Chapter 19: Binomial Heaps

We will study another heap structure called, the binomial heap.

The binomial heap allows for efficient union, which can not be done efficiently in the binary heap. The extra cost paid is the minimum operation, which now requires $O(\log n)$.

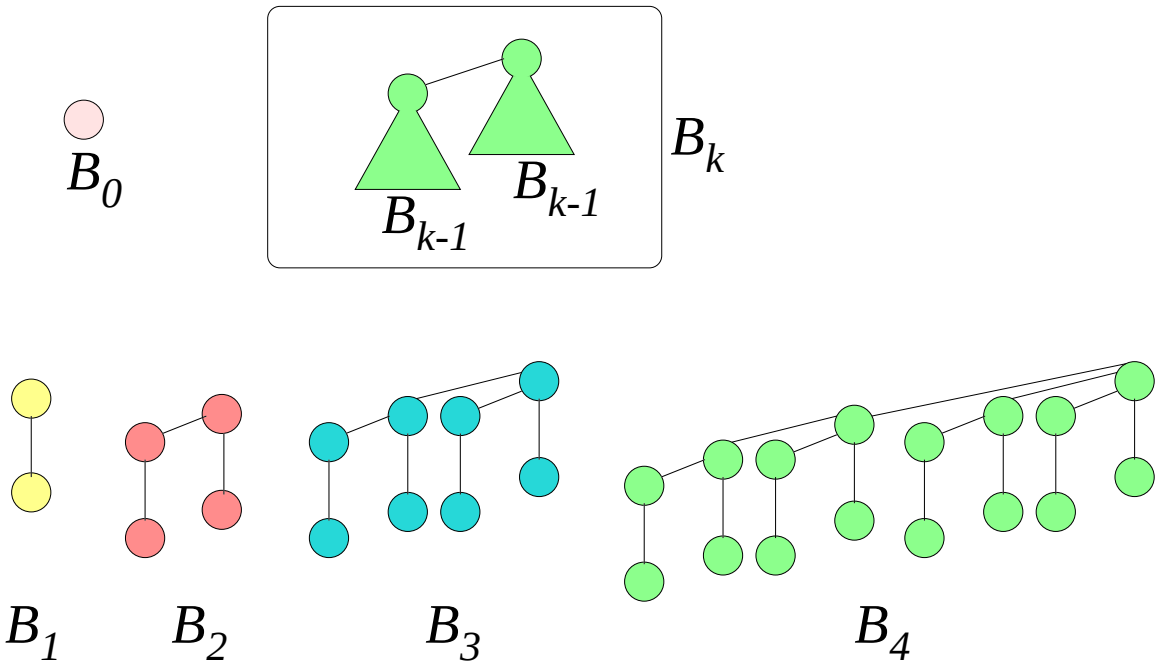
Comparison of Efficiency

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

Definition

A binomial tree B_k is an ordered tree defined recursively.

- B_0 consists of a single node.
- For $k \geq 1$, B_k is a pair of B_{k-1} trees, where the root of one B_{k-1} becomes the leftmost child of the other.


B_0
B_k
B_{k-1}
B_{k-1}
B_1
B_2
B_3
B_4

Properties of Binomial Trees

Lemma A For all integers $k \geq 0$, the following properties hold:

1. B_k has 2^k nodes.
2. B_k has height k.
3. For $i = 0, \ldots, k$, B_k has exactly $\binom{k}{i}$ nodes at depth i.
4. The root of B_k has degree k and all other nodes in B_k have degree smaller than k.
5. If $k \geq 1$, then the children of the root of B_k are $B_{k-1}, B_{k-2}, \cdots, B_0$ from left to right.

Corollary B The maximum degree of an n-node binomial tree is $\lg n$.

Properties of Binomial Trees

For $i = 0, \ldots, k$, B_k has exactly $\binom{k}{i}$ nodes at depth i.

