Algorithmen und Datenstrukturen (für ESE) WS 2010 / 2011

Vorlesung 6, Montag, 29. November 2010

(Prioritätswarteschlangen)

Prof. Dr. Hannah Bast
Lehrstuhl für Algorithmen und Datenstrukturen
Institut für Informatik
Universität Freiburg

UNI FREIBURG

Blick über die Vorlesung heute

UNI FREIBURG

- Organisatorisches
  - Morgen keine Übungsgruppe, dafür Fragetermin später
  - Ihre Erfahrungen mit dem 5. Übungsblatt
- Prioritätswarteschlangen (priority queues)
  - Ebenfalls eine Datenstruktur, die man sehr häufig braucht
  - Wir werden ein Beispiel sehen, wo man sie braucht
  - Und dann erklären, wie man selber eine bauen kann
  - Die Übungsaufgabe ist es dann, auf der Grundlage dieser Erklärungen, eine Klasse PriorityQueue zu schreiben
  - Am Ende noch kurz etwas zur std::priority_queue

Morgen keine Übungsgruppe

- Grund
  - Das letzte Mal kam nur eine Person, der Bedarf scheint sich also in engen Grenzen zu halten
- Stattdessen
  - Fragestunde mit einem Tutor später in der Woche
  - Schreiben Sie eine Mail an Sebastian Sester (Adresse siehe Wiki) wenn Sie Bedarf haben, mit Terminwunsch
  - Termin wird dann im Laufe der Woche bekannt gegeben

Ihre Erfahrungen mit dem 5. Ü-Blatt

UNI FREIBURG

- Zusammenfassung von Ihrem Feedback
  - Blatt war ok, aber wieder etwas schwieriger als das letzte
  - Fehlende Programmierpraxis macht es bei einigen langwierig
  - Die Mathe-Option (Option 1) haben nur wenige gemacht

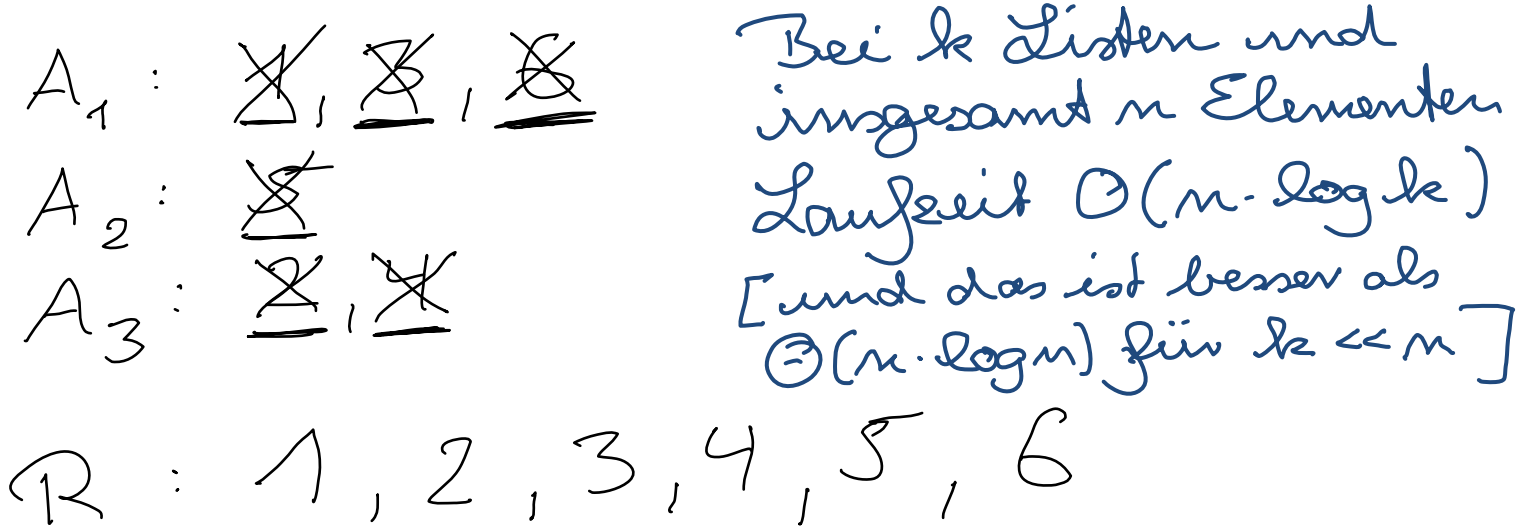
Prioritätswarteschlangen

Transitiv: $x < y \wedge y < z \Rightarrow x < z$
Antisymm.: $x < y \Rightarrow \neg(y < x)$

