anschliesendem Rücksprung die Kanten aus $M(\vec{e}_{act})$ gelöscht werden, die nicht mehr auf der Front zwischen besuchten und unbesuchten Kanten liegen. In Abbildung 4.9 ist diese Situation abstrakt dargestellt.

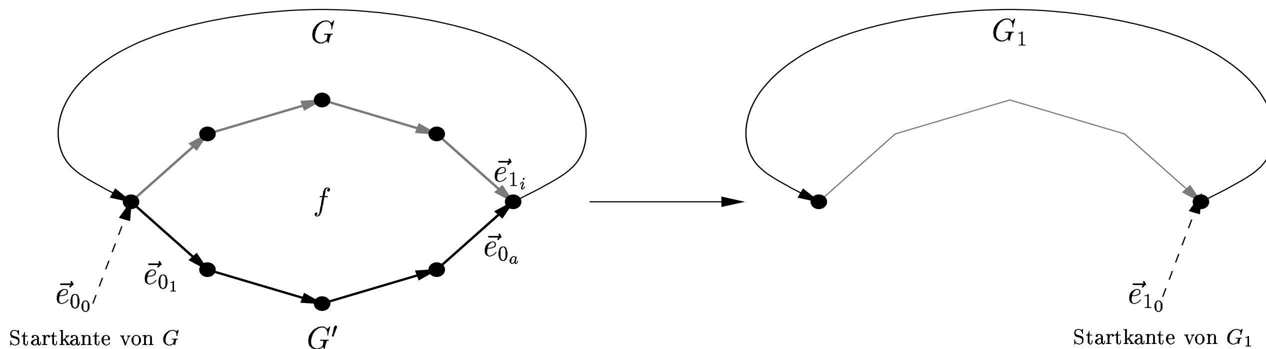


Abbildung 4.9: Sprung über die Kante $\vec{e}_{0_a}$. In M_j sind danach alle Randkanten von G zusammen mit den dunkelgrauen Kanten enthalten, die Kanten $\vec{e}_{0_1}$ bis $\vec{e}_{0_{a-1}}$ befinden sich in der Menge M_i.

Wir wollen diese beiden Fälle eines Sprunges, bei denen entweder M_i leer ist oder keine Kante enthält, von der aus eine unbesuchte Kante gefunden werden kann, als triviale Sprünge bezeichnen. Ein *trivialer Sprung* bewirkt im wesentlichen nur, daß die Sprungkante und Kanten, die vor ihr in die aktuelle Menge eingefügt wurden, aus der Menge entfernt werden, da von ihnen aus sowieso nichts mehr gefunden werden kann. Sind dagegen die Kanten aus M_i inzident zu noch unbesuchten Teilen des Graphen, so fährt die Tiefensuche mit der Bearbeitung dieser Bereiche fort (Schritt (9) bis (15)).
Nach einem Sprung werden die neuen Kanten in die aktuelle Menge M_i eingefügt, die Menge, welche die Kanten unterhalb der Sprungkante enthält. Im Laufe eines Algorithmus werden oft viele Sprünge ausgeführt und dementsprechend viele $SUCC$-Zeiger angelegt und umgehängt. Das folgende Lemma gibt Auskunft über die besondere Eigenschaft der $SUCC$-Zeiger.

Lemma 4.4.2
Für jede Kante $\vec{e} \in M, M = \cup M_k$ mit $NIL \neq SUCC[\vec{e}] \in M$ gilt, daß

$$AGE[\vec{e}] < AGE[SUCC[\vec{e}]]$$

Beweis:
Wir zeigen, daß die Eigenschaft erfüllt ist, wenn ein neuer $SUCC$-Zeiger angelegt wird und daß sie bei jeder Art des Umhängens erhalten bleibt.
Teil 1)
Ein neuer $SUCC$-Zeiger wird angelegt, wenn die Jump-E-DFS$_{\circlearrowleft}$ einen Sprung durchführt. Bei einem Sprung über die Kante $\vec{e}_k$ wird die aktuelle Menge M_k in die Mengen M_i und M_j

unterteilt. Der $SUCC$-Zeiger wird von $\vec{e}_{i_{max}}$ oder einer anderen Kante, falls schon $SUCC$-Zeiger existieren, auf $\vec{e}_{j_{max}}$ gesetzt. Dabei ist $\vec{e}_{j_{max}}$ die Kante aus M, die den größten AGE-Wert von allen Kanten aus M besitzt. Die geforderte Eigenschaft ist somit automatisch erfüllt. Die Tiefensuche macht nun von M_i aus weiter, und jede Kante, die neu eingefügt wird, hat einen größeren AGE-Wert als alle Kanten aus M_j.

Teil 2)

Ein $SUCC$-Zeiger wird umgehängt. Das Umhängen eines $SUCC$-Zeigers ist dann nötig, wenn eine Kante $\vec{e}_i$ aus M entfernt wird, die nicht die letzte Kante aus $M(\vec{e}_i)$ ist und welche ein nicht leeres $SUCC$-Feld hat. Diese Situation kann beim Entfernen der aktuellen Kante oder bei einem Sprung über $\vec{e}_n$, eintreten. Die beiden Fälle unterscheiden sich nicht, da man $\vec{e}_n$ auch als oberste Kante von M_i ansehen kann, die dann noch in demselben Schritt gelöscht wird. Der Nachfolger der obersten Kante wird also an eine andere Kante gehängt. Diese andere Kante ist die letzte in der $SUCC$-Kette der Kante, die unterhalb der zu löschenden Kante im Stack liegt. Bezeichne $\vec{e}_{top}$ die oberste Kante im Stack, die Kante, welche gelöscht wird. Es gilt $AGE[\vec{e}_{top}] < AGE[SUCC[\vec{e}_{top}]]$. Sei $\vec{e}_i$ die Kante im Stack, die direkt unterhalb von $\vec{e}_{top}$ liegt, dann wissen wir, daß $AGE[\vec{e}_i] < AGE[\vec{e}_{top}]$. Der Beweis der Behauptung folgt durch Induktion über die Länge der $SUCC$-Kette von $\vec{e}_i$. Aus Teil 1 wissen wir für jede Kante $\vec{e}_k$ in M mit $SUCC[\vec{e}_k] \neq NIL$ daß jede Kante, die oberhalb von $\vec{e}_k$ in $M(\vec{e}_k)$ liegt, einen größeren AGE-Wert besitzt als $SUCC[\vec{e}_k]$.

Sei $SUCC[\vec{e}_i] = \vec{e}_k \neq NIL$, dann wird $SUCC[\vec{e}_k] := SUCC[\vec{e}_{top}]$ gesetzt und aus $AGE[\vec{e}_k] < AGE[\vec{e}_{top}] < AGE[SUCC[\vec{e}_{top}]]$ folgt sofort, daß $AGE[\vec{e}_k] < AGE[SUCC[\vec{e}_k]]$.

Sei nun die Länge der Kette gleich a und für alle Kanten $\vec{e}_j, 1 \leq j \leq a$ mit $\vec{e}_j = SUCC[\vec{e}_{j-1}]$, $\vec{e}_0 = \vec{e}_i$ gilt $AGE[\vec{e}_{j-1}] < AGE[\vec{e}_j]$. Die Kante $\vec{e}_a$ ist die zuletzt eingefügte beziehungsweise umgehängte Kante in der Kette. Wurde $\vec{e}_a$ neu angehängt, so gilt wieder automatisch, daß alle Kanten, die oberhalb von $\vec{e}_0 = \vec{e}_i$ neu in $M(\vec{e}_i)$ eingefügt werden, einen größeren AGE-Wert besitzen als $\vec{e}_a$. Wird also ein Nachfolger $\vec{e}$ einer Kante oberhalb von $\vec{e}_0$ an $\vec{e}_a$ gehängt, so gilt sicherlich $AGE[\vec{e}_a] < AGE[SUCC[\vec{e}_a]] = AGE[\vec{e}]$. War $\vec{e}_a$ vorher schon Nachfolger einer anderen Kante $\vec{e}_m$ und ist von dieser aus an $\vec{e}_{a-1}$ angehängt worden, so gilt für alle Kanten, die oberhalb von $\vec{e}_m$ in $M(\vec{e}_i)$ liegen, also auch für $\vec{e}_{top}$, daß sie einen größeren AGE-Wert haben als $\vec{e}_a = SUCC[\vec{e}_m]$. Daraus ergibt sich, daß $AGE[\vec{e}_i] < AGE[\vec{e}_a] < AGE[\vec{e}_{top}] < AGE[SUCC[\vec{e}_{top}]]$ und die Behauptung ist bewiesen. □

Um einen besseren Einblick zu bekommen mache man sich das Setzten der $SUCC$-Zeiger anhand Abbildung 4.10 klar. An dem Beispiel eines Graphen mit einer wesentlichen Brücke ist in der Abbildung dargestellt, wie die $SUCC$-Zeiger die einzelnen Komponenten verbinden und dadurch gewährleisten, daß die Tiefensuche zwischen den Komponenten hin und her springen kann.

