technische universität dortmund

SoSe 2018

M. Buchin
A. Krivošija, D. Mezlaf
L. Pfahler, A. Rey

DAP2 – Heimübung 6

Ausgabedatum: 11.05.2018 — Abgabedatum: Di. 22.05.2018 bis 12 Uhr

Abgabe:

Schreiben Sie unbedingt immer Ihren vollständigen Namen, Ihre Matrikelnummer und Ihre Gruppennummer auf Ihre Abgaben! Beweise sind nur dort notwendig, wo explizit danach gefragt wird. Eine Begründung der Antwort wird allerdings *immer* verlangt.

Aufgabe 6.1 (5 Punkte): (Gierige Algorithmen: ein Scheduling-Problem)

Auf einer Maschine sind m Jobs mit Laufzeiten $p_1, \ldots, p_m$ auszuführen. Wir bezeichnen als *Schedule* für m Jobs eine Reihenfolge, in der diese Jobs auf der Maschine ausgeführt werden. Der Zeitpunkt, zu dem in einem Schedule der Job i beendet ist, sei mit c_i bezeichnet. Gesucht ist ein Schedule, der die summierten Beendigungszeitpunkte von allen Jobs, d. h. $\sum_{i=1}^{m} c_i$ minimiert.

a) Formulieren Sie – sowohl in eigenen Worten als auch in Pseudocode – einen **gierigen** Algorithmus, der bei Eingabe eines Arrays $P[1..m]$, in dem die Laufzeiten p_i, $1 \leq i \leq m$, gespeichert sind, die minimale Summe der Beendigungszeitpunkte für alle Jobs in einem Schedule bestimmt. Für die volle Punktzahl wird ein Algorithmus erwartet, dessen Worst-Case-Laufzeit durch $\mathcal{O}(m \log m)$ beschränkt ist.

b) Analysieren Sie die Laufzeit Ihres Algorithmus.

c) Beweisen Sie, dass dieser Algorithmus den optimalen Wert liefert.

Lösung:

a) Eine optimale Strategie ist *SPT* (shortest processing time first): Die Idee des Algorithmus besteht darin, zuerst die kürzesten Jobs auszuführen und danach in aufsteigender Reihenfolge der Ausführungszeiten. Dazu müssen die Jobs zuerst entsprechend sortiert werden. Die Optimalität wird in Aufgabenteil c) bewiesen. Die gesuchte Summe (gespeichert in Variable c) setzt sich zusammen aus den einzelnen Beendigungszeitpunkten, also der aktuellen Laufzeit aufaddiert auf den zuletzt verwendeten Zeitpunkt (z).

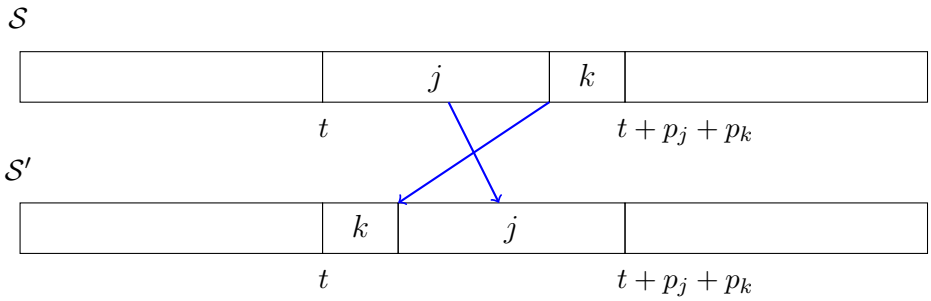
SPT(Array P):
1. Sortiere das Array P aufsteigend bzgl. der Ausführungszeiten $P[i]$, $1 \leq i \leq$ length
2. $c \leftarrow 0$
3. $z \leftarrow 0$
4. **for** $i \leftarrow 1$ **to** m **do**
5. $\quad z \leftarrow z + P[i]$
6. $\quad c \leftarrow c + z$
7. **return** c

