Binomial Heap

- A kind of **Mergeable Heaps**,
- Support operations
  - MAKE-HEAP(), creates and returns a new heap containing no elements.
  - INSERT(H, x), insert node x into heap H.
  - MINIMUM(H), returns a pointer to the minimum in H.
  - EXTRACT-MIN(H) deletes the minimum from H, retunrs a pointer pointing to the node.
  - UNION(H_1, H_2), creates and returns a new heap that contains all the nodes of heaps H_1 and H_2. H_1 and H_2 are destroyed.

- Also support
  - DECREASE-KEY(H, x, k), assign to node x within heap H the new key value k, which is assumed to be no gerater than its current key value.
  - DELETE(H, x), delete x from heap H.
- SEARCH cannot be done fast
- DECREASE-KEY and DELETE, we assume that there is a pointer pointing to the node.

Procedure	Binary heap (Worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$
UNION	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$

Binomial Tree and Binomial Heap

- Binomial Tree, B_k
- An ordered tree, defined recursively
- B_0 consists of a single node
- B_k consists of two binomial trees B_{k-1} that are linked together.
- Root of one is the left most child of the root of the other.

Lemma, Properties of binomial trees

For the binomial tree B_k,

1. there are 2^k nodes,
2. the height of the tree is k,
3. there are exactly $\begin{pmatrix} k \\ i \end{pmatrix}$ (binomial coefficient) nodes at depth i for $i = 0, 1, \ldots, k$, and
4. the root has degree k, which is greater than that of any other node; moreover if the children of the root are numbered from left to right by $k-1$, $k-2$, $\ldots$, 0, child i is the root of the subtree B_i.

Proof of the 4 Properties

- By induction on k.
- The basis is the binomial tree B_0.
- Verifying that each property hoids for B_0 is trival.
- For the inductive step, we assume that the lemma holds for B_{k-1}

Property 1: Binomial tree B_k consists of two copies of B_{k-1} (each has 2^{k-1} nodes, from the inductive basis). B_k has $2^{k-1} + 2^{k-1} = 2^k$ nodes.

Proof of the 4 Properties

Property 2: B_k is obtained by linking two B_{k-1}s together, one is the left most child of the other. Thus the depth of B_k is one greater that B_{k-1}. B_{k-1} has depth $(k-1)$ from the inductive basis. Depth of $B_k = (k-1)+1 = k$.

Proof of the 4 Properties

Property 3: Let $D(k, i)$ be the number of nodes at depth i of B_k. Depth i in B_k consists of nodes in depth i in one B_{k-1} and depth $i-1$ in another B_{k-1}. They have $D(k-1, i)$ and $D(k-1, i-1)$ nodes respectively (induction hypothesis).

$$\begin{aligned} D(k,i) &= D(k-1,i) + D(k-1,i-1) \\ &= \begin{pmatrix} k-1 \\ i \end{pmatrix} + \begin{pmatrix} k-1 \\ i-1 \end{pmatrix} \\ &= \begin{pmatrix} k \\ i \end{pmatrix} \end{aligned}$$

Proof of the 4 Properties

Property 4: Root of B_k (connecting two B_{k-1}) has one more degree than root of B_{k-1} (degree k-1).

Children of B_{k-1} from left to right are B_{k-2}, B_{k-3}, ..., B_1, B_0.

B_k is obtained by connecting a B_{k-1} to that B_{k-1}, we have the subtree from left to right B_{k-1}, B_{k-2}, B_{k-3}, ..., B_1, B_0.

The maximum degree of any node in an n-node binomial tree is $\lg n$.

Proof Immediate from Properties 1 and 4 of previous lemma.

Binomial Heap

A binomial heap H is a set of binomial trees that satisfies the *binomial-heap* properties.

1. Each binomial tree in H obeys the *min-heap property*: the key of a node is greater than or equal to the key of its parent. We say that each such tree is *min-heap-ordered*.
2. For any nonnegative integer k, there is at most one binomial tree in H whose root has degree k.

From 1, the root of a min-heap-ordered tree contains the smallest key. From 2, an n-node binomial heap H consists of at most $\lfloor \lg n \rfloor + 1$ binomial trees.

Representing Binomial Heaps

- A binomial tree is stored in the left-child, right-sibling representation.
- Each node has a key field and any other satellite information.
- a node x has a pointers $p[x]$ to its parent, $child[x]$ to its leftmost child, abd $sibling[x]$ to the sibling of x immediately to its right.
- x also has a field $degree[x]$, which is the number of children of x.

