	Binary Heap (worst case)	Binomial Heap (worst case)	Fibonacci Heap (amortized)
Make-Heap	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Minimum	$\Theta(1)$	$\Theta(lgn)$	$\Theta(1)$
Extract-Min	$\Theta(lgn)$	$\Theta(lgn)$	$\Theta(lgn)$
(Union)	$\Theta(n)$	$\Theta(lgn)$	$\Theta(1)$
Decrease-Key	$\Theta(lgn)$	$\Theta(lgn)$	$\Theta(1)$
Delete	$\Theta(lgn)$	$\Theta(lgn)$	$\Theta(lgn)$
Insert	$\Theta(lgn)$	$\Theta(lgn)$	$\Theta(1)$

Mergeable Heaps

Union(H1, H2) Creates and returns a new heap containing all nodes from heaps H1 and H2

Binary Heaps: Union = $\Theta(n)$ worst case

Binomial Heaps: Union = $O(\lg n)$ worst case

Fibonacci Heaps: Union = $\Theta(1)$ amortized

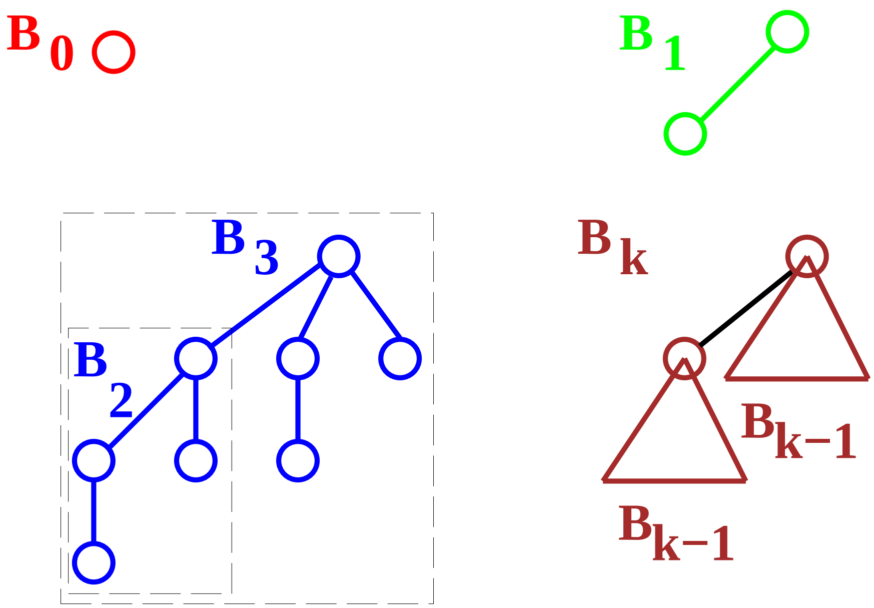
Other operations:

Extract-Min maintains partial ordering over keys.
This is useful for many graph algorithms.

Binomial Heaps

A binomial heap is a set of **binomial trees**.

A **binomial tree** B_K is an ordered tree such that



B_K properties

1. There are 2^k nodes
2. Height of tree = k
3. There are exactly $\binom{k}{i}$ nodes at depth i (this is why the tree is called a "binomial" tree)

   Review: this is $\frac{k!}{i!(k-i)!}$
4. Root has degree k (children) and its children are $B_{k-1}, B_{k-2}, .., B_0$ from left to right

---

Prove properties by induction on k

Base Case: Holds for B_0.

Assume: Holds for $B_0 \ldots B_{k-1}$.

1. B_k is 2 copies of B_{k-1}, so $2^{k-1} + 2^{k-1} = 2^k$ nodes.
2. Depth of B_k is one greater than maximum depth of B_{k-1}.
   Add one more level: height = (k-1) + 1 = k.
3. See book (Lemma 20.1).
4. True for children $B_{k-1}, B_{k-2}, .., B_0$ from left to right.
   B_{k-1} is left child of B_k, root is also root of B_{k-1} (minus left child), so degrees are $B_{k-1}, B_{k-2}, .., B_0$.
   The root of B_k is a B_{k-1} with one more child (the left child), so root of B_k has degree (k-1) + 1 = k.

