Fibonacci Heap

Thanks to Sartaj Sahni for the original version of the slides

Fibonacci heaps

• Similar to binomial heaps, consists of a collection of trees, each arranged in a heap-order (each node is smaller than each of its children)

• Unlike binomial heaps, can have many trees of the same cardinality, and a tree does not have to have exactly $2^i$ nodes.

• Main idea – laziness is welcomed. Try to postpone doing the hard work, until no other solution works.

Node Structure

• Very similar to binomial heaps

• Each node $v$ stores its
  • degree,
  • $a$ points to its parent,
  • $a$ points to a child,
  • data,
  • Pointers to left and right sibling used for circular doubly linked list of siblings, called the sibling list.

More – in next slide

Priority Queues

<table>
<thead>
<tr>
<th>Operation</th>
<th>Make-Heap</th>
<th>Find-Min</th>
<th>Delete-Min</th>
<th>Decrease-Key</th>
<th>Delete</th>
<th>Is-Empty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>1</td>
<td>$\log N$</td>
<td>$N$</td>
<td>$1$</td>
<td>$N$</td>
<td>$1$</td>
</tr>
</tbody>
</table>

General Structure

• Very similar to Binomial heaps

• Main structure: A collection of trees, each in a heap-order.

• All root are stored in a doubly connected list, called the roots-list.

• Every node points to one of its childrens. All the children are stored in a doubly connected list, called the sibling-list.

• A pointer $\text{min}(H)$ always points to the min element.

Node Structure

• Each node $v$ stores a flag $\text{ChildCut}$ (not existing in binomial heaps)
  • $\text{True}$ only if $v$ is not a root, and $v$ has lost a single child since became a child of its current parent.
  • We say that $v$ is marked in this case.

• Will see: $\text{Extract_Min}$ is the only operation that makes one node a child of another, and then flag might change.

• Flag is undefined for a root node (not used)
**Fibonacci Heap Representation**

**Potential Function**

Some nodes would be marked (to be explained later)
We use the potential functions for the heap $H$

$$\Phi(H) = t(H) + 2 \cdot m(H)$$

Where $t(H)$ is the number of trees in $H$
And $m(H)$ is the number of marked nodes in $H$.

**Insert($x$)**

Create a new tree consisting of a single node $v$ whose key is $x$.
Add $v$ to the roots list. (always unmarked)

Actual time $w_i$ needed for the operation is 1

Number of trees increased by 1.

Changes in potential function

$$\Phi(H') - \Phi(H) = \Delta \Phi(H) = t(H') - t(H) + 2 \cdot m(H') - 2 \cdot m(H) = 1$$

So the amortized work $a_i = w_i + \Delta \Phi(H) = 2$

**DecreaseKey($\text{theNode}$, $\text{theAmount}$)**

If $\text{theNode}$ is not a root and new key < parent key,
remove subtree rooted at $\text{theNode}$ from its sibling list.
Insert $\text{theNode}$ into roots-list.
Perform cascading cut from parent($\text{theNode}$) (described later)

**Cascading Cut**

- When $\text{theNode}$ is cut out of its sibling list in a decrease key operation, follow path from parent of $\text{theNode}$ upward toward the root.
- Encountered nodes (other than root) with ChildCut = true are cut from their sibling lists and inserted into roots-list.
- Stop at first node with ChildCut = false.
- For this node, set ChildCut = true. (since it just lost exactly one child)

- In other words, if a node lost two children since it became a child, it must move itself from the the parent to the roots-list.

The result the tree rooted at 0 used to be a subtree of 1.

Convince yourself that all pointers modifications are double in $O(1)$.
Cascading Cut Example

Decrease key by 2.

Note – a node that moves to the root lists looses its mark (becomes unmarked).

Amortized time

Note that the number of marked nodes decreases by $k$ or $k+1$, and the number of trees increased by $k+1$.

Let $H'$ to be $H$ denote the heap before and after the $\text{Decrease\_min}$ operation, then $t(H') = t(H) + k + 1$ and $m(H') = m(H) - k$

The change in the potential function is (denoting $H'$ to be $H$ after the $\text{Decrease\_min}$) is

$$\Phi(H') - \Phi(H) = (t(H') + 2m(H') - (t(H) + 2 m(H)) = -k + 2$$

And

$$a_i = w_i + \Phi(H') - \Phi(H) = k + (k+2) = 2$$

Actual time complexity of the cascading_cut of a path of length $k$ is $\Theta(k+1)$ (can be $\Theta(h)$ in the worst case, where $h$ is the height)

Assume we specify the time of an elementary operations, so that this time $w_i$ is $k+1$.