Representing Binomial Heaps

- Roots of the binomial trees within the binomial heaps are organized in a linked list, which we refer to as the *root list*. The degree of the roots strictly increase as we traverse the root list.
- a pointer $head[H]$ points to the first root in the root lost.
- By 2nd binomial heap property, degrees of roots are a subset of $\{0, 1, \ldots, \lfloor \lg n \rfloor\}$ (length of the root list is $O(\log n)$).

Operations in Binomial Heaps

- Creating a new binomial heap, return $head[H]=$ Nil, running time is $\Theta(1)$.
- Finding the minimum key, check all roots along the root list. There are at most $\lfloor \lg n \rfloor + 1$ roots, can be done in $O(\lg n)$ time.
- Uniting two binomial heaps, biasic idea is to "merge" tow root list, then fix the root list to meet the 4th property of binomial tree. If there are trees have same number of nodes (two B_{k-1}), link them to form a B_k. After that there are at most 3 B_ks, can be done in $O(\lg n)$ time.

Operations in Binomial Heaps

- Inserting a node, create a one node binominal heap H' in $O(1)$ time, then uniting with the n-node binomial heap H in $O(\lg n)$ time.
- Extracting the node with minimum key,
  - Remove the tree, let it be B_k, has the smallest from the root list (findi it in $O(\lg n)$ time, remove it in $O(1)$ time).
  - Delete the root, we then have a forest, from let to right are trees B_{k-1}, B_{k-2}, $\ldots$, B_0 (properyt 4).
  - Reverse them and link them into a root list to form a binomial heap ($O(\lg n)$) time.
  - Uniting the two binominal heap in $O(\lg n)$ time.

Operations in Binomial Heaps

- SEARCH is hard, so we assuome that there is a pointer pointing to node to be decreasing key or deleting.
- Decreasing a key, same as the binary heap, bubbling up the node toward the root of the tree, height is $O(\lg n)$.
- Deleting a key, make the key value to be $-\infty$ then decreasing key. The node goes to the root, then extracting the node with minimum key.

Fibonacci Heap

A kind of Mergeable Heaps, support operations

- Make-Heap()
- Insert(H, x)
- Minimum(H)
- Extract-Min(H)
- Union(H_1, H_2)
- Decrease-Key(H, x, k)
- Delete(H, x)

Procedure	Binary heap (Worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$
UNION	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$

Fibonacci Heap

- Good for the cases where EXTRACT-MIN and DELETE are few, better if there are many DECREASE-KEYs.
- DECREASE-KEY can be done in $\Theta(1)$ amortized time. Case happens when we are looking for single-source shortest path in a graph.
- Theoretical interest.

Fibonacci Heap

A Collection of rooted min-heap-ordered trees, not constrained to be binomial trees. Each Node x,

- $x.p$, p a pointer to its parent.
- $x.child$, *child* is a pointer to one of its children.
- *children* of x are in a circular doubly linked list, called *child list* of x.
- $x.degree$, *degree* number of nodes in the *child list*.
- $x.mark$ indicates that whether x has lost a child since the last time x was made the child of another node.
  - x is unmarked if it is newly created.
  - x is unmarked if is made child of another node.

- A Fibonacci heap H, $X.min$ points to the root of the tree that has the smallest key. This node is called the *minimum node*.
- The roots of all the trees are linked together into a circular, doubly linked list, called the *root list*.
- $H.n$, the number of nodes currently in the heap.

Potential Function

- Given a Fibonacci heap H
  - $t(H)$, the number of trees in the root list of H.
  - $m(H)$, the number of marked nodes in H.
  - Potential function

$$\Phi(H) = t(H) + 2m(H). \tag{1}$$

  - The potential of a set of Fibonacci heaps is the sum of the potentials of its constituent Fibonacci heap.
  - Assume that a unit of potential can pay for a constant amount of work.
- Potential of the heap in Figure 19.2 is 5+2*3=11.

Potential Function

- Assume that we start with an empty heap.
- Above equation is non-negative at all subsequence time, the amortized cost covers the actual cost.
- Assume that there is known upper bound for $D(n)$ on the maximum degree of any node in an n-node Fibonacci heap (for amortixed analysis purpose).
- $D(n) \leq \lfloor \lg n \rfloor$ at this time, state without proof.

Mergeable-Heap Operations

- Delay work as long as possible.
- INSERT and UNION work on the root list, make sure $H.min$ points to the minimum.
- Work delay to DELETE or EXTRACT-MIN, after deleting, CONSOLIDATE is required.
- After CONSOLIDATE, each node in the root list has a degree that is unique within the root list, thus the size of the root list is at most $D(n) + 1$.

Create and Minimum

- Create a new Fibonacci heap, actual cost $O(1)$, potential $\Phi(H) = 0$, amortized cost is $O(1)$.
- Finding the minimum, follow the pointer $min[H]$. Potential does not change, actual cost is $O(1)$, thus amortized cost is $O(1)$.

