Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ulf Leser
E. Angriman, A. Ermshaus, F. Nelles, M. Sänger

Berlin, den 31.05.2021

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 4

Abgabe: Montag, den 14.06.2021, bis 11:10 Uhr über Moodle. Die Übungsblätter sind in Gruppen von zwei (in Ausnahmefällen drei) Personen zu bearbeiten. Die Lösungen sind auf nach Aufgaben getrennten **PDFs** über Moodle abzugeben. Alle Teilaufgaben einer Aufgabe sind in einem PDF hochzuladen. Pro Gruppe genügt es, wenn eine Person die Lösung der Gruppe abgibt. Zur Bewertung wird der zuletzt hochgeladene Stand herangezogen. Vermerken Sie auf allen Abgaben Ihre **Namen**, Ihre **CMS-Benutzernamen** und Ihre **Abgabegruppe (z.B. AG123)** aus Moodle. Benennen Sie die hochgeladenen PDF-Dateien nach dem Schema *A<Aufgabe>-<Person1>-<Person2>.pdf*, bspw. A03-Musterfrau-Beispiel.pdf für Aufgabe 3 von Lisa Musterfrau und Mark Beispiel. Die Auflistung der Namen kann in beliebiger Reihenfolge erfolgen.
Beachten Sie die Informationen im Moodle-Kurs (https://hu.berlin/algodat21).

Konventionen:

- Für ein Array A ist $|A|$ die Länge von A, also die Anzahl der Elemente in A. Die Indizierung aller Arrays beginnt bei 1 (und endet also bei $|A|$). Bitte beginnen Sie die Indizierung der Arrays in Ihren Lösungen auch bei 1.
- Mit der Aufforderung "Analysieren Sie die Laufzeit" ist gemeint, dass Sie eine möglichst gute obere Schranke der (Worst Case) Zeitkomplexität angeben und diese begründen sollen.
- Die Menge der natürlichen Zahlen $\mathbb{N}$ enthält die Zahl 0.

Aufgabe 1 (Fibonacci- und Interpolations-Suche) — 4+6+4=14 Punkte

Gegeben ist das folgende sortierte Array natürlicher Zahlen:

$A =$

2	3	5	7	11	13	17	19

1. Führen Sie einen Schreibtischtest für den Algorithmus *Fibonacci-Search* aus der VL durch, bei dem das Array A nach dem Wert $c = 5$ durchsucht wird. Geben Sie die aktuellen Belegungen der Variablen $fib2$, $fib3$, und m vor jedem Aufruf von Zeile 8 im Pseudocode von Folie 13 an.

2. Die Fibonacci-Zahlen sind wie folgt definiert: $F_1 = 1$, $F_2 = 1$ und $F_k = F_{k-2} + F_{k-1}$ für alle $k > 2$. Zeigen Sie unter Verwendung dieser Definition für Fibonacci-Zahlen, dass für $k \geq 1$ die Beziehung $F_k \geq \frac{1}{3}F_{k+2}$ gilt. *Hinweis*: Eine geeignete Beweismethode ist die vollständige Induktion.

3. Führen Sie einen Schreibtischtest für den Algorithmus *Interpolation-Search* aus der VL durch, bei dem das obige Array A nach dem Wert $c = 3$ durchsucht wird. Geben Sie die aktuellen Belegungen der Variablen f, r, und m vor jedem Aufruf von Zeile 6 im Pseudocode von Folie 8 (rechts) an. Berechnen Sie m wie auf Folie 20 definiert und runden Sie ggfs. m auf natürliche Zahlen ab.

Aufgabe 2 (Implementierung von Suchverfahren) **6+6=12 Punkte**

Implementieren Sie die nachfolgenden zwei Suchverfahren für die Suche eines Werts k in einem aufsteigend sortierten Array A mit Werten vom Typ Integer:

1. *Binäre Suche*: Die binäre Suche ist der aus der VL bekannte Suchalgorithmus. Der Algorithmus nimmt als Eingabe ein sortiertes Array A aus (möglicherweise negativen) ganzen Zahlen und eine ganze Zahl k und gibt einen Index i mit $A[i] = k$ zurück, wenn $k \in A$, sonst -1. Um die volle Punkzahl zu erhalten, soll die Laufzeit Ihrer Lösung in $\mathcal{O}(\log n)$ und der zusätzliche Speicherplatzbedarf in $\mathcal{O}(1)$ liegen. Beachten Sie dabei, dass bei einer rekursiven Lösung auch die rekursiven Funktionsaufrufe Speicherplatz benötigen.

2. *Suche in einem dünnbesetzten Array*: Dieses Suchverfahren soll eine effiziente Suche in Arrays, in denen Elemente auch `null` (im Sinne von "kein Wert vorhanden") sein können, ermöglichen. Dies kann bspw. der Fall sein, wenn in einem sortierten Array nachträglich einige der Elemente gelöscht werden und man aus Effizienzgründen auf das "Zusammenschieben" des Arrays verzichtet hat. Ihr Algorithmus nimmt als Eingabe ein sortiertes Array A von (möglicherweise negativen) ganzen Zahlen, in dem Elemente `null` sein können, und eine ganze Zahl $k \neq$ `null`. Wenn $k \in A$, dann gibt der Algorithmus einen Index i mit $A[i] = k$ zurück, sonst -1. Ein Beispiel für die Implementierung einer solchen Suche finden Sie in dem vorgegebenen Quellcode-Gerüst in der Klasse `LinearSparseSearch`. Um die volle Punktzahl zu erhalten, soll die Laufzeit Ihrer Lösung für Arrays, die keine oder eine konstante Anzahl an `null` Elementen aufweisen, in $\mathcal{O}(\log n)$ und der zusätzliche Speicherbedarf in $\mathcal{O}(1)$ liegen. Ihr Algorithmus darf eine lineare Worst Case Laufzeitkomplexität haben, wenn A eine lineare Anzahl (in Bezug auf n) an `null` Einträgen besitzt.