Binomial Heap Properties

1. Each binomial tree is heap-ordered (key(x) $\geq$ key(parent(x)).
   This is the opposite of previous heap properties.
2. There never exist two or more trees with the same degree in the heap.

A binomial heap with n nodes has at most $\lfloor \lg n \rfloor + 1$ binomial trees.

n in binary = $< b_k, b_{k-1}, .., b_0 >$ bits
k = $\lfloor \lg n \rfloor$, n = $\Sigma_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$

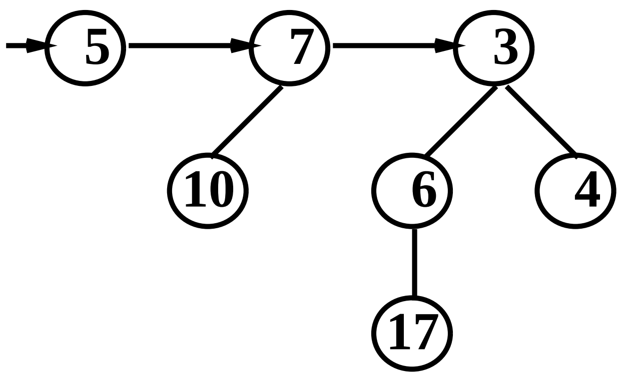
There is a one-to-one mapping between the binary representation and binomial trees in a binomial heap.

If $b_i = 1$, then B_i is in the heap

Recall that there are 2^i nodes in B_i

At most $\lfloor \lg n \rfloor + 1$ bits are needed to express n base 2

Example: Binomial Heap H



$B_0 \longrightarrow B_1 \longrightarrow B_2$ (____________________ nodes)
$B_0 \longrightarrow B_2 \longrightarrow B_5$ (_______________________ nodes)

Operations

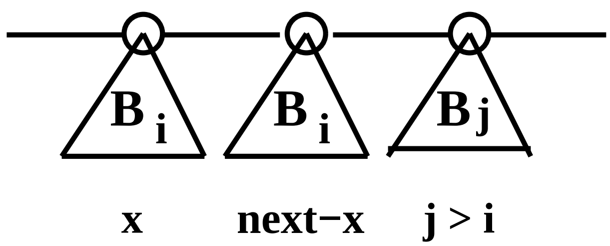
Make-Heap() ($\Theta(1)$)

Minimum(H) ($O(\lg n)$)
Find minimum of roots of binomial trees in H

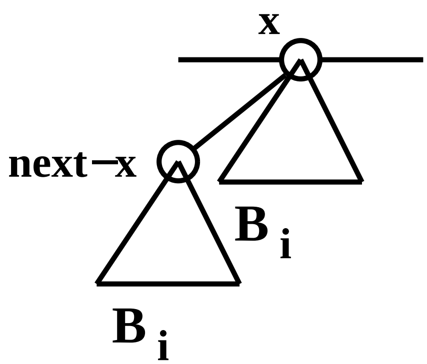
Operations

Union(H_1, H_2)

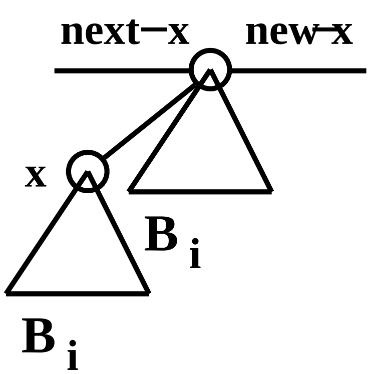
Union(H_1, H_2)
  H = new heap containing trees of H_1 and H_2 merged in
    non-decreasing order by degree of root
      ; Similar to Merge used in MergeSort
      ; O(lg n): at most two roots of each degree,
      ;   O(lg n) possible degrees
      ; No more than 2 B_i trees in H at this point
      ; Could be 3 after linking two B_{i-1} trees together
  prev-x = NIL ; three-tree window
  x = head(H) ; look for:



  next-x = sibling(x)
  while next-x $\neq$ NIL
    if degree(x) $\neq$ degree(next-x) or
      degree(x) = degree(next-x) = degree(sibling(next-x))
    then move window right by one
    else if key(x) $\leq$ key(next-x)
      then:



else:



advance window

Running time = O(lg n) if $n = n_1 + n_2$ nodes in H.

