Datenstrukturen und Algorithmen, SS 2024
Prof. Dr. P. Rossmanith
D. Cloerkes, M. Gehnen, D. Mock, N. Speetzen

Abgabe: keine :)

Übungsblatt mit Lösungen 13

Aufgabe T44

Gegeben seien n Steine, die auf mehrere Haufen verteilt sein können. Es gibt die Operation *Aufteilen*, die die Steine eines Haufens nimmt, und auf die anderen Haufen verteilt. Dabei darf auf jeden Haufen maximal ein Stein gelegt werden (es können auch neue Haufen aufgemacht werden).

Wir gehen davon aus, dass das Bewegen jedes Steines eine Zeiteinheit benötigt.

Beweisen Sie, dass diese Operation m mal in in $O(m\sqrt{n})$ Zeit ausgeführt werden kann, bzw. dass die amortisierte Zeit einer Operation $O(\sqrt{n})$ ist, wenn die Haufen anfangs Höhe 1 haben.

Definieren und verwenden Sie dazu eine Potentialfunktion Φ.

Lösungsvorschlag

Intuitiv ist es also teuer, wenn wir einen sehr großen Haufen aufteilen, kleinere Haufen sind günstiger. Allerdings kann man nicht zu oft zu große Haufen aufteilen, da die Steine vom aufteilen eines großen Haufens zu breit verteilt werden. Mit etwas Ausprobieren merkt man, dass der (so gesehen) teuerste Fall eine Aufteilung der Steine in eine Pyramide (Dreieckszahlen) ist. Wird dann der größte Stapel gleichmäßig auf die weiteren aufgeteilt, so erhält man die gleiche Pyramide erneut. Durch die Potentialfunktion und amortisierte Analyse wollen wir nun auch die Operationsreihenfolgen, die nicht so gleichmäßig aufgebaut sind, analysieren.

Als Potential der Potentialfunktion Φ zählen wir alle Steine auf den Haufen, die über $\sqrt{n}$ hinausgehen.

Diese Potentialfunktion bietet sich an, weil es maximal $\sqrt{(n)}$ viele Haufen mit mindestens $\sqrt{n}$ Steinen gibt.

Wird daher ein Haufen aufgeteilt, erlaubt uns die Laufzeit von $O(\sqrt{n})$ bis zu $\sqrt{n}$ Steine von einem Haufen zu entfernen, und gleichzeitig auch bis zu $\sqrt{n}$ Steine auf einen Haufen mit mehr als $\sqrt{n}$ vielen Steinen zu legen. Ist der Haufen größer, der aufgeteilt wird, so wird die Potentialfunktion dafür entsprechend reduziert werden.

Aufgabe T45

Nach einigen Jahren harter Arbeit als Fotograf haben Sie es geschafft: Ihre Arbeit ist gefragt, allein für nächste Woche Samstag haben dutzende Leute nach einem Shooting gefragt.

Jeder potentielle Kunde hat allerdings auch genaue Vorstellungen: Das Shooting i müsste zu einem Zeitpunkt s_i beginnen und die Dauer d_i sein. Dafür wäre der Kunde bereit, p_i Euro für ihre Arbeit zu bezahlen.

Leider können Sie nicht alle Aufträge annehmen: Einige Aufträge überschneiden sich, und Sie können nur einen Auftrag zur gleichen Zeit bearbeiten. Wie viel Geld können Sie verdienen? Welche Aufträge i müssten Sie dafür annehmen?

Konstruieren Sie einen Graphen, sodass ein (aus der Vorlesung bekannter) Graphalgorithmus ihr Problem effizient löst!

Lösungsvorschlag

Das Problem kann als Shortest-Path Problem auf einem gerichteten Graphen modelliert werden, welches mit Bellman-Ford effizient gelöst werden kann:

Wir definieren für jeden Startzeitpunkt eines Jobs s_i einen Knoten, sowie einen Knoten k am Ende des Tages.

Zwischen zwei zeitlich aufeinanderfolgenden Knoten wird je eine Kante (vom früheren zum späteren Knoten) eingeführt, die das Gewicht 0 bekommt.

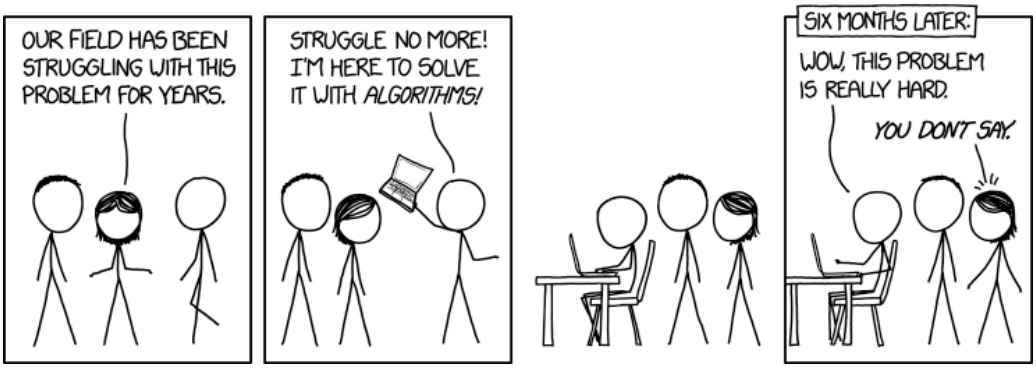
Weiterhin wird für jeden möglichen Auftrag i eine Kante eingefügt - vom Startzeitpunkt s_i bis zu dem Knoten, der zeitlich als nächstes nach $s_i + d_i$ folgt. Das Gewicht der Kante ist $-p_i$.

