Karlsruher Institut für Technologie

Institut für Theoretische Informatik
T.T.-Prof. Dr. Thomas Bläsius
https://scale.iti.kit.edu

Übungsblatt 13

Algorithmen I - Sommersemester 2024

Abgabe im ILIAS bis 26.07.2024, 18:00 Uhr

Die Abgabe erfolgt alleine oder zu zweit als eine PDF-Datei über das Übungsmodul in der Gruppe eures Tutoriums im ILIAS. Beschriftet die Abgabe deutlich mit Matrikelnummer und Name.

- Achtet bei handschriftlichen Abgaben auf Lesbarkeit.
- Achtet darauf, ob Algorithmen in Worten oder Pseudocode beschrieben werden sollen.
- Es werden immer asymptotisch möglichst effiziente Algorithmen erwartet, wenn nicht anders angegeben.
- Wenn Korrektheits- oder Laufzeitanalysen gefordert sind, behandelt diese separat von der Algorithmenbeschreibung.

Gesamtpunkte: 20 (+ 6 Bonus)

Aufgabe 1 - Biber im Sägewerk Bad Segeberg (10 Punkte)

Neben ihrer Arbeit für Dr. Meta wollen sich die Biber noch etwas Taschengeld dazu verdienen. Dafür haben sie sich einen Baumstamm der Länge n Meter geklaut. Der Händler, dem sie den Stamm verkaufen wollen bietet ihnen nun verschiedene Tarife an. Für jedes $i \in \{1, \dots, n\}$ nennt er den Bibern eine Summe, die er für einen Baumstamm der Länge i zahlen würde. Die Biber überlegen nun, wie sie den Baumstamm zernagen müssen, um möglichst viel Geld zu erhalten. Durch jahrelange Übung können die Biber einen Baumstamm so zernagen, dass kein Material verloren geht.

Etwas formaler formuliert stellt sich für die Biber die folgende Problemstellung. Gegeben sind eine Zahl $n \in \mathbb{N}_+$ und ein Array offer : $[\mathbb{N}_0; n+1]$. Gesucht ist eine Aufteilung von n in Zahlen $l_1, \dots, l_k$ mit $l_i \geq 1, \forall i \in \{1, \dots, k\}$ mit Summe $\sum_{i=1}^{k} l_i = n$, so dass und $\sum_{i=1}^{k} \text{offer}[l_i]$ maximal ist. Beachte hierbei, dass das Angebot für einen Baumstamm der Länge i in offer$[i]$ gespeichert ist und offer$[0] = 0$. Wir wollen das Problem mittels eines Dynamischen Programms lösen.

a) Gib für $n = 7$ und offer $= [0, 1, 3, 2, 4, 6, 7, 4]$ die optimale Aufteilung der Zahl n an. Gib außerdem den erzielten Gewinn an. (1.5 Punkte)

b) Gib an, in welche Teilprobleme du das Gesamtproblem aufteilen kannst und welche Werte du dann für eine Teillösung speichern musst. (1.5 Punkte)

c) Welche Teillösungen stellen den Basisfall deines DPs dar? Welche Werte werden diesen zugewiesen? (1 Punkt)

d) Gib eine Rekurrenz an, die aus den Werten der bisherigen Teillösungen den Wert der nächsten Teillösung berechnet. (2 Punkte)

e) Beschreibe, wie du mithilfe der Rekurrenz den maximal möglichen Gewinn berechnen kannst. Begründe außerdem die Laufzeit, die das DP benötigt, um den maximalen Gewinn zu bestimmen. (2 Punkte)

f) Welche Information musst du zusätzlich für jede Teillösung speichern, damit du auch die optimale Aufteilung rekonstruieren kannst? Erkläre, wie du in $\mathcal{O}(n)$ die optimale Aufteilung bestimmen kannst, falls die Werte des DPs und die zuätzlichen Informationen bereits berechnet wurden. (2 Punkte)

Lösung 1

a) Wir können den Baumstamm in drei Stücke der Länge 2 und ein Stück der Länge 1 aufteilen. Somit erzielen wir einen Gewinn von 10.

