9. Übung Informatik I

Marcus Rickert

30. Dezember 1995

Aufgabe 1

Erläuterungen

Ich bin bei dem Algorithmus davon ausgegangen, daß die Struktur des Graphen in einem Feld *kanten* abgelegt ist, wo jeder Eintrag einer Kante entspricht und folgende Informationen enthält:

- $knotennr1$: Nummer des 1. Knotens der Kante,
- $knotennr2$: Nummer des 2. Knotens der Kante,
- *kosten*: Kosten für diese Kante.

Für die Knoten steht ebenfalls ein Feld *knoten* zur Verfügung mit folgenden Informationen:

- *in_w*: Flag, ob Knoten in W,
- *kantenliste*: Zeiger, auf Liste mit allen Nummern der Kanten, die von diesem Knoten ausgehen.

Die obige Liste der Kantennummern muß am Anfang nicht vorhanden sein, sondern wird durch meinen Algorithmus aufgebaut.

Zu den einzelnen Abschnitten des Algorithmus:

- Bei der Initialisierung werden erst einmal die *in_w*-Flags aller Knoten auf FALSE gesetzt. Dann werden aus der Feld *kanten* alle Kantenlisten des Feldes *knoten* aufgebaut (falls diese nicht als bekannt vorrausgesetzt worden sind). Da die Teilmenge W von V am Anfang noch leer ist muß für die richtige Arbeitsweise des Algorithmus diejenige Kante gefunden werden mit den geringsten Kosten von allen. Dies geschieht der Einfachheit halber durch lineares Durchsuchen des Feldes *kanten*. Die beiden Knoten die an dieser Kante hängen werden in W übernommen, d.h. ihr *in_w*-Flag wird gesetzt. Anschließend werden die Kantenlisten der beiden Knoten durchgegangen. Jede Kante wird dann mit ihren Kosten als Schlüssel in einen geordneten (A,B)-Baum eingefügt, falls sie noch nicht vorhanden ist. Ist sie schon vorhanden, so wird sie aus dem Baum gelöscht. Diese Vorgehensweise hat zur Folge, daß am Ende nur die Kanten im Baum abgelegt sind, von denen ein Knoten in W ist und der andere in $V \setminus W$. Unter diesen Kanten ist dann der jeweilige Kandidat zu suchen, der mit den geringsten Kosten ein Knoten aus W mit einem Knoten aus $V \setminus W$ verbinden kann.

- Es folgt der Schleifenanfang, wo ein Zähler auf 2 gesetzt wird, der die in W vorhandenen Elemente zählt.

- In der eigentlichen Schleife wird die Kante e_0, der Kanten im (A,B)-Baum bestimmt, die die geringsten Kosten hat. Mit Hilfe der Flags *in_w* kann festgestellt werden, welcher der beiden Knoten, die zu e_0 gehören, schon in W war (k_i) und welche außerhalb lag (k_a). Die Kante e_0 wird dann in eine lineare Liste abgespeichert, die die spätere Lösungsmenge enthält.

- Nun wird wie bei der Initialisierung die Kantenliste des Knotens k_a Kante für Kante durchgegangen. Ist eine Kante noch nicht im (A,B)-Baum vorhanden, so wird sie eingefügt. Ist sie schon vorhanden, so liegt sie jetzt innerhalb von W und wird gelöscht. Anschließend wird das *in_w*-Flag von k_a auf TRUE gesetzt.

- Die Schleife wird so oft durchlaufen, bis der Zähler auf $|V|$ steht.

Algorithmus

Sei $n := |V|$ und $m := |E|$.

