# Aufgabensammlung ADS-Repetitorium 2021

Amortisierte Analyse – Dynamische Programmierung

**Aufgabe 1: Bank-Schließfächer und Heaps**

Bei einer Bank können Sie Schließfächer mieten. In jedes Schließfach passt genau ein Gegenstand. Der Service ist aber nicht kostenlos. Für das Einlagern des $n$. Gegenstands stellt die Bank $\log n$ Euro in Rechnung. Ebenfalls werden bei der Rückgabe des $n$. Gegenstands $\log n$ Euro fällig.

(a) Geben Sie die Gesamtkosten für das Ein- und Auslagern den $n$. Gegenstands in $\Theta$-Notation an.

(b) Wie muss die Bank ihre Bezahlpolitik ändern, sodass das Abholen der Gegenstände kostenlos ist, die Bank aber trotzdem denselben Gewinn macht?

(c) Geben Sie die Kosten für das Ein- und Auslagern mit der neuen Bezahlpolitik in $\Theta$-Notation an.

(d) Übertragen Sie Ihre Überlegungen aus den vorherigen Teilaufgaben auf einen MaxHeap. Zeigen Sie mit der Buchhaltermethode, dass die Insert-Methode amortisiert eine Laufzeit von $\mathcal{O}(\log n)$ und ExtractMax eine konstante Laufzeit hat.

(e) Lösen Sie die Aufgabe nun mit der Potentialmethode.

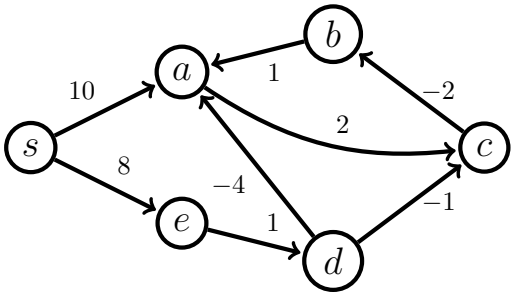
**Aufgabe 2: Kürzeste Wege mit negativen Kanten**

Gestern haben wir festgestellt, dass die Ergebnisse von Dijkstra auf Graphen mit negativen Kanten unter Umständen nicht die kürzesten Wege repräsentieren. In dieser Aufgabe wollen wir einen Algorithmus finden, der auf einem Graphen $G = (V, E)$ mit der Gewichtsfunktion $w : V \times V \to \mathbb{R}$ einen kürzesten Weg zwischen zwei Knoten $s$ und $t$ findet. Wir nehmen an, dass der Graph $G$ keine von $s$ erreichbaren, negativen Kreise enthält.

(a) Zeigen Sie, dass das Problem eine optimale Substruktur aufweist. Sie müssen also zeigen, dass der kürzeste Weg zwischen $s$ und $t$ aus kleineren Teillösungen desselben Problems berechenbar ist. In welchen Fällen ist es besonders einfach, den kürzesten Weg zwischen $s$ und $t$ zu berechnen?

(b) Wie viele Kanten kann jeder kürzeste Weg im Graphen $G = (V, E)$ maximal haben? Angenommen, der Distanzwert $v.d$ ist für alle $v \in V$ mit $\infty$ intialisiert und $s.d = 0$. Was passiert auf jeden Fall, wenn Sie die Relax-Methode (siehe Dijkstra) nun auf *jede* Kante in beliebiger Reihenfolge aufrufen? Was passiert, wenn Sie dies erneut tun?

(c) Wir betrachten nun eine zweidimensionale Tabelle $T$. Jede Spalte steht für einen Knoten (in beliebiger Reihenfolge), und es gibt $|V| - 1$ Zeilen. Die Zelle $T(i, j)$ enthält die *Länge* des kürzesten Weges von $s$ zum $i$-ten Knoten, nachdem $j$ Mal die Relax-Methode auf *alle* Kanten aufgerufen wurde. Stellen Sie die Tabelle für folgenden Graphen auf und füllen Sie sie zeilenweise aus. Relaxieren Sie die Kanten in alphabetischer Reihenfolge nach ihrem Startknoten.




| Iteration | $s.d$ | $a.d$ | $b.d$ | $c.d$ | $d.d$ | $e.d$ |
|---|---|---|---|---|---|---|
| 0 | 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |

(d) Geben Sie nun den Algorithmus in Pseudocode an. Welche Laufzeit hat er?