Nun kann der kürzeste Pfad zwischen dem ersten Knoten und k gesucht werden:

Der Bellman-Ford Algorithmus darf verwendet werden, da der gerichtete Graph keine Kante enthält, die von einem späteren zu einem früheren Zeitpunkt führt. Daher gibt es keine Zyklen, und insbesondere auch keine mit negativem Gewicht.

Die Kanten für die Aufträge auf dem kürzesten Pfad entsprechen nun den Aufrägen, die ausgewählt werden sollten, um möglichst viel Gewinn zu machen. Die Kanten mit Gewicht von 0 zwischen zwei aufeinanderfolgenden Knoten entsprechen Phasen mit Leerlauf, wo kein Auftrag durchgeführt wird.

Zwei Aufträge mit Überscheinung können aufgrund der Konstruktion nicht beide ausgeführt werden; die Optimalität der Auswahl folgt aufgrund der Minimalität des Pfades durch den Algorithmus.



xkcd 1831: Here to Help. *"We TOLD you it was hard." "Yeah, but now that I'VE tried, we KNOW it's hard."*

Hinten geht es weiter.

Aufgabe T46

a) Wann ist ein Sortierverfahren „in-place“?

b) Wann ist ein Sortierverfahren stabil?

c) Beantworten Sie die Fragen für alle Sortierverfahren in folgender Tabelle. Gehen Sie davon aus, dass ein Vergleich in konstanter Zeit durchgeführt wird und die Anzahl der zu sortierenden Elemente n beträgt. Für Laufzeiten tragen Sie eine Funktion $f(n)$ in die Tabelle ein, um eine Laufzeit von $O(f(n))$ auszudrücken. Schätzen Sie dabei die Laufzeit möglichst präzise ab. Für die Sortierverfahren, welche nicht vergleichsbasiert sind, drücken Sie die Laufzeiten durch Funktionen $f(n, w)$ aus, wobei w die Wortlänge der zu sortierenden Elemente in Bits ist. Durchschnittliche Laufzeiten beziehen sich auf n verschiedene Elemente, die zufällig permutiert sind.

	Quicksort	Heapsort	Mergesort	Insertion-Sort	Straight-Radix	Radix-Exchange
In-place						
Stabil						
Laufzeit (Worst-case)						
Laufzeit (Durchschnitt)						
Vergleichsbasiert						

Lösungsvorschlag

a) Ein Sortierverfahren ist „in-place“, wenn es nur konstant viel zusätzlichen Speicher zur Ausführung benötigt. Hierbei wird das unsortierte Eingabearray mit dem sortierten Ausgabearray überschrieben. Auch logarithmisch viel zusätzlicher Speicher wäre richtig. Kein zusätzlicher Speicher dagegen ist falsch.

b) Ein Sortierverfahren ist stabil, wenn die Reihenfolge gleicher Schlüssel im Ausgabearray der Reihenfolge dieser Schlüssel im Eingabearray entspricht.
Typische Fehler: Es wird verlangt, dass die Reihenfolge zwischen zwei Elementen zu jedem Zeitpunkt gleich bleibt, dies muss jedoch nur nach der Ausführung des Algorithmus gelten. Weiterhin falsch ist es zu verlangen, dass der absolute Abstand zwischen zwei Elementen identisch bleibt.

c) Für Quicksort und Radix-Exchange gibt es jeweils Implementierungen die In-place als auch nicht In-place sind. Daher sind hier beide Antworten richtig.

Bei den Radix-Sortierverfahren bezeichne w die Wortlänge. Quicksort und Radix-Exchange-Sort sind wegen des benötigten Stacks nicht in-place. Beide lassen sich jedoch so implementieren, dass der Stack in rekursiven Funktionsaufrufen „versteckt“ wird, während jeder einzelne Aufruf nur konstant viel Platz benötigt. Bei Sortierverfahren sagen manche, dass sie nicht in-place sind, wenn sie $\Omega(n)$ viel Platz zusätzlich brauchen, andere, schon bei mehr als konstant vielem Zusatzplatz.

	Quicksort	Heapsort	Mergesort	Insertion-Sort	Straight-Radix	Radix-Exchange
In-place	??	J	N	J	N	??
Stabil	N	N	J	J	J	N
Laufzeit (Worst-case)	n^2	$n \log n$	$n \log n$	n^2	nw	nw
Laufzeit (Durchschnitt)	$n \log n$	$n \log n$	$n \log n$	n^2	nw	nw
Vergleichsbasiert	J	J	J	J	N	N

* * *

Wir wünschen euch eine schöne vorlesungsfreie Zeit und viel Erfolg beim Lernen für die Klausuren!

* * *