Informatik II

Übung 9

Giuseppe Accaputo, Felix Friedrich, Patrick Gruntz, Tobias Klenze, Max Rossmannek, David Sidler, Thilo Weghorn

FS 2017

Heutiges Programm

1. Binäre Suchbäume
2. Heaps

Baumknoten in Java

```
public class SearchNode {
  int key;     // Schluessel
  SearchNode left;    // linker Teilbaum
  SearchNode right;   // rechter Teilbaum

  // Konstruktor: Knoten ohne Nachfolger
  SearchNode(int k){
    key = k;
    left = right = null;
  }
}
```

Suchbaum und Suchen in Java

```
public class SearchTree {
  SearchNode root = null; // Wurzelknoten

  // Gibt Knoten mit Schluessel k zurueck.
  // Wenn nicht existiert: null.
  public SearchNode Search (int k){
    SearchNode n = root;
    while (n != null && n.key != k){
      if (k < n.key) n = n.left;
      else n = n.right;
    }
    return n;
  }
  ... // Einfuegen, Loeschen
}
```

Rekursion auf Bäumen

Aufgabe: Schreiben Sie die (rekursive) Funktion `int height(SearchNode n)` zur Bestimmung der Höhe eines (Teil-)Suchbaumes.

[Starten Sie hier: `https://codeboard.io/projects/48851`]

Knoten Einfügen in Java

```
public SearchNode Insert (int k) {
  if (root == null) { return root = new SearchNode(k); }
  SearchNode t=root;
  while (true) {
    if (k == t.key) { return null; }
    if (k < t.key) {
      if (t.left == null) { return t.left = new SearchNode(k); }
      else { t = t.left; }
    }
    else { // k > t.key
      if (t.right == null) { return t.right = new SearchNode(k); }
      else { t = t.right; }
    }
  }
}
```

Knoten Löschen in Java

```
public void Delete (int k) {
  SearchNode n = root;
  if (n != null && n.key == k) {
    root = SymmetricDesc(root);
  } else {
    while (n != null) {
      if (n.left != null && k == n.left.key) {
        n.left = SymmetricDesc(n.left); return;
      } else if (n.right != null && k == n.right.key) {
        n.right = SymmetricDesc(n.right); return;
      } else if (k < n.key) { n = n.left;
      } else { n = n.right; }
    }
  }
}
```

SymmetricDesc in Java

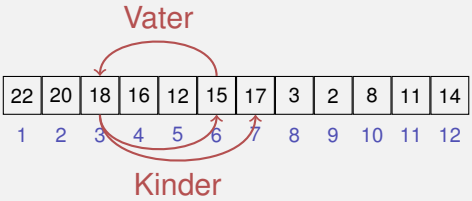
```
public SearchNode SymmetricDesc(SearchNode node) {
  if (node.left == null) { return node.right; }
  if (node.right == null) { return node.left; }
  SearchNode n = node;
  SearchNode parent = null;
  n = n.right;
  while (n.left != null) { parent = n; n = n.left; }
  if (parent != null) {
    parent.left = n.right;
    n.left = node.left;
    n.right = node.right;
  } else { n.left = node.left; }
  return n;
}
```

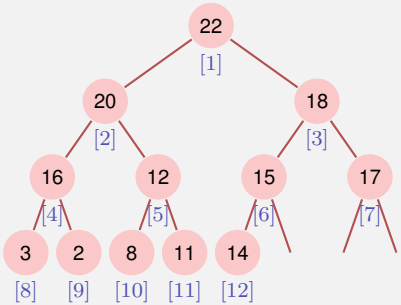
Dieser Algorithmus gibt den symmetrischen Nachfolger zurück. Aber tut noch mehr: er behandelt auch die Fälle mit einem oder keinem Nachfolger. Ausserdem entfernt er den Symmetrischen Nachfolger und setzt dessen Nachfolgeknoten.

Heap und Array

Baum $\rightarrow$ Array:

- Kinder$(i) = \{2i, 2i+1\}$
- Vater$(i) = \lfloor i/2 \rfloor$

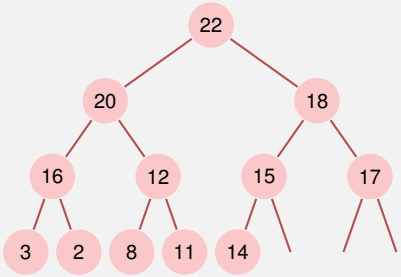




Abhängig von Startindex![1]

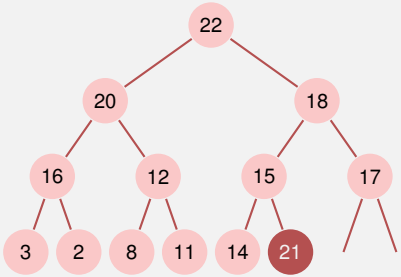
[1] Für Arrays, die bei 0 beginnen: $\{2i, 2i+1\} \rightarrow \{2i+1, 2i+2\}$, $\lfloor i/2 \rfloor \rightarrow \lfloor (i-1)/2 \rfloor$

Einfügen



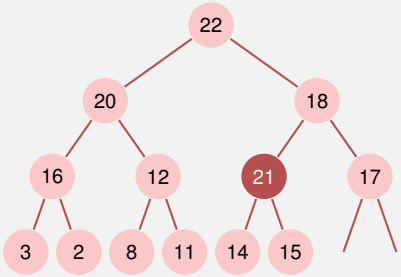
Einfügen

- Füge neues Element an erste freie Stelle ein. Verletzt Heap Eigenschaft potentiell.