b) Wir betrachten einen Stamm der Länge ℓ für $\ell \in \{1, \ldots, n\}$ als Teilproblem. Die Funktion $W : \mathbb{N}_+ \to \mathbb{N}_0$ gibt den maximal möglichen Gewinn für einen Stamm einer gegebenen Länge an. Wir nutzen somit $W[\ell]$ als Wert der Teillösung mit Länge ℓ.

c) Der Basisfall ist ein Stamm der Länge 1. Dieser kann nicht weiter aufgeteilt werden. Es gilt daher $W[1] = \text{offer}[1]$.

d) Wir können aus einem gegebenen Baumstamm ein Stück der Länge i an einem Ende entfernen und dann für den verbleibenden Baumstamm die optimale Aufteilung nachschlagen. Wenn wir dies für alle möglichen Längen i berechnen, finden wir auch die optimale Aufteilung des gegebenenen Stamms. Dies ist dadurch begründet, dass optimale Aufteilungen des restlichen Stammes auch Teil der optimalen Aufteilung des gesamten Stammes sind. Es ergibt sich somit die folgende Rekurrenz für W:

$$W[l] = \max_{i \in \{1,\ldots,l\}} (\text{offer}[i] + W[l - i])$$

e) Wir bestimmen mithilfe der Rekurrenz die Werte aller Teillösungen beginnend bei 1. Der Wert der optimalen Gesamtlösung ist dann der Wert der Teillösung mit der Länge n, also $W[n]$.

   Wir benötigen Laufzeit $\mathcal{O}(n^2)$, da wir n Teillösungen bestimmen müssen und für jede Teillösung das Maximum über $\mathcal{O}(n)$ Werte bilden müssen. Der Zugriff auf die Arrayeinträge erfolgt in konstanter Zeit.

f) Wir speichern uns für jede Teillösung, für welches i der Wert maximal wurde. Nun können wir bei der Teillösung für n beginnen und dann die Aufteilung rekursiv zurückverfolgen. Wenn wir die Teillösung l betrachten und dort ein Stück der Länge i gewählt wurde, können wir i zur Aufteilung hinzufügen und die Rekonstruktion an Teillösing $l - i$ fortsetzen. Dies ist offensichtlich in Linearzeit möglich, da wir Teillösungen in absteigender Länge betrachten und pro Teillösungen konstante Laufzeit benötigen.

Aufgabe 2 - All in! (10 Punkte)

Nachdem Dr. Meta nun wiederholt kurz vor Schluss (mit deiner Hilfe) aufgehalten wurde, schlägt er nun vor, die Sache in einem direkten Duell zu klären. Wenn er gewinnt, darf er ungestört mit seinen bösen Plänen fortfahren. Gewinnst du, muss Dr. Meta sich während der Klausurenphase ruhig verhalten. Dafür schlägt Dr. Meta das folgende Spiel vor:

- Auf dem Tisch vor euch stehen n KIT-Rucksäcke in einer Reihe. Jeder Rucksack ist mit einer großgeschriebenen Nummer versehen, die den Wert des Inhalts repräsentiert. Wir stellen diese Werte durch eine Folge von natürlichen Zahlen $\langle r_0, \ldots, r_{n-1} \rangle$ dar, wobei r_i den Wert des i-ten Rucksacks angibt.
- Dr. Meta und du seid abwechselnd an der Reihe. Dr. Meta lässt dich beginnen.
- Der Spieler, der an der Reihe ist, wählt einen Rucksack von einem der beiden Enden (links oder rechts) der Reihe aus.
- Die Zahl auf dem gewählten Rucksack wird zur Punktzahl des Spielers addiert. Der gewählte Rucksack wird dann aus der Reihe entfernt, sodass die Reihe für den nächsten Spielzug kürzer wird.
- Das Spiel endet, wenn alle Rucksäcke gewählt wurden.
- Der Gewinner des Spiels ist der Spieler mit der höheren Gesamtpunktzahl am Ende. Bei gleicher Punktzahl gewinnst du.

