Datenstrukturen und Algorithmen, SS 2019
Prof. Dr. P. Rossmanith
S. Dollase, J. Dreier, H. Lotze

Datum: 4.7.2019

Übungsblatt mit Lösungen 12

Aufgabe T42

Beim Divide-and-Conquer-Algorithmus für das Finden der nächsten Nachbarn ist es ja ziemlich unangenehm, den mittleren Streifen noch einmal extra zu betrachten. Ein Schlaumeier schlägt folgenden Algorithmus vor:

Falls es weniger als, sagen wir, zehn Punkte gibt, dann lösen wir das Problem in konstanter Zeit indem wir alle Paare betrachten. Andernfalls teilen wir die Punkte horizontal in etwa drei gleich große Mengen A, B und C auf, anstatt in nur zwei. Dann rufen wir den Algorithmus zweimal rekursiv auf: Einmal auf $A \cup B$ und einmal auf $B \cup C$. Von beiden Antworten ist die kleinere dann die korrekte Lösung.

Ist der Algorithmus korrekt? Falls nein, kann der Fehler leicht behoben werden?

Lösungsvorschlag

Der Algorithmus ist wohl nicht korrekt: B könnte aus Punkten bestehen, die alle übereinander liegen und recht großen Abstand voneinander haben. Dann könnte ein Punkt von A und ein Punkt von C ganz nah beieinander liegen und der Algorithmus würde das nicht merken.

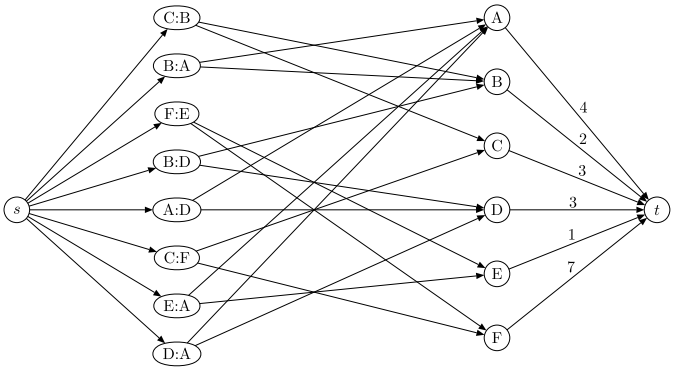
Es ist nicht klar, wie das Problem behoben werden könnte, ohne wieder einen Streifen zu betrachten. Außerdem ist die Laufzeit des Schlaumeieralgorithmus sowieso viel zu schlecht.

Aufgabe T43

Wir sind mitten in einem Fußballturnier. Außer unserer Mannschaft G gibt es noch die Mannschaften A,. . . ,F. Bei einem Gewinn bekommt die siegreiche Mannschaft zwei Punkte, bei einem Unentschieden beide einen Punkt. Wir möchten wissen, ob unsere Mannschaft das Turnier noch gewinnen kann. Unter der Annahme, dass wir alle unserer vier eigenen Spiele gewinnen, geht das nur wenn A höchstens noch vier Punkte gewinnt, B noch höchstens zwei, und so weiter.

Die anderen Mannschaften spielen insgesamt noch acht Spiele. Bei einem Gewinn bekommt die siegreiche Mannschaft zwei Punkte, bei einem Unentschieden beide einen Punkt. Wie kann das abgebildete Flussnetzwerk uns sagen, ob wir noch eine Chance haben? Die unbeschrifteten Kanten haben Kapazität 2.

Team	Punkte
E	16
B	15
C	14
D	14
A	13
F	10
G	10



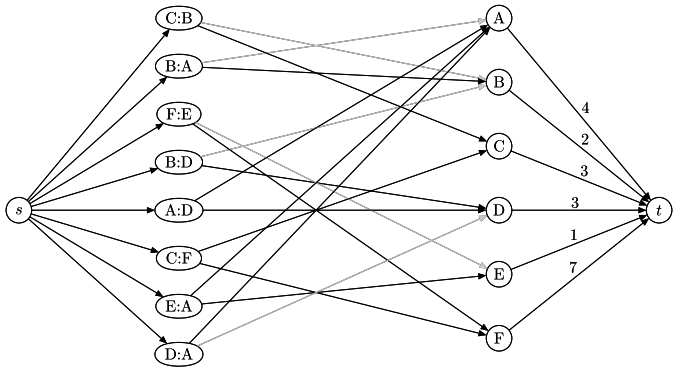
Lösungsvorschlag

Man kann sich vorstellen, dass in jedes Spiel zwei Punkte reinfließen können, wir möchten diese nun auf die Mannschaften verteilen. Die Punkte, die jede Mannschaft erhalten darf, sind aber

beschränkt dadurch, dass die Mannschaften nicht insgesamt mehr Punkte erhalten dürfen als unsere Mannschaft theoretisch maximal erreichen kann (durch Gewinnen jedes Spiels). Da unsere Mannschaft noch vier Spiele spielt und wir davon ausgehen, dass wir jedes Spiel gewinnen, können wir maximal 18 Punkte erreichen. Das bedeutet, dass jede andere Mannschaft höchstens 17 Punkte erreichen darf (man kann argumentieren das es auch Gleichstand geben darf, aber wir wollen einen eindeutigen Gewinner haben).