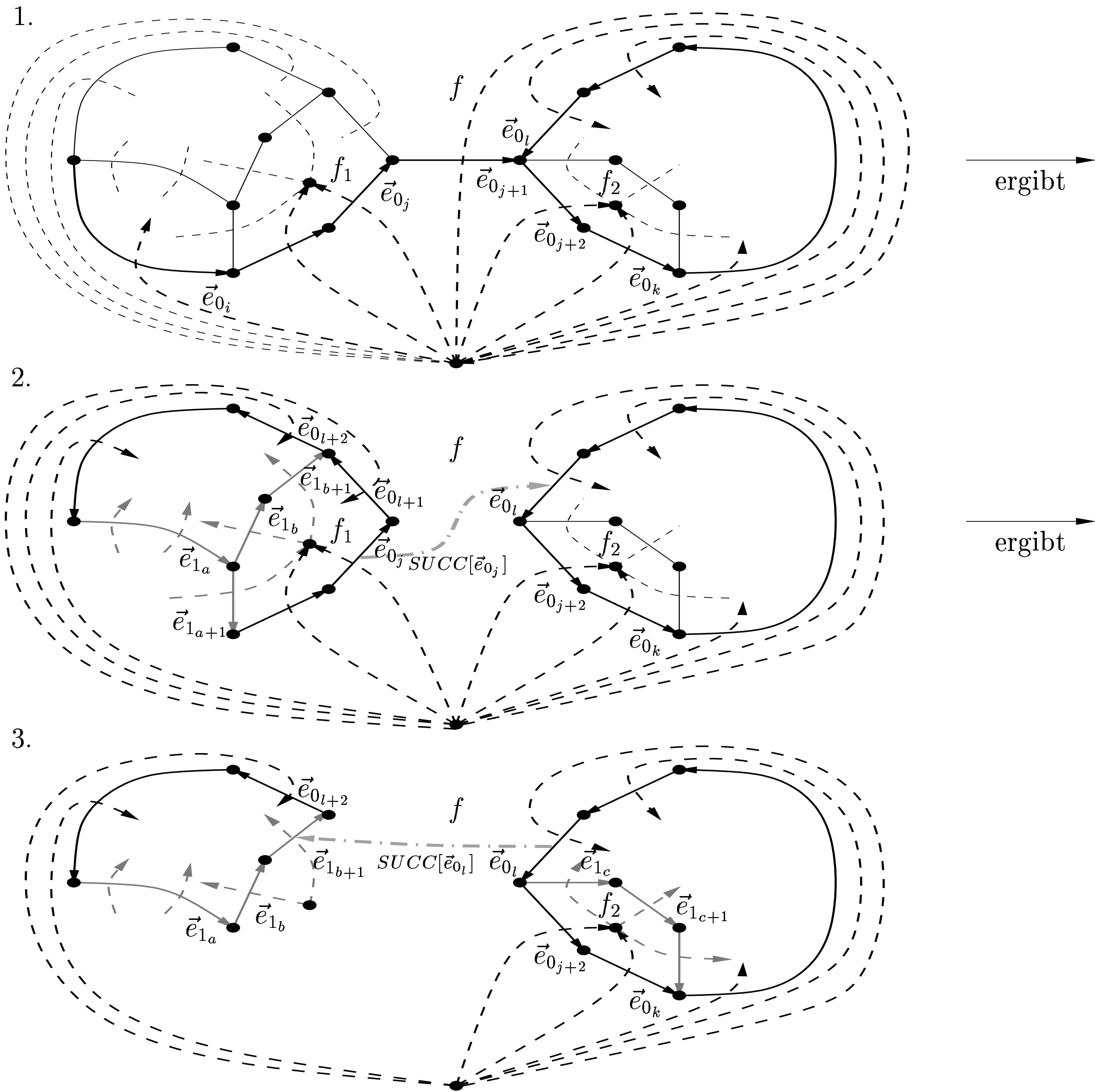


Abbildung 4.10: Das Setzen der $SUCC$-Zeiger durch die Jump-E-DFS$_\circlearrowleft$

Die Menge M wird im Verlauf des Algorithmus in mehrere kleinere Mengen M_k unterteilt. Die Kanten in jeder der Mengen M_k sind nach ihrem Alter sortiert. Betrachtet man M_k als Stack, so sind die Kanten aufsteigend sortiert, je weiter oben im Stack eine Kante liegt, umso größer ist ihr AGE-Wert. Die oberste Kante einer jeden Menge M_k außer der aktuellen Menge $M(\vec{e}_{act})$ ist Nachfolger einer Kante aus einer anderen Menge $M_j \cap M_k = \emptyset$. Jede Menge M_k außer der aktuellen Menge $M(\vec{e}_{act})$ bildet einen geschlossenen Linkskreis von besuchten Kanten. Eine Menge $M_k \neq M(\vec{e}_{act})$ wird erst dann bearbeitet, wenn über einen $SUCC$-Zeiger zu ihr übergegangen wird.

Durch die $SUCC$-Zeiger und die Sortierung der Kanten in den einzelnen Mengen M_k wird eine Sortierung aller Kanten aus M induziert. Die sortierte Menge M läßt sich aus den einzelnen Mengen M_k wie folgt zusammenbauen:
Setze $M := M(\vec{e}_{act})$. Solange es noch Kanten in M mit nicht leerem $SUCC$-Wert gibt, füge oberhalb der Kante $\vec{e}$ mit $SUCC[\vec{e}] = \vec{e}_s$ die Menge $M(\vec{e}_s)$ ein und setze $SUCC[\vec{e}] := NIL$. Abbildung 4.11 soll die Verbindung der Mengen untereinander und die Sortierung der Kanten nach ihrem AGE-Wert veranschaulichen.

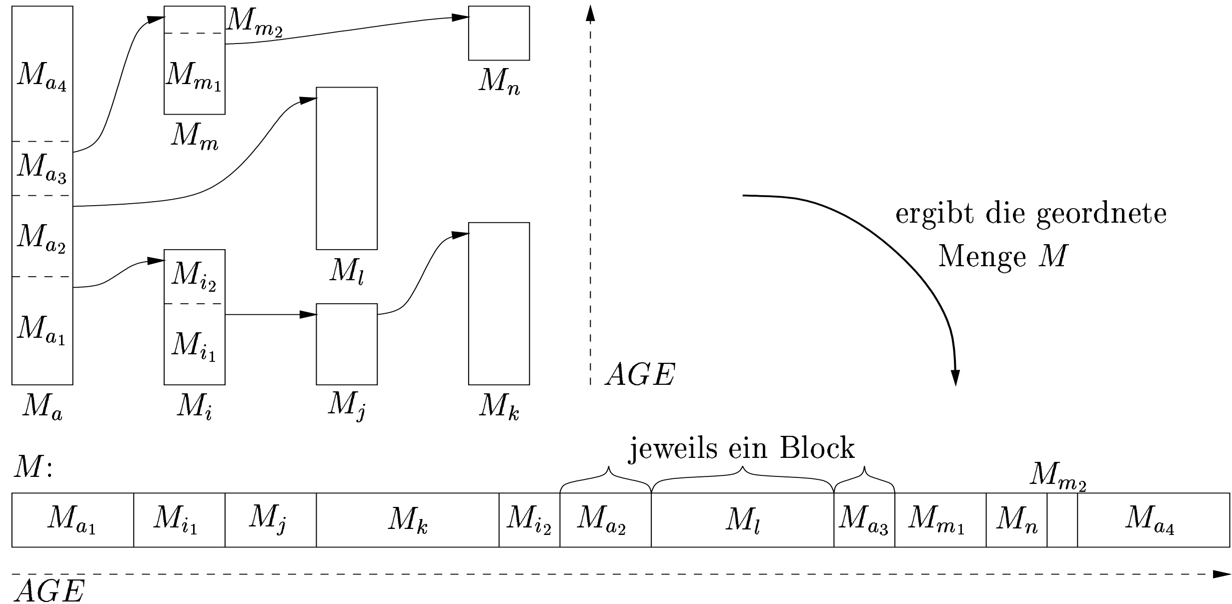


Abbildung 4.11: Mengen M_t und die Sortierung ihrer Kanten nach aufsteigendem AGE-Wert. Die aktuelle Menge $M(\vec{e}_{act})$ ist mit M_a bezeichnet.

Mit den nun gebenen Hilfsmitteln können wir den zentralen Satz dieses Abschnittes relativ leicht zeigen.

Satz 4.4.3
Gegeben ein planar eingebetteter Graph $G = (V, E)$ und ein ausgezeichneter Knoten $v_s \in V$ auf dem Rand der äußeren Facette. Die zur BFS$_\circlearrowleft$ duale Suche ist eine Jump-E-DFS$_\circlearrowleft$.

Beweis:
Um im Beweis wieder die Struktur des Schachtelungsbaumes verwenden zu können, werden wir die Rolle des Graphen G und seines Dualgraphen G^* vertauschen. Das ist keine Einschränkung, da der Dualgraph von G^* ja wieder G ist, gibt uns aber die Möglichkeit, schon bewiesene Eigenschaften zu verwenden.
Die Jump-E-DFS$_\circlearrowleft$ starte auf G mit einer Dummykante $\vec{e}_s := (x, v_s)$ in der äußeren Facette. Die BFS$_\circlearrowleft$ ihrerseits starte auf G^* mit der Dualkante $\vec{e}_s^*$ dieser Startkante. Der Beweis ergibt sich wieder aus der Betrachtung des Schachtelungsbaumes. Wie wir bereits wissen, verhalten sich Tiefen- und Breitensuche auf einem Rand, der nur einen Nachfolger im

Schachtelungsbaum besitzt, dual. Wir müssen uns daher anschauen, wie sich die Jump-E-DFS$_{\circlearrowright}$ verhält, wenn der Schachtelungsbaum verzweigt ist.
Analysieren wir das Vorgehen der Jump-E-DFS$_{\circlearrowright}$. Mit der Startkante $\vec{e}_s$ beginnend arbeitet sie sich im Uhrzeigersinn auf dem Rand der äußeren Facette entlang. Trifft sie dabei zum ersten Mal auf eine schon besuchte Kante $\vec{e}_i$, so ist diese entweder die Startkante selbst, eine wesentliche Brücke des Randes B_0 oder eine Kante der Antenne, auf der v_s liegt, sofern v_s auf einer Antenne liegt.
Ist $\vec{e}_i = \vec{e}_s$ die Startkante, so wird ein trivialer Sprung ausgeführt, durch den $\vec{e}_s$ aus M entfernt wird. Danach enthält M den geschlossenen Linkskreis des einfachen Randes $\hat{B}_0$.
Ist $\vec{e}_i$ eine wesentliche Brücke, so führt die Jump-E-DFS$_{\circlearrowright}$ einen nicht trivialen Sprung aus. Dabei bleibt in M_j der geschlossenen Linkskreis des einfachen Randes der einen Komponente zurück und von M_i aus, also von dem bereits besuchten Teilstück des Randes der anderen Komponente, wird der Rand B_0 weiter abgearbeitet, bis ebenfalls die Startkante erreicht wird.
Liegt v_s auf einer Antenne, so findet die Tiefensuche die zur Wurzel $\hat{v}_s$ inzidente besuchte Kante $\vec{e}_i$. Wieder führt sie einen Sprung aus, der entweder trivial ist, falls v_s das rechteste Blatt der Antenne war, oder nicht. War v_s nicht das rechteste Blatt, so durchsucht die Jump-E-DFS$_{\circlearrowright}$ die noch ungesehenen Bereiche der Antennen von rechts nach links, in der gleichen Reihenfolge wie die Breitensuche die Dualkanten. Ist die Antenne vollständig abgearbeitet, so werden die Kanten nacheinander aus M_i gelöscht und über den $SUCC$-Zeiger zu der letzten gefundenen Kante vor dem Sprung, $\vec{e}_{j_{max}}$, zurückgesprungen. Es ergibt sich die gleiche Situation wie im Fall $\vec{e}_i = \vec{e}_s$.
Nach Bearbeitung des Randes B_0 sind in M alle Kanten des einfachen Randes $\hat{B}_0$ gespeichert. Enthält B_0 wesentliche Brücken, so zerfällt M in mehrere Mengen M_k, wobei k gleich der Anzahl der Komponenten ist, in die G durch Entfernen der wesentlichen Brücken zerfällt. Die geordnete Menge M (man vergleiche hierzu Abbildung 4.11) enthält die Kanten nach aufsteigendem AGE-Wert sortiert. Die direkt hintereinander gefundenen Teilstücke der Linkskreise sind durch senkrechte Linien voneinander abgesetzt. Ein solches Teilstück nennen wir Block. Die Kanten einer Menge M_i sind also zu Blöcken zusammengefaßt, welche die direkt hintereinander gefundenen Kanten enthalten.
Betrachten wir nun die weitere Vorgehensweise der Jump-E-DFS$_{\circlearrowright}$. Die Suche arbeitet sich nun Facette für Facette voran. Die nächste Facette, deren Rand im Uhrzeigersinn abgelaufen wird, ist der Rand der Facette f, die links der untersten Kante aus Stack M liegt. Die Dualkante dieser Kante ist gerade die nächste aktuelle Kante der Breitensuche, denn es ist die erste Kante, über die diese Facette besucht wurde. Man beachte, daß die unterste Kante aus M die Kante mit minimalem AGE-Wert unter allen Kanten aus M ist und, da Tiefen- und Breitensuche die Kanten bisher in der gleichen Reihenfolge gefunden haben, ist es auch die Kante mit kleinstem AGE-Wert für die Breitensuche. Die Jump-E-DFS$_{\circlearrowright}$ läuft den noch unbesuchten Teil des Randes von f ab und trifft dann auf eine schon besuchte Kante $\vec{e}_i$. Durch den Sprung über $\vec{e}_i$ wird sozusagen auf der Kante rückwärts gegangen, um zu sehen, ob der Rand von f vollständig bearbeitet ist. Dabei kann nun einer von insgesamt drei Fällen eintreten:
1. Fall:

