Algorithmen / Algorithms II

Peter Sanders

Exercise:
Daniel Seemaier, Tobias Heuer

Institute of Theoretical Informatics

Web:
http://algo2.iti.kit.edu/AlgorithmenII_WS20.php
2 Advanced Data Structures

Here using the example of priority queues.

Further examples:

- Monotone integer priority queues  chapter: shortest paths
- Perfect Hashing  chapter: randomized algorithms
- Search trees with advanced operations  see book
- External priority queues  chapter: external algorithms
- Geometric data structures  chapter: geom. algorithms
2.1 Addressable Priority Queues – Operations

Procedure \textbf{build}(\{e_1, \ldots, e_n\}) \quad M := \{e_1, \ldots, e_n\}

Function \textbf{size} \quad \textbf{return} \quad |M|

Procedure \textbf{insert}(e) \quad M := M \cup \{e\}

Function \textbf{min} \quad \textbf{return} \quad \min M

Function \textbf{deleteMin} \quad e := \min M; \quad M := M \setminus \{e\}; \quad \textbf{return} \quad e

Function \textbf{remove}(h : \text{Handle}) \quad e := h; \quad M := M \setminus \{e\}; \quad \textbf{return} \quad e

Procedure \textbf{decreaseKey}(h : \text{Handle}, k : \text{Key}) \quad \textbf{assert} \quad \text{key}(h) \geq k; \quad \text{key}(h) := k

Procedure \textbf{merge}(M') \quad M := M \cup M'

Addressable Priority queues – Use Cases

- Dijkstra’s algorithm shortest paths
- Jarník-Prim algorithm for minimum spanning trees
- Here: hierarchy construction for route planning
- Here: graph partitioning
- Here: disk scheduling

In general:
Greedy algorithms, where priorities change (within limits).
Basic Data Structure

A forest of heap-ordered trees

```
  minPtr
    d   a   e
      g   f   c   b
           h   i
```

Generalization of binary heaps:

- Tree $\rightarrow$ forest
- Binary $\rightarrow$ arbitrary node-degrees
Processing Forests

Cut:

Link:

union(a, b): link(min(a, b), max(a, b))
Pairing Heaps

[Fredman Sedgewick Sleator Tarjan 1986]

**Procedure** `insertItem(h : Handle)`

`newTree(h)`

**Procedure** `newTree(h : Handle)`

`forest := forest ∪ {h}`

`if *h < min then minPtr := h`

Attention: Simple implementation from the 1. English edition. Further editions differ (e.g. the German edition).
Pairing Heaps

**Procedure** decreaseKey$(h : \text{Handle}, k : \text{Key})$

key$(h) := k$

if $h$ is not a root then cut$(h)$

else update minPtr

**Procedure** cut$(h : \text{Handle})$

remove the subtree rooted at $h$

newTree$(h)$
Pairing Heaps

**Function** deleteMin : Handle

\[
m := \text{minPtr} \\
\text{forest} := \text{forest} \setminus \{m\} \\
\text{foreach child } h \text{ of } m \text{ do newTree}(h) \\
\text{perform pair-wise union operations on the roots in forest} \\
\text{update minPtr} \\
\text{return } m
\]
Pairing Heaps

Procedure $\text{merge}(o : \text{AdressablePQ})$

if $*\text{minPtr} > *(o.\text{minPtr})$ then $\text{minPtr} := o.\text{minPtr}$

$\text{forest} := \text{forest} \cup o.\text{forest}$

$o.\text{forest} := \emptyset$
Pairing Heaps – Representation

Roots: in a doubly linked list

Tree items:
- one child
- left sibling or parent
- right sibling
- data

(left sibling, right sibling, data, parent)
Pairing Heaps – Analysis

insert, merge: \(O(1)\)

deleteMin, remove: \(O(\log n)\) amortized

decreaseKey: unknown! \(O(\log \log n) \leq T \leq O(\log n)\) amortized, but fast in practice.

Proofs: not here.
Fibonacci Heaps [Fredman Tarjan 1987]

**Rank:** Save the number of (immediate) children.

**Union-by-rank:** Only call union on roots with the same rank.

**Mark:** Mark nodes that have lost a child.

**Cascading cuts:** Cut at marked nodes
   (i.e. nodes that have lost two children)

**Theorem:** Amortised complexity $O(\log n)$ for deleteMin and remove $O(1)$ for all other operations.

(i.e. total time $= O(o + d \log n)$ if $d = \#\text{deleteMin}, o = \#\text{otherOps}, n = \max |M|$)
Fibonacci Heaps – Representation

**Roots:** in a doubly linked list

(and a temporary array for deleteMin)

