



Prof. Dr. Sebastian Wild
Dr. Nikolaus Glombiewski

Übungen zur Vorlesung
Effiziente Algorithmen

Abgabe: 08.11.2024,
bis **spätestens** 19:00 Uhr
über die ILIAS Plattform

Übungsblatt 2

Aufgabe 2.1: Amortisierte Analyse

(3 Punkte)

Wir betrachten einen Stack S der die folgenden Operationen unterstützt:

1. $\text{PUSH}(S, x)$: Legt das Element x auf den Stack S .
2. $\text{POP}(S)$: Entfernt das oberste Element vom Stack S .
3. $\text{MULTIPOP}(S, k)$: 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 der 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 $O(n^2)$.

Aufgabe 2.2: Binäre Bäume (2+2+2)

(6 Punkte)

- a) Gegeben sei die Menge von Schlüsseln $\{2, 5, 6, 11, 17, 18, 22\}$. Zeichnen Sie binäre Suchbäume mit den Höhen 2, 3, 5 und 6.
- b) Bei einem *erweiterten Binärbaum* ist die Anzahl von Kindern entweder 0 oder 2, d.h. es darf keine unären Knoten geben. Zeigen Sie durch strukturelle Induktion: Die Anzahl Knoten (Blätter und innere Knoten) in einem erweiterten Binärbaum ist ungerade.
- c) Gegeben sei folgender Sortieralgorithmus: Für eine Menge von n Zahlen wird ein (unbalancierter) binärer Suchbaum aufgebaut, indem n Mal die Einfüge-Operation ausgeführt wird. Anschließend wird eine *in-order* Traversierung des Baums durchgeführt. Geben Sie die Best- und Worst-Case Laufzeiten des Sortieralgorithmus an. Begründen Sie Ihre Antwort.

Aufgabe 2.3: Entwurf von Datenstrukturen

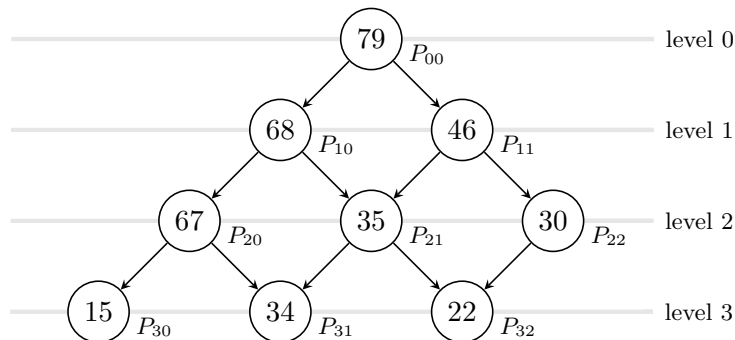
(3 Punkte)

Entwerfen Sie eine Datenstruktur, welche die folgenden Operationen in konstanter Zeit und konstantem Speicheraufwand unterstützt: *push*, *pop*, *getMin*. Begründen Sie Ihre Antwort.

Aufgabe 2.4: Priority Queues (1+2+2+3)

(8 Punkte)

Eine *Pyramide* ist eine Datenstruktur, welche die Basis für eine Implementierung des abstrakten Datentyps Priority Queue bildet. Im Folgenden ist ein Beispiel für eine Pyramide mit 9 Knoten:



Eine Pyramide hat die folgenden beiden Eigenschaften:

- **Struktur:** Eine Pyramide besteht aus $\ell \geq 0$ Leveln. Das i -te Level, für $0 \leq i < \ell$, enthält höchstens $i + 1$ Einträge, welche als $P_{i,0}, \dots, P_{i,i}$ bezeichnet werden.
Bis auf potentiell das letzte Level ist jedes Level vollständig gefüllt. Das letzte Level ist links-bündig, d.h. Knoten sind von links nach rechts befüllt.
- **Ordnung:** Jeder Knoten $P_{i,j}$ hat höchstens zwei Kinder: $P_{i+1,j}$ und $P_{i+1,j+1}$, falls diese Knoten existieren. Die Priorität eines Knoten ist immer größer oder gleich der Priorität seiner beiden Kinder.

Sie dürfen annehmen, dass alle Elemente einer Pyramide paarweise verschieden sind.

- a) Zeigen Sie, dass eine Pyramide mit n Kindern die Höhe $\Theta(\sqrt{n})$ hat.
- b) Angenommen sie können auf die Elemente *key*, *leftChild*, *rightChild*, *leftParent*, and *rightParent* eines Knotens in $O(1)$ Laufzeit zugreifen.

Geben Sie eine Implementierung in Pseudocode für die *deleteMax* Operation in Pyramiden an. Die Laufzeit Ihres Algorithmus muss linear in der Höhe der Pyramide sein.

Hinweis: Orientieren Sie sich an der deleteMax Operation von Heaps.

- c) Geben Sie eine Implementierung in Pseudocode für die *insert* Operation in Pyramiden an. Die Laufzeit Ihres Algorithmus muss linear in der Höhe der Pyramide sein.
- d) Die Operation *contains*(x) nimmt eine Priorität x entgegen, und gibt zurück, ob eine Priority Queue einen Schlüssel mit der Priorität x enthält.

In einem binären Heap muss im Worst Case der gesamte Heap durchsucht werden. In Pyramiden lässt sich die *contains* Operation effizienter implementieren.

Beschreiben Sie eine Implementierung für *contains* in Pyramidem mit einer Laufzeit von $O(\sqrt{n} \log n)$. Für Ihre Implementierung dürfen Sie annehmen, dass es eine Funktion $\mathbf{get}(i, j) = P_{i,j}$ gibt, welche einen Knoten mit den gegebenen Koordinaten in $O(1)$ Laufzeit zurückgibt.