Der Rand von f ist besucht. Dann werden alle Kanten aus M_i gelöscht und die $SUCC$-Zeiger jeweils entsprechend umgehängt. Sind alle Kanten entfernt, bilden die $SUCC$-Zeiger eine Kette von der letzten entfernten Kante aus. Zu der ersten Kante in dieser Kette wird nun zurück gesprungen. Es sind zwei Unterfälle zu unterscheiden:
a) Die Kette besteht nur aus einem Glied, das heißt, es wird zu M_j beziehungsweise zu $\vec{e}_{j_{max}}$ zurück gesprungen. Dann war die Sprungkante $\vec{e}_i$ eine Kante innerhalb des untersten Blocks M_0 von M, und nicht die oberste dieses Blocks.
b) Die Kette besteht aus mehreren Gliedern und es wird zu der ersten Kante dieser Kette gesprungen. Der Block, der diese Kante enthält, ist nun der jüngste, beziehungsweise unterste Block von Kanten in M, da der bis dahin darunter gelegene Block gerade entfernt wurde. Durch den Rücksprung wird erreicht, daß von Kanten aus weiter gemacht wird, die von der zuletzt aktuellen Menge M_j aus nicht erreichbar sind.
2. Fall:
Der Rand von f enthält noch unbesuchte Kanten. Dann geht die Tiefensuche solange auf den schon besuchten Kanten des Randes der Facette rückwärts und entfernt sie dabei aus M_i, bis wieder eine unbesuchte Kante rechts wegführt. Dies entspricht dem Vorgehen der Breitensuche, welche auch die gesamte Adjazenzliste der Facette durchgeht, bevor sie ihre Referenzkante aus M entfernt. Die Tiefensuche arbeitet sich weiter im Uhrzeigersinn auf dem Rand der Facette entlang, bis sie - eventuelle über weitere besuchte Kanten - durch einen letzten trivialen Sprung die Kante des Randes von f aus M löscht, die minimalen AGE-Wert besitzt. Der letzte Sprung ist also wieder wie Fall 1 zu behandeln, da dann alle Kanten auf dem Rand von f besucht sind.
Insgesamt ergibt sich, daß nach vollständiger Bearbeitung des Randes einer Facette ein trivialer Sprung ausgeführt wird. Die besuchten Kanten, die dadurch gelöscht werden, enthalten die Information, ob von der Kante, von der aus der Sprung durchgeführt wurde, weitergegangen wird, oder ob in eine andere Komponente gesprungen werden muss. Das ist genau die Information, die die Tiefensuche benötigt, um denselben Weg zu gehen wie die Breitensuche.
Besteht ein $SUCC$-Zeiger zu dem untersten Block M_0 in M, so arbeitet die Tiefensuche von der maximalen Kante $\vec{e}_{0_{max}}$ dieses Blocks aus weiter und kommt mit der nächst rechten Kante auf den Rand der Facette, die links von der minimalen Kante $\vec{e}_{0_{min}}$ des Blocks M_0 liegt. Man beachte, daß die Blöcke Linkskreise bilden. Die Breitensuche macht mit der minimalen Kante aus M_{BFS} als aktueller Kante weiter, welche gerade die Dualkante von $\vec{e}_{0_{min}}$ ist. Die beiden Suchen bearbeiten folglich den Rand derselben Facette.
Besteht kein $SUCC$-Zeiger, so springt die Tiefensuche zu $\vec{e}_{j_{max}}$ zurück und es gibt keine minimaleren Kanten, als $\vec{e}_{j_{min}}$, welche gerade auf $\vec{e}_{j_{max}}$ in dem durch die Kanten aus M_j geschlossenen Linkszykel folgt. Beide Suchen bearbeitet nun den Rand der Facette links von $\vec{e}_{j_{min}}$. Damit ergibt sich die Dualität von BFS$_{\circlearrowright}$ und Jump-E-DFS$_{\circlearrowleft}$. □

Wir haben uns in diesem Kapitel mit der BFS$_{\circlearrowright}$ und der E-DFS$_{\circlearrowleft}$ beschäftigt und gesehen, daß sie im allgemeinen nicht dual zueinander sind. Es ist uns jedoch gelungen, zu jeder der beiden Suchen eine duale Suche vorzustellen, die jeweils durch eine kleine Modifikation aus der Originalversion hervorgeht. Hierbei haben wir uns auf die Betrachtung

der Kanten-Tiefensuche beschränkt, da die Knoten-Tiefensuche praktisch keinerlei Dualität zur Breitensuche aufweist. Im folgenden Kapitel wollen wir uns nun intensiv mit der Knoten-Tiefensuche befassen.

Kapitel 5

Dualität des Tiefensuchbaumes

In diesem Kapitel werden wir uns eine besondere Eigenschaft der Knoten-Tiefensuche auf planaren Graphen anschauen. Die Kanten, über welche jeder Knoten zum ersten Mal gefunden wurde, bilden einen Baum, den Tiefensuchbaum. Der Tiefensuchbaum, der durch eine Right-First-Knoten-Tiefensuche entsteht, weist einige spezielle Merkmale auf, die wir im folgenden genauer betrachten werden.

5.1 Die Right-First-Knoten-Tiefensuche

In 3.1 haben wir Knoten-Tiefensuche definiert und einen Algorithmus vorgestellt. Da wir weiterhin auf planaren Graphen arbeiten, werden wir wieder eine feste kombinatorische Einbettung des Eingabegraphen voraussetzen. Die Tiefensuche, die wir in diesem Kapitel betrachten, ist die Right-First-Knoten-Tiefensuche. Wir wollen uns die besonderen Eigenschaften des durch den V-DFS$_{\circlearrowleft}$-Algorithmus entstehenden Tiefensuchbaumes anschauen.

Algorithmus 5.1.1 (V-DFS$_{\circlearrowleft}$)
Eingabe: planar eingebetteter Graph $G = (V, E)$ und ein Startknoten $v_s \in V$ auf dem Rand der äußeren Facette.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$ in der äußeren Facette.

Knotenauswahl

(1) Endknoten v der Kante (u, v), die zuletzt in M eingefügt wurde

Kantenauswahl$(v; (u, v))$

(1) Falls v aktiv

(2) wähle die im Gegenuhrzeigersinn von (u, v) aus nächste unbesuchte zu v inzidente Kante $\{v, w\}$ und orientiere sie: (v, w)

(3) falls w unbesucht

(4) lege (v, w) in M ab

(5) gib (v, w) aus

(6) ansonsten

(7) entferne (u, v) aus M

Die in M gespeicherten Kanten ergeben einen Right-First-Tiefensuchbaum. In Abbildung 5.1 ist das Vorgehen der Knoten-Tiefensuche und der entstehende Suchbaum an einem Beispiel aufgezeigt.

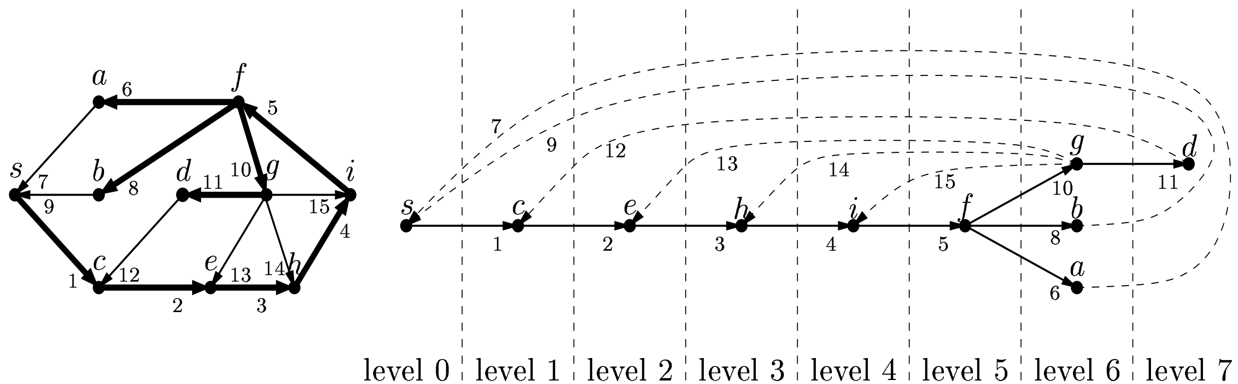


Abbildung 5.1: Right-First-Knoten-Tiefensuche angewendet auf planaren Graphen und der zugehörige Suchbaum mit gestrichelten Nichtbaumkanten. Der Suchbaum ist im Graphen durch dicke Linien gekennzeichnet. Die Pfeile und Zahlen geben die Orientierung und den AGE-Wert der Kanten an.

5.2 Dualitätseigenschaft des Tiefensuchbaumes

Betrachten wir den V-DFS$_{\circlearrowleft}$-Baum T etwas genauer. Der Knoten-Tiefensuchbaum ist wie alle Suchbäume ein aufspannender Baum im Graphen $G = (V, E)$. Betrachtet man den Dualgraphen $G^* = (V^*, E^*)$ von G und in diesem den Untergraphen, der durch die Dualkanten der Nichtbaumkanten induziert wird, so stellt man schnell fest, daß dieser Untergraph in G^* ebenfalls einen aufspannenden Baum bildet. Schauen wir uns dazu in Abbildung 5.2 ein Beispiel an.

Wie man an den Kantennumerierungen sofort erkennt, ist die Kantenaufnahmereihenfolge der beiden Suchen auf G und G^* völlig verschieden. Wir werden uns im folgenden auch nicht die Kantenaufnahmereihenfolge ansehen, sondern die Struktur der durch die V-DFS$_{\circlearrowleft}$ entstehenden Suchbäume in Graph und Dualgraph betrachten. Dazu werden wir zuerst den folgenden Satz beweisen, welcher ein älteres wohlbekanntes Resultat benennt.