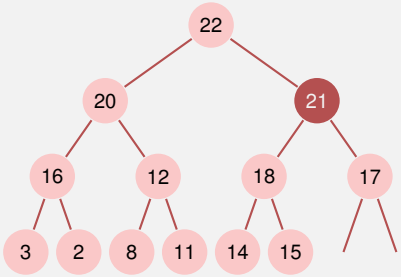
Einfügen

- Füge neues Element an erste freie Stelle ein. Verletzt Heap Eigenschaft potentiell.
- Stelle Heap Eigenschaft wieder her: Sukzessives Aufsteigen.



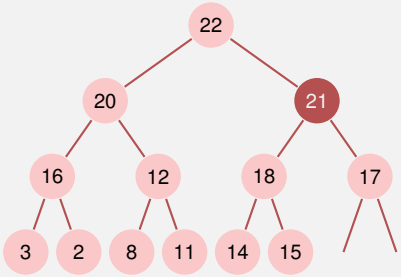
Einfügen

- Füge neues Element an erste freie Stelle ein. Verletzt Heap Eigenschaft potentiell.
- Stelle Heap Eigenschaft wieder her: Sukzessives Aufsteigen.

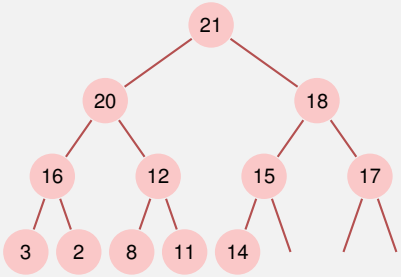


Einfügen

- Füge neues Element an erste freie Stelle ein. Verletzt Heap Eigenschaft potentiell.
- Stelle Heap Eigenschaft wieder her: Sukzessives Aufsteigen.
- Anzahl Operationen im schlechtesten Fall: $\mathcal{O}(\log n)$

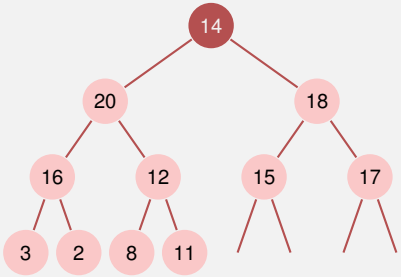


Maximum entfernen



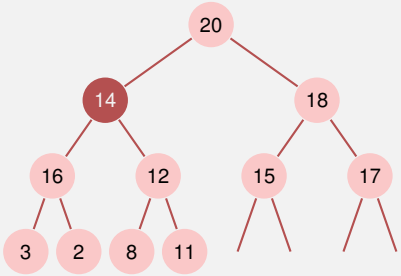
Maximum entfernen

- Ersetze das Maximum durch das unterste rechte Element.



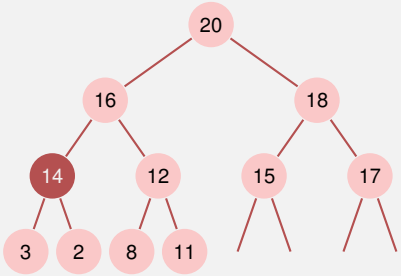
Maximum entfernen

- Ersetze das Maximum durch das unterste rechte Element.
- Stelle Heap Eigenschaft wieder her: Sukzessives Absinken (in Richtung des grösseren Kindes).



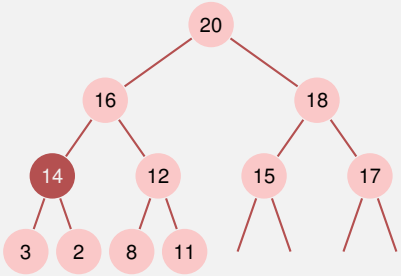
Maximum entfernen

- Ersetze das Maximum durch das unterste rechte Element.
- Stelle Heap Eigenschaft wieder her: Sukzessives Absinken (in Richtung des grösseren Kindes).



Maximum entfernen

- Ersetze das Maximum durch das unterste rechte Element.
- Stelle Heap Eigenschaft wieder her: Sukzessives Absinken (in Richtung des grösseren Kindes).
- Anzahl Operationen im schlechtesten Fall: $\mathcal{O}(\log n)$



Aufgabe

Implementieren Sie die Operation `Changekey(int position, double key)`: Der Schlüssel an einem Eintrag wird geändert.[2] Überlegen Sie zuerst konzeptuell, was gemacht werden soll.

[Startpunkt: `https://codeboard.io/projects/48852`]

Welche Komplexität hat Ihre Lösung?

[2] Diese Operation heisst im Zusammenhang mit Min-Heaps DecreaseKey und wird typischerweise nur für die Verkleinerung des Schlüssels benötig. Hier sollen aber beide Richtungen implementiert werden.

Fragen oder Anregungen?