- Definition
  - Eine **Prioritätswarteschlange** (PW) speichert eine Menge von (key, value) Paaren (wie ein assoziatives Array auch)
  - Es gibt eine transitive, antisymm. Ordnung **<** auf den Keys
    - Bei uns sind die Keys immer Zahlen mit dem normalen **<**
  - Die PW unterstützt auf dieser Menge folgende Operationen
    - getMin: liefert das Paar mit dem kleinsten Key
    - deleteMin: entferne das Paar mit dem kleinsten Key
    - insert(key, value): füge das gegebene Paar ein
  - Bemerkung: mehrere Paare mit demselben Key möglich; gibt es mehrere mit dem kleinsten Key, gibt getMin irgendeins davon zurück (und deleteMin löscht eben das)

PWs — Anwendungen 1/2

- Wo braucht man PWs? Ein Beispiel
  - Berechnung der Vereinigungsmenge von k sortierten Listen (sogenannter multi-way merge oder k-way merge)
  - Die implementieren wir jetzt mit unserer eigenen PriorityQueue (die Sie für das 6. Übungsblatt implementieren sollen)
  - Vorab noch einmal die Grundidee des k-way merge:



PWs — Anwendungen 2/2

- Viele weitere Anwendungen
  - Zum Beispiel für Dijkstra's Algorithmus zur Berechnung kürzester Wege → spätere Vorlesung
  - Unter anderem kann man damit auch einfach sortieren:

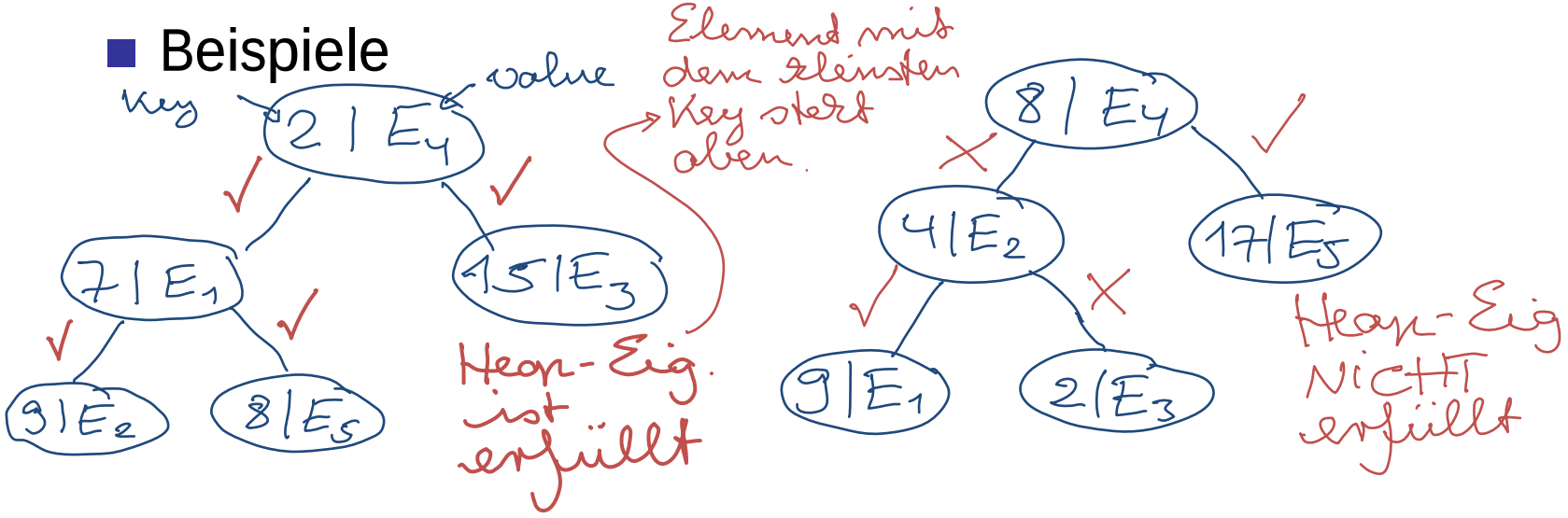
```
void pqSort(vector<int>& A)
{
  PriorityQueue<int> pq;
  for(size_t i=0; i<A.size(); i++)
    pq.insert(A[i], EGAL);
  for(size_t i=0; i<A.size(); i++)
    A[i] = pq.getMin.first;
    pq.deleteMin();
}
```