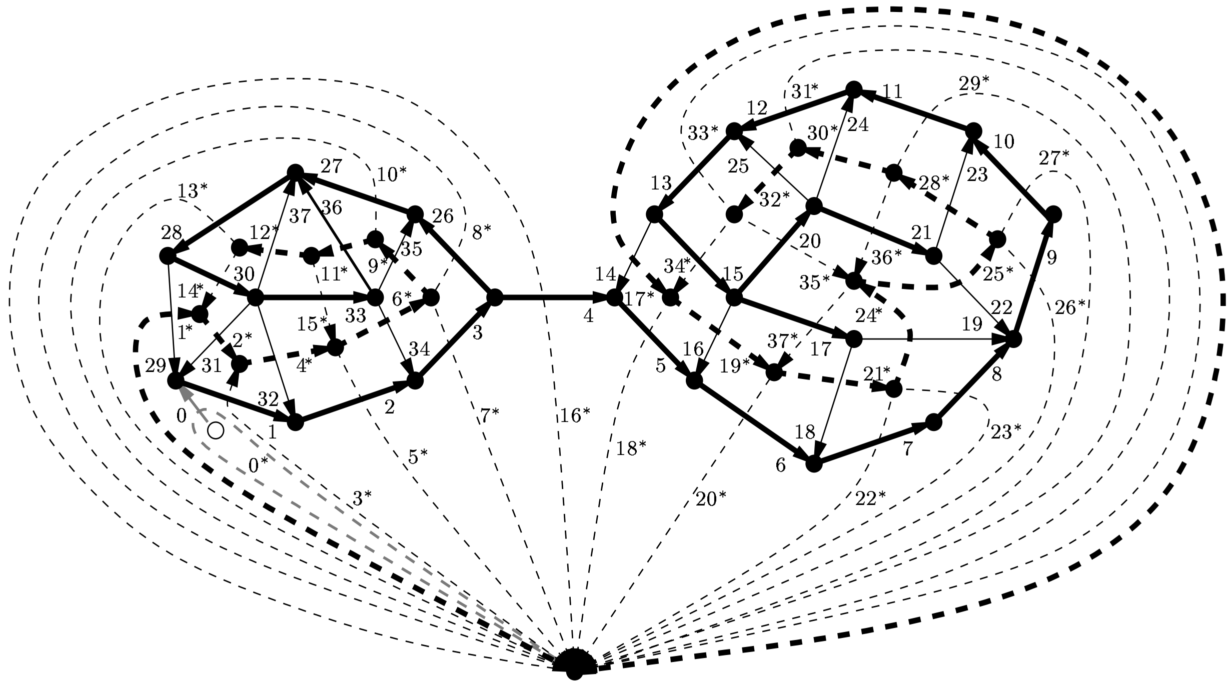


Abbildung 5.2: Right-First-Knoten-Tiefensuche auf planarem Graph und auf dem Dualgraph. Die dicken Linie kennzeichnen die entandenen Suchbäume und Zahlen an den Kanten geben die Kantennumerierungen gemäß den Suchen an.

Satz 5.2.1
Gegeben ein planar eingebetteter zusammenhängender Graph $G = (V, E)$. Sei T ein aufspannender Baum in G. Der durch die Menge der Dualkanten der Nichtbaumkanten induzierte Untergraph T^* von $G^* = (V^*, E^*)$ ist ein aufspannender Baum in G^*.

Beweis:
Es ist zu zeigen, daß T^* zusammenhängend ist. Außerdem muß gezeigt werden, daß T^* genau $|V^*| - 1$ viele Kanten besitzt.
1. Für einen aufspannenden Baum muß $|V^*| = |V(T^*)|$ gelten. Angenommen T^* wäre nicht zusammenhängend, dann existiert ein Schnitt $(S, V^* \backslash S)$ in G^*, der wenigstens zwei der Zusammenhangskomponenten von T^* trennt. Die Menge $E_S := \{ \{u, v\} \mid u \in S, v \in V^* \backslash S \}$ der Schnittkanten enthält keine Kanten aus $E(T^*)$. Die Dualkanten der Kanten aus E_S sind also Baumkanten in G. Nach dem Satz von Menger (siehe zum Beispiel [Jun94]) bilden die Dualkanten eines Schnittes einen Kreis im Dualgraphen. Dies ist im Widerspruch dazu, daß T ein Baum und somit kreisfrei ist.
2. Es bleibt somit noch zu zeigen, daß T^* genau $|V^*| - 1$ viele Kanten besitzt. Hierzu schauen wir uns die schon aus Abschnitt 2.5 bekannte Eulersche Polyederformel an. Wie gewohnt sei $n = |V|$, $m = |E|$ und f die Anzahl der Facetten von G. Daraus folgt, daß

$f = |V^*|$. Außerdem ist $m = |E(T)| + |E(G\backslash T)| = |E(T)| + |E(T^*)|$.

$$\begin{aligned}
n - m + f &= 2 \\
\iff \quad |V| - (\,|E(T)| + |E(T^*)|\,) + |V^*| &= 2 \\
\iff \quad |V| - |E(T)| - 2 + |V^*| &= |E(T^*)| \\
\iff \quad |V| - |V(T)| + 1 - 2 + |V^*| &= |E(T^*)| \\
\iff \quad |V| - |V| - 1 + |V^*| &= |E(T^*)| \\
\iff \quad |V^*| - 1 &= |E(T^*)|
\end{aligned}$$

T^* hat also genau $|V^*| - 1$ Kanten, $|V^*|$ Knoten und ist zusammenhängend. Somit ist T^* ein aufspannender Baum in G^*. □

Dieses Resultat ist sozusagen Folklore und wird im allgemeinen nicht explizit angegeben. Wir wissen nun daß der durch die Dualkanten der Nichtbaumkanten induzierte Teilgraph von G^* ein aufspannender Baum ist. Im folgenden werden wir zeigen, daß dieser aufspannende Baum durch eine V-DFS$_{\circlearrowleft}$ auf G^* mit entsprechendem Startknoten induziert wird. Die folgenden Lemmas werden uns den Beweis des zentralen Satzes vereinfachen.

Lemma 5.2.2
Gegeben ein planarer Graph $G = (V, E)$ und ein ausgezeichneter Startknoten $v_s \in V$ auf dem Rand der äußeren Facette. Die Dummykante werde wie üblich in die äußere Facette eingebettet. Jede Nichtbaumkante schließt einen Linkskreis, dessen Kanten bis auf die schließende Nichtbaumkante nur aus Baumkanten besteht.

Beweis:
Es ist leicht zu sehen, daß jede Nichtbaumkante einen Kreis schließt. Da der entstehende Suchbaum T ein aufspannender Baum in G ist, gibt es zwischen je zwei Knoten $v, w \in V(G)$ einen Weg W in T. Jede beliebige Nichtbaumkante $e \in E(G\backslash T)$ zwischen zwei Knoten $v, w \in V$ schließt dann den Kreis bestehend aus dem Weg W in T und der Kante e.
Schauen wir uns nun das Vorgehen der V-DFS$_{\circlearrowleft}$ genauer an. Wie auch die E-DFS$_{\circlearrowleft}$ arbeitet sie sich von der Dummykante aus auf dem Rand der äußeren Facette entlang. Trifft sie nun auf einen schon besuchten Knoten v, so ist dieses entweder der Startknoten oder ein Separator. Die neu aufgenommene Kante (u, v) schließt einen Kreis. Da die Dummykante in der äußeren Facette liegt, , kann sie nicht innerhalb des Kreises liegen. Angenommen der Kreis wäre ein Rechtskreis, dann ist am Knoten v zu einem früheren Zeitpunkt die Kante (v, w) gewählt worden, im Widerspruch zur Right-First-Auswahlregel, welche die zu diesem Zeitpunkt unbesuchte und weiter rechts gelegene Kante (v, u) gewählte hätte. Man vergleiche hierzu auch Abbildung 3.5 in Lemma 3.2.3. Demnach ist der geschlossene Kreis ein Linkskreis.
Danach arbeitet sich die Tiefensuche ins Innere des Kreises vor. Löscht man die gefundene Nichtbaumkante aus G, so befindet sich die Suche wieder auf dem Rand der äußeren Facette. Es ergibt sich also die gleiche Situation wie oben.
Da ein aufspannender Baum maximal kreisfrei ist, folgt, daß jede Nichtbaumkante e einen Kreis schließt, der nur aus Baumkanten und der Kante e besteht. □

*

Lemma 5.2.3
Direkt nacheinander geschlossene Linkskreise sind entweder ineinander geschachtelt, der später geschlossene in den davor geschlossenen, oder sie sind facetten- und kantendisjunkt. Sind die beiden Linkskreise ineinander geschachtelt, so grenzt das neue Wegstück des kleineren Kreises außen nur an eine einzige Facette.

Beweis:
Sei Kante $\vec{e}_i$ die Nichtbaumkante durch die der letzte Linkskreis C_i geschlossen wurde. Wir betrachten nun die innerhalb liegenden Facetten und haben dabei zwei Fälle zu unterscheiden.
Fall a)
Das Innere von C_i besteht aus mehreren Facetten. Da der gerade geschlossene Linkskreis der rechteste unbesuchte Weg war, gibt es keine zu einem Knoten des Kreises, außer dem Endknoten von $\vec{e}_i$, inzidente außerhalb des Kreises liegende unbesuchte Kanten. Die innerhalb liegenden Kanten sind dementsprechend unbesucht. Vom Anfangsknoten von $\vec{e}_i$ aus sucht die Tiefensuche nun die nächst rechte unbesuchte Kante. In Abbildung 5.3 sind drei der vier Varianten für einen nächsten Linkskreis dargestellt.