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

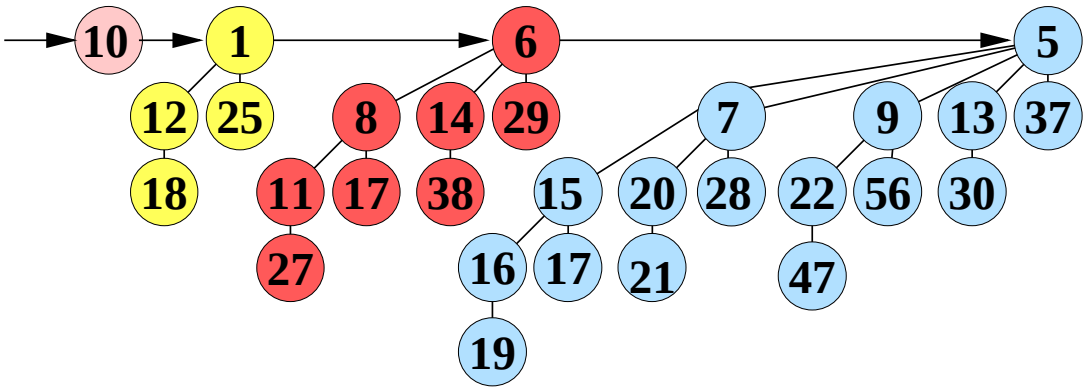
The Binomial Heap

A binomial heap is a collection of binomial trees that satisfies the following binomial-heap properties:

1. No two binomial trees in the collection have the same size.
2. Each node in each tree has a key.
3. Each binomial tree in the collection is heap-ordered in the sense that each non-root has a key strictly less than the key of its parent.

By the first property we have the following: For all $n \geq 1$ and $k \geq 0$, B_k appears in an n-node binary heap if and only if the $(k+1)$st bit of the binary representation of n is a 1.

This means that the number of trees in a binomial heap is $O(\log n)$.