Du befürchtest, dass Dr. Meta viel Erfahrung besitzt und daher **in jedem Zug optimal wählt**. Verhindere, dass Dr. Meta die Klausurenphase manipuliert und entwirf daher ein DP, das dir die optimalen Spielzüge berechnet!

a) Gegeben sei die Zahlenfolge $\langle 3, 5, 1, 2 \rangle$. Gib an, welche Zahlen von welchem Spieler in welchem Zug gewählt werden, wenn beide Spieler optimal wählen. Bestimme außerdem den Sieger dieses Spiels. (1 Punkt)

b) Gib für die Fälle $n = 1$ und $n = 2$ an mit welchen Zügen du das Spiel immer gewinnst. (1 Punkt)

c) Kannst du für $n = 3$ immer Züge so wählen, dass du das Spiel garantiert gewinnst? Gib, wenn möglich, an wie die Züge gewählt werden können und begründe ihre Optimalität. Anderenfalls gib ein Beispiel an, in dem Dr. Meta unabhängig von deiner Auswahl immer gewinnt. (1 Punkt)

Du möchtest nun ein dynamisches Programm entwickeln, mit dem du die optimalen Spielzüge identifizieren kannst. Hierfür betrachtest du die verschiedenen Teilfolgen der ursprünglichen Zahlenfolge als Teillösung. Jeder Teilfolge soll die maximale Punktzahl zugewiesen werden, welche du erreichen kannst, wenn die betrachtete Teilfolge übrig ist und du gerade am Zug bist. Diese Punktzahl soll in einem zweidimensionalen Array X gespeichert werden: $X[i, j]$ enthält die maximale erreichbare Punktzahl für die Person die am Zug ist, wenn die Folge $\langle r_i, \ldots, r_j \rangle$ übrig ist. Wir bezeichnen $\langle r_i, \ldots, r_j \rangle$ auch als Teilfolge mit Startindex i und Endindex j. Außerdem bezeichnen wir die Summe aller Zahlen der Teilfolge mit $G[i, j]$.

d) Stelle die Rekurrenz auf, mit deren Hilfe das Array X korrekt ausgefüllt werden kann. (3 Punkte)

Hinweis: Achte darauf, auch die Basisfälle der Rekurrenz anzugeben, falls nötig.

e) Gib einen Algorithmus in Pseudocode an, der als Eingabe die Elemente $r_1, \ldots, r_n$ erhält und in $\mathcal{O}(n^2)$ Zeit ausgibt, wie viele Punkte du erreichst, wenn du r_1, beziehungsweise r_n wählst. Begründe außerdem wieso der Algorithmus die geforderte Laufzeitschranke einhält. (4 Punkte)

Verwende folgende Signatur: ERREICHBAREPUNKTE$(\langle r_0, \ldots, r_{n-1} \rangle) : \mathbb{N}_0, \mathbb{N}_0$

Lösung 2

a) Du beginnst und wählst die 2, dann wählt Dr. Meta die 3. Es verbleiben die 5 für dich und die 1 für Dr. Meta. Somit hast du mit 7 Punkten gegen Dr. Metas 4 Punkte gewonnen.

b) Für $n = 1$ kannst du die einzige Zahl wählen, für $n = 2$ kannst du die größere der beiden Zahlen wählen. Im ersten Fall kann Dr. Meta keine Zahl wählen, im zweiten Fall kann er nur eine wählen, die kleiner als deine Zahl ist. Sind für $n = 2$ beide Zahlen gleich groß gewinnst ergibt erzielen beide Spieler die gleiche Punktzahl und du gewinnst.

c) Für die Folge $\langle 1, 10, 2\rangle$ kann keine solche Strategie angegeben werden. Dr. Meta erreicht stets 10 und du stets 3 Punkte. Allgemein tritt dieser Fall immer auf, falls die mittlere Zahl größer, als die Summe der äußeren ist.

d) Wenn wir X bereits für alle kürzeren Teilfolgen ausgefüllt haben, können wir für die aktuelle Teilfolge die erreichbare Punktzahl beim Wählen vom linken oder rechten Ende bestimmen. In beiden Fällen erhalten wir den Wert des gewählten Rucksacks und die Punkte, die Punkte die wir auf der verbleibenden Teilfolge erreichen können. Letztere ergeben sich als die verbleibende Gesamtpunktzahl (gespeichert in $G[i, j]$) minus die vom Gegenspieler erreichbare Punktzahl (gespeichert in $X[i, j]$).