Inserting a node

- Allocate a new node x, x is unmarked, and insert x into the root list. Update $H.min$ if necessary.
- Make no attempt to consolidate the trees within the Fibonacci heap.
- The amortized cost:
  - H the input, H' the resulting Fibonacci heap
  - $t(H') = t(H) + 1$, $m(H') = m(H)$
  - Increase in potential is $((t(H) + 1 + 2m(H)) - (t(H) + 2m(H))) = 1$.
  - Actual cost is 1, thus amortized cost is $O(1)$.

Uniting two Fibonacci heaps

- Given H_1, H_2, concatenate the root list of H_1 and H_2, then determine the new minimum node.
- No consolidation of trees occurs.
- Potential change $\Phi(H) - (\Phi(H_1) + \Phi(H_2)) = 0$, since $t(H) = t(H_1) + t(H_2)$ and $m(H) = m(H_1) + m(H_2)$.
- The actual cost is $O(1)$ since doubly linked list is used.
- Amortized cost is $O(1)$.

Extracting the minimum node

- Delete the minimum z, making all z's children roots of H.
- $H.min$ points to a node ($z.right$) in the root list, not necessary to be the minimum.
- Then reduce the number of trees in the root list, i.e., Consolidating the root list.
- Finally, $H.min$ points to the minimum and return z.

Extracting the minimum node

- CONSOLIDATE
  1. Find two roots x and y in the root list with the same degree, where $x.key \leq y.key$.
  2. Link y to x: remove y from the root list and make y a child of x.
  3. $x.degree$ is incremented and mark on y clear.
- CONSOLIDATE needs an auxiliary array $A[0..D(n(H))]$,
- $A[i] = y$, y is currently a root with $y.degree = i$.

Extracting the minimum node

- H the Fibonacci heap just prior to deleting the minimum.
- The actual cost
  - $O(D(n))$ contribution comes from there being at most $D(n)$ children of z, and last step to consolidate.
  - Scan the root list and trees merging:
    - Size of the root list before conolidation: at most $D(n) + t(H) - 1$; used to be $t(H)$, "-1" remove z, z has at most $D(n)$ children that become roots of trees.
    - Each merging removes a tree, total number of mergings $< D(n) + t(H)$.
- Actual cost is $O(D(n) + t(H))$.

Extracting the minimum node

- Potential changes
  - Before extracting the minimum node, $t(H) + 2m(H)$.
  - After extracting the minimum node, at most $(D(n) + 1) + 2m(H)$ since at most $D(n) + 1$ roots remains and no nodes become marked.
- Amortized cost

$$
\begin{aligned}
& O(D(n) + t(H)) + ((D(n) + 1) + 2m(H) - (t(H) + 2m(H))) \\
= \; & O(D(n)) + O(t(H)) - t(H) \\
= \; & O(D(n))
\end{aligned}
$$

Decreasing a key

- Check whether decreasing key change anything (if $x.key$ still greater than $x.p.key$, or x is the root).
- Decreasing the key in node x
- Cutting x, cut the link between x and its parent y,
- Making x a root, unmark x,
- Mark y if y is unmarked, otherwise CASCADING-CUT, i.e., cut y and recursively do so.
- Finally, change $H.min$ if necessary (since only key of x was decreased, $H.min$ remains or points to x).

Decreasing a key

- The *mark* field records history information of the node x. Suppose that he following events have happened to node x,
  1. at some time, x was a root,
  2. then x was linked to another node,
  3. two children of x were removed by cuts.
- x becomes a new root once its second child is removed.

Decreasing a key

- Actual cost
  - Decrease key takes $O(1)$ time +
  - Cascading cuts, if CASCADING-CUT is recursively called c times, it takes $O(c)$ time
- Changes in Potential
  - Each recursive call of CASCADING-CUT, except the last one, cuts a marked node and clears the marked field.
  - Afterward, there are $t(H) + c$ trees (used to be $t(H)$ trees, $c - 1$ trees produced by cascading cuts, and x).
  - At most $m(H) - c + 2$ marked nodes ($c - 1$ were unmarked by cascading cuts, last call of CASCADING-CUT makrs a node)

Decreasing a key

- Changes of Potential: $((t(H) + c) + 2(m(H) - c + 2))$ - (t(H)+2m(H)) $= 4 - c$.
- Amortized cost, $O(c) + 4$ - $c = O(1)$.

- Decreasing the key in x to ∞,
- then extracting minimum.
- The armotized cost are respectively $O(1)$ and $O(D(n))$.

