Algorithmen und Datenstrukturen
Sommersemester 2024

Alexandros Paraskevaidis
Elias Gestrich

3. Übungsblatt

Aufgabe 1 - Pipelines

Man kann mittels QuickSelect den Median in O(n) finden und dort seine Pipeline verlegen. Dafür erstellen wir ein Array mit der gleichen Größe wie die Anzahl der y-Koordinaten. Wir suchen uns ein zufälliges Pivot Element aus und ordnen alles kleinere links davon und alles größere rechts davon. Falls unser Pivot Element in der Mitte des Arrays landet sind wir fertig. Sonst führen wenden wir unseren bisherigen Algorithmus auf die linke oder rechte Hälfte je nachdem, ob unser Pivot Element kleiner oder größer ist als das Element in der Mitte. Wir machen solange weiter bis unser Pivot Element in der Mitte des Subarrays ist. (Bei Arrays mit gerader Anzahl von Elementen nehmen wir das Element bei $\lfloor (n+1)/2 \rfloor$)
Die Pipeline legen wir auf der y-Koordinate. Alle y-Anlegekoordinaten oberhalb würden die Länge der unteren Hälfte der Zuleitungen verlängern und alles unterhalb die Länge der oberen Hälfte verlängern.

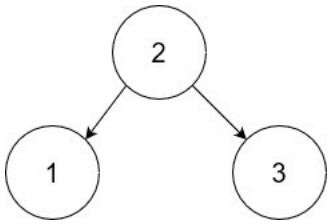
Aufgabe 2 - Vorsortiert

A, B sortiert mit jeweils n Elementen $\rightarrow A\left[\left\lfloor \frac{n}{2} \right\rfloor\right]$ Media von A, $B\left[\left\lfloor \frac{n}{2} \right\rfloor\right]$ Median von $B \rightarrow$ Media liegt zwischen $A\left[\left\lfloor \frac{n}{2} \right\rfloor\right]$ und $B\left[\left\lfloor \frac{n}{2} \right\rfloor\right]$, da sonst mehr als $\frac{n}{2} + \frac{n}{2}$ größer/kleiner - Widerspruch
Wenn $A\left[\left\lfloor \frac{n}{2} \right\rfloor\right] = B\left[\left\lfloor \frac{n}{2} \right\rfloor\right]$, dann haben wir unseren Median schon gefunden. Sonst Œ$A\left[\left\lfloor \frac{n}{2} \right\rfloor\right] < B\left[\left\lfloor \frac{n}{2} \right\rfloor\right]$, der Median liegt also in $A\left[\left\lfloor \frac{n}{2} \right\rfloor, \ldots, n\right]$ oder $B\left[1, \ldots, \left\lfloor \frac{n}{2} \right\rfloor\right]$
Wenn wir $A\left[\left\lfloor \frac{3n}{4} \right\rfloor\right]$ und $B\left[\left\lfloor \frac{n}{2} \right\rfloor\right]$ vergleichen und sie gleich sind dann haben wir den Media gefunden, da $\frac{3n}{4} + \frac{n}{4}$ kleiner/größer.
Wenn $A\left[\left\lfloor \frac{3n}{4} \right\rfloor\right] < B\left[\left\lfloor \frac{n}{4} \right\rfloor\right]$, dann liegt der Median in $A\left[\left\lfloor \frac{3n}{4} \right\rfloor, \ldots, n\right]$ oder $B\left[1, \ldots, \left\lfloor \frac{n}{4} \right\rfloor\right]$, sonst in $A\left[\left\lfloor \frac{n}{2} \right\rfloor, \ldots, \left\lfloor \frac{3n}{4} \right\rfloor\right]$, oder $B\left[\left\lfloor \frac{n}{4} \right\rfloor, \ldots, \left\lfloor \frac{n}{2} \right\rfloor\right]$

Aufgabe 3 - Heaps und Suchbäume

a) 1) So ein Heap ist nicht möglich, da wir immer erst links einfügen müssten aufgrund der notwendigen Linksvollständigkeit für einen Heap. Diese Elemente müssen aber größer sein als ihr Elternknoten, wodurch die Eigenschaft des Binärensuchbaums verletzt wird.

2) Sobald ein kleineres Kind links von einem Elternknoten eingefügt wird, wird die Heapeigenschaft verletzt. Somit ist ein solcher Baum leicht konstruierbar.



3) So ein BST ist nicht umsetztbar, da ein linksvollständiger BST zwangsweise Kinderknoten enthält, die größer sind als ihre Elternknoten, womit die Heap Eigenschaft verletzt wird.

b) Best Case: am wenigsten Vergleiche: Für das $(k+1)$-te Element, sind schon k Elemente im Binärem Suchbaum. D.h. im Best Case $\lfloor \log_2(k) \rfloor$, also $\lfloor \log_2(k) \rfloor$ Vergleiche. Also $\Omega(n \log(n))$.
Würde es nicht balanciert, dann müsste für ein Element, das es unbalanciert macht, mindestens $\lfloor \log_2(n) \rfloor + 1$ Vergleiche gemacht werden. Also ist der Best Case $\Omega(n \log(n))$

Aufgabe 4 - Pfade im Suchbaum

a) 1) Ein solcher BST existiert, da jeder linke Teilbaum nur Elemente enthält, die kleiner sind als der Elternknoten und jeder rechter Teilbaum nur Elemente enthält, die größer als ihr Elternknoten sind. 2) Ein solcher BST kann aus demselben Grund existiert wie in 1) 3) Ein solcher BST ist nicht valide, da bei der Einfügung von Schlüssel 666, der im linken Teilbaum vom Schlüssel 652 auftaucht und somit größer als der Knoten 652 ist, die BST Eigenschaft verletzt. 4) Ein solcher BST existiert, da keine Eigenschaften des BST verletzt werden.

b) Man fängt vom Ende des Arrays an und läuft nach links und merkt sich Minimum und Maximum. Beim Finden eines größeren/kleineren Wertes aktualisiert man entsprechend den aktuellen Max./Min. Wert. Wenn ein Wert auftaucht, der zwischen Maximum und Minimum liegt kann es einen solchen Baum nicht geben, da entsprechend, die Eigenschaft des BST verletzt wird, dass im linken Teilbaum nur größere Elemente vorkommen oder dass im rechten Teilbaum nur kleinere Elemente stehen können.

Algorithmus 1: BSTCheck (int[] A)

```
1 max ← A[k]
2 min ← A[k]
3 for i = k − 1 to A[0] do
4 |   if A[i] <A[i-1] then
5 |   |   min ← A[i]
6 |   else if A[i] >A[i-1] then
7 |   |   max ← A[i]
8 |   else
9 |   |   return false
```