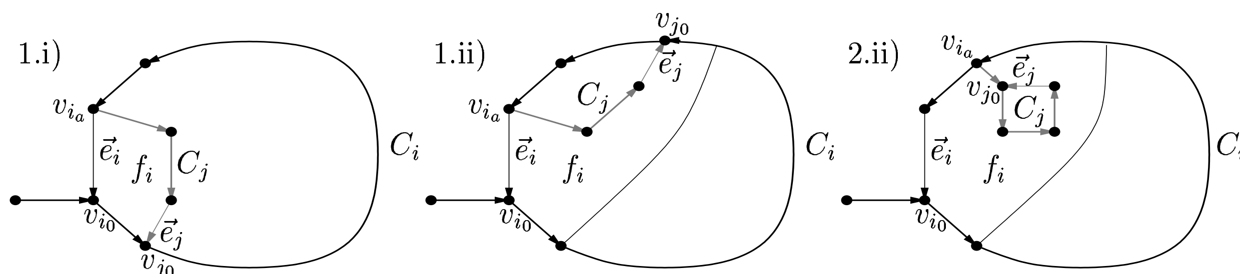


Abbildung 5.3: Linkkreis C_i und darauffolgender innen gelegener Linkskreis C_j

Nachdem die V-DFS$_{\circlearrowleft}$ eine Nichtbaumkante gefunden hat, macht sie vom Endknoten der zuletzt aufgenommenen Baumkante aus weiter und geht den nächst rechten Weg um die Facette f_i rechts von der Nichtbaumkante $\vec{e}_i$ herum. Auf diesem Weg trifft sie irgendwann auf eine Kante mit schon besuchtem Endknoten. Dieser Knoten v_{j_0} liegt entweder auf dem Kreis C_i, wie in Skizze 1.i und 1.ii der Abbildung 5.3, oder nicht, so wie in Skizze 2.ii). Es ist klar, daß in allen vier Fällen das neue Wegstück rechts nur an eine einzige Facette angrenzt, da im Inneren des Kreises C_i noch alle Kanten unbesucht sind und durch die Right-First-Auswahl der rechteste Weg gewählt wird. Somit bleibt die Tiefensuche auf dem Rand der Facette f_i. Die drei in Abbildung 5.3 dargestellten Varianten unterscheiden sich wie folgt:
1.i) Beim Schließen des Kreises C_j durch die Nichtbaumkante $\vec{e}_j$ ist der Rand der Facette f_i vollständig besucht. Die Linkskreise C_i und C_j haben das gerichtete Wegstück zwischen v_{j_0} und v_{i_a} gemeinsam. Wenn die Tiefensuche das gesamte Innere von C_j besucht hat, wird sie durch Backtrack Schritte nicht nur außerhalb von C_j zurückkehren müssen, sondern sogar außerhalb von C_i, da zwischen C_i und C_j keine unbesuchten Kanten mehr existieren.

1.ii) Wie in 1.i) trifft die V-DFS$_{\circlearrowleft}$ auf einen Knoten von C_i. Diesmal ist aber der Rand der Facette f_i noch nicht vollständig abgearbeitet. Es gibt aber keine unbesuchte Verbindung mehr in die anderen Facetten des Inneren von C_i. Die beiden Kreise haben wieder das Wegstück zwischen v_{j_0} und v_{i_a} gemeinsam. Hat die Tiefensuche das Innere von C_j abgearbeitet, so wird sie zum Rand von C_i zurückkehren und von dort aus die Ränder der anderen Facetten im Inneren des Kreises C_i besuchen.
2. Die dritte Möglichkeit ist, daß die V-DFS$_{\circlearrowleft}$ nicht auf einen Knoten des Kreises C_i trifft, sondern auf einen Knoten des neuen Wegstückes. Dann besitzen die Kreise C_i und C_j kein gemeinsames Wegstück und C_j liegt komplett in Facette f_i. Es gibt wieder die beiden Möglichkeiten i) und ii) wie in 1. In Fall i) ist $\vec{e}_j$ die letzte unbesuchte Kante auf dem Rand der Facette f_i und die Tiefensuche wird bis außerhalb von C_i zurückkehren müssen um eventuell noch vorhandene unbesuchte Kanten zu finden. In Fall ii) ist der Rand von f_i noch nicht ganz bearbeitet und die Tiefensuche findet nach Abarbeitung des Inneren von C_j vom Rand von C_i aus noch unbesuchten Kanten innerhalb des Kreises C_i.
Fall b)
Das Innere von C_i besteht aus einer einzigen Facette. Da die Tiefensuche die Right-First-Auswahlregel benutzt, gibt es rechts des Kreises keine unbesuchte Kante mehr, sonst wäre beim Ablaufen des Kreises diese Regel verletzt worden. Innerhalb liegen nach Voraussetzung eventuell noch Antennen aber keine zwei unterschiedlichen Facetten. Das heißt, es kann innerhalb von C_i kein Kreis mehr geschlosssen werden. Falls es außer den eventuell vorhandenen Antennen innerhalb von C_i noch unbesuchte Kanten in G gibt, so grenzen diese nicht an einen Knoten von C_i. Die Tiefensuche muß demnach durch Backtrack Schritte bis ganz außerhalb des Kreises C_i zurückkehren um noch unbesuchte Kanten zu finden. Die Tiefensuche entfernt so lange die zuletzt eingefügten Kanten aus M, bis sie auf dem Rand eines Kreises C_k angelangt, dessen Inneres noch nicht abgearbeitet ist, oder bis sie auf dem Rand der äußeren Facette landet.
Ist das Innere des Kreises C_k noch nicht vollständig bearbeitet, so findet die Tiefensuche von C_k aus noch unbesuchte Kanten. Wie in 1.ii und 2.ii aus Fall a) zu sehen ist, sind diese Kanten nur über den Kreis C_k mit dem schon besuchten Kreis C_i verbunden. Der neu geschlossene Linkskreis C_j und C_i sind also facetten- und auch kantendisjunkt.
Muß die Tiefensuche bis auf den Rand der äußeren Facette zurück gehen, so ist der Algorithmus entweder beendet, falls der Rand der äußeren Facette schon vollständig abgearbeitet ist, oder der Graph besitzt einen Separator, von dem aus eine durch ihn abgetrennte Komponente nun besucht wird. Zwei durch einen Separator getrennte Komponenten sind natürlich facetten- sowie kantendisjunkt, womit das Lemma bewiesen ist. □

Aus diesem Lemma folgt sofort, daß zwei beliebige durch die V-DFS$_{\circlearrowleft}$ geschlossene Linkskreise entweder ineinander geschachtelt oder facetten- sowie kantendisjunkt sind.
Das nächste Lemma macht bereits eine Aussage über das Verhalten der Tiefensuche auf G^*. Um die Suchen auf den Graphen G und G^* auseinander halten zu können, werden wir den zugrundeliegenden Graphen jeweils in Klammern angeben.

Lemma 5.2.4
Zu dem Zeitpunkt, da die V-DFS$_{\circlearrowleft}(G^*)$ eine neue Baumkante $\vec{e}^*$ in die Menge M aufnimmt,

sind alle Facetten, die direkt rechts an den durch $\vec{e}$ geschlossenen Linkskreis C von G angrenzen, besucht und alle Facetten innerhalb dieses Kreises unbesucht.

Beweis:
Der Beweis erfolgt per Induktion über die aufgenommenen Baumkanten aus G^*.
Ausgangslage: Die äußere Facette f_s ist besucht. Die erste aufgenommene Baumkante $\vec{e}_1^*$ führt von f_s in eine innere Facette. Diese Kante $\vec{e}_1^*$ ist die Dualkante der Kante des einfachen Randes von G, die durch die V-DFS$_{\circlearrowleft}(G)$ zuletzt aufgenommen wird. Sie ist die Nichtbaumkante von G mit Endknoten $\hat{v}_s$ und schließt Kreis C_1. Zu dem Zeitpunkt da $\vec{e}_1^*$ aufgenommen wird, sind höchstens einige Schleifen inzident zu f_s gefunden worden. Demnach sind alle Facetten innerhalb des Randes von G unbesucht und alle außen an C_1 angrenzenden Facetten sind besucht, dies ist nur f_s, und die Bedingung ist erfüllt.
Induktionsschritt: Sei die Voraussetzung für Baumkante $\vec{e}_i^*$ erfüllt, das heißt, die außen an den durch $\vec{e}_i$ geschlossenen Linkskreis C_i angrenzenden Facetten sind besucht; alle Facetten innerhalb sind noch unbesucht. Die V-DFS$_{\circlearrowleft}$ nimmt die Baumkante $\vec{e}_i^*$ auf und sucht von ihrem Endknoten f_i aus weiter. Damit sind alle außen an C_i angrenzenden Facetten und außerdem Facette f_i besucht. Es gibt nun die Möglichkeit, daß f_i die einzige innere Facette von C_i ist, oder daß noch unbesuchte Facetten im Inneren von C_i existieren.
a) f_i ist einzige innere Facette von C_i.
Dann kann die Tiefensuche keine weitere Baumkante mehr finden. Sie bearbeitet eventuell vorhandene Schleifen inzident zu f_i und muß dann durch Backtrack Schritte auf dem bisher zurückgelegten Weg rückwärts gehen. Dabei kommt sie entweder irgendwann zur Startkante zurück, falls die Komponente, zu der C_i gehört, vollständig abgearbeitet ist, oder sie macht von einer Baumkante $\vec{e}_k^*$ aus weiter.
Ist die C_i enthaltende Komponente abgearbeitet, so ist entweder G^* vollständig durchsucht und die V-DFS$_{\circlearrowleft}(G^*)$ ist fertig, oder sie findet von f_s aus eine noch unbesuchte Kante, die in das Innere einer noch nicht besuchten Komponente führt. Dann sind die in dieser Komponente gelegenen Facetten unbesucht und sie grenzt außen nur an f_s an. Die Behauptung ist also richtig.
Kommt die V-DFS$_{\circlearrowleft}$ durch Backtrack Schritte zu $\vec{e}_k^*$ zurück, so ist ein Teil der inneren Facetten des zugehörigen Kreises C_k abgearbeitet. Diese Facetten bilden das Innere eines oder mehrerer Linkskreises C_{l_r}, welche vollständig durchsucht sind. Der Endknoten von $\vec{e}_k^*$ werde mit f_k bezeichnet. Findet die Tiefensuche von f_k aus eine neue Baumkante $\vec{e}_j^*$, so betritt sie das Innere eines anderen Linkskreises C_j, der ebenfalls innerhalb von C_k liegt. Aus Lemma 5.2.3 und der Tatsache, daß C_k der kleinste Kreis ist, dessen Inneres noch nicht vollständig abgearbeitet ist, ergibt sich, daß C_j und das Innere der Kreise C_{l_r} facettendisjunkt sind. Das Innere von C_j ist folglich noch unbesucht. Das Äußere ist entweder nur die Facette f_k, wie in Fall a) 2. des Beweises zu Lemmas 5.2.3, oder aber f_k zusammen mit einem Teil des Äußeren von C_k, welches nach Voraussetzung besucht ist. Damit stimmt die Behauptung auch in diesem Fall.
b) Es existieren noch unbesuchte Facetten innerhalb des Kreises C_i.
Bis auf f_i sind diese nach Voraussetzung alle unbesucht. Wählt die V-DFS$_{\circlearrowleft}$ als nächste Baumkante $\vec{e}_j^*$, so betritt sie das Innere eines Kreises C_j. Dieser grenzt außen entweder

nur an Facette f_i oder an f_i und an das Äußere von C_i an. Vergleiche hierzu 1. und 2. aus Fall a im Beweis von Lemma 5.2.3. Die außen an C_j angrenzenden Facetten sind folglich besucht. Das Innere ist unbesucht und somit ist das Lemma bewiesen. □

Satz 5.2.5
Gegeben ein planar eingebetteter Graph $G = (V, E)$ und ein Startknoten $v_s \in V$ auf dem Rand der äußeren Facette. T sei der Tiefensuchbaum einer V-DFS$_{\circlearrowleft}$ auf G. Dann ist der Baum T^*, der durch die Dualkanten der Nichtbaumkanten $E(G \backslash T)$ in G^* induziert wird, der Tiefensuchbaum einer V-DFS$_{\circlearrowleft}$ in G^*.