	Befehle	einzeln	insgesamt
	Setze die *in_w*-Flags von allen Knoten auf FALSE	$O(1)$	$O(n)$
	Initialisiere geordneten (A,B)-Baum		$O(1)$
	Initialisiere Lösungsliste der Kanten		$O(1)$
	for $i := 1$ to m do		
	Baue Knoten i in Kantenliste von $knotennr1(i)$ ein	$O(1)$	$O(m)$
	Baue Knoten i in Kantenliste von $knotennr2(i)$ ein	$O(1)$	$O(m)$
	Suche Kante e_0 mit minimalen Kosten (linear)		$O(m)$
	Durchsuche Kantenlisten von $knotennr1(e_0)$ und $knotennr2(e_0)$		
	Kante schon im (A,B)-Baum ?	$O(\log m)$	$O(m \log m)$
	Dann lösche Kante	$O(\log m)$	$O(m \log m)$
	Sonst füge Kante ein	$O(\log m)$	$O(m \log m)$
	$in\_w[knotennr1(e_0)] := TRUE$		$O(1)$
	$in\_w[knotennr2(e_0)] := TRUE$		$O(1)$
	$z\ddot{a}hler := 2$		$O(1)$
	while $z\ddot{a}hler < n$ do		
	Hole Kante e_0 mit geringsten Kosten aus (A,B)-Baum	$O(\log m)$	$O(n \log m)$
	Füge Kante in Lösungsliste ein		$O(1)$
	Falls $in\_w[knotennr1(e_0)] = FALSE$	$O(1)$	$O(n)$
	Dann $k_a := knotennr1(e_0)$	$O(1)$	$O(n)$
	Sonst $k_a := knotennr2(e_0)$	$O(1)$	$O(n)$
	Durchsuche die Kantenliste von k_a		
*	Kante schon im (A,B)-Baum ?	$O(\log m)$	$O(m \log m)$
*	Dann lösche Kante	$O(\log m)$	$O(m \log m)$
*	Sonst füge Kante ein	$O(\log m)$	$O(m \log m)$
	$in\_w[k_a] := TRUE$	$O(1)$	$O(n)$
	$z\ddot{a}hler := z\ddot{a}hler + 1$	$O(1)$	$O(n)$

Laufzeit

In der zweiten Spalte von rechts steht jeweils die Laufzeit für einen Durchlauf der Schleife, in der rechtesten Spalte die Laufzeit summiert über alle Schleifendurchläufe. Bei denen mit $*$ versehenen Zeilen ergibt sich die Laufzeit zu $O(m \log m)$ und nicht zu $O(mn \log m)$ — wie man es eigentlich aus der zweifachen Verschachtelung ableiten würde — aus folgendem Grund:

Jede Kante wird nur ein einziges mal in den (A,B)-Baum aufgenommen und aus ihm gelöscht, da das Löschen den Übergang zu einer inneren Kante bedeutet. Beim Durchlaufen der äußeren Schleife wird daher jede Kante (bis auf die, bei denen das schon bei der Initialisierung geschehen ist) einmal eingefügt und einmal gelöscht unabhängig davon wie viele Kanten auf die einzelnen Knoten und damit auf die innere Schleife entfallen.

Als Laufzeit erhält man insgesamt: $O(n \log m + m \log m)$. Da aber der Graph aus zusammenhängend vorrausgesetzt worden ist, gilt immer: $m \geq n - 1$. Man erhält als neue obere Grenze: $O(m \log m)$.

Bemerkungen

- Bei den Algorithmen zum Einfügen und Löschen in (A,B)-Bäumen ist bis jetzt davon ausgegangen worden, daß die Schlüssel paarweise verschieden sind. Dies ist aber hier auf den ersten Blick nicht der Fall, denn für verschiedene Kanten dürfen im allgemeinen die gleichen Kosten auftreten. Für den korrekten Ablauf des Algorithmus ist es jedoch essentiell, daß Kanten gleicher Kosten unterschieden werden können. Dies ist z.B. so zu realisieren, indem man nicht die Kosten alleine als Schlüssel benutzt, sondern eine Kombination aus Kosten und der Kantennummer:

  - $schlüssel_{neu} := kosten * MAX\_KANTENNUMMER + kantennummer$

  Da die Kosten den höherwertigen Anteil darstellen, wird immer noch richtig nach den Kosten sortiert, ohne daß die Information über die einzelne Kante wegfällt. Die ursprünglichen Informationen erhält man auf bekannte weise:

  - $kosten := schlüssel_{neu}$ div $MAX\_KANTENNUMMER)$
  - $kantennummer := schlüssel_{neu}$ mod $MAX\_KANTENNUMMER.$

