Lecture 14

Instructor: Karteek Sreenivasaiah

12th October 2018

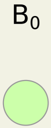
Binomial Tree

is a rooted tree with a recursive construction.

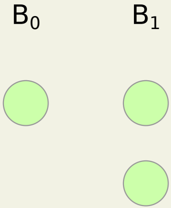
Defined recursively as follows:

- Constructing B_0: Just one node.
- B_k:
  Take two B_{k-1} trees T_1, T_2.
  Make T_1 the leftmost child of the root T_2.

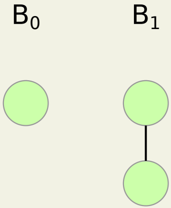
Example



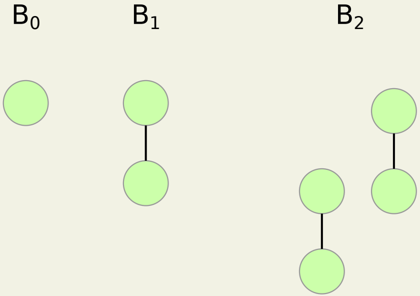
Example



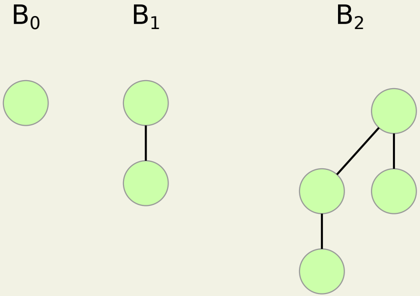
Example



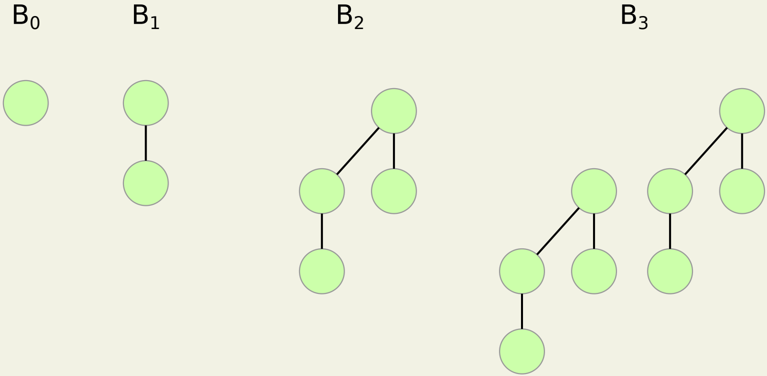
Example



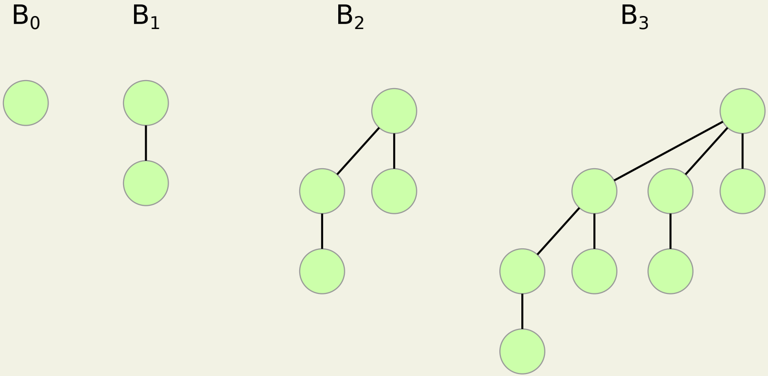
Example



Example

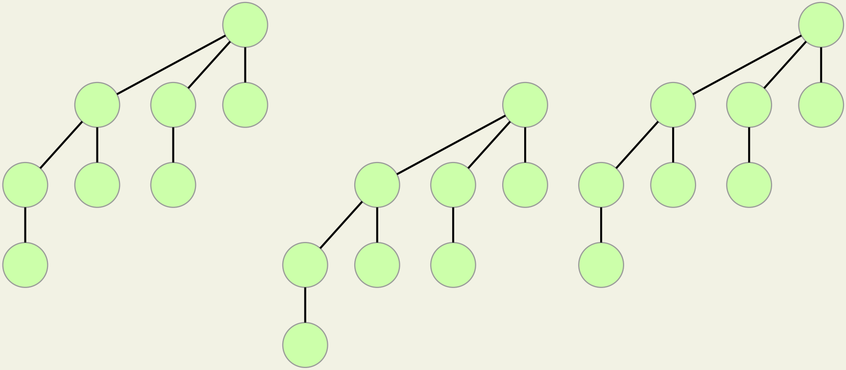


Example



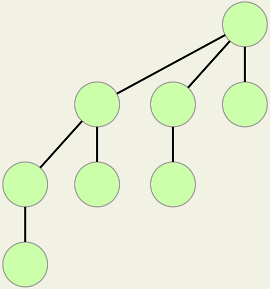
Example

B_3