geht in $O(n \log n)$ Zeit → spätere Folie

PWs — Realisierung 1/4

$x \leq y :\Leftrightarrow \neg(y < x)$

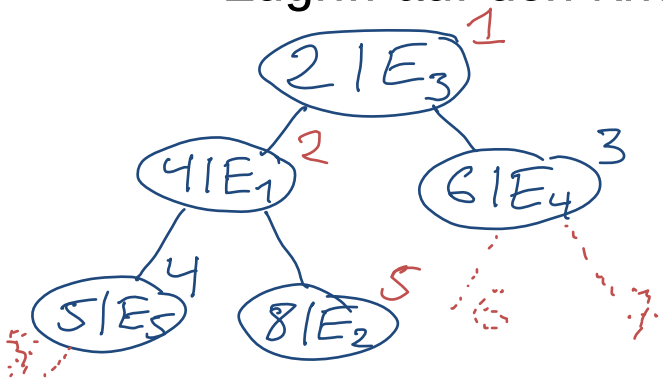
- Grundidee
  - Elemente (Paare) in einem **binären Baum** speichern
    - Ein binärer Baum ist ein Baum, bei dem jeder Knoten höchtens zwei Kinder hat
  - Es gilt die sogenannte **Heap-Eigenschaft**
    - Der Key eines Knotens ist **≤** die Keys seiner Kinder
  - Entsprechend nennt man das ganze **binären Heap**
- Beispiele

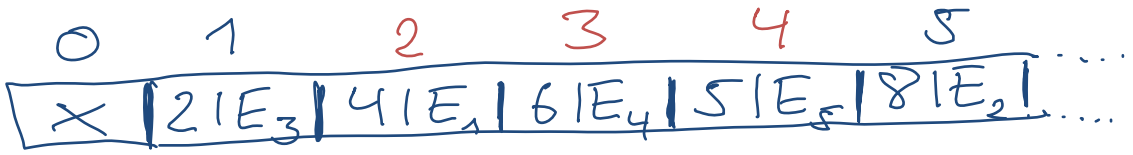


PWs — Realisierung 2/4

- Wie speichert man einen binären Heap
  - Wir numerieren die Knoten von oben nach unten und links nach rechts durch, beginnend mit 1
  - Dann sind die Kinder von Knoten i die Knoten 2i und 2i + 1
  - Und der Elternknoten von einem Knoten i ist floor(i/2)
  - Wir können die Paare dann einfach in einem normalen Array speichern:

    vector<pair<KeyType, ValueType> > _heap;
  - Zugriff auf den Knoten i einfach mit _heap[i]





PWs — Realisierung 3/4

- Wie erhält man die Heapeigenschaft?
  - Beim insert von einem neuen Element, fügen wir es einfach am Ende des Arrays ein

    ```
    _heap.push_back(make_pair(key, value));
    ```

  - Nach deleteMin setzen wir einfach das letzte Element an die erste Stelle

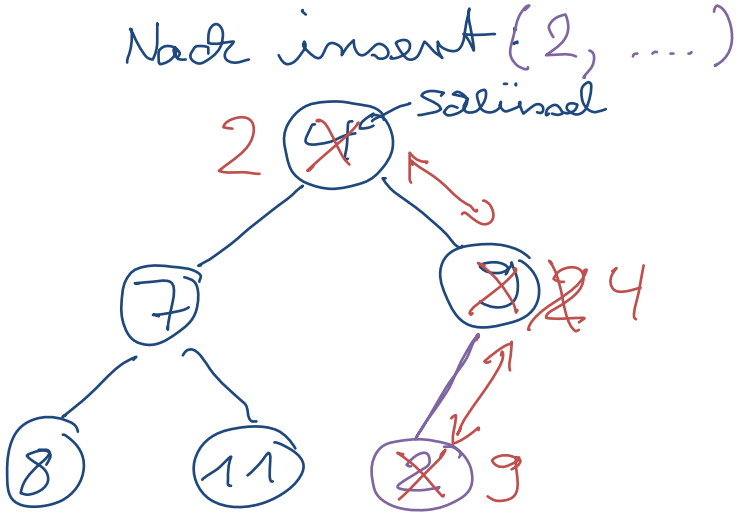
    ```
    _heap[1] = _heap[_size];
    _heap.resize(_heap.size() -1);
    _size--;
    ```

