Prof. Dr. Christian Sohler
Nicole Funk, Dr. Matthijs Ebbens, Sebastian Zaun

Sommersemester 2023

8. Übungsblatt

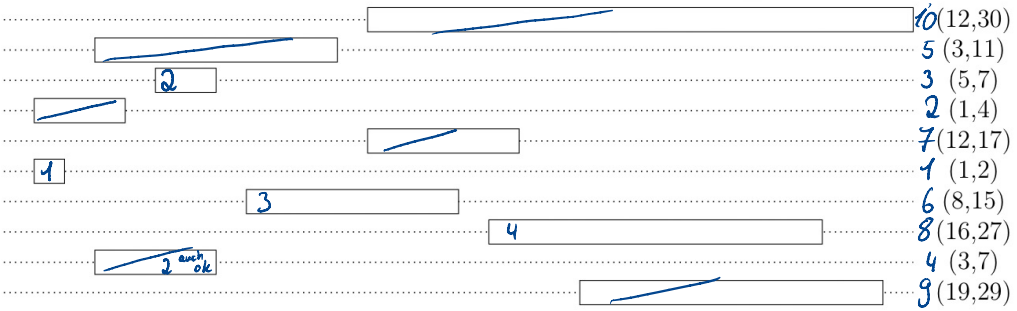
zur Vorlesung

Grundzüge der Informatik I

Abgabe über Ilias bis zum 7.6. 14:00 Uhr.
Besprechung in Kalenderwoche 24.

Aufgabe 1 *Scheduling (4 Punkte)*

Betrachten Sie die folgenden dargestellten Aufgaben, die am rechten Rand auch als Tupel (Anfangszeitpunkt, Endzeitpunkt) angegeben sind. Geben Sie an, in welcher Reihenfolge die Aufgaben betrachtet werden müssen, damit der Algorithmus *IntervalScheduling* aus der Vorlesung anwendbar ist. Führen Sie dann den Algorithmus aus, wobei die Daten in der von Ihnen angegebenen Reihenfolge bearbeitet werden. Geben Sie die Ausgabe des Algorithmus und alle Aufgaben, die nicht erfüllt werden, an.



- nach Endzeitpunkten sortieren
- nächstmögliches nehmen

Ausgabe: {1, 3, 6, 8}

Aufgabe 2 *Gierige Algorithmen (4 + 2 Punkte)*

Wir betrachten ein Problem aus dem Bereich *Scheduling*:

Auf einer einzelnen Maschine sind n Jobs mit Bearbeitungszeiten $p_1, \ldots, p_n \in \mathbb{R}^+$ auszuführen. Die Maschine kann zu jedem Zeitpunkt höchstens einen der Jobs bearbeiten und es ist nicht erlaubt, die Bearbeitung eines Jobs zugunsten eines anderen Jobs zu unterbrechen und zu einem späteren Zeitpunkt wiederaufzunehmen. Ein *Schedule* beschreibt nun einen Plan, in welchen Zeiträumen die Jobs auf der Maschine bearbeitet werden. Der Zeitpunkt, zu dem in einem Schedule der Job i beendet ist, wird mit c_i bezeichnet. Wir suchen einen Schedule, der die summierten Beendigungszeitpunkte von allen Jobs, d.h. $\sum_{i=1}^{n} c_i$ minimiert.

Beispiel: Gegeben seien drei Jobs mit den Bearbeitungszeiten $p_1 = 1, p_2 = 2$ und $p_3 = 3$. Wenn die Jobs in der Reihenfolge $2, 3, 1$ ohne Leerlauf bearbeitet werden, ergibt sich $c_2 + c_3 + c_1 = 2 + 5 + 6 = 13$ für die summierten Beendigungszeitpunkte.

a) Entwickeln Sie einen gierigen Algorithmus, der ein Schedule findet, das die summierten Beendigungszeitpunkte von allen Jobs minimiert, und erklären oder kommentieren Sie diesen.

b) Analysieren Sie die asymptotische Worst-Case-Laufzeit Ihres Algorithmus.

A2

a) geg: Bearb.zeiten $p = [p_1, \dots, p_n]$

ges: Schedule / Reihenfolge der n Jobs s.d.

$\sum_{i=1}^{n} c_i$ minimal ist $\quad c_i$ = Beendigungszeit von J_i

$=$

$n \cdot p_1 + (n-1) \cdot p_2 \dots p_n$

Beh: Summe minimal wenn Jobs nach Bearbeitungszeit aufsteigend sortiert sind.

b)

Algo(p)

1. sortiere p — nicht aufsteigend — $O(n \log n)$ via MergeSort
2. return p — $O(1)$ / $O(n)$

$O(n \log n)$

Beweis: nicht gefordert: Ann: Die optimale Reihenfolge S wird nicht von dem Algo aus a) berechnet.

$\Rightarrow$ $\exists$ (Gegenbsp) Jobs j, ℓ mit $p_j > p_\ell$ und j kommt vor ℓ in S vor

Dann ist Job ℓ nach der Zeit $c_\ell = p_j + p_\ell + t$ fertig

Wobei t der Zeitpkt ist wenn j bearbeitet wird

Sei S' die Reihenfolge die man durch S bekommt wenn man j und ℓ tauscht.