Beweis:
Sei $\vec{e}_0$ die von der V-DFS$_{\circlearrowleft}(G)$ in die äußere Facette von G eingefügte Dummykante. Der die äußere Facette von G repräsentierenden Knoten f_s ist der Startknoten und die Dualkante von $\vec{e}_0$ die einzufügende Dummykante der V-DFS$_{\circlearrowleft}(G^*)$.
Der Beweis gliedert sich in zwei Teile: Im ersten Teil wollen wir die Wahl der Startkante $\vec{e}_0^*$ der V-DFS$_{\circlearrowleft}(G^*)$ überprüfen. Der zweite Teil besteht dann darin, das Vorgehen der Tiefensuche in G^* zu betrachten.
Teil 1) Wahl des Startknotens und der Dummykante für die V-DFS$_{\circlearrowleft}(G^*)$.
Die Dummykante der Tiefensuche auf G ist in die äußere Facette eingebettet. Ihre Dualkante ist demnach eine Schleife an dem die äußere Facette repräsentierenden Knoten f_s. Damit ist f_s der Startknoten der Tiefensuche in G^*. Liegt der Startknoten v_s auf einer Antenne, so wollen wir die Wurzel dieser Antenne betrachten, denn alle Dualkanten zu Kanten einer Antenne sind Schleifen und somit keine Baumkanten. Wir bezeichnen die Wurzel der Antenne wieder mit $\hat{v}_s$. Liegt v_s selbst auf dem echten Rand, so ist $\hat{v}_s = v_s$.

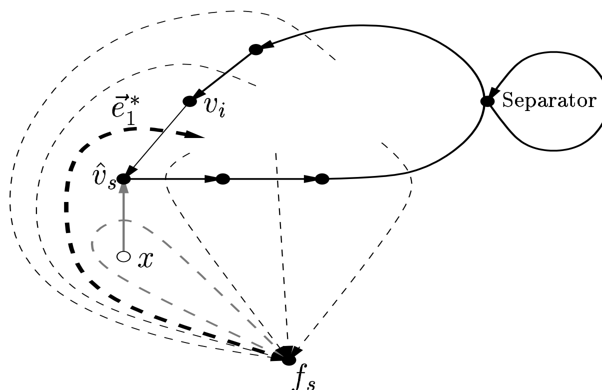


Abbildung 5.4: Die erste Baumkante $\vec{e}_1^*$ (dick) der V-DFS$_{\circlearrowleft}$ in G^*. Die jeweiligen Dummykanten sind grau eingezeichnet.

Die Tiefensuche auf G geht von v_s aus auf dem Rand B entlang, so herum, daß die äußere Facette rechts des Weges liegt. Hat der Graph G keinen Separator, so kommt die Tiefensuche irgendwann wieder bei $\hat{v}_s$ an. Besitzt der Graph einen Separator, so wird erst die durch ihn von der den Startknoten enthaltenen Komponente getrennte Komponente abgearbeitet und dann zu dem Separator durch Backtrack Schritte zurückgegangen. Von dort aus geht

die Tiefensuche weiter auf dem Rand entlang. Irgendwann hat sie alle durch Separatoren abgetrennten Komponenten bearbeitet und durchläuft das letzte Teilstück des Rands. Die letzte Kante auf diesem Teilstück hat den Endknoten $\hat{v}_s$ und ist somit eine Nichtbaumkante. Die Dualkante zu dieser Kante ist die erste Baumkante im Tiefensuchbaum der V-DFS$_{\circlearrowleft}(G^*)$. In Abbildung 5.4 ist die Situation graphisch dargestellt.
Von der Startkante $\vec{e}_0^*$ der Tiefensuche in G^* aus ist die im Gegenuhrzeigersinn nächste unbesuchte Kante, deren Endknoten nicht f_s ist, gerade die Dualkante $\vec{e}_1^*$ der letzten durch die V-DFS$_{\circlearrowleft}(G)$ aufgenommene Kante des Randes B des Graphen. Somit erfüllt die gewählte Startkante die Bedingung, daß die erste aufgenommene Baumkante die Dualkante der Nichtbaumkante mit Endknoten $\hat{v}_s$ ist.
Teil 2)
Schauen wir uns nun das Vorgehen der V-DFS$_{\circlearrowleft}(G^*)$ mit der entsprechenden Startkante genauer an. Zu Beginn ist nur der die äußere Facette von G repräsentierende Knoten f_s besucht. Die Kanten des Graphen G seien gemäß der V-DFS$_{\circlearrowleft}(G)$ durchnumeriert. In Lemma 5.2.4 haben wir gezeigt, daß die V-DFS$_{\circlearrowleft}(G^*)$ über die Baumkanten Linkskreise betritt, deren außen angrenzenden Facetten schon besucht sind. Daraus ergibt sich, daß sie zuerst das gesamte Innere eines solchen Linkskreises abarbeiten muss, bevor sie ihn durch Backtrack Schritte wieder verlassen kann. Erreicht die Tiefensuche durch Backtrack Schritte einen Kreis C_k, dessen Inneres noch nicht abgearbeitet ist, so haben wir in Lemma 5.2.3 gesehen, daß dann die inneren Linkskreise durch die Facette f_k vollständig voneinander getrennt werden. Es bleibt zu zeigen, daß die jeweilige Nichtbaumkante der Tiefensuche in G die rechteste Möglichkeit ist, in einen solchen neuen Linkskreis vorzudringen.
Hierzu müssen wir unterscheiden, ob es sich a) um einen Linkskreis bestehend aus Kanten des Randes handelt, oder b) um einen inneren Linkskreis.
Fall a) Enthält der Rand keinen Separator, so ist nichts zu zeigen, da die Startkante durch die Startkante der Tiefensuche auf G festgelegt wird. Enthält der Graph einen Separator, so gilt dieses Argument auch für die durch den Separator abgetrennte Komponente, welche die Startkante enthält. Betrachten wir nun eine durch den Separator v_i abgetrennte Komponente $Komp_i$. Die Tiefensuche auf G geht wie bekannt von der Startkante aus im Uhrzeigersinn um den Rand der äußeren Facette. Nach Umrundung der Komponente $Komp_i$ findet sie mit Nichtbaumkante $\vec{e}_i$ den Separator v_i. Kante $\vec{e}_i$ ist also im Uhrzeigersinn die letzte Kante auf dem Rand der äußeren Facette, die noch zu $Komp_i$ gehört. Somit ist sie im Gegenuhrzeigersinn die erste Kante auf dem Rand der Komponente. Die V-DFS$_{\circlearrowleft}(G^*)$ wird also in der Adjazenzliste von f_s ihre Dualkante $\vec{e}_i^*$ vor den anderen Kanten mit Endknoten in $Komp_i$ finden.
Fall b) Sei $\vec{e}_i^*$ die letzte durch die V-DFS$_{\circlearrowleft}(G^*)$ aufgenommene Baumkante. Die Tiefensuche befindet sich also innerhalb des Linkskreises C_i und sucht nun weiter nach Kanten mit noch unbesuchtem Endknoten. Jeder der direkt innerhalb von C_i gelegenen Linkskreise C_{j_r} grenzt mit einem Teilstück seines Randes an f_i, dem Endknoten von $\vec{e}_i^*$. Diese Teilstücke werden von der V-DFS$_{\circlearrowleft}(G)$ im Laufe des Algorithmus von f_i aus gesehen im Uhrzeigersinn durchaufen. Die Nichtbaumkante von G ist dann immer die letzte Kante des Wegstücks. Die V-DFS$_{\circlearrowleft}(G^*)$ durchläuft die Adjazenzliste von f_i im Gegenuhrzeigersinn. Die erste Kante eines jeden Wegstücks ist somit genau die Dualkante der Nichtbaumkante

in G. Also bildet eine Nichtbaumkante in G immer gerade die rechteste Möglichkeit in einen noch unbesuchten Linkskreis vorzudringen und der Satz ist bewiesen. □

Die Reihenfolge, in der die Tiefensuche auf G und die Tiefensuche auf G^* die Kanten aufnehmen, ist nicht die gleiche. Dennoch sind gewisse Regelmäßigkeiten erkennbar. Tatsächlich kann man aus dem Tiefensuchbaum der V-DFS$_\circlearrowleft(G)$ die Aufnahmereihenfolge der Kanten einer V-DFS$_\circlearrowleft(G^*)$ mit derselben Dummykante ablesen. Um dies zeigen zu können, werden wir zuvor noch ein Lemma beweisen.

Lemma 5.2.6
Die Nichtbaumkanten in G^* sind die entgegengerichteten Dualkanten der Baumkanten von G.

Beweis:
Bei einer Tiefensuche werden Kanten immer vom zuletzt gefundenen Knoten aus besucht. Daraus folgt sofort, daß Nichtbaumkanten immer von später gefundenen Knoten zu davor gefundenen Knoten gerichtet sind. Aus Lemma 3.2.3 wissen wir, daß die Tiefensuche in G mit jeder Nichtbaumkante einen Linkskreis schließt. Das Innere eines Kreises liegt also links des Weges. Das Lemma 5.2.4 besagt, daß zu dem Zeitpunkt, da die Tiefensuche in G^* über die Dualkante einer Nichtbaumkante von G einen Linkskreis betritt, alle außen angrenzenden Facetten besucht sind und alle inneren unbesucht. Damit ergibt sich, daß alle inneren Facetten eines Kreises später gefunden werden als die angrenzenden äußeren. Daraus folgt mit dem Ergebnis von oben, daß die Nichtbaumkanten von G^* immer von innen nach außen gerichtet sind und somit von links nach rechts über die Baumkanten von G hinweglaufen. Die Dualkante $\vec{e}^*$ einer gerichteten Kante $\vec{e}$ ist aber als von rechts nach links gerichtet definiert worden, also gerade entgegengesetzt. □

Um den Tiefensuchbaum analysieren zu können wollen wir uns zuerst Gedanken darüber machen, wie er gezeichnet werden soll. Der V-DFS$_\circlearrowleft$-Baum soll von unten nach oben aufgebaut werden, so daß der Startknoten $v_s \in V$, welcher die Wurzel des Baumes bildet, als unterster Knoten zu liegen kommt. Der rechteste Weg im Graphen bilde den rechtesten Ast im Baum. Alle nach Backtrack Schritten gefundenen Wege liegen links des rechtesten Astes. Die Nichtbaumkanten werden links des Baumes vorbeigeführt, so daß sie von links auf ihren Endknoten treffen. In jedem Knoten werden die Adjazenzreihenfolgen gewahrt, das heißt, daß die Kanten im Baum in derselben Reihenfolge an ihren Endknoten hängen wie im Graph. Diese Zeichnung des Baumes läßt sich planar einbetten, da der Originalgraph planar ist und die Adjazenzreihenfolgen beibehalten werden. Man betrachte Abbildung 5.1 und drehe den Tiefensuchbaum um 90° im Gegenuhrzeigersinn. Dadurch werden die Facetten erhalten und nur ihre Form verändert. Wenn wir im folgenden vom Tiefensuchbaum reden, so gehen wir davon aus, daß er in der beschriebenen Weise gezeichnet ist.
Wir stellen nun einen Kanten-Tiefensuchalgorithmus mit Left-First-Auswahlregel vor, mit dessen Hilfe wir die Kantenaufnahmereihenfolge der V-DFS$_\circlearrowleft$ in G^* aus dem Tiefensuchbaum von G rekonstruieren können.