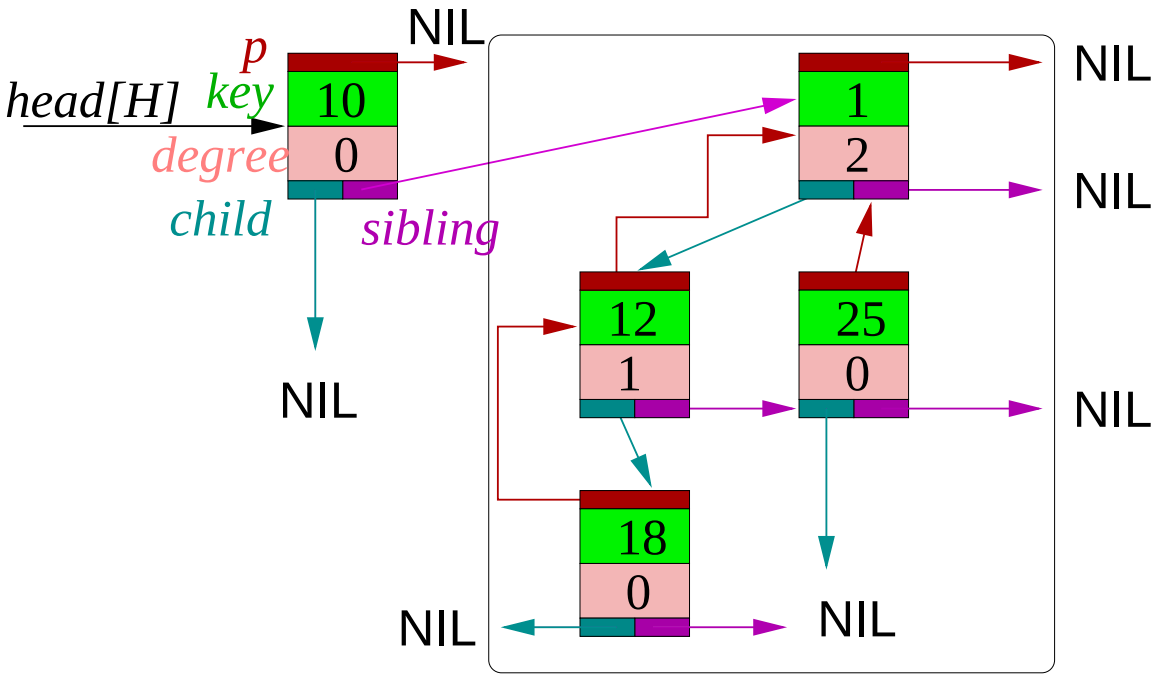

10
1
6
5
12
25
8
14
29
7
9
13
37
18
11
17
38
15
20
28
22
56
30
27
16
17
21
47
19

Implementation of a Binomial Heap

Keep at each node the following pieces of information:

- a field key for its key,
- a field $degree$ for the number of children,
- a pointer $child$, which points to the leftmost-child,
- a pointer $sibling$, which points to the right-sibling, and
- a pointer p, which points to the parent.

The roots of the trees are connected so that the sizes of the connected trees are in decreasing order. Also, for a heap H, $head[H]$ points to the head of the list.


p
key
head[H]
10
degree
0
child
sibling
NIL
NIL
NIL
1
2
NIL
12
1
25
0
NIL
18
0
NIL
NIL

Operations on Binomial Heaps

1. creation of a new heap,
2. search for the minimum key,
3. uniting two binomial heaps,
4. insertion of a node,
5. removal of the root of a tree,
6. decreasing a key, and
7. removal of a node.

1. Creation of a New Heap

To do this, we create an object H with $head[h] = \mathbf{nil}$.

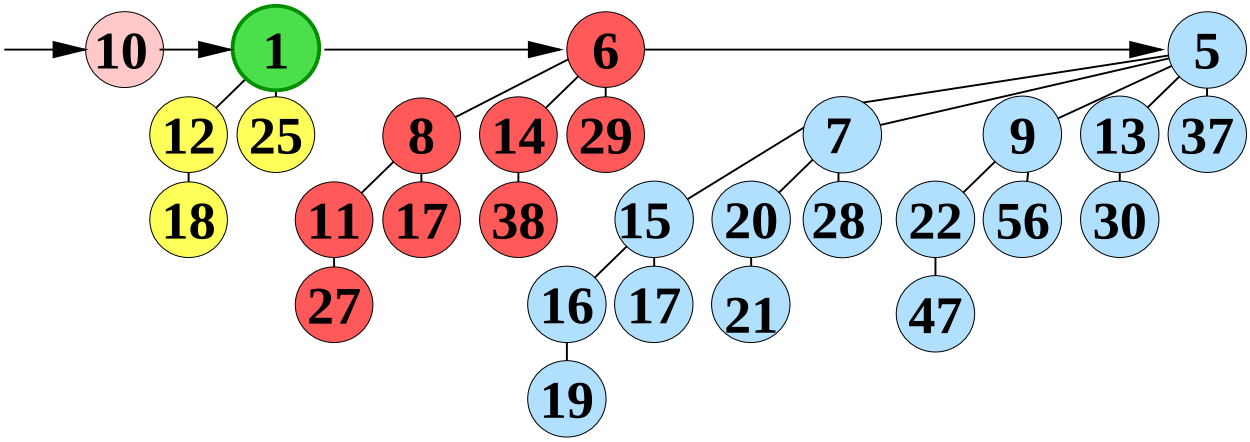
2. Search for the Minimum Key

To do this we find the smallest key among those stored at the roots connected to the head of H.

What's the cost of minimum-search?

What's the cost of minimum-search?