Unterschied von S und S' ist c_j und c_ℓ

1. in S: $c_j + c_\ell = (t + p_j) + (t + p_j + p_\ell) = 2(t + p_j) + p_\ell = A$

2. in S': $c_\ell + c_j = (\dots \quad \dots) = 2(t + p_\ell) + p_j = B$

$A - B = p_j - p_\ell > 0$ nach Ann. $\Rightarrow$ Zielfkt. von S ist größer als von S' ↯

Aufgabe 3 *Gierige Algorithmen (4 + 2 + 2 + 2 Punkte)*

Bob hat einen kleinen Unverpacktladen, in dem er verschiedene Lebensmittel wie Gemüse oder Getreide unverpackt verkauft. Er ist beim Großmarkt, um Waren für seinen Laden einzukaufen. Beim Großmarkt hat Bob die Möglichkeit jede der n möglichen Waren in 100g Einheiten zu kaufen. Die unterschiedlichen Waren sind mit verschiedenen Kennzahlen in $\{1, \ldots, n\}$ bezeichnet. Leider hat der Großmarkt aber nur eine begrenzte Menge der einzelnen Waren, da es ein paar Lieferschwierigkeiten gab.

Nun muss Bob entscheiden, welche Waren er für seinen Laden kauft. Allerdings hat Bob nur einen sehr alten Lastwagen und kann so nur ein Maximalgewicht G an Waren transportieren. Das Gewicht der Verpackungen kann dabei vernachlässigt werden, da Bob selber genug Behälter dabei hat, die bereits im Maximalgewicht G berücksichtigt werden. Zuerst schreibt Bob sich die vorhandenen Mengen in einem Feld $g[1 \ldots n]$ mit $g[i] \in \mathbb{N}$ auf.

Bob muss sich keine Gedanken über sein Budget für den Einkauf oder darüber machen, welche Produkte seine Kunden kaufen, da das Geschäft gut läuft und alle möglichen Waren sehr begehrt bei seinen Kunden sind. Allerdings möchte er natürlich möglichst viel Gewinn mit den ausgewählten Waren machen. Deswegen notiert sich Bob auch den Gewinn, den er für 100g der einzelnen Waren erbringen kann, in einem Feld $p[1 \ldots n]$ mit $p[i] \in \mathbb{N}$.

Bob's Notizen könnten dabei z.B. wie folgt aussehen:

Ware	**ID** i	**verfügbare Menge** $g[i]$	**Gewinn pro 100g** $p[i]$
Vollkornspaghetti	1	1200g	$0,4$ €
Kartoffeln	2	1500g	$0,2$ €
Basmatireis	3	300g	$0,8$ €
⋮	⋮	⋮	⋮

Also möchte Bob nun die Mengen an Waren auswählen, die ihm möglichst viel Gewinn bringen, aber noch von seinem Lastwagen transportiert werden können.

a) Entwickeln Sie einen gierigen Algorithmus mit einer Laufzeit in $O(n \log n)$, der Bob die Entscheidung abnimmt, und erklären oder kommentieren Sie diesen. Dabei können sie davon ausgehen, dass die Gewichtskapazität G des Lastwagens ein Vielfaches von 100g ist (also $G\%100 = 0$) und die Einträge in g in Gramm sind.

b) Analysieren Sie die asymptotische Worst-Case-Laufzeit Ihres Algorithmus.

c) Zeigen Sie die Korrektheit Ihres Algorithmus mit einem Widerspruchsbeweis.

(Hinweis: Stellen Sie sich folgende Fragen:

1. welche strukturelle Eigenschaft hat die von Ihrem Algorithmus berechnete Lösung?

2. Was würde folgen, wenn es eine optimale Lösung gibt, die diese strukturelle Eigenschaft nicht besitzt?)

d) Diskutieren Sie, ob es ebenfalls möglich ist eine optimale Lösung mit Hilfe eines gierigen Algorithmus zu finden, wenn Sie nur die gesamten Mengen eines Produktes abnehmen können. Geben Sie entweder einen kommentierten Algorithmus an oder ein Gegenbeispiel, für das Ihr gieriger Algorithmus keine optimale Lösung findet.

a)

b)

```
                GreedyBob(G, g, p, n)
O(n)    1       ID = new array [1..n]   // Index array
O(n)    2       for i=1 to n do
O(n)    3           ID[i] = i           // füllen von ID mit den Indizes
O(n log n)  4   sortiere ID absteigend nach p[ID[i]]
O(n)    5       L = new array [1..n]    // Lösungsarray
O(1)    6       j = G                   // Resthapazität des Wagens
        7       for i to n
        8           c = [illegible] - g[ID[i]]   // zu nehmende Menge
        9           if j-c < 0
O(n)    10              c = j           // nur so viel nehmen wie der Wagen noch transportieren kann
        11          L[ID[i]] = c        // speichern der Menge
        12          j = j - c           // Update der Restkapaz
O(1)    13      return L
```

O(n log n) weil das sortieren am aufwändigsten ist.

d)

optimale Lösung ist nicht möglich!

Gegenbeispiel:

G = 900g		Greedy	Opt
Spaghetti	700g	0,3[illegible]€	X
Reis	300g	0,25€	X
Bulgur	300g	0,25€	X
Mehl	300g	0,25€	X
		2,1€	2,25€

c) Beh GreedyBob(G, g, p, n) gibt ein Array L der Länge n aus, in dem an Stelle i für $1 \leq i \leq n$ die Menge der i-ten Ware, die Bob kaufen muss, um seinen Profit zu maximieren

Bew

Annahme L beinhaltet nicht die optimalen Menge der Waren

=> $\exists$ x (400g einer Ware, die nicht in L gewählt wurde) und $\exists$ y (400g einer Ware, die in L gewählt wurde) sodass x statt y wählen zu einem besseren Ergebnis führt.

↳ dann sollte die Ware zu x vor y in der sortierten Liste ID stehen, da ihr Profit p höher sein muss ↯

dies ist nicht möglich, da der Algorithmus in der for-Schleife in Z 7-13 y erst wählt, wenn alle Waren mit höherem Profit bereits in L gewählt sind

=> Also kann solch ein x nicht geben und die Annahme gilt nicht.

=> Also beinhaltet L die optimalen Längen der Waren. □