b) Der Algorithmus SPT arbeitet in $O(m \log m)$, denn: In Zeile 1 wird das Array der Länge m etwa mit Merge-Sort aufsteigend sortiert, was eine Laufzeit in $\mathcal{O}(m \log m)$ benötigt. Zeilen 2,3 und 7 benötigen eine konstante Anzahl an Rechenschritten. Zeile 4 wird $m+1$ mal aufgerufen, Zeilen 5 und 6 benötigen je einen Rechenschritt, werden also insgesamt $2m$ mal aufgerufen.

c) *Behauptung*: Der von Algorithmus SPT berechnete Wert ist optimal bzgl. des Minimierungsziels $\sum_{i=1}^{n} c_i$.

Beweis: Wir führen einen Widerspruchsbeweis. Wir nehmen dazu an, es gäbe einen optimalen Schedule $\mathcal{S}$, der *nicht* nach der SPT-Strategie entstanden ist. Dann muss es mindestens zwei *aufeinanderfolgende* Jobs, j gefolgt von k, geben, sodass $p_j > p_k$ gilt. Dies wird durch folgendes Widerspruchsargument deutlich. Der kleinste Abstand zwischen zwei Jobs, die nicht der SPT-Regel genügen, sei $\ell > 1$. Diese Jobs bezeichnen wir mit j und k und es gilt $p_j > p_k$. Dann hat der rechte Nachbar von j entweder eine kleinere Ausführungszeit als p_j, oder seine Ausführungszeit ist größer oder gleich p_j und somit ist auch seine Ausführungszeit größer als p_k. Im beiden Fällen haben wir dann einen kleineren Abstand zwischen zwei Jobs, die nicht der SPT-Regel genügen.

Tauschen wir diese beiden Jobs aus, so entsteht ein Schedule $\mathcal{S}'$, in dem alle übrigen Jobs ihre Position beibehalten. Wichtig ist nun zu erkennen, dass sich die Beendigungszeiten der Jobs, die vor und nach j und k ausgeführt werden, unter dem Austausch nicht ändert. Die Differenz der Zielfunktionswerte hängt also einzig und allein von den Jobs j und k ab.



Wir vergleichen ihre Beiträge bzgl. der Schedules $\mathcal{S}$ und $\mathcal{S}'$. Dazu sei t der Startzeitpunkt des ersten der beiden Jobs. Unter dem Schedule $\mathcal{S}$ erhalten wir einen Beitrag zur Zielfunktion von

$$(t + p_j) + (t + p_j + p_k),$$

während wir unter dem Schedule $\mathcal{S}'$ einen Beitrag von

$$(t + p_k) + (t + p_j + p_k)$$

erhalten.

Da wir $p_k < p_j$ angenommen hatten, hat $\mathcal{S}'$ einen echt besseren Funktionswert als $\mathcal{S}$. Dies ist ein Widerspruch zur Optimalität von $\mathcal{S}$. □

Aufgabe 6.2 (5 Punkte): (Gierige Algorithmen: ein Subset-Sum-Problem)

Eine endliche, natürliche Zahlenfolge $a_1, a_2 \ldots, a_n$ heißt *superwachsend*, falls jede Zahl größer als die Summe der vorherigen Zahlen ist, also

$$a_i > \sum_{j=1}^{i-1} a_j$$

für alle i, $2 \leq i \leq n$, gilt. Es sei $a_1 > 0$.