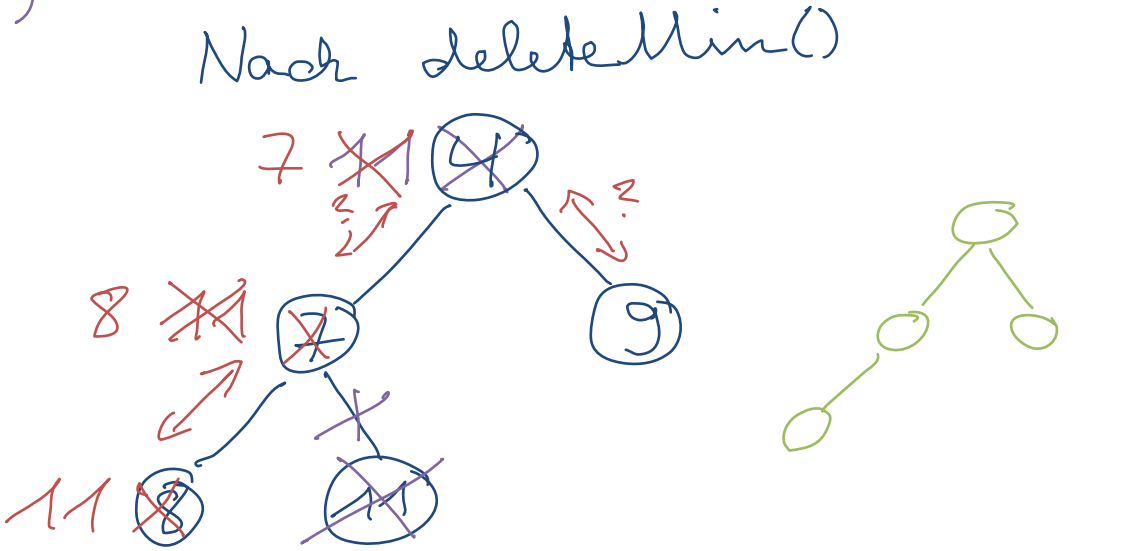
  - In beiden Fällen ist danach die Heapeigenschaft wahrscheinlich verletzt
  - Wie stellen wir Sie dann (effizient) wieder her?

PWs — Realisierung 4/4

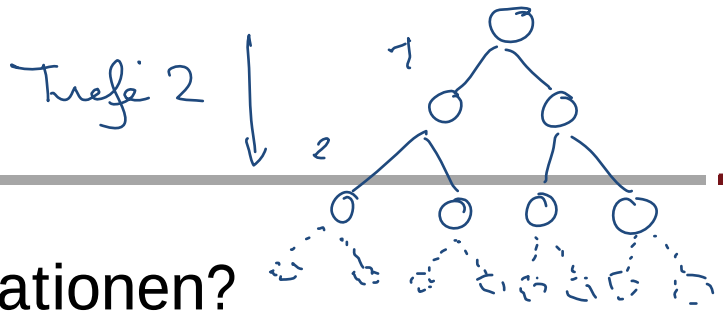
- Wiederherstellen der Heapeigenschaft
  - Erklärung anhand von Beispielen:

Ich lasse hier mal die values ($E_1, E_2, \ldots$) weg





PWs — Komplexität



- Wie teuer sind die PW-Operationen?
  - Vorweg: ein vollständiger binärer Baum der Tiefe d hat $2^{d+1} - 1$ Knoten (zum Beispiel: Tiefe 2 → 7 Knoten)
  - Sei n die Anzahl der Elemente in der PW, dann ist die Anzahl der Elemente auf einem Pfad von einem beliebigen Element zur Wurzel O(log n)
  - Damit ist klar:
    - getMin hat Kosten O(1)
    - deleteMin hat Kosten O(log n)
    - insert hat Kosten O(log n)
  - Bemerkung: **Fibonacci-Heaps** schaffen getMin, deleteMin, insert in O(1), O(log n), O(1), sind aber deutlich komplizierter

Prioritätswarteschlangen in der STL

UNI FREIBURG

- Im Prinzip dasselbe, aber etwas anderes Interface
  - Element-Typ unterscheidet nicht zwischen Key und Value

    std::priority_queue<T> pq;
  - Es wird die Ordnung **>** auf diesem Typ genommen, d.h. es wird das **größte** und nicht das kleinste Element geliefert
  - Man kann sich aber eine beliebige Ordnung definieren, das ist allerdings etwas tricky → siehe Codebeispiel
  - Außerdem heißen die Operationen anders
    - getMin heißt top (und liefert das größte Element)
    - deleteMin heißt pop (und entfernt das größte Element)
    - insert heißt push

Literatur / Links

- Prioritätswarteschlangen
  - In Mehlhorn/Sanders:

    6 Priority Queues [einfache und fortgeschrittenere Varianten]
  - In Cormen/Leiserson/Rivest

    20 Binomial Heaps [gleich die fortgeschrittenere Variante]
  - In Wikipedia

    http://de.wikipedia.org/wiki/Vorrangwarteschlange

    http://en.wikipedia.org/wiki/Priority_queue
  - In C++ und in Java

    http://www.sgi.com/tech/stl/priority_queue.html

    http://download.oracle.com/javase/1.5.0/docs/api/java/util/PriorityQueue.html