Example

Operations

Insert(H, x)

```
Insert(H,x)
    H' = x
    H = Union(H, H')
```

Running time = O(lg n)

Extract-Min(H)

```
Extract-Min(H)
    Find root x with minimum key in H          ; O(lg n)
    Remove x from H                            ; Θ(1)
    H' = children of x in reverse order        ; O(lg n)
            ; because children are B_{k-1}, B_{k-2}, ..., B_0
    Union(H, H')                               ; O(lg n)
```

Running time = O(lg n)

Operations

Decrease-Key(H, x, k), where k $\leq$ key(x)

```
Decrease-Key(H, x, k)
    key(x) = k
    while parent(x) ≠ NIL and key(x) < key(parent(x))
        ;; "bubble" new key up
        swap(key(x), key(parent(x)))
        x = parent(x)
```

Max depth = $\lfloor lgn \rfloor$
Running time = O(lg n)

Procedure	Binomial Heap (worst case)	Fibonacci Heap (amortized)
Make-Heap	$\Theta(1)$	$\Theta(1)$
Insert	$O(\lg n)$	$\Theta(1)$
Minimum	$\Theta(\lg n)$	$\Theta(1)$
Extract-Min	$\Theta(\lg n)$	$O(\lg n)$
Union	$O(\lg n)$	$\Theta(1)$
Decrease-Key	$\Theta(\lg n)$	$\Theta(1)$
Delete	$\Theta(\lg n)$	$O(\lg n)$

Delete(H, x)

```
Delete(H, x)
    Decrease-Key(H, x, −∞)        ; O(lg n)
    Extract-Min(H)                ; O(lg n)
```

Running time = O(lg n)

Fibonacci Heaps

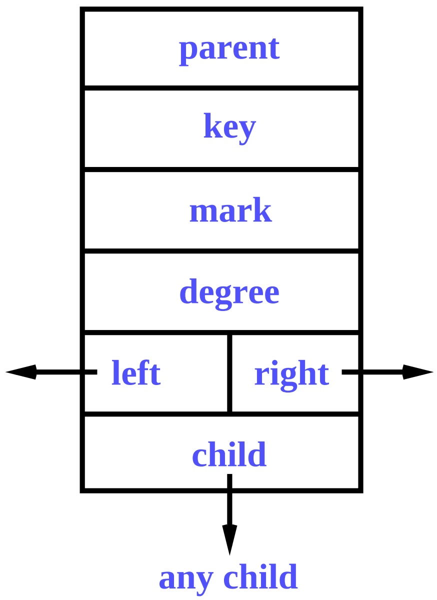
- If nodes are never removed, then yields $\Theta(1)$ performance
- Not designed for efficient search

Structure of Fibonacci Heaps

A **Fibonacci Heap** is a set of heap-ordered trees. Trees are not ordered binomial trees, because

1. Children of a node are unordered
2. Deleting nodes may destroy binomial construction

Node Structure:



The field "mark" is True if the node has lost a child since the node became a child of another node.

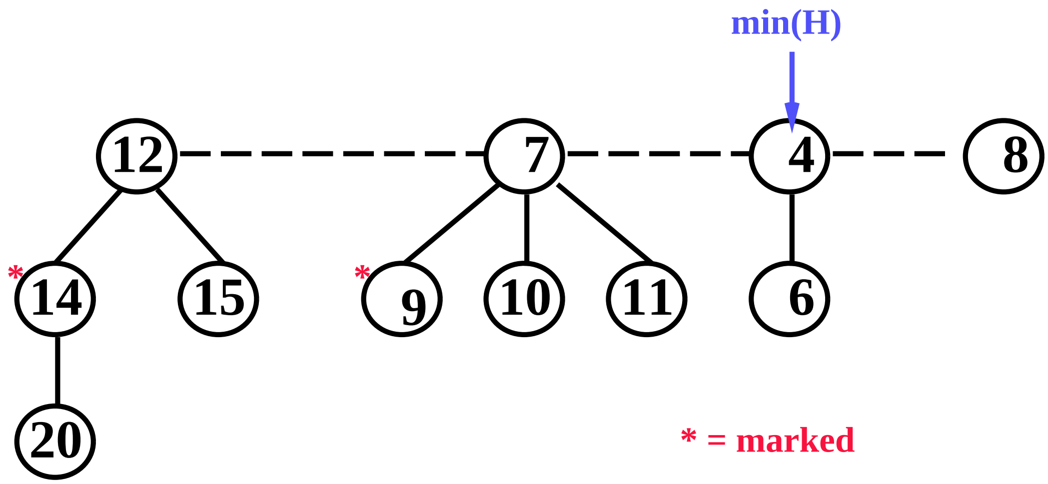
The field "degree" contains the number of children of this node.

