Algorithmen und Datenstrukturen

Übungsblatt 06

th357, jr574, ro46

June 14, 2022

Aufgabe 2

a)

Die Funktion iteriert über jeden Pfad und vergleicht jeden Pfad nach der Anzahl der Knoten. Am Ende nimmt man den längsten Pfad und und macht die Anzahl der Knoten - 1 = Tiefe des Baumes. Laufzeit:
Worstcase:
Ein voll besetzter Binärbaum, bei der jeder Pfad die gleiche länge besitzt. Alle Pfade besitzen die länge n und die Anzahl der Pfade ebenfalls n. Dadurch ist die Laufzeit quadratisch, da man n-mal n Knoten durchgeht, alo n $\cdot n = n^2$
Bestcase:
Es existiert nur ein Pfad der länge n. Laufzeit ist linear mit der Tiefe des Baumes.

b)

Algorithmus: Wir starten beim Wurzelknoten und gehen so lange zum jeweiligen linken Kindknoten, bis der aktuell bertachtete Knoten keinen linken Kindknoten mehr hat (current.left = None). Dann ist current der Knoten mit dem kleinsten Schlüssel im Baum. Wir speichern diesen Schlüssel in einer Liste und löschen den Knoten.
Danach führen wir das selbe Vorgehen für den nun um einen Knoten gekürzten Baum aus. Wir wiederholen diese Schritte k mal (oder so lange, bis keine Knoten mehr in unserem Baum sind).
Danach sind die k kleinsten Schlüssel aufsteigend sortiert in unserer Liste gespeichert.

Laufzeit:
Im worst case besteht unser Baum nur aus einem einzigen Pfad. Den müssen wir dann für jeden der k Schlüssel einmal komplett bis zum tiefsten verbleibenden Schlüssel durchlaufen. Die Laufzeit für das Löschen eines Elements beträgt schon O(Tiefe des aktuellen Baumes), um einen Schlüssel zu finden benötigen wir ebenfalls O(Tiefe des aktuellen

Baumes).
Unsere Gesamtlaufzeit ist somit quadratisch mit der Tiefe des Baumes.

c)

Wir führen zunächst eine Binary Search mit x durch (und merken uns den jeweils aktuellen Vorgängerknoten):

```
current = root
parent = None
u = None
w = None
// Binary Search.
while current is not None and current.key != x:
  parent = current
  if current.key > x:
    current = current.left
  else:
    current = current.right
// If Binary Search found the key, return the node.
if current is not None:
  return current
// Else search the next bigger and next smaller key.
else:
  while u is None or w is None:
    if parent.key < x:
      if u is None:
        u = parent
    else:
      if w is None:
        w = parent
    // If no smaller or no bigger key is in the tree,
    // we will reach the root:
    if parent.parent is None:
      // Then return None for the value we did not find.
      break
    // Move upwards the tree to search for u and w
    parent = parent.parent
  return (u, w)
```

Da der Algorithmus den Baum im ersten Teil (binary search) genau auf einem Pfad traversiert, ist die Laufzeit hierfür linear mit der Länge des Pfades, also im worst case linear mit der Tiefe des Baumes. Das ggf. notwendige Finden von u und w traversiert danach den selben Pfad in umgekehrter Reihenfolge, jedoch nur so lange, bis u und w gefunden sind. Im worst case (u oder w existiert nicht) muss der ganze Pfad bis zur

Wurzel zurückgegangen werden, d.h. auch hierfür beträgt unsere worst case Laufzeit O(Tiefe des Baumes). Insgesamt ist die worst case Laufzeit unseres Algorithmus also O(2*Tiefe des Baumes), also linear mit der Tiefe des Baumes bzw. O(Tiefe des Baumes). Im best case liegt der gesuchte Wert in der Wurzel und wir erhalten eine konstante Laufzeit von 1.