- Prove the amortized time of a FIB-HEAP-EXTRACT-MIN and FIB-HEAP-DELETE is $O(\lg n)$,
- To show the upper bound $D(n)$ on the degree of any node of a n node Fibonacci Heap is $O(\lg n)$.
- If trees in Fibonacci are unordered binomial tree, we have $D(n) = \lfloor \lg n \rfloor$.
- But CUT make trees that are not unordered binominal tree.
- To prove the $\lg n$ upper bound becomes not trival.

To bound $D(n)$

- Show that because we cut a node from it parent as soon as it loses two children, $D(n)$ is $O(\lg n)$.
- In particular $D(n) \leq \lfloor \log_\phi n \rfloor$, where $\phi = (1 + \sqrt{5})/2$
- Key idea of th proof, for each node x, define $size(x)$ to be the number of nodes, including x, in the subtree rooted at x.
- And show that $size(x)$ is exponential in $degree[x]$.

Let x be in a Fibonacci heap, and suppose that $degree[x] = k$. Let $y_1, y_2, \ldots, y_k$ denote the children on x in the order in which there were linked to x. Then, $degree[y_1] \geq 0$ and $degree[y_i] \geq i - 2$ for $i = 2, 3, \ldots, k$.

Proof Obviously, $degree[y_1] \geq 0$. For $i \geq 2$, note that when y_i was linked to x, all of $y_1, y_2, \ldots, y_{i-1}$ were children of x, so $degree[x] = i - 1$. Node y_i is linked to x only if $degree[x] = degree[y_i]$, so we must have had $degree[y_i] = i - 1$. Since then y_i could lost one child, since it would have been cut from x if it had lost two children. We conclude that $degree[y_1] \geq i - 2$.

For $k = 0, 1, 2, \ldots,$ the kth Fibonacci number is defined recursive

$$F_k = \begin{cases} 0 & \text{if} \quad k = 0, \\ 1 & \text{if} \quad k = 1, \\ F_{k-1} + F_{k-2} & \text{if} \quad k \geq 2. \end{cases}$$

The lemma gives another way to express F_k.

Lemma For all integer $k \geq 0$,

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i.$$

Induction on k: When $k = 0$

$$\begin{aligned} 1 + \sum_{i=0}^{0} F_i &= 1 + F_0 = 1 + 0 = 1 \\ &= F_2. \end{aligned}$$

Assume the inductive hypothesis that $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$, then we have

$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} \\ &= F_k + (1 + \sum_{i=0}^{k-1} F_i) \\ &= 1 + \sum_{i=0}^{k} F_i \end{aligned}$$

Use the inequality

$$F_{k+2} \geq \phi^k$$

to show the following lemma,
where $\phi = (1 + \sqrt{5})/2 = 1.61803$ is the golden ratio.
And we shall argue that $Size[x] > F_{k+2}$.

Let x be any node in a Fibonacci heap, and let $k = degree[x]$.
Then $size(x) \geq F_{k+2} \geq \phi^k$.
Proof Let s_k denote the minimum possible value of $size(z)$ over all nodes z, s.t. $degree[z] = k$.
$s_0 = 1$, $s_1 = 2$, and $s_2 = 3$.
Let $y_1, y_2, \ldots, y_k$ denote the children of x in the order in which there were linked to x. We would like to compute the lower bound on $size(x)$.

1. count 1 for x,
2. count 1 for the first child y_1 (since $size(y_1) \geq 1$).

We have

$$
\begin{aligned}
size(x) &\geq s_k && (2)\\
&= 2 + \sum_{i=2}^{k} s_{degree[y_i]} && (3)\\
&\geq 2 + \sum_{i=2}^{k} s_{i-2} && (4)
\end{aligned}
$$

We then show by induction on k that $s_k \geq F_{k+2}$, $\forall k \geq 0$.

Bases, $k = 0$, $k = 1$, trival.
Inductive step, assume that $k \geq 2$ and that $s_i \geq F_{i+2}$.

$$\begin{aligned} s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} && (5) \\ &\geq 2 + \sum_{i=2}^{k} F_i && (6) \\ &= 1 + \sum_{i=0}^{k} F_i && (7) \\ &= F_{k+2}. && (8) \end{aligned}$$

We conclude that $size(x) \geq s_k \geq F_{k+2} \geq \phi^k$.

Corollary The maximum degree $D(n)$ of any node in an n-node Fibonacci heap is $O(\lg n)$.

Proof x a node in n-node Fibonacci heap and $k = \mathit{degree}[x]$. Above lemma says $n \geq \mathit{size}(x) \geq \phi^k$. Taking base-$\phi$ logarithms results $k \leq \log_\phi n$, i.e., the maximum degree $D(n)$ of any node is $O(\lg n)$.