The structure contains a doubly-linked list of sibling nodes.

Heap Structure

min(H) pointer to node in root list having smallest key in heap H

n(H) number of nodes in heap H



Potential Function

$$\Phi(H) = \text{t(H)} + 2\text{m(H)}$$
$$\text{t(H)} = \#\text{trees in root list of heap H}$$
$$\text{m(H)} = \#\text{mark nodes in heap H}$$

Example

t(H) = 4, m(H) = 2, $\Phi(H) =$ _

Empty heap, $\Phi(H_0) = 0$, $\Phi(H_i) \geq 0$

$$\Sigma_{i=1}^{n} \hat{c}_i \text{ is an upper bound on } \Sigma_{i=1}^{n} c_i$$

Maximum Degree

D(n) = upper bound on degree of a node in a Fibonacci Heap with n nodes

By showing D(n) = O(lg n), we can constrain running times for node removal.

1. Make node root
2. Delete
3. Add O(lg n) children to root list

Mergeable Heap Operations

Make-Heap, Insert, Minimum, Extract-Min, Union

These always yield unordered binomial trees; thus, they maintain the binomial tree properties.

1. 2^k nodes
2. k = height of tree
3. $\binom{k}{i}$ nodes at depth i
4. Unordered binomial tree U_k has root with degree k greater than any other node. Children are trees $U_0, U_1, .., U_{k-1}$ in some order.

For n-node Fibonacci Heap, D(n) is largest if all nodes are in one tree.

The maximum degree is at depth=1, $\binom{k}{1}$ = k for tree with 2^k nodes.

If n = 2^k, then k = lg n

$$D(n) \leq k = \lg n$$
$$D(n) = O(\lg n)$$

Strategy

- Do not merge trees until necessary
- Merging done in Extract-Min, where new minimum is needed

Operations

Make-Heap

Make-Heap()
    allocate(H)
    min(H) = NIL
    n(H) = 0

Analysis:
t(H) = m(H) = 0
$\Phi(H)$ = t(H) + 2m(H) = 0
$\hat{c}_i = c_i$ = O(1)
Amortized cost equals actual cost.

Operations

Insert

Insert(H, x)
    set x's fields appropriately
    add x to root list of H ; O(1)

reset min(H) if needed
n(H) = n(H) + 1

Analysis:
H = initial heap with t(H) trees and m(H) marked nodes
H' = new heap, t(H') = t(H) + 1, m(H') = m(H)

$$\hat{c}_i = c_i + \Phi(H') - \Phi(H)$$
$$= O(1) + [t(H) + 1 + 2m(H)] - [t(H) + 2m(H)]$$
$$= O(1) + 1 = O(1)$$

Operations

Minimum

Minimum(H)
return min(H)

Analysis:
H = H'
$\Phi(H) = \Phi(H')$
$\hat{c}_i = c_i = O(1)$

Operations

Union

Union(H_1, H_2)
H = new heap whose root list contains roots from H_1 and H_2

n(H) = n(H_1) + n(H_2)
min(H) = min(H_1)
if (min(H_1) = NIL) or (min(H_2) $\neq$ NIL and min(H_2) < min(H_1))
then min(H) = min(H_2)

Analysis:
t(H) = t(H1) + t(H2)
m(H) = m(H1) + m(H2)

$$\begin{aligned}
\hat{c_i} &= c_i + \Phi(H) - (\Phi(H_1) + \Phi(H_2)) \\
&= O(1) + [t(H_1) + t(H_2) + 2(m(H_1) + m(H_2))] \\
&\quad -[t(H_1) + 2m(H_1) + t(H_2) + 2m(H_2)] \\
&= O(1) + 0 \\
&= O(1)
\end{aligned}$$