Algorithmus 5.2.7 ($\vec{E}$-DFS$_\circlearrowright$)
Eingabe: planar eingebetteter gerichteter Graph $D = (V, \vec{E})$ und ein Startknoten $v_s \in V$ auf dem Rand der äußeren Facette.
Initialisierung: $M := \{(x, v_s)\}$ mit Dummyknoten $x \notin V$ in der äußeren Facette.

Knotenauswahl

(1) Endknoten v der Kante (u, v), die zuletzt in M eingefügt wurde.

Kantenauswahl(v; (u, v))

(1) Falls noch unbesuchte zu v inzidente (ein- oder auslaufende) Kante existiert
(2) wähle linkeste unbesuchte zu v inzidente Kante $\vec{e}$
(3) falls $\vec{e} = (v, w)$ auslaufende Kante
(4) lege (v, w) in M ab
(5) gib $\vec{e}$ aus
(6) ansonsten
(7) entferne (u, v) aus M

Erläuterung: Die $\vec{E}$-DFS$_\circlearrowright$ ist eine Left-First-Kanten-Tiefensuche auf einem gerichteten, planaren Graphen. Normalerweise betrachten Suchen auf gerichteten Graphen nur die aus dem aktuellen Knoten auslaufenden Kanten. Die $\vec{E}$-DFS$_\circlearrowright$ dagegen arbeitet auf den kompletten Adjazenzlisten der Knoten, in denen alle Kanten unabhängig von ihrer Orientierung gemäß der kombinatorischen Einbettung gespeichert sind. Vom Startknoten aus geht sie die Adjazenzliste von v_s im Uhrzeigersinn durch bis sie eine auslaufende Kante findet. Auf auslaufenden Kanten geht sie vorwärts zum nächsten Knoten, wie eine gewöhnliche Tiefensuche. Bei einlaufend orientierten Kanten bleibt sie auf dem aktuellen Knoten stehen, was man ebenfalls als Tiefensuche interpretieren kann, wenn man festsetzt, daß eine Tiefensuche immer vom Endknoten der neuen Kante aus weiter macht. Die $\vec{E}$-DFS$_\circlearrowright$ besucht jede Kante genau einmal und führt dabei eine feste Anzahl $\mathcal{O}(1)$-Operationen durch. Für die Suche ergibt sich daraus eine Komplexität in $\mathcal{O}(n)$, wie auch bisher bei den anderen Suchen auf planaren Graphen.

Satz 5.2.8
Führt man den $\vec{E}$-DFS$_\circlearrowright$-Algorithmus auf dem V-DFS$_\circlearrowright$-Suchbaum von G mit derselben Startkante und demselben Startknoten durch und läßt sich bei einer einlaufenden Kante $\vec{e} = (v, w)$ die Dualkante $\vec{e}^*$ und bei einer auslaufenden Kante $\vec{e} = (w, v)$ die entgegengerichtete Dualkante $\vec{e}_r^* = (v, w)^*$ ausgeben, so erhält man die Kantenreihenfolge einer V-DFS$_\circlearrowright$ auf G^* mit der äußeren Facette als Startknoten und der Dualkante der Dummykante der Tiefensuche in G als Dummykante in G^*.

Beweis:
Lemma 5.2.6 besagt, daß die Baumkanten in G gerade die Dualkanten der Nichtbaumkanten in G^* sind. Daraus ergibt sich unmittelbar, daß die Richtung der ausgegebenen Kanten korrekt ist.
Es bleibt zu zeigen, daß die $\vec{E}$-DFS$_{\circlearrowright}$ die Kanten in der richtigen Reihenfolge aufnimmt. Wie wir bereits wissen, ist eine Left-First-Breitensuche an einem einzelnen Knoten f dual zu einer Right-First-Tiefensuche auf dem Rand der durch f repräsentierten Facette des Dualgraphen. Außerdem haben wir in Satz 5.2.5 gezeigt, daß der durch die Dualkanten der Nichtbaumkanten von G in G^* induzierte Untergraph T^* ein Right-First-Tiefensuchbaum ist. Aus dem Lemma 5.2.4 folgt für die Einbettung des Tiefensuchbaumes, daß rechts des aktuellen Astes die angrenzenden Facetten besucht sind und links alle Facetten außerhalb der nächsten aufzunehmenden Kante $\vec{e}$. Alle Facetten innerhalb des durch $\vec{e}$ geschlossenen Kreises sind unbesucht.
Betrachten wir zuerst die Situation am Startknoten v_s. Von der Startkante aus führt die $\vec{E}$-DFS$_{\circlearrowright}$ so lange eine Left-First-Tiefensuche aus, bis sie die erste Nichtbaumkante findet. Die V-DFS$_{\circlearrowleft}$ ihrerseits führt an den Schleifen inzident zu f_s eine Right-First-Breitensuche durch, bis sie die erste Baumkante findet. Die Nichtbaumkanten der Antenne, auf der v_s liegt, werden demzufolge in der gleichen Reihenfolge aufgenommen.

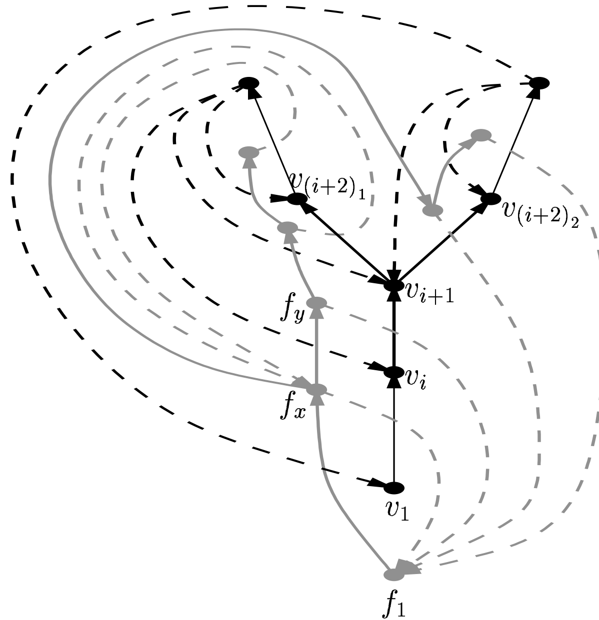


Abbildung 5.5: Ausschnitt aus einem Knoten-Tiefensuchbaum T (schwarz) und dem Dualgraphen T^* (grau)

Sei v_i ein beliebiger innerer Knoten von T. Zum Vergleich betrachte man Abbildung 5.5. Sind die im Uhrzeigersinn ersten Kanten inzident zu v_i einlaufende Nichtbaumkanten, so sucht die $\vec{E}$-DFS$_{\circlearrowright}$ diese im Uhrzeigersinn ab, bis sie auf eine Baumkante $\vec{e}_{i_{b1}} = (v_i, v_{i+1})$ stößt. Entsprechend nimmt die V-DFS$_{\circlearrowleft}$ die Baumkanten auf dem Rand der durch v_i repräsentierten Facette im Uhrzeigersinn auf. Findet sie dabei die Nichtbaumkante $\vec{e}^*_{i_{b1}}$, so

wird diese aufgenommen aber vom aktuellen Knoten aus weitergesucht. Sie wechselt also über auf den Rand der Facette links von $\vec{e}^*_{i_{b1}}$. Dieser Übergang wird von der $\vec{E}$-DFS$_\circlearrowright$ dadurch simuliert, daß sie über die Baumkante $\vec{e}_{i_{b1}}$ zum nächsten Knoten v_{i+1} übergeht, der die Facette links von $\vec{e}^*_{i_{b1}}$ repräsentiert. Die beiden Suche verhalten sich dual.
Überlegen wir nun noch, was an den Verzweigungen im Baum passiert. Sei Knoten v_{i+1} der linkeste oberste Knoten im Baum mit einer Verzweigung, das heißt es existieren mindestens zwei Baumkanten $\vec{e}_{(i+1)_{b1}} = (v_{i+1}, v_{(i+2)_1})$ und $\vec{e}_{(i+1)_{b2}} = (v_{i+1}, v_{(i+2)_2})$. Der Ast, der an $\vec{e}_{(i+1)_{b2}}$ hängt, wird erst dann bearbeitet, wenn die $\vec{E}$-DFS$_\circlearrowright$ den ganzen Ast, der an $\vec{e}_{(i+1)_{b1}}$ hängt, bearbeitet hat und durch Backtrack Schritte zu v_{i+1} zurück kehrt. Von dort aus nimmt sie wie gewohnt die inzidenten Nichtbaumkanten im Uhrzeigersinn ab $\vec{e}_{(i+1)_{b1}}$ auf, bis sie $\vec{e}_{(i+1)_{b2}}$ findet und über diese im Baum nach oben läuft. Nach vollständiger Bearbeitung des Astes an $\vec{e}_{(i+1)_{b1}}$ muß auch die V-DFS$_\circlearrowleft$ Backtrack Schritte durchführen, bis zu der ersten Facette, die noch unbesuchte inzidente Kanten besitzt. Dies ist nach Wahl von v_{i+1} gerade die Facette rechts von $\vec{e}_{(i+1)_{b1}}$. Somit wandert die V-DFS$_\circlearrowleft$ von dort aus weiter im Uhrzeigersinn auf dem Rand der Facette entlang, die durch v_{i+1} repräsentiert wird. Sind beide Suchen am Ende des Astes an $\vec{e}_{(i+1)_{b2}}$ angekommen, so ist eventuell ein anderer Knoten linkester oberster Knoten mit noch nicht abgearbeiteter Adjazenzliste. Das Argument wiederholt sich entsprechend. Somit arbeitet die $\vec{E}$-DFS$_\circlearrowright$ auf dem gerichteten Graphen $G^{V\text{-DFS}}$ dual zur V-DFS$_\circlearrowleft$ auf dem ungerichteten Dualgraphen G^* und die Kantenaufnahmereihenfolge der V-DFS$_\circlearrowleft(G^*)$ ergibt sich aus der im Satz formulierten Version der $\vec{E}$-DFS$_\circlearrowright$. □