Wir betrachten nun das folgende Problem: Gegeben sei eine solche superwachsende endliche, natürliche Zahlenfolge und eine natürliche Zahl q, $a_1 \leq q \leq a_n$, für die es eine Teilmenge der Indizes $I \subseteq \{1, 2, \ldots, n\}$ gibt, sodass $\sum_{i \in I} a_i = q$ gilt. Gesucht ist die minimale Größe einer solchen Indexmenge.

a) Beschreiben Sie – mit eigenen Worten und als Pseudocode – einen **gierigen** Algorithmus, der bei Eingabe eines Arrays $A[1..n]$, in dem eine solche Zahlenfolge gespeichert ist, und einer natürlichen Zahl q, $A[1] \leq q \leq A[n]$, die Größe einer minimalen Indexmenge zur Lösung des oben beschriebenen Problems ausgibt. Für die volle Punktzahl wird ein Algorithmus erwartet, dessen Worst-Case-Laufzeit durch $\mathcal{O}(n)$ beschränkt ist.

b) Analysieren Sie die Laufzeit Ihres Algorithmus.

c) Beweisen Sie, dass Ihr Algorithmus die optimale Lösung zurückgibt.

Lösung:

a) Eine gierige Herangehensweise ist, den jeweils größtmöglichen Wert a_i, der kleiner ist als der aktuelle Zielwert, hinzuzunehmen, d. h. i in die Indexmenge aufzunehmen, und als neuen Zeilwert die Differenz zwischen dem aktuellen Zielwert und der aktuellen Zahl a_i zu betrachten. Aufgrund der Eigenschaft, dass die Zahlenfolge superwachsend ist, liefert dies eine eindeutige und damit optimale Lösung bezüglich der Anzahl der gewählten Indizes. Die Korrektheit wird in Aufgabenteil c) bewiesen.

In Pseudocode erhalten wir folgenden Algorithmus.

SubsetSum(Array $A[1..n]$**, int** q**):**

```
1. I ← ∅
2. m ← 0
3. for i ← n to 1 do
4.     if A[i] ≤ q then
5.         I ← I ∪ {i}
6.         q ← q − A[i]
7.         m ← m + 1
8. if q = 0 then
9.     return m
```

b) Die Laufzeit dieses Algorithmus ist durch die For-Schleife in Zeilen 3 bis 7 bestimmt, da diese Schleife n Wiederholungen enthält, und, da im Worst Case die Bedingung in Zeile 4 immer wahr ist, jede Wiederholung konstante Zeit benötigt; also insgesamt $\mathcal{O}(n)$. Die Anweisungen in Zeilen 1, 2 und 8 benötigen konstante Zeit. Am Ende wird entweder Zeile 9 oder Zeile 11 mit je einem Rechenschritt ausgeführt. Insgesamt ergibt sich eine Laufzeit in $\mathcal{O}(n)$.

c) Sei I die Indexmenge, die der Algorithmus berechnet. Wir zeigen (I), dass, wenn I eine gültige Lösung ist, der Wert $m = |I|$ in Zeile 9 ausgegeben wird. Außerdem zeigen wir (II), dass, wenn es eine Lösung gibt, m optimal ist.

Beweis von (I): Sei q' die initiale Belegung von q. Vor der j. Schleifeniteration, $1 \leq j \leq n+1$ (das entspricht Laufvariable $i = n-j+1$, $1 \leq j \leq n$, und dem Schleifenaustritt für $j = n+1$) gelten $q' = q + \sum_{k \in I} A[k]$ $q < A[i+1] = A[n-j+2]$ (für $i < n$, $q \leq A[n]$ sonst), es kann keinen Index $k > i = n-j+1$, $k \notin I$, geben, der Teil einer Lösung für q' ist, und $m = |I|$. Dies können wir induktiv zeigen: Induktionsanfang: Für $j = 1$ gilt initial, dass $q' = q + 0$. Laut Aufgabenstellung ist $q \leq A[n]$. Es gibt keinen größeren gütligen Index als $i = n$. Es gilt $m = 0 = |I|$. Induktionsschritt: Vorausgesetzt, die Invariante gelte vor einem Iteration j, $1 \leq q \leq n$, dann gilt sie auch nach Iteration j, also vor Iteration $j+1$ (oder beim Schleifenaustritt, falls $j = n$):

Falls $A[i] > q$ ist, wird zu I kein Index hinzugefügt und q und m bleiben unverändert. Es gilt also $q < A[i] = A[n-(j+1)+2]$. Die Aussagen für q und m bleiben erhalten. Es gibt weiterhin keinen Index mit $k > i = n-j+1$, $k \notin I$, der Teil einer Lösung für q' ist, Für $k = i$ gilt $k \notin I$ und i kann nicht Teil einer anderen Lösung sein: Angenommen, es gibt eine solche Lösung J, die i enthält. Nach Induktionsvoraussetzung ist $\{k \in J \mid k > i\} \subseteq I$. Falls $\{k \in J \mid k > i\} = I$, ist $\sum_{k \in J} a_k = \left(\sum_{k \in Jk<i} a_k\right) + a_i + \left(\sum_{k \in Jk>i} a_k\right) = \left(\sum_{k \in Jk<i} a_k\right) + a_i + \left(\sum_{k \in Ik>i} a_k\right) = \left(\sum_{k \in Jk<i} a_k\right) + a_i + q' - q > q'$. Also ist J keine Lösung. Falls $\{k \in J \mid k > i\} \subset I$, gibt es ein $\ell \in I$, $\ell > i$, sodass $\sum_{k \in J} a_k = \left(\sum_{k \in Jk<i} a_k\right) + a_i + \left(\sum_{k \in Jk>i} a_k\right) \leq \left(\sum_{k \in Jk<i} a_k\right) + a_i + \left(\sum_{k \in Ik>i} a_k\right) - a_\ell < a_\ell + q' - q - a_\ell \leq q'$.

Falls $A[i] \leq q$, werden Zeilen 5 bis 7 ausgeführt und es gelten anschließend $q + \sum_{k \in I} A[k] = q_{\text{vorher}} - A[i] + (A[i] + \sum_{k \in I \setminus \{i\}}) = q'$; die Aussage, dass alle Indizes $k > i$, $k \notin I$, nicht Teil einer Lösung sein können, bleibt erhalten. Den Index $i \in I$ betrifft das nicht. Es folgt, dass der neue Wert $q < A[i]$ ist, ansonsten kann es keine Lösung bestehend aus kleineren Indizes geben. Der Wert $m = |I|$ wird korrekt aktualisiert.

Beim Schleifenaustritt gelten also nun $q' = q + \sum_{k \in I} A[k]$ mit $q < A[n-(n+1)+2] = A[1]$ und kein Index $k > 0$, $k \notin I$, kann Teil der Lösung sein und $m = |I|$. Da es eine Lösung gibt, muss daher $q = 0$ sein. Also ist wegen $q' = q + \sum_{k \in I} A[k]$ eine korrekte Indexmenge I gefunden und es wird in Zeile 9 korrekt $m = |I|$ zurückgegeben.

Beweis von (II): Angenommen, es gibt eine andere Lösung $J \neq I$ mit $\sum_{k \in J} = q$ und $|J| < |I|$. Dann gibt es einen Index in I, der nicht in J vorkommt. Sei i der größte solche Index. Dann gilt

$$a_i + \left(\sum_{k \in I, k>i} a_k\right) + \left(\sum_{k \in I, k<i} a_k\right) = q = \left(\sum_{k \in J, k>i} a_k\right) + \left(\sum_{k \in J, k<i} a_k\right).$$

Da die Summen über die Indizes größer als i jeweils gleich sind, bleibt $a_i + \left(\sum_{k \in I, k<i} a_k\right) = \left(\sum_{k \in J, k<i} a_k\right)$. Allerdings ist $\sum_{k \in J, k<i} a_k \leq \sum_{k=1}^{i-1} a_k < a_i$, da die Folge superwachsend ist, ein Widerspruch zur Gleichheit. Also gibt es keine bessere Lösung als I. □