Nutzen Sie zur Implementierung der Suchverfahren die beiden privaten Klassen `BinarySearch` und `SparseSearch` in der in Moodle bereitgestellten Vorlage `SortedSearch.java`. Sie können beliebige neue Variablen und Hilfsmethoden zur Klasse hinzufügen, dürfen jedoch keine außer den von Java bereitgestellten Standard-Bibliotheken verwenden. Stellen Sie sicher, dass alle Testfälle in der `main`-Methode der Datei `SortedSearch.java` erfolgreich durchlaufen. Achten Sie außerdem auf Randbedingungen und Spezialfälle, die von den Testfällen vielleicht nicht vollständig abgedeckt werden.

Hinweis: Ihr Java-Programm muss auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, sodass die einzelnen Schritte nachvollziehbar sind. Die Abgabe des von Ihnen modifizierten Quellcodes SortedSearch.java erfolgt über Moodle.

Aufgabe 3 (Suchen und Auswahl) **8 + 4 = 12 Punkte**

Angenommen, Sie haben ein *unendlich* langes Array A gegeben, d.h., die Länge des Arrays n kann nicht bestimmt werden (beispielsweise kommen die Werte über ein Netzwerk ohne dass Sie wissen, wann der Datenstrom enden wird). In A kommen nur die Buchstaben **a**, **b** und **c** vor. Die Elemente sind so geordnet, dass zuerst alle **a**s kommen, dann an Position p ein einzelnes **b**, und danach ab Position $p+1$ nur noch **c**s. Sie können annehmen, dass Sie auf alle Positionen in konstanter Zeit zugreifen können (z.B., indem das schon gelesene Array gepuffert wird und das Warten auf eine noch nicht eingetroffene Position nur mit konstanter Arbeit bewertet wird).

1. Geben Sie einen Algorithmus in Pseudocode an, der in $\mathcal{O}(\log p)$ vielen Vergleichen die Position p bestimmt, an der das eine **b** steht.

2. Begründen Sie, warum die Anzahl der Vergleiche Ihres Algorithmus in $\mathcal{O}(\log p)$ liegt.

Aufgabe 4 (Laufzeitanalyse) **3+3+3+3=12 Punkte**

Anton will seine Sammlung alter Liebesbriefe in einem (feuerfesten) Papierkorb verbrennen, der anfangs leer ist. Er tut dies mit Hilfe von zwei Operationen:

`rein`: Anton benötigt eine Sekunde, um einen Liebesbrief in den Papierkorb zu werfen. Es sei angenommen, dass der Papierkorb für eine beliebige Anzahl Liebesbriefe Platz hat.

`brand`: Anton setzt die Liebesbriefe im Papierkorb in Brand. In diesem Fall wartet Anton sicherheitshalber immer, bis alle Liebesbriefe im Papierkorb rückstandslos verbrannt sind. Liegen k Liebesbriefe im Papierkorb, so dauert dies genau k Sekunden.

a) Zeigen Sie zuerst, dass Anton im Rahmen einer konventionellen Worst Case Analyse $\mathcal{O}(n^2)$ Sekunden benötigt, um eine beliebige Folge von $n > 0$ Operationen `rein` und `brand` auszuführen.

Wir wollen nun nun mit Hilfe der Potentialmethode zeigen, dass Anton im Worst Case tatsächlich nur $\mathcal{O}(n)$ Sekunden benötigt, um eine beliebige Folge von $n > 0$ Operationen `rein` und `brand` auszuführen. Im Folgenden bezeichnen wir mit D_0 den leeren Anfangszustand des Papierkorbs und mit D_i den Zustand des Papierkorbs nach Ausführung der ersten i Operationen einer Folge Q von Operationen.

b) Beschreiben Sie zuerst eine geeignete Potentialfunktion, die jedem Zustand D des Papierkorbs ein Potential $\Phi(D)$ zuordnet.

c) Gegeben sei die Folge Q der Operationen {`rein`, `rein`, `rein`, `brand`, `rein`, `brand`} der Länge $n = 6$. Notieren Sie tabellarisch für jede Operation der Folge Q die folgenden drei Werte:

- die realen Kosten c_i (in Sekunden) der i-ten Operation von Q,
- das Potential $\Phi(D_i)$ des Papierkorbs nach Ausführung der i-ten Operation und
- die amortisierten Kosten $d_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$ der i-ten Operation von Q.

d) Zeigen Sie mit Hilfe der Potentialmethode, dass die tatsächlichen Kosten zur Ausführung aller n Operationen der Folge Q in $\mathcal{O}(n)$ liegen, d.h., dass

$$\sum_{i=1}^{n} c_i \in \mathcal{O}(n) .$$

Hinweis: Zeigen Sie dafür zunächst, dass für $1 \leq i \leq n$ die amortisierten Kosten d_i der i-ten Operation aus Q höchstens 2 betragen.