Falls wir alle aus s ausgehenden Kanten saturieren, können wir tatsächlich alle Punkte verteilen, ohne das eine Mannschaft zu viele Punkte erreicht. Sollte der maximale Fluss nicht alle aus s ausgehenden Kanten saturieren, ist es nicht möglich die Punkte so zu verteilen, dass unsere Mannschaft noch Meister wird.

Die Beispielinstanz hat tatsächlich eine Lösung und theoretisch kann unsere Mannschaft gewinnen. Es gibt einen Fluss mit Wert 16, was den 16 vergebenen Punkten entspricht. Die folgende Zeichnung zeigt, auf welchen Kanten etwas fließt. Daraus kann man ablesen, wie die einzelnen Spiele ausgehen sollen, um keiner Mannschaft mehr als die erlaubte Punktzahl zu gewähren.

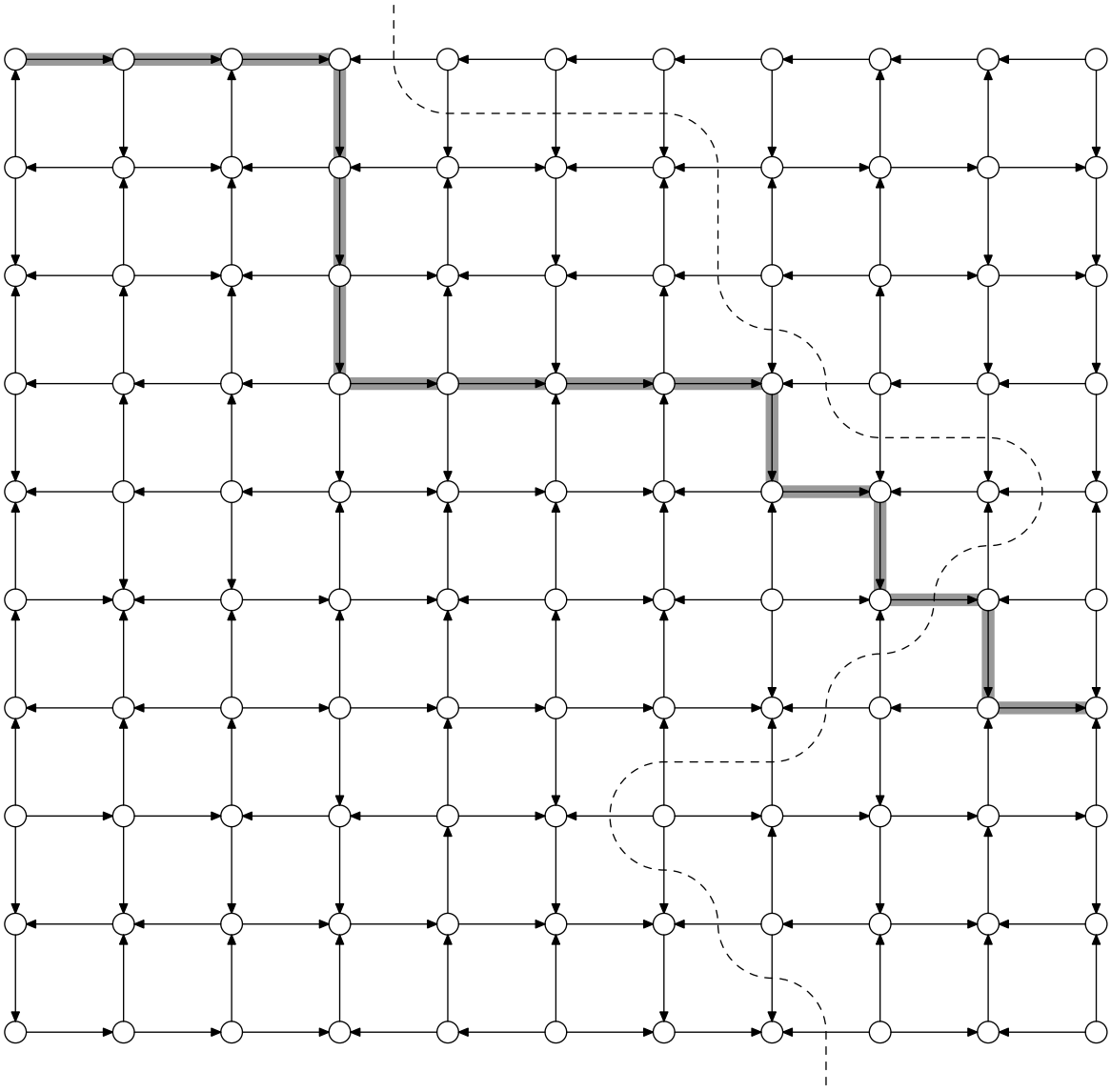


Im Spiel C:B beispielsweise fließen beide Punkte zu C, welche also dieses Spiel gewinnt. Im Spiel C:F dagegen fließen je ein Punkt zu C und F, das Spiel endet also unentschieden. Alle 16 Punkte werden so vergeben und keine Mannschaft erhält zu viele Punkte.