Somit ergibt sich die folgende Rekurrenz.

$$X(i,j) = \begin{cases} r_i & \text{falls} i = j, \\ \max(r_i + \text{rest}_1, r_j + \text{rest}_2) & \text{falls} j \geq i. \end{cases}$$

wobei

$$\begin{aligned} \text{rest}_1 &= G[i+1, j] - X[i+1, j], \\ \text{rest}_2 &= G[i, j-1] - X[i, j-1]. \end{aligned}$$

e) Wir geben den folgenden Pseudocode an:

```
1:  erreichbarePunkte((⟨r_1, ..., r_n⟩)): ℕ_0, ℕ_0
2:      G : [[ℕ_0; n]; n]
3:      for i ∈ {0, ..., n − 1} do
4:          G[i][i] ← r_i
5:      end
6:      for i ∈ {0, ..., n − 1} do
7:          for j ∈ {i + 1, ..., n − 1} do
8:              G[i][j] ← G[i][j − 1] + r_j
9:          end
10:     end
11:     X : [[ℕ_0; n]; n]
12:     for length ∈ {0, ..., n − 1} do
13:         for i ∈ {0, ..., n − 1} do
14:             if length = 0 then
15:                 X[i][i] ← r_i
16:             else
17:                 if i + length ≤ n − 1 then
18:                     choose_left ← r_i + (G[i + 1, i + length] − X[i + 1, i + length])
19:                     choose_right ← r_{i+length} + (G[i, i + length − 1] − X[i, i + length − 1])
20:                     X[i][i + length] ← max(choose_left, choose_right)
21:                 end
22:             end
23:         end
24:     end
25:     return G[0, n − 1] − X[1, n − 1], G[0, n − 1] − X[0, n − 2]
```

Wir führen zwei Schritte aus. Zuerst berechnen wir in G die summierten Punktzahlen jeder Teilfolge. Hierfür durchlaufen wir das Array zeilenweise und addieren je zwei Werte. Im zweiten Schritt wird die Rekurrenz ausgewertet. Hierfür betrachten wir jeden Eintrag $X[i, j]$ mit $i < j$ genau einmal und berechnen den Wert der Rekurrenz. Hierfür lesen wir konstant viele Werte aus und addieren bzw. subtrahieren diese. Somit erfüllen wir die Laufzeitschranke.

Aufgabe 3 - Zum Knobeln (0 Punkte + 6 Bonus)

Wir wollen nun nochmal einen Rückblick auf das in diesem Semester Gelernte wagen:

a) **$\mathcal{O}$-Kalkül**: Welche der Funktionen $\sqrt{n \cdot \log(n)}$ und $n \cdot \log(\sqrt{n})$ wächst asymptotisch schneller? Begründe deine Antwort. (0.5 Punkte)

b) **Amortisierte Analyse**: Wir rufen auf einer leeren Datenstruktur in beliebiger Reihenfolge m mal die Operation `doThis` und n mal die Operation `doThat` auf. Die Gesamtlaufzeit der Operationsfolge für beliebige n und m ist $\mathcal{O}(n \cdot \sqrt{n} + m \cdot \log(m))$. Welche amortisierte Laufzeiten haben die Operationen jeweils? (0.5 Punkte)

c) **Sortieren**: Erkläre, welche Voraussetzung erfüllt sein muss, damit du n Zahlen mit BucketSort in $\mathcal{O}(n)$ sortieren kannst. (0.5 Punkte)

d) **Hashing**: Welche Bedingungen muss eine Universelle Familie von Hashfunktionen erfüllen? (0.5 Punkte)

e) **Graph Algorithmen**: welcher Algorithmus ist am besten geeignet um für einen gegebenen Graphen G und einen Knoten $s \in V(G)$ alle kürzesten Wege von s zu anderen Knoten zu bestimmen? (0.5 Punkte)

f) **Algorithmus von Dijkstra**: Gib einen Beispielgraphen ohne negative Kreise mit beliebigen Kantengewichten an, auf dem Dijkstras Algorithmus die kürzesten Wege **nicht** findet. (0.5 Punkte)

g) **Heaps**: Nenne die Heap-Eigenschaft in Min-Heaps. (0.5 Punkte)