Wenn man sich das Vorgehen der $\vec{E}$-DFS$_\circlearrowright$ noch einmal vor Augen führt, so könnte man sie auch als modifizierte Left-First-Breitensuche ansehen. Die Modifikation besteht diesmal darin, daß man der Breitensuche über die Richtung der inzidenten Kanten die Information zugänglich macht, ob die Facette rechts der gewählten Kante durch die duale Suche besucht ist oder nicht. Diese Information ist notwendig für das korrekt duale Verhalten der Left-First-Breitensuche. Es ist unklar, ob diese Information über den Zustand der Facette rechts der Kante auf irgendeine Weise während des Algorithmus bestimmt werden kann. Nach intensiver Beschäftigung mit diesem Problem liegt die Vermutung nahe, daß die Information nur durch ein Preprocessing mit Laufzeit $\mathcal{O}(n)$ geliefert werden kann. Die Komplexität der $\vec{E}$-DFS$_\circlearrowright$ liegt natürlich wie die der anderen Suchen in $\mathcal{O}(n)$. Insgesamt ergibt sich demnach für eine zur V-DFS$_\circlearrowleft$ duale Suche eine Komplexität von $\mathcal{O}(n)$, aber der Koeffizient ist wesentlich größer als bei den in Kapitel 4 vorgestellten Graphsuchen. Die Berechnung des Dualgraphen liegt ebenfalls in $\mathcal{O}(n)$. Ein Preprocessing mit Aufwand $\mathcal{O}(n)$ bringt also keine Vorteile gegenüber der herkömmlichen Methode, bei welcher der Dualgraph aufgestellt und dann auf diesem die Suche gestartet wird.

Es sei zum Schluß noch erwähnt, daß für den Breitensuchbaum keine entsprechende Aussage gemacht werden kann, wie man an dem einfachen Beispiel in Abbildung 5.6 sofort sieht.

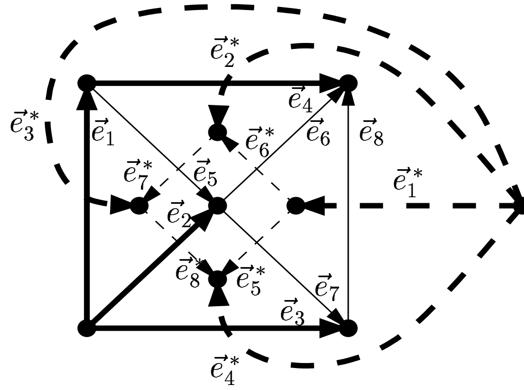


Abbildung 5.6: BFS$_{\circlearrowright}$ auf Graph G und dem Dualgraphen G^* (gestrichelt) durchgeführt. Die dicken Kanten markieren die entstandenen Breitensuchbäume. Die Dualkanten zu dicken Baumkanten sind nicht unbedingt dünne Nichtbaumkanten.

Kapitel 6

Schlußbemerkungen

Nach eingehender Untersuchung des Verhaltens von Breiten- und Tiefensuche auf einem planaren Graphen und seinem Dualgraphen haben wir verschiedene Erkenntnisse gewonnen.
Wir haben gesehen, daß $\mathrm{BFS}_{\circlearrowright}$ und E-$\mathrm{DFS}_{\circlearrowleft}$ auf dem Rand einer Facette und auf Graphen, deren Einbettung ganz bestimmte Eigenschaften erfüllen, dual zueinander sind. Das Hauptmerkmal der Einbettung solcher Graphen war, daß ihr Schachtelungsbaum unverzweigt ist. Dadurch bleibt der Untergraph, der aus den noch unbesuchten Kanten besteht, zusammenhängend und die Tiefensuche kann nicht in einer Komponente stecken bleiben während die Breitensuche zwischen den Komponenten hin und her springt. Die Einbettungen beliebiger Graphen erfüllen diese Voraussetzung im allgemeinen nicht.
Nachfolgend haben wir die Breitensuche so modifiziert, daß sie die Kanten in der gleichen Reihenfolge aufnimmt wie eine E-$\mathrm{DFS}_{\circlearrowleft}$. Das Ergebnis war die Jump-$\mathrm{BFS}_{\circlearrowright}$, welche auf dem Schachtelungsbaum des Graphen eine Tiefensuche durchführt, wobei sie die Kanten der durch die Knoten des Schachtelungsbaumes repräsentierten Ränder gemäß einer $\mathrm{BFS}_{\circlearrowright}$ besucht. Die Information, wann sie in einer bestimmten Komponente verweilen muß, erhält die Jump-$\mathrm{BFS}_{\circlearrowright}$ durch die Betrachtung der bereits besuchten in einen Knoten einlaufenden Kanten.
Auch die Jump-E-$\mathrm{DFS}_{\circlearrowleft}$ benutzt diese in einen Knoten einlaufenden Kanten. Sie geht auf diesen zurück und kontrolliert, ob in der durch sie abgetrennten Komponente noch unbesuchte Kanten liegen. Durch diese einfache Modifikation der Tiefensuche verhält sie sich dual zur $\mathrm{BFS}_{\circlearrowright}$. Die Jump-$E$-$\mathrm{DFS}_{\circlearrowleft}$ führt also auf dem Schachtelungsbaum eine Breitensuche durch, wobei sie die nacheinander abzuarbeitenden Teilstücke der Ränder gemäß einer E-$\mathrm{DFS}_{\circlearrowleft}$ besucht.
Zur Lösung vieler Aufgaben sind die Informationen einer Tiefen- oder Breitensuche auf dem Dualgraphen erforderlich. Auch Weihes Linearzeitalgorithmus [Wei94] zur Bestimmung kantendisjunkter s-t-Pfade in ungerichteten planaren Graphen benötigt die Durchführung einer Breitensuche auf dem Dualgraphen. Mit Hilfe der hier vorgestellten Algorithmen kann nun die explizite Aufstellung des Dualgraphen eingespart werden, da die gewünschte Information direkt aus dem Originalgraphen gewonnen werden kann.

Im letzten Kapitel haben wir unser Augenmerk auf die Knoten-Tiefensuche gelenkt. Die V-DFS$_{\circlearrowleft}$ verhält sich nicht dual zu einer der beiden anderen Suchen, wenn man von der Bearbeitung eines einfachen Kreises absieht. Dafür haben wir gesehen, daß der durch eine V-DFS$_{\circlearrowleft}$ entstandene Tiefensuchbaum eine besondere Dualitätseigenschaft besitzt. Die Dualkanten der Nichtbaumkanten eines V-DFS$_{\circlearrowleft}$-Suchbaumes induzieren einen V-DFS$_{\circlearrowleft}$-Suchbaum im Dualgraphen. Für planare Graphen G ist bekannt, daß die Dualkanten der Nichtbaumkanten eines aufspannenden Baumes im Dualgraphen ebenfalls einen aufspannenden Baum bilden. Ist nun der aufspannende Baum in G ein Tiefensuchbaum, so ist der resultierende Baum in G^* ebenfalls ein Tiefensuchbaum.

Wir haben außerdem gesehen, daß sich die Aufnahmereihenfolge der Kanten durch die V-DFS$_{\circlearrowleft}$ auf G^* aus dem Tiefensuchbaum zusammen mit den Rückwärtskanten in G rekonstruieren läßt und haben einen entsprechenden Algorithmus vorgestellt. Dieser Algorithmus ist im wesentlichen eine Left-First-Breitensuche, welche allerdings den aktuellen Knoten wechselt, sobald sie in der Adjazenzliste des aktuellen Knotens eine auslaufende Kante findet. Diese Suche verhält sich auf dem Graphen G dual zur V-DFS$_{\circlearrowleft}$ auf G^*, ist aber auf die Orientierung der Kanten von G durch eine V-DFS$_{\circlearrowleft}$ angewiesen. Somit benötigt die $\vec{E}$-DFS$_{\circlearrowright}$ also eine Vorarbeit vom Umfang einer Suche und bietet nicht den Vorteil, daß ein aufwendiges Preprocessing wie das Aufstellen des Dualgraphen eingespart werden kann. Im Gegensatz dazu konnten die beiden in Kapitel 4 vorgestellten Suchen alle benötigte Information aus dem Status der nächsten Kante gewinnen. Die von einer zur V-DFS$_{\circlearrowleft}$ dualen Suche benötigte Information müßte Auskunft über den Status der rechten Facette einer Kante geben. Für alle auslaufenden Kanten, über welche die Suche zum nächsten Knoten weiterläuft, gilt, daß die rechte Facette durch die duale Suche besucht ist. Es bleibt weitergehenden Untersuchungen vorbehalten, ob sich diese Information oder eine zu ihr äquivalente auch ohne Preprocessing während der Suche gewinnen läßt.

Literaturverzeichnis

[BW97] Ulrik Brandes and Dorothea Wagner. A linear time algorithm for the arc disjoint menger problem in planar directed graphs. *Lecture Notes in Computer Science*, 1284:64–77, 1997.

[Cou97] Laurent Coupry. A simple linear algorithm for the edge-disjoint (s,t)-paths problem in undirected planar graphs. *Information Processing Letters*, 64(2):83–86, November 1997.

[Eul36] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Comment. Acad. Sci. Imper. Petropol.*, 8:128–140, 1736.

[GS85] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

[Har69] F. Harary. *Graph theory.* Addison Wesley, Reading, Massachusetts, 1969.

[HT74] John E. Hopcroft and Robert E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, October 1974.

[Jun94] Dieter Jungnickel. *Graphen, Netzwerke und Algorithmen.* BI-Wissenschaftsverlag, Mannheim, Leipzig, Wien, Zürich, 1994.

[Kö36] Dénes König. *Theorie der endlichen und unendlichen Graphen.* Akademische Verlagsgesellschaft M.B.H., Leipzig, 1936.

[RLWW93] Heike Ripphausen-Lipa, Dorothea Wagner, and Karsten Weihe. The vertex-dijoint menger problem in planar graphs. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '93)*, pages 112–119, Austin, TX, USA, January 1993. SIAM.

[RLWW95] Heike Ripphausen-Lipa, Dorothea Wagner, and Karsten Weihe. Efficient algorithms for disjoint paths in planar graphs. In L. Lovasz W. Cook and P. Seymour, editors, *DIMACS Volume 20 - from the Special Year of Combinatorial Optimization*, pages 295–354. AMS, 1995.

[Shi69] R. W. Shirey. *Implementation and Analysis of Efficient Graph Planarity Testing Algorithms*. Doctoral thesis, Computer Sciences Department, University of Wisconsin, Madison, 1969.

[Tar72] Robert E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.

[Wei94] Karsten Weihe. Edge-disjoint (s, t)-paths in undirected planar graphs in linear time. *Springer Lecture Notes in Computer Science*, 855:130–140, 1994.