Aufgabe T44

Ermitteln Sie einen maximalen Fluss für das Netzwerk auf der folgenden Seite, wobei die Quellen links und die Senken rechts sind. Beweisen Sie die Maximalität durch Angabe eines gleichgroßen Schnitts. Alle Kapazitäten sind 1.

Lösungsvorschlag



Aufgabe H34 (10 Punkte)
Geben Sie eine Folge von Union- und Find-Operationen an, die zu einem Baum der Höhe vier führt (vier Knoten übereinander). Verwenden Sie dabei sowohl die Rangheuristik als auch Pfadkompression.
Nehmen Sie an, dass bei der Operation $union(A, B)$ der Baum B in A eingehängt wird, falls beide gleichen Rang haben. Geben Sie das Zwischenergebniss nach jeder Operation an.

Lösungsvorschlag
Wir betrachten die Menge $\{1, \ldots 16\}$. Da sich bei Benutzung der Rangheuristik der Rang nur erhöht, wenn wir *union* auf zwei Bäumen mit dem gleichen Rang anwenden, werden wir keine Bäume mergen die unterschiedlichen Rang haben.
Desweiteren wollen wir nicht, dass *find* auf Elemente ausgeführt wird die nicht in der Wurzel stehen, damit die Bäume nicht durch Pfadkompression geglättet werden.
Der erste durchlauf ist also: *union*(1,2), *union*(3,4), *union*(5,6), *union*(7,8), ..., *union*(15,16).
Um Bäume mit jeweils Rang eins zu erzeugen.
Und jetzt führen wir weitere merges benachbarter Bäume durch.
union(1,3), *union*(5,7), *union*(9,11), *union*(13,15).
union(1,5), *union*(9,13).
union(1,9).
Da sich bei jedem *union* der Rang um eins erhöht ist die Gesamthöhe danach vier.

Aufgabe H35 (12 Punkte)

Entwerfen Sie einen Algorithmus, der in einem gegebenen Wort mit Länge n das längste Unterwort findet, das ein Palindrom ist. Die Laufzeit des Algorithmus sollte höchstens $O(n^2)$ sein. Sie dürfen einen deterministischen oder einen randomisierten Algorithmus erfinden.

Lösungsvorschlag

Wir iterieren über alle möglichen $O(n)$ Mittelpunkte eines Palindroms. Diese können entweder zwischen zwei Buchstaben oder auf einem Buchstaben liegen. Für einen gegebenen Mittelpunkt finden wir nun das längste Palindrom mit diesem Mittelpunkt in $O(n)$ Zeit auf folgende Weise: Teste ob die nächsten Buchstaben links und rechts gleich sind und wenn ja füge sie zum Wort hinzu.

Aufgabe H36 (8 Punkte)

	Liste	Sortierte Liste / Array	Sortiertes Array	Binärer Suchbaum	AVL-Baum	Splay-Tree	Treap	Skip-List	Hashtabelle	Min-Heap
Randomisiert										
Einfügen										
Suchen										
Löschen										
Minimum										
Maximum										
Sort. Ausgeben										

Beantworten Sie die Fragen für alle Datenstrukturen bzw. geben Sie eine obere Schranke für den Aufwand an. Gehen Sie davon aus, dass die Anzahl der enthaltenden Elemente n beträgt. Für Laufzeiten tragen Sie eine Funktion $f(n)$ in die Tabelle ein, um eine Laufzeit von $O(f(n))$ auszudrücken. Bei randomisierten Datenstrukturen ist die erwartete obere Schranke gemeint, bei amortisierten Analysen die amortisierte Laufzeit.

Lösungsvorschlag

	Liste	Sortierte Liste / Array	Sortiertes Array	Binärer Suchbaum / AVL-Baum	Splay-Tree	Treap	Skip-List	Hashtabelle	Min-Heap
Random	N	N	N	N	N	N	J	J	J	N
Einfügen	1	n	1	n	n	$\log n$	$\log n$	$\log n$	$\log n$	1	$\log n$
Suchen	n	n	n	$\log n$	n	$\log n$	$\log n$	$\log n$	$\log n$	1	n
Löschen	n	n	n	n	n	$\log n$	$\log n$	$\log n$	$\log n$	1	$\log n$
Minimum	n	1	n	1	n	$\log n$	$\log n$	$\log n$	1	n	1
Maximum	n	1	n	1	n	$\log n$	$\log n$	$\log n$	$\log n$	n	n
Sort. Aus.	$n \log n$	n	$n \log n$	n	n	n	n	n	n	$n \log n$	$n \log n$