Algorithmendesign

Lösungen zu Übungsblatt 8
WS 13/14

Gruppe 2
Max Bannach

Markus Richter (614027)

Aufgabe 8.1 Union-Find-Darstellungen

Teilaufgabe 1 und 2)

siehe Anhang.

Aufgabe 8.2 Ein Spiel

Teilaufgabe 1

In der ersten Runde sollte man unbedingt $k = 1$ wählen und als Schätzung $x_1 = 2$. Nur so ist garantiert, dass man auf keinen Fall bereits in der ersten Runde verliert, falls z. B. der Gegenspieler eine 1 wählt. Hat der Gegner die 1 gewählt so gewinnt man in der nächsten Runde, bei 2 sofort und in allen anderen Fällen hat man sich für die nächste Runde mindestens $k = 3$ erkauft.

Teilaufgabe 2

Die Strategie besteht darin zunächst ein Intervall zu finden in dem die gesuchte Zahl liegt. Anschließend sucht man in Einzelschritten nach y. Das k benutzt man als Anzahl der Folgenglieder einer fortschreitend exponentiell zur Basis 2 steigenden Folge, d. h. der Suchraum wächst fortwährend exponentiell pro Runde. Man kann auch einfach sagen der Suchraum wächst gemäß itexp.

Man geht folgendermaßen vor:

1. Beginne mit $k = 1$ und $x_1 = 2$
2. Falls $y > x_1$: Setze für nächste Runde $k = 3$ und $x_1 = 4, x_2 = 8, x_3 = 16$
3. Falls nun $y > x_3$: Setze für nächste Runde $k = 17$ und $x_1 = 32, x_2 = 64 \ldots$. usw.
4. Falls aber in einer Runde für ein $x_i \in \{x_1 \ldots x_k\}$ gilt: $x_{i-1} < y$ und $x_i > y$ so ist das Intervall gefunden: $y \in]x_{i-1}; x_i[$. Das ist auch für rundenübergreifende Werte möglich, z. B. wenn in Runde j gilt $x_k < y$ und in Runde $j+1$ gilt $x_1 > y$.
5. Hat man ein Intervall $]x_{i-1}; x_i[$ gefunden, so hat man in der nächsten Runde $x_{i-1} + 1$ Einzelschritte zwischen x_{i-1} und x_i um y zu finden. Das ist auf jeden Fall ausreichend. Man wählt also zum Schluss in der letzten Runde $k = x_{i-1} + 1$ und $x_1 = x_{i-1} + 1, x_2 = x_1 + 1, \ldots, x_{k' < k} = x_i - 1$. Einer dieser Werte wird der gesuchte Wert y sein.

Dadurch, dass der Suchraum gemäß itexp wächst wird das Intervall definitiv in $\mathcal{O}(\text{itlog} y)$ Runden gefunden. Zusätzlich kommt eine Runde, die aber als Konstante in der $\mathcal{O}$-Notation keine Bedeutung hat.

Beispiel Gesucht sei $y = 177$

Runde	k	Schätzungen	Antwort
1	1	2	$y > 2$
2	3	$4, 8, 16$	$y > 4, y > 8, y > 16$
3	17	$32, 64, 128, 256, \ldots$	$y > 32, y > 64, y > 128, y < 256$
4	$(256 - 129 = 127)$	$129, 130, \ldots, 177, \ldots, 256$	$y > 129, y > 130, \ldots, y = 177$ (!)

Aufgabe 8.3 Minimal spannende Bäume

Teilaufgabe 1

Sei K ein nicht minimaler spannender Baum, der durch den Kruskal Algorithmus für eine bestimmte Eingabe erzeugt wurde. Sei weiterhin L ein minimaler spannender Baum für dieselbe Eingabe. Wir betrachten jetzt die erste Kante $e = (x, y) \in E$, bei der sich beide Bäume zum ersten Mal unterscheiden. Es gilt $e \in K$ und $e \notin L$.

L ist minimal aufspannend und enthält nicht e, das bedeutet, dass $L \cup \{e\}$ einen Kreis C bilden würde. Sei also C dieser Kreis. Die Folgerung daraus ist nun, dass es eine Kante f in $C \setminus \{e\}$ geben muss, deren Gewicht höher ist als e, andernfalls hätte Kruskal f bereits betrachtet. L hat nun eine höhere Gewichtung, kann also nicht optimal sein. Sei nun $L' = L \cup \{e\} \setminus \{f\}$, so ist L' wieder optimal, allerdings entspricht L' nun genau K. Daraus folgt, das K optimal sein muss und damit, dass der Kruskal-Algorithmus immer einen minimal spannenden Baum liefert.

■

Teilaufgabe 2

Ein naiver Kreistest würde, meiner Meinung nach – denn ich weiß nicht wie ein allgemein naiver Kreistest aussieht –, erst einmal das e_i hinzufügen und dann jeden Pfad ausgehend in eine Richtung von e_i verfolgen und beim ersten Wiedereintreffen bei e_i einen Kreis melden. Im schlimmsten Fall müsste dieser Test alle Kanten durchgehen. Da Kruskal bereits alle Kanten betrachtet würde der Algorithmus mit der oben beschriebenen Kreissuche $\mathcal{O}(|E| \cdot |E|)$ benötigen.

Teilaufgabe 3

Algorithm 1 KRUSKAL UNION-FIND

Input: Ungerichteter, zusammenhängender Graph $G = (V, E)$ Kantengewichtsfunktion γ
Output: A

1: $A \leftarrow \varnothing$
2: Sortiere die Kanten aufsteigend nach ihrem Gewicht und erhalte die Reihenfolge $e_1, \ldots, e_{|E|}$
3: **for** jeden Knoten $v \in V$ **do**
4: MAKE-SET(v)
5: **end for**
6: **for** $i = 1$ to $|E|$ sei $e = e_i = (u, v)$ **do**
7: **if** FIND-SET(u) $\neq$ FIND-SET(v) **then**
8: $A = A \cup \{e\}$
9: UNION(u, v)
10: **end if**
11: **end for**
12: **return** A

Korrektheit: Der Algorithmus erzeugt für jeden Knoten eine Menge, sodass theoretisch auch jeder Knoten mit jedem anderen vereinigt werden kann. Weiterhin werden die Kanten nach ihrem Gewicht sortiert, was sicherstellt, dass die Kanten mit geringerem Gewicht zuerst betrachtet werden. Darüber hinaus überprüft der Algorithmus für zwei zu vereinende Knoten u, v ob diese zur selben Menge gehören und vereinigt erst wenn das ausgeschlossen werden kann, d. h. u und v haben dann nicht denselben Repräsentanten. Dieses Verfahren schließt Kreise aus. Da nun alle Kanten betrachtet werden ist auch sichergestellt, das immer ein korrektes und sogar optimales Ergebnis zurückgegeben wird.

Laufzeit: MAKE-SET erzeugt für alle $v \in V$ insgesamt einen linearen Aufwand. Das Sortieren kostet $\mathcal{O}(n \log n)$. Die zweite For-Schleife benötigt $|E|$ mal nahezu konstanten Aufwand, da ein FIND-SET $\mathcal{O}(1)$ benötigt. Die Gesamtlaufzeit beträgt daher $\mathcal{O}(|E|)$.