(e) Modifizieren Sie Ihren Algorithmus so, dass er auch die optimale Lösung selbst, also den kürzesten Weg berechnet. Welche Methode aus der Vorlesung können Sie dafür verwenden?

(f) Unter welchen Umständen kann der Algorithmus vorzeitig abgebrochen werden? Wie kann mithilfe des Algorithmus ein negativer Zykel detektiert werden?

**Aufgabe 3: Palindrome Subsequenzen**

Ein *Palindrom* ist eine Zeichenkette, die von vorne und von hinten gelesen das gleiche ergibt, zum Beispiel das Adjektiv „soldlos“. Eine *Subsequenz* ist ein String, der nach Weglassen beliebig vieler Zeichen aus einem String hervorgeht, beispielsweise das Wort „Baum“ aus „Brauchtum“. Wir suchen nun einen effizienten Algorithmus, der eine längste Subsequenz einer Zeichenkette $s = s_0 \dots s_n$ findet, die gleichzeitig ein Palindrom ist. Die längste palindrome Subsequenz in „Amortisierte Laufzeit“ ist „tieteit“.

(a) Erklären Sie kurz, wie ein Brute-Force-Algorithmus vorgehen würde, um das Problem zu lösen. Was ist die Laufzeit dieses Algorithmus?

(b) Gegeben sei eine Zeichenkette $s = s_0 \dots s_n$ und ihre längste palindrome Subsequenz $p = p_0 \dots p_m$. Beschreiben Sie wie $p$ aus einer kleineren Instanz desselben Problems hervorgeht. Betrachten Sie dazu einen Teilstring $s'$ von $s$, und erklären Sie, wie $p$ zu $s'$ steht.

(c) Wir definieren $l(i, j)$ als die Länge der längsten palindromen Subsequenz im Substring $s_i \dots s_j$. Was ist $l(i, i)$ und $l(i, i+1)$? Dies sind die Basisfälle und sind einfach anzugeben. Überlegen Sie sich nun, wie Sie für allgemeine $i, j$ mit $i < j$ den Wert $l(l, j)$ berechnen können. *Tipp*: Machen Sie eine Fallunterscheidung nach $s_i = s_j$ bzw. $s_i \neq s_j$ und greifen Sie auf $l(i', j')$ zu, wobei $i' < i$ oder $j' < j$.

(d) Legen Sie eine Matrix, die $l(i, j)$ für alle $0 \leq i \leq j \leq n$ repräsentiert, für die beiden Sequenzen „anna“ und „graphalgo“ an und füllen Sie sie aus. Wie lang sind die längsten palindromen Subsquenzen in den Wörtern? Wo steht der Wert der Lösung in der Matrix?

(e) Formulieren Sie jetzt einen Algorithmus, der eine solche Matrix automatisch ausfüllt und den *Wert* der Lösung zurückgibt.

(f) Jetzt möchten Sie nicht nur den Wert ermitteln, sondern auch die längste palindrome Subsequenz selbst. Beschreiben Sie, wie Sie mit einer zweiten Matrix die längste palindrome Subsequenz ermitteln können.

(g) Zeichnen Sie auch die ergänzte Matrix für die Wörter „anna“ und „graphalgo“. Was ist die jeweils längste palindrome Subsequenz?

(h) Ändern Sie Ihren Algorithmus so, dass er die längste palindrome Subsequenz zurückgibt.

(i) Überlegen Sie sich, wie Sie Ihren Algorithmus ändern können, sodass er den längsten palindromen *Substring* findet. Im Wort „stirnlappenbasilisk“ ist der längste palindrome Substring „silis“, während die bisher betrachtete palindrome Subsequenz „silappalis“ ist. Formulieren Sie die Matrix-Rekurrenzen aus Teilaufgabe c) um und beschreiben Sie in Worten, wo sich in der Matrix jetzt der Wert der Lösung befindet und wie Sie die Lösung rekonstruieren können.

(j) Falls Sie noch Zeit haben, geben Sie einen Brute-Force-Algorithmus an, der eine längste palindrome Subsequenz findet.