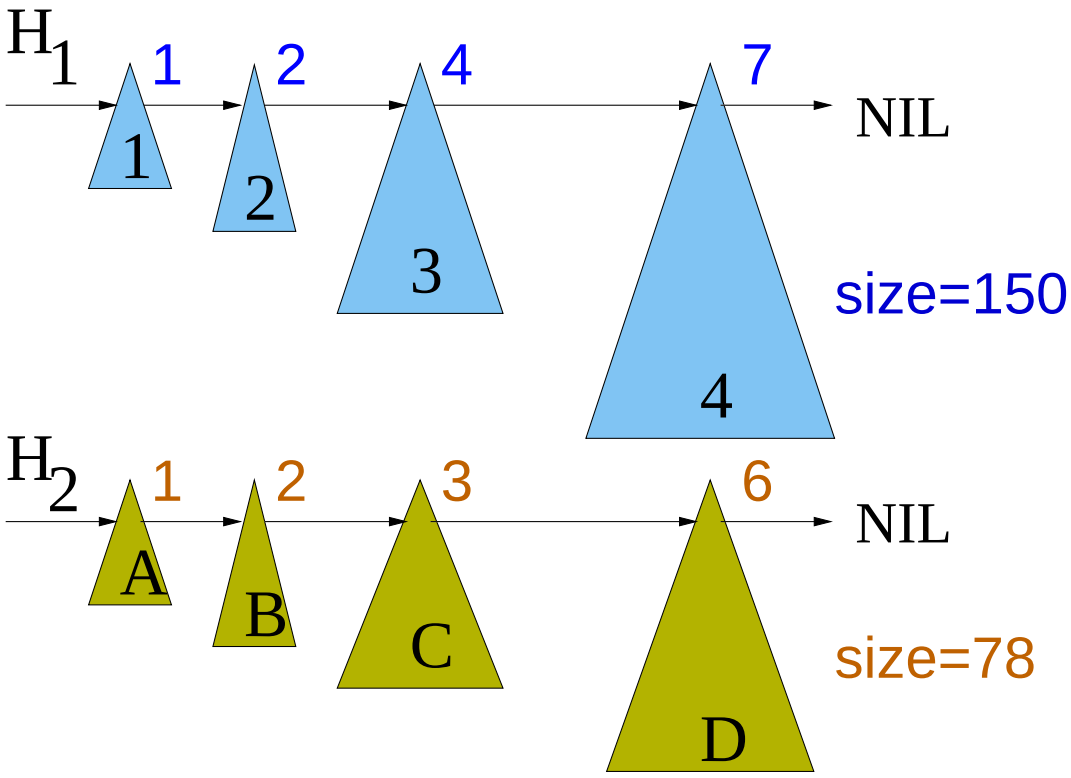
The cost is $O(\log n)$ because there are $O(\log n)$ heaps, in each tree the minimum is located at the root, and the roots are linked.


