### 8.2 Binomial Heaps

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binary Heap</th>
<th>BST</th>
<th>Binomial Heap</th>
<th>Fibonacci Heap*</th>
</tr>
</thead>
<tbody>
<tr>
<td>build</td>
<td>$n$</td>
<td>$n \log n$</td>
<td>$n \log n$</td>
<td>$n$</td>
</tr>
<tr>
<td>minimum</td>
<td>$1$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$1$</td>
</tr>
<tr>
<td>is-empty</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>insert</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$1$</td>
</tr>
<tr>
<td>delete</td>
<td>$\log n^{**}$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>delete-min</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
</tr>
<tr>
<td>decrease-key</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$\log n$</td>
<td>$1$</td>
</tr>
<tr>
<td>merge</td>
<td>$n$</td>
<td>$n \log n$</td>
<td>$\log n$</td>
<td>$1$</td>
</tr>
</tbody>
</table>
Binomial Trees

8.2 Binomial Heaps
Binomial Trees

Properties of Binomial Trees

- $B_k$ has $2^k$ nodes.
- $B_k$ has height $k$.
- The root of $B_k$ has degree $k$.
- $B_k$ has $\binom{k}{\ell}$ nodes on level $\ell$.
- Deleting the root of $B_k$ gives trees $B_0, B_1, \ldots, B_{k-1}$.
Deleting the root of $B_5$ leaves sub-trees $B_4$, $B_3$, $B_2$, $B_1$, and $B_0$. 
Deleting the leaf furthest from the root (in $B_5$) leaves a path that connects the roots of sub-trees $B_4$, $B_3$, $B_2$, $B_1$, and $B_0$. 
The number of nodes on level $\ell$ in tree $B_k$ is therefore

$$\binom{k-1}{\ell-1} + \binom{k-1}{\ell} = \binom{k}{\ell}$$
The binomial tree $B_k$ is a sub-graph of the hypercube $H_k$.

The parent of a node with label $b_n, \ldots, b_1, b_0$ is obtained by setting the least significant 1-bit to 0.

The $\ell$-th level contains nodes that have $\ell$ 1’s in their label.
How do we implement trees with non-constant degree?

- The children of a node are arranged in a circular linked list.
- A child-pointer points to an arbitrary node within the list.
- A parent-pointer points to the parent node.
- Pointers $x.\text{left}$ and $x.\text{right}$ point to the left and right sibling of $x$ (if $x$ does not have siblings then $x.\text{left} = x.\text{right} = x$).
8.2 Binomial Heaps

- Given a pointer to a node \( x \) we can splice out the sub-tree rooted at \( x \) in constant time.
- We can add a child-tree \( T \) to a node \( x \) in constant time if we are given a pointer to \( x \) and a pointer to the root of \( T \).
In a binomial heap the keys are arranged in a collection of binomial trees.

Every tree fulfills the heap-property

There is at most one tree for every dimension/order. For example the above heap contains trees $B_0$, $B_1$, and $B_4$. 
Given the number $n$ of keys to be stored in a binomial heap we can deduce the binomial trees that will be contained in the collection.

Let $B_{k_1}, B_{k_2}, B_{k_3}, k_i < k_{i+1}$ denote the binomial trees in the collection and recall that every tree may be contained at most once.

Then $n = \sum_i 2^{k_i}$ must hold. But since the $k_i$ are all distinct this means that the $k_i$ define the non-zero bit-positions in the binary representation of $n$. 
Binomial Heap

Properties of a heap with $n$ keys:

- Let $n = b_d b_{d-1} \ldots b_0$ denote binary representation of $n$.
- The heap contains tree $B_i$ iff $b_i = 1$.
- Hence, at most $\lceil \log n \rceil + 1$ trees.
- The minimum must be contained in one of the roots.
- The height of the largest tree is at most $\lceil \log n \rceil$.
- The trees are stored in a single-linked list; ordered by dimension/size.
The merge-operation is instrumental for binomial heaps.

A merge is easy if we have two heaps with different binomial trees. We can simply merge the tree-lists.

Otherwise, we cannot do this because the merged heap is not allowed to contain two trees of the same order.

Merging two trees of the same size: Add the tree with larger root-value as a child to the other tree.

For more trees the technique is analogous to binary addition.
$S_1. \text{merge}(S_2)$:

- Analogous to binary addition.
- Time is proportional to the number of trees in both heaps.
- Time: $\Theta(\log n)$.
All other operations can be reduced to \texttt{merge}().

\textbf{S. insert($x$):}

- Create a new heap $S'$ that contains just the element $x$.
- Execute \texttt{S.merge($S'$)}.
- Time: $\mathcal{O}(\log n)$. 
8.2 Binomial Heaps

\textbf{S. minimum():}

\begin{itemize}
  \item Find the minimum key-value among all roots.
  \item Time: $\Theta(\log n)$.
\end{itemize}
8.2 Binomial Heaps

*S. delete-min():*

- Find the minimum key-value among all roots.
- Remove the corresponding tree $T_{\text{min}}$ from the heap.
- Create a new heap $S'$ that contains the trees obtained from $T_{\text{min}}$ after deleting the root (note that these are just $O(\log n)$ trees).
- Compute $S.\text{merge}(S')$.
- Time: $\Theta(\log n)$.
8.2 Binomial Heaps

**S. decrease-key(handle \( h \)):**

- Decrease the key of the element pointed to by \( h \).
- Bubble the element up in the tree until the heap property is fulfilled.
- Time: \( \mathcal{O}(\log n) \) since the trees have height \( \mathcal{O}(\log n) \).
8.2 Binomial Heaps

\[ S. \text{ delete(handle } h): \]

\begin{itemize}
  \item Execute \texttt{S. decrease-key}(h, \(-\infty\)).
  \item Execute \texttt{S. delete-min}().
  \item Time: \(\mathcal{O}(\log n)\).
\end{itemize}