Marburg University

Department of Computer Science
Prof. Dr. Sebastian Wild

Date: 2025-10-24
Version: 2025-10-24 01:47

Übungsblatt 2 zur Vorlesung Effiziente Algorithmen (Winter 2025/26)

Abgabe: *Bis 2025-10-31 18:00, on ILIAS.*

1. Aufgabe

10 + 10 Punkte

Beweisen Sie die folgenden Aussagen, jeweils für $n \to \infty$.

a) Sei $P(n) = a_0 + a_1 n + \cdots + a_k n^k$ ein beliebiges Polynom mit $a_k > 0$.
Dann gilt $P(n) \sim a_k n^k$.

b) Für alle $\alpha, \varepsilon \in \mathbb{R}_{>0}$ gilt $n^\alpha \in o(n^{\alpha+\varepsilon})$.

2. Aufgabe

10 + 10 + 20 + 20 Punkte

Beweisen oder widerlegen Sie folgende Behauptungen, jeweils für $n \to \infty$.

a) $2^{2n} \sim 4^n$.

b) $\sqrt[n]{n} = o(1)$.

c) Für beliebige Funktionen: $f, g : \mathbb{N} \to \mathbb{R}$ gilt $f(n) = O(g(n))$ oder $g(n) = O(f(n))$.

d) $(n + m)^4 \in \Theta(n^4)$ für $m \in \mathcal{O}(1)$.

3. Aufgabe

10 + 20 + 20 + 10 Punkte

In der Vorlesung wurden mehrere Lösung für das Maximum Subarray Problem vorgestellt. In dieser Aufgabe sollen Sie eine Lösung für die 2-dimensionale Variante des Problems erarbeiten. Die Definition für das 2D Maximum Subarray Problem ist wie folgt:

Gegeben sei eine ganzzahlige $n \times n$ Matrix M, sodass $M[i, j]$ mit $0 \leq i, j \leq n$ den Wert der Matrix in der i-ten Zeile und der j-ten Spalte liefert. Sei $z_k(x_1, x_p) = \sum_{i=x_1}^{x_p} M[i, k]$ die Summe der Zeilen x_1 bis x_p der k-ten Spalte der Matrix. Ferner sei $S(x_1, x_p, y_1, y_q) =$

$\sum_{i=y_1}^{y_q} z_i(x_1, x_p)$ die Summe der Submatrix von M der Spalten y_1 bis y_q mit jeweils den Zeilen x_1 bis x_p. Bestimmen Sie die Koordinaten der maximalen Submatrix $S_{\max}$ in M, d.h.

$$S_{\max} = \left\{(x_1, x_p, y_1, y_q) \;\middle|\; \forall(x_1', x_p', y_1', y_q') : S(x_1, x_p, y_1, y_q) \geq S(x_1', x_p', y_1', y_q')\right\}.$$

Sie finden die Code Basis für diese Aufgabe auf der Kurswebseite. Ferner ist die Code Basis auch eingebettet in diesem PDF (letzteres könnte bei einigen PDF Viewern nicht funktionieren):

SubArrayProblem.java

Anmerkung: $S_{\max}$ gibt eine Menge zurück, im Code und in dieser Aufgabenstellung reicht es eine mögliche Lösung zurückzugegeben.

a) Die `bruteForce` Methode der `SubArrayProblem` Klasse liefer eine (sehr naive) Lösung für 2D Maximum Subarray Problem zurück. Geben Sie eine möglichst kleine obere Schranke in O-Notation für den `bruteForce` Algorithmus an. Begründen Sie Ihre Lösung.

b) Beschreiben Sie textuell einen Algorithmus, welcher das Problem in Zeit $\mathcal{O}(n^3)$ löst.

   Hinweis: Sie können die $\Theta(n)$ Lösung für das 1-dimensionale Problem aus der Vorlesung verwenden. Begründen Sie die Laufzeit Ihrer Lösung.

c) Implementieren Sie die Ihre Lösung aus b) in einer Methode `efficient` in der Klasse `SubArrayProblem`.

d) Erstellen Sie in der Klasse `SubArrayProblem` ein geeignetes Experiment, um die Laufzeiten von `bruteForce` und `efficient` experimentell zu ermitteln. Plotten Sie einen Vergleich der Ergebnisse und beschreiben Sie diese.

4. Aufgabe

30 Punkte

Wir betrachten einen Stack S der die folgenden Operationen unterstützt:

1. Push(S, x): Legt das Element x auf den Stack S.
2. Pop(S): Entfernt das oberste Element vom Stack S.
3. MultiPop(S, k): Entfernt die obersten k Elemente vom Stack S.

Push und Pop haben eine Laufzeit von $O(1)$, d.h. eine Sequenz von n Push- oder Pop-Operationen hat eine Laufzeit von $\Theta(n)$. MultiPop entfernt die (maximal) obersten k Elemente des Stacks S der größe s in einer Folge von Pop-Operationen und hat entsprechend die Kosten $\min(s, k)$.

Wir wollen nun eine Sequenz von PUSH-, POP- und MULTIPOP-Operationen auf einen initial leeren Stack betrachten. Da der Stack höchstens die Höhe n hat, hat MULTIPOP im Worst-Case die Laufzeit $O(n)$. Entsprechend hat eine Folge von n dieser drei Operationen höchstens die Laufzeit $O(n^2)$. Diese obere Schranke ist allerdings nicht sehr nah an den tatsächlichen Kosten.

Bestimmen Sie die amortisierten Kosten mit der Potentialmethode: Bestimmen Sie eine geeignetes Potential Φ, welches den Stack D_i, der nach Anwendung er i-ten Stack-Operation entsteht, die nicht-negative Zahl $\Phi(D_i)$ zuordnet. Berechnen Sie damit die amortisierten Kosten der drei Stack-Operationen und folgern Sie eine niedrigere obere Schranke für die Kosten von n Operationen als die ursprünglichen $O(n^2)$.