10
1
6
5
12
25
8
14
29
7
9
13
37
18
11
17
38
15
20
28
22
56
30
27
16
17
21
47
19

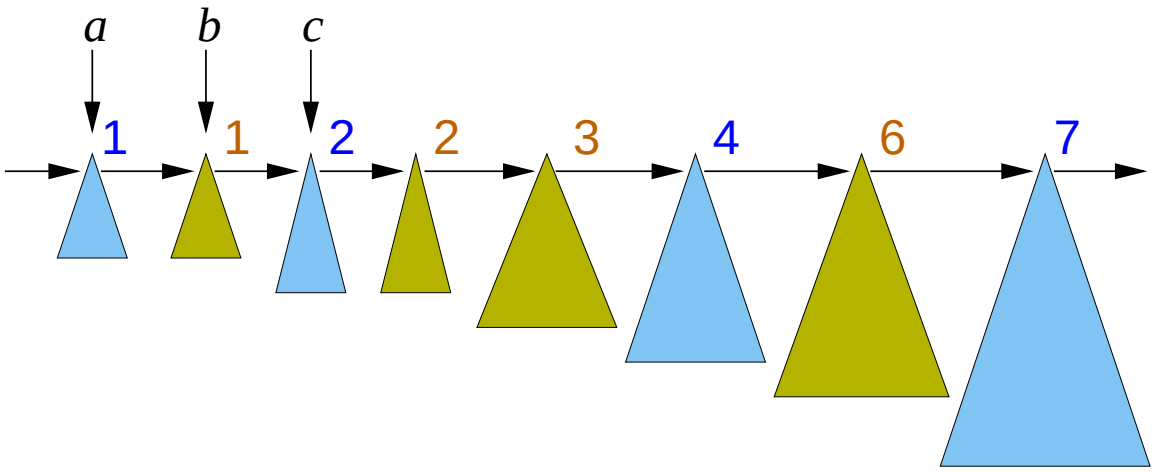
3. Uniting Two Binomial Heaps

Suppose we wish to unite two binomial heaps, H_1 and H_2, having size n_1 and n_2, respectively. Call the new heap H_0.

Recall that the list of the root degrees of a binary heap is the sorted list of all positions in which the binary representation of the heap size has a bit 1 and the order and that the positions appear in increasing order. We will simulate addition of binary numbers.


H1
1
2
4
7
NIL
1
2
3
4
size=150
H2
1
2
3
6
NIL
A
B
C
D
size=78

- Merge H_1 and H_2 into H_0 so that the tree sizes are in nondecreasing order (in the case of a tie the tree from H_1 precedes the one from H_2).
- Sweep-scan H_0 with pointers three points, a, b, and c, that are set on three consecutive trees. Here a is the closest to the start and c to the end. The scanning process is terminated as soon as a becomes **nil**.
- While scanning preserve:
  1. $degree[a] \leq degree[b] \leq degree[c]$,
  2. no two trees that have been left behind have an equal size, and
  3. no more than three trees on the list have an equal size.


a
b
c
1
1
2
2
3
4
6
7

We will study three cases:

Case I: Either $degree[a] < degree[b]$ or $b = \mathbf{nil}$.

Case II: $degree[a] = degree[b] = degree[c]$.

Case III: $degree[a] = degree[b]$ and (either $degree[b] < degree[c]$ or $c = \mathbf{nil}$).

Case I: Either $degree[a] < degree[b]$ or $b = \mathbf{nil}$.

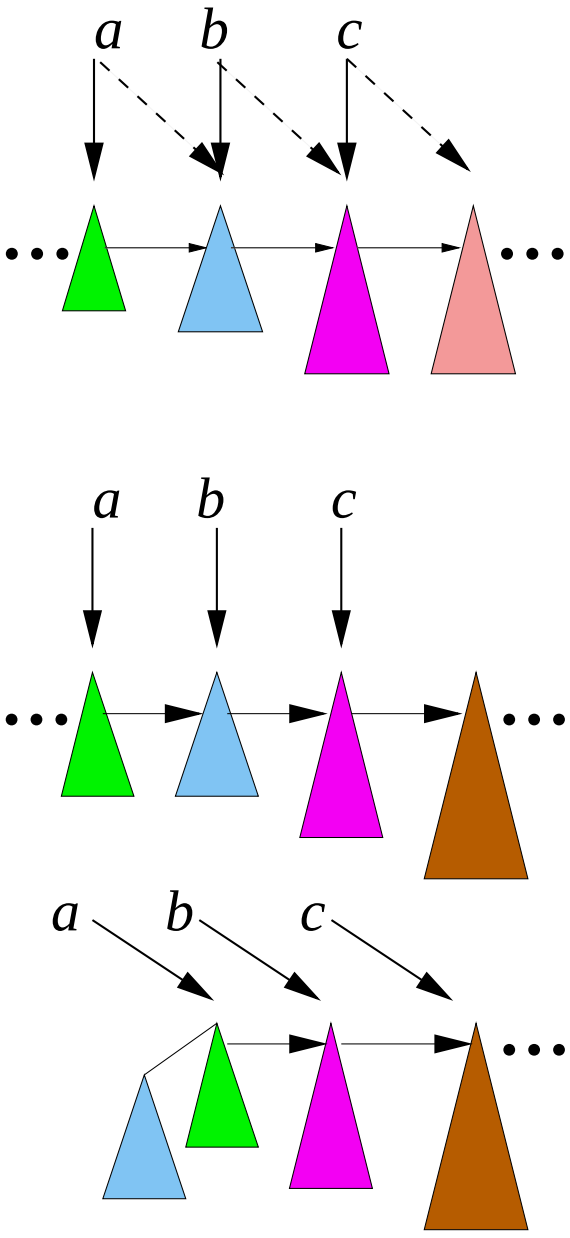
We will leave behind the tree at a and move each of the three pointers to the next tree.