h) **Sortierte Folgen**: Wie stellen wir sicher, dass $(2, 3)$-Bäume stets logarithmische Tiefe haben? (0.5 Punkte)

i) **DFS**: In welcher Reihenfolge musst du dich laut Vorlesung morgens anziehen? (0.5 Punkte)

j) **DFS**: Wie nennt man einen zusammenhängenden Graphen bei dem in einer DFS Traversierung für jeden Knoten $v \in V$ gilt $\text{low}(v) = \text{dfs}(v)$? (0.5 Punkte)

k) **MST**: Was besagt die Schnitteigenschaft? (0.5 Punkte)

l) **DP**: Welche drei Schritte musst du beim Lösen eines Problems mittels DP beachten? (0.5 Punkte)

Lösung 3

a) Es gilt $n \cdot \log(\sqrt{n}) = n \cdot \log(n^{\frac{1}{2}}) = \frac{1}{2} \cdot n \cdot \log(n)$. Somit wächst diese Funktion schneller als $\sqrt{n \cdot \log(n)}$, da hier keine Wurzel das Wachstum verringert.

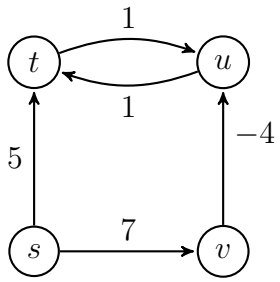
b) Es ergeben sich amortisierte Laufzeiten von $\sqrt{n}$ für `doThat` und $\log(m)$ für `doThis`.

c) Die Zahlen müssen alle ganzzahlig sein und von oben linear in n beschränkt sein.

d) Sei H eine Menge von Hashfunktionen. H ist eine universelle Familie, wenn für alle $k_1, k_2 \in U$ mit $k_1 \neq k_2$ und ein zufälliges $h \in H$ gilt, dass $\Pr[h(k_1) = h(k_2)] \leq \frac{1}{m}$, wobei m die Anzahl Buckets ist.

e) Für einen Graphen ohne Kantengewichte eignet sich die Breitensuche (BFS) am besten.

f) Betrachte den folgenden Graphen $G = (V, E)$ mit $s, t \in V$:



g) Die Heapeigenschaft besagt, dass der Wert des Elternknoten immer kleiner als der Wert des Kindkonten ist.

h) Wir stellen sicher, dass zu jedem Zeitpunkt alle Blätter die selbe Distanz zur Wurzel aufweisen. In Kombination damit, dass innere Knoten mindestenz 2 Kinder haben müssen, ergibt sich somit eine Tiefe von höchstens $\log_2 n$.

i) Die Vorlesung empfiehlt Unterhose, Socken, Unterhemd, Hose, T-Shirt, Pullover und abschließend Schuhe. Andere topologische Sortierungen sind auch korrekt.

j) Baum.

k) Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist. Beachte, dass wir eindeutige Gewichte annehmen.

l) Die Schritte sind: Spezifikation der Teilprobleme, Aufstellung der Rekurrenz für Teillösungen und Iterative Berechnung der Teillösungen.

Auf Wunsch der Fachschaft möchten wir auf folgende Veranstaltung hinweisen:

Orientierungsveranstaltung – Mit Schwung ins dritte Semester

Du bist gerade im zweiten Semester Informatik und willst Tipps zu der kommenden Prüfungsphase und dem dritten Semester? Dann komm einfach zur Orientierungsveranstaltung der Fachschaft Mathe/Info am

24.07. um 17:30 Uhr in Raum -101 im Infobau (50.34).

Dort beantworten wir Fragen wie:

Wie bereite ich mich auf Klausuren vor? Was mache ich, wenn ich eine Klausur/einen Übungsschein nicht bestanden habe? Welche Ergänzungsfächer gibt es? Welche Möglichkeiten zur Unterstützung gibt es?

Außerdem geben wir dir einen generellen Überblick über die Vorlesungen im dritten Semester, Tipps für PSE sowie zu Tutorenstellen im 3. Semester und vielem mehr.

Zudem sind viele Fachschaftler:innen anwesend, die dir weitere Fragen im Anschluss bei Snacks und Getränken persönlich beantworten können.

Wir freuen uns auf dich!