Note that the number of trees increases by $k+1$, and the number of marked nodes decreases by either $k$ or $k+1$.
Deletion of a node $v$

- Perform $\text{DecreaseKey}(v)$ to $-\infty$.
- Perform $\text{ExtractMin}(H)$ - seen next.

Extract $\min$

- Remember - there is a pointer ($\text{min}[H]$) pointing to the min.
- Set theNode $\text{min}[H]$.
- Remove theNode from its sibling list.
- Free theNode.
- Perform $\text{Consolidate}(H)$ - merging trees.

Extract $\min$ – example (1)

- $\text{ExtractMin}$ operation (not including consolidation (counted later))
- Let $\deg(v)$ is the number of children of $v$.
- Lemma: (CLRS 20.3): The number of nodes in a tree the tree rooted at $v$ is $\geq \emptyset$
  \[ \deg(v) \geq 1.5 \deg(v) \]
- Conclusion 1: $\deg(v) = O(\log n)$, for every node $v$.
- The actual time $w_j$ needed for disconnecting $v$ from its children and adding them to the roots list – $O(\log n) = \deg(v)$
Union of two trees.
(Need for Consolidation)
• (Similar (but not identical) operation was seen in the binomial heaps)
• Degree of a tree is defined as the degree of the root of the tree.
• Given two trees with the same degree of their roots, connect the root of one as a child of the other root.
• There is always a way to do so while maintaining the heap order:

Point of potential confusion: For Binomial heaps, trees have the same size iff they have the same degree.
Not true here

Extract_min – cont: Consolidation.
• Each extract_min is followed by the consolidation operation:
• This operation repeatedly joins trees with same degree, using the tree-union operation:
  • Repeatedly pick two trees with the same degree, and merge them:
  • ... but trees are not sorted by degree, (as oppose to Binomial heaps) and there are many of them – how can this be done efficiently ? (on board)
  • Finish when no two trees with the same degree exist.
  • Recover the new minimum while doing so.
• Actual Time \( t_w \) – proportional to the number of trees
  • (since every operation reduces the number of trees by one, and takes a constant time).

Extract_min – cont: Consolidation.
(Continued)
• Each extract_min is followed by the consolidation operation:
• This operation repeatedly joins trees with the same degree, using the tree-union operation:
  • Repeatedly pick two trees with the same degree, and merge them:
  • ... but trees are not sorted by degree, (as oppose to Binomial heaps) and there are many of them – how can this be done efficiently ? (on board)
  • Finish when no two trees with the same degree exist.
  • Recover the new minimum while doing so.
• Actual Time \( t_w \) – proportional to the number of trees
  • (since every operation reduces the number of trees by one, and takes a constant time).

Time analysis for consolidation
• The consolidation takes actual time \( t(H) \) time.
• In \( H' \), there at most one tree for each degree of its root (followed from conclusion 1), so \( t(H') = O(\log n) \).
• The number of marked nodes is not changed.
• \( \Phi(H') - \Phi(H) = (t(H') + 2m(H')) - (t(H) + 2m(H)) = t(H') - t(H) = O(\log n) - t(H) \)
• The amortized work is therefore
  \[ t(H) + (O(\log n) - t(H)) = O(\log n) \]

Time analysis for Delete
• Deletion consists of
  • first DecreaseKey (amortized time \( O(1) \) ) and then
  • ExtractMin (amortized time \( O(\log n) \) )
• Total amortized work: \( O(\log n) \)

Toward proving lemma CLRS 20.3
• Claim: Let \( F_i = F_{i+1} \) and \( F_{i+2} = F_{i+1} + F_i \). Then (induction) \( F_{i+2} = \Theta^n \)
  • Lemma 20.2: \( F_{i+2} = 1 + \sum_{i} F_i \)
  • Proof – by induction, on the board

Proving lemma CLRS 20.3
• Let \( s_i \) denote the minimum number of nodes at a tree of degree \( k \).
  • Lemma 20.3 : \( s_k = \Theta^k \)
• Proof: Let \( y_{j_1}, \ldots, y_{j_k} \) denote its children of a node \( x \), in the order they joined \( x \).
  • The degree of \( y_j \) is \( \geq i-L \),
  • hence containing \( \geq F_{i-1} \) nodes,
  • \( s_k = 1 (\text{root}) + \text{sum of number of nodes in subtrees} \)