Case II: $degree[a] = degree[b] = degree[c]$.

The same as Case I.

Case III: $degree[a] = degree[b]$ and either $degree[b] < degree[c]$ or $c = \mathbf{nil}$:

We link the trees at a and b into a new tree and set a to this new tree. Then we move b and c to the next one each.


a
b
c
a
b
c
a
b
c

4. Insertion of a Node.

Suppose we wish to insert a node x into a binomial heap H.

We create a single node heap H' consisting of x and unite H and H'.

5. Removing the Root of a Tree T

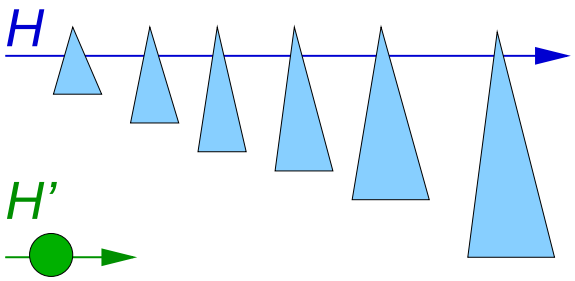
We eliminate T from H and create a heap H' consisting solely of the children of T. Then we unite H and H'.

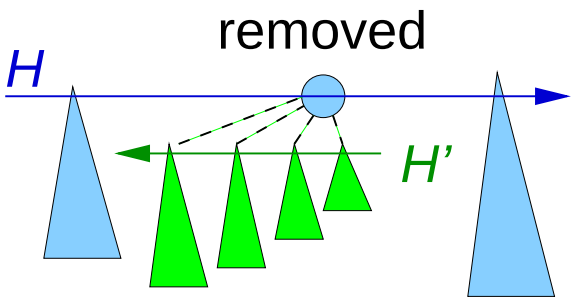
6. Decreasing a Key

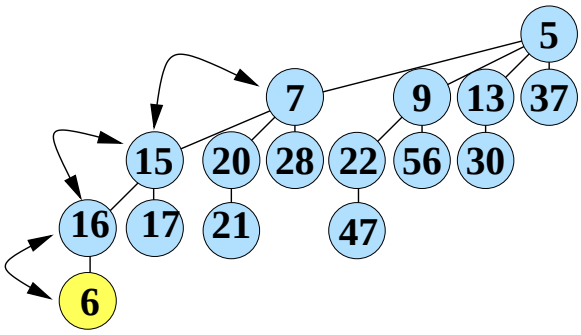
We decrease the key and then keep exchanging the keys upward until violation of the heap property is resolved.

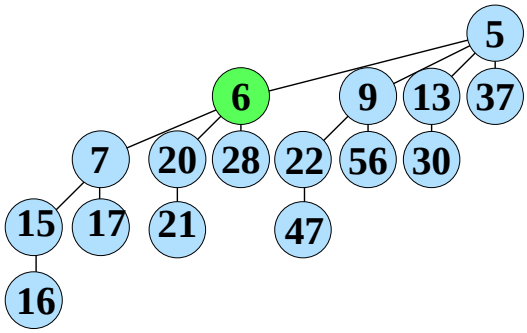
7. Deletion of a Key

We decrease the key to $-\infty$ to move the key to the root position. Then we use the root removal.


H
H'


removed
H
H'


5
7
9
13
37
15
20
28
22
56
30
16
17
21
47
6


5
6
9
13
37
7
20
28
22
56
30
15
17
21
47
16