Gruppennummer: 22

Datenstrukturen und Algorithmen, Abgabe 2

Beccard, Vincent, 377979; Braun, Basile 388542; Brungs, Simon, 377281

3. Mai 2018

1 Binäre Bäume

a Beweis: Ein Baum der Höhe h enthält höchstens 2h - 1 innere Knoten

Ein Baum der Höhe h hat maximal $2^{h+1} - 1$ Knoten. Dies ist erreicht, wenn der Baum vollständig ist. Es gilt zudem, dass Ebene d höchstens 2^d Knoten hat. Da die Ebene d die Blätter darstellt, muss diese von der gesamten Anzahl der Knoten abgezogen werden. Dies führt zur Formel $2^{h+1} - 2^d$.
Da der Baum vollständig ist, ist die unterste Ebene d gleich der Höhe h.
$\Rightarrow 2^{h+1} - 2^d = 2^h - 1$
Da von der maximalen Anzahl an gesamt Knoten und Knoten pro Ebene ausgegangen wird folgt, dass $2^h - 1$ die maximale Anzahl an inneren Knoten beschreibt.

b Beweis:

Noch nichts.

2 DAG-Traversierung

3 Abstrakte Datentypen

a

Wenn eine doppelt verkettete Liste benutzt wird, hat jedes Element einen Vorgänger und einen Nachfolger. Zudem gibt ein head-Element und ein tail-Element. Bei isEmpty muss also nur überprüft werden ob der head leer ist. Dies ist unabhängig von der Länge der Eingabe. Bei den anderen Operationen müssen immer mit die ersten bzw. die letzten beiden Element verändert werden, da man weiß welches Element Vorne bzw. Hinten liegt und die Nachfolger bzw. Vorgänger bestimmt sind. durchlaufen werden. Dies hat zur Folge, dass die Länge der Eingabe unabhängig von der Laufzeit ist und somit wie isEmpty in O(1) liegt.

b

Die Operationen könne nicht alle in O(1) ausgeführt werden, wenn ein unbeschränktes Array und ein Zeiger zur Hilfe genommen werden. Dies liegt daran, dass bei enqueueBack und daqueueBack das komplette Array durchlaufen werden muss um das letzte Element zu bearbeiten. Somit müssen diese mindestens in O(n) liegen.

c

Um herauszufinden ob ein Element e schon in einem Set s vorhanden ist muss im worst-case das gesamte Set durchlaufen werden. Somit sind comtains und add höchstens in O(n). In union muss s2 zu s1 hinzugefügt und s1 ausgegeben werden. Sei $n1$ die Länge von s1 und $n2$ die Länge von s2, so muss um schlimmsten Fall n2 mal n1 durchgelaufen werden, da bei jedem Element aus s2 add benutzt wird um zu sichern, dass kein Element zweimal vorkommt. Somit ist die Laufzeit $n1 \cdot n2$ dies ist $\geq n$ welches $n1 + n2$ ist. Da die Laufzeit von union jedoch nur ein Vielfaches von n ist, ist die Laufzeit auch in O(n).
Damit ist die Aussage bewiesen.

Es gibt keine Implementierung, sodass die Operation union des ADT Set nur O(1) benötigt. Dies liegt daran, dass für union für jedes Element aus dem einem Set überprüft werden muss ob es schon in dem anderen Set vorkommt. Somit muss jedes mal max das andere Set durchgelaufen werden. Damit ist die Laufzeit von der Eingabelänge abhängig und kann nicht konstant sein.s

4 Laufzeitanalyse

a Speicherdarf

Sowohl die Worst-Case als auch Best-Case Speicherkomplexität ist k + 1, da egal wie meine Eingabe der Liste l aussieht, legt der Algorithmus einen array mit k Stellen an und die Variable i.

b Laufzeit

Die Best-Case Laufzeit ist 2.
Die Worst-Case Laufzeit ist n.
Die Average-Case Laufzeit ist $A(n) = \sum_{i=1}^{n} 0.5^i$.

c Konstanter Speicherverbrauch im Worst-Case

Beccard, Vincent, 377979; Braun, Basile 388542; Brungs, Simon, 377281;
Nummer der Übungsgruppe: 22

```
Eingabe: Liste l der Laenge n von Zahlen zwischen 1 und k
Ausgabe: Gibt es Duplikate von Zahlen in l


for i := 0 .. l.length -1
  for j := 0 .. i
    if l[i] = l[j]
      return true

return false
```

Die asymptotische Worst-case Laufzeit ist $A(n) = \sum_{i=0}^{n} i$