**Rree items:**

```
+------------------+
| data     | rank | mark |
+----------+-------+-------|
| left sibling |
+----------+-------+-------|
| parent |
+----------+-------+-------|
| one child |
+----------+-------+-------|
| right sibling |
+------------------+
```

**insert, merge:** as before, in time $O(1)$
**deleteMin with Union-by-Rank**

**Function** deleteMin : Handle

\[ m := \text{minPtr} \]

forest := forest \ \{m\}

**foreach** child \( h \) of \( m \) **do** newTree(\( h \))

**while** \( \exists a, b \in \text{forest} : \text{rank}(a) = \text{rank}(b) \) **do**

\[ \text{union}(a, b) \]  \( // \) increments rank of surviving root

update minPtr

**return** \( m \)
Fast Union-by-Rank

An array that is addressed by the rank.
Execute link until a free entry is found.

Analysis: Time $O(\#\text{unions} + |\text{forest}|)$
Amortised Analysis for deleteMin

\[
\text{maxRank} := \max_{a \in \text{forest}} \text{rank}(a) \text{ (after)}
\]

Lemma: \( T_{\text{deleteMin}} = O(\text{maxRank}) \)

Proof: Using the accounting method. One token per Root

\[ \text{rank} (\text{minPtr}) \leq \text{maxRank} \]

\( \Rightarrow \) costs of \( O(\text{maxRank}) \) for newTrees and a new token.

Union-by-rank: token pays for

\( \square \) union operations (a token becomes free) and

\( \square \) iterating through roots (old and new).

At the end there are \( \leq \text{maxRank} \) roots.
Why is $\text{maxRank}$ logarithmic? – Binomial Trees

$2^k + 1 \times \text{insert}, 1 \times \text{deleteMin} \sim \text{rank } k$

[Vuillemin 1978] PQ (only) with binomial trees, $T_{\text{decreaseKey}} = O(\log n)$. 

Problem: Cuts can lead to high ranking trees.
Cascading Cuts

**Procedure** decreaseKey($h$ : Handle, $k$ : Key)

key($h$) := $k$

cascadingCut($h$)

**Procedure** cascadingCut($h$)

if $h$ is not a root then

$p$ := parent($h$)

unmark $h$

cut($h$)

if $p$ is marked then

cascadingCut($p$)

else mark $p$

We will show: cascading cuts keep maxRank logarithmic
Lemma: decreaseKey has amortised complexity $O(1)$.

Accounting Method: ($\approx 1$ Token per cut or union)

1 token per root

2 tokens for every marked node

Looking at decreaseKey with $k$ consecutive marked predecessors:

2$k$ token becomes free (nodes become unmarked)

2 token needed for new marks

$k+1$ tokens needed for the new roots

$k+1$ tokens pay for the cuts

Thus, there remains a cost of 4 tokens $+O(1)$ time for decreaseKey
Here is where Mr. Fibonacci comes in.

\[ F_i := \begin{cases} 
0 & \text{für } i = 0 \\
1 & \text{für } i = 1 \\
F_{i-2} + F_{i-1} & \text{else}
\end{cases} \]

Known: \( F_{i+1} \geq ((1 + \sqrt{5})/2)^i \geq 1.618^i \) for all \( i \geq 0 \).

We show:

A subtree with root \( v \) and \( \text{rank}(v) = i \) contains \( \geq F_{i+2} \) elements.

\( \Rightarrow \)

logarithmic time for deleteMin.
Proof:

Looking at the moment when the $j$-th child $w_j$ of $v$ was added:

- $w_j$ and $v$ had the same rank $\geq j - 1$ (since $v$ already had $j - 1$ children)
- rank$(w_j)$ was reduced by at most one (cascading cuts)

$\Rightarrow$ rank$(w_j) \geq j - 2$ and rank$(v) \geq j - 1$

$S_i :=$ lower bound for the # of nodes whose root has rank $i$:

$S_0 = 1$

$S_1 = 2$

$S_i \geq 1 + 1 + S_0 + S_1 + \cdots + S_{i-2}$

for $i \geq 2$

This recurrence has the solution $S_i \geq F_{i+2}$
Addressable Priority Queues – More

- Lower bound $\Omega (\log n)$ for deleteMin (comparison based)
  
  Proof: exercise

- Worst case Bounds: not here

- Monotone PQs with integer keys (stay tuned)

Open Problems:

Analysis of pairing heaps (simplification of Fibonacci Heaps)
Recap Data Structures

- In this lecture, we focused on the example of priority queues (see shortest path algorithms and external algorithms).

- Heap concept can take you far.

- Sibling-pointers can be used to represent arbitrary trees with a constant number of pointers per item.

- Fibonacci heaps – a non-trivial example for amortised analysis