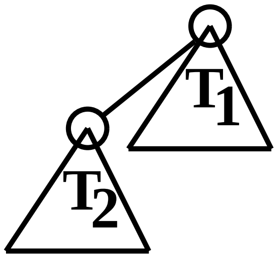
Operations

Extract-Min

Extract-Min(H)
z = min(H)
add z's children to root list ; O(D(n(H)))
remove z from root list
if root list $\neq$ {}
then Consolidate(H) ; O(D(n(H)))
else min(H) = NIL
n(H) = n(H) - 1

Consolidate

Consolidate(H)
while two trees in H (T1,T2) have same degree
change root list to following using Link(H, T2, T1):



for i = 0 to D(n(H))
if tree T of degree i has root-key < min(H)
then min(H) = T

Example

Click mouse to advance to next frame.

Analysis

$$\text{n(H)} = \text{n}$$
$$\text{length(rootlist)} \leq \text{D(n)} + \text{t(H)} - 1$$
$$\text{T(while loop)} \leq \text{D(n)} + \text{t(H)}$$
$$c_i = \text{O(D(n)} + \text{t(H))}$$

$$\Phi(H) = \text{t(H)} + \text{2m(H)}$$
$$\Phi(H') \leq \text{D(n)} + 1 + \text{2m(H)}$$

$$\hat{c}_i = c_i + (\mathrm{D(n)} + 1 + 2\mathrm{m(H)}) - (\mathrm{t(H)} + 2\mathrm{m(H)})$$
$$= \mathrm{O(D(n) + t(H)) + D(n) + 1 - t(H)}$$
$$= \mathrm{O(D(n))}$$

Assuming adjustment of potential coefficients to dominate coefficients hidden in O(t(H)).

Operations

Decrease-Key

```
Decrease-Key(H, x, k)
    key(x) = k
    p = parent(x)
    if p ≠ NIL and key(x) < key(p)
    then Cut(H, x, p)
        Cascading-Cut(H, p)
    if key(x) < key(min(H))
    then min(H) = x

Cut(H, x, p)
    remove x from children of p
    add x to root list of H
    mark(x) = False

Cascading-Cut(H, p)
    next-p = parent(p)
    if next-p ≠ NIL
    then if mark(p) = False
        then mark(p) = True
```

```
else Cut(H, p, next-p)
     Cascading-Cut(H, next-p)
```

Analysis

$$\text{Let } c_i = \text{O(c) be the number of cascading cuts}$$
$$\Phi(H') = \text{(t(H) + c) + 2(m(H) - c + 2), c-1 unmarked, 1 marked}$$
$$\hat{c_i} = \text{O(c) + (t(H) + c) + 2(m(H) - c + 2) - (t(H) + 2m(H))}$$
$$= \text{O(c) + 4 - c}$$
$$= \text{O(1)}$$

Example

Click mouse to advance to next frame.

Operations

Delete

```
Delete(H, x)
     Decrease-Key(H, x, -∞)      ; O(1) amortized
     Extract-Min(H)              ; O(D(n)) amortized
```

Analysis:
Running time = O(D(n)) amortized

Bounding Maximum Degree D(n)

Lemma 21.1

$F_{k+2} \; = \; 1 \; + \; \Sigma_{i=0}^{k} F_i,$ where F_k is a Fibonacci number.

$$F_k = \begin{cases} k & \text{if } k < 2 \\ F_{k-1} + F_{k-2} & \text{if } k \geq 2 \end{cases}$$

Lemma 21.3

For a node x in a Fibonacci heap, where k = degree(x),
size(x) $\geq F_{k+2} \geq \phi^k$, where $\phi \; = \; \frac{1+\sqrt{5}}{2}$
size(x) = #nodes in subtree rooted at x

Corollary 21.4

D(n) = O(lgn)

By Lemma 21.3, n $\geq$ size(x) $\geq \phi^k$, where n = nodes in Fibonacci Heap and k = degree of any node x.

Then $\log_\phi n \;\; \geq \;\; k$, and k = $O(\log_\phi n)$ = O(lg n).

Applications