- Sollte die Struktur des Graphen am Anfang nicht in einem Kantenfeld, sondern in Form von Nachfolgerlisten aller Knoten vorliegen, so muß das für den Algorithmus benötigte Feld erst erzeugt werden. Ein einfaches Durchsuchen der Nachfolgerlisten und Eintragen in ein lineares Feld ist jedoch nicht ohne weiteres möglich, da dann jede Kante zweimal in das Feld aufgenommen würde. Es ist daher notwendig, daß überprüft wird, ob die jeweilige Kante der Liste schon im Feld vorhanden ist. Realisiert werden könnte dieses Verfahren durch erstmaliges Abspeichern in einen geordneten (A,B)-Baum, wo als Schlüssel eine eindeutige Kombination aus beiden Knotennummern dient (siehe erste Bemerkung). Doppelte Kanten können dann in $O(\log m)$ ermittelt werden. Insgesamt sind höchstens $2m$ Schritte nötig, so daß die Laufzeit für die Erzeugung des (A,B)-Baumes bei $O(m \log m)$ liegt. Anschließend muß noch der Baum ausgelesen und in das *kanten*-Feld übertragen werden. Dies ist jedoch in $O(m)$ möglich. Die Gesamtlaufzeit des Algorithmus zum Finden des minimal auspannenden Baumes wird demnach nicht durch die Erzeugung des *kanten*-Feldes beeinflußt.

Aufgabe 3

Vorbemerkung (Kummerkasten)

In der vorliegenden Aufgabenstellung ist ein allgemeiner gerichteter, azyklischer Graph vorrausgesetzt worden. Es widerspricht nicht der Definition der Vorlesung für einen solchen Graphen, daß *mehr als ein* Knoten existiert, der keine Vorgänger hat. Es gäbe also Fälle, wo mehr als ein Knoten ausgezeichnet ist (im Sinne der Bezeichnungsweise im Dijekstra-Algorithmus). Wie soll nun aber die Definition des kürzesten Weges auf einen solchen Fall erweitert werden? Man könnte meiner Meinung nach einerseits annehmen, daß

1. man den kürzesten Weg eines jeden Knotens zum *allernächsten* ausgezeichneten Knoten ermitteln soll, oder

2. die kürzesten Wege eines jeden Knotens zu *allen* ausgezeichneten Knoten ermittelt, die diesen Knoten in ihrer Nachfolgermenge haben.

Der zweite Fall scheint mir nicht plausibel zu sein, da man dort im ungünstigsten Fall für n Knoten $n-1$ Zeiger auf den Vorgänger zu $n-1$ ausgezeichneten Knoten haben müßte.

Erläuterungen

Mein Algorithmus basiert darauf, daß der Dijekstra-Algorithmus suksessiv für alle ausgezeichneten Knoten durchgeführt wird. Den ersten ausgezeichneten Knoten S_1 erhält man durch das Unterprogramm aus der 3. Übung, das die Knoten so sortiert (Sortierung wird im folgenden Vorgängersortierung genannt), daß der erste Knoten keine Vorgänger hat (ich weiß nicht , warum in der aktuellen Aufgabenstellung genau die umgekehrte Sortierung beschrieben wird, da dies nur einer Vertauschung der Indices entspricht). Mit diesem Knoten wird dann der Dijekstra-Algorithmus in der Abwandlung vom 8. Übungsblatt aufgerufen. Die Menge U wird deshalb statt der Menge M benutzt, da die Abbruchbedingung $M = V$ nicht mehr unbedingt eintritt. Es kann nämlich vorkommen, daß Knoten existieren, die nicht Nachfolger von S_1 sind. Diese würden durch den alten Dijekstra-Algorithmus nie erreicht werden.

Anschließend wird dieser Schritt für alle Knoten v aus der Vorgängersortierung durchgeführt deren $vor(v) = 0$ ist.

Die Initialisierung für den Dijekstra-Algorithmus unterscheidet sich etwas von der ursprünglichen Version, da nicht automatisch alle Nachfolger des aktuellen ausgezeichneten Knotens diesen als Vorgänger haben, sondern erst überprüft werden muß, ob die alten Wege zu den vorherigen ausgezeichneten Knoten kürzer sind.

Im folgenden Algorithmus bezeichnet $v_j(u)$ den j-ten Nachfolger des Knotens u. Diese Zugriffsweise steht nicht im Widerspruch zur Listenstruktur, in der die Nachfolger abgelegt sind, weil auf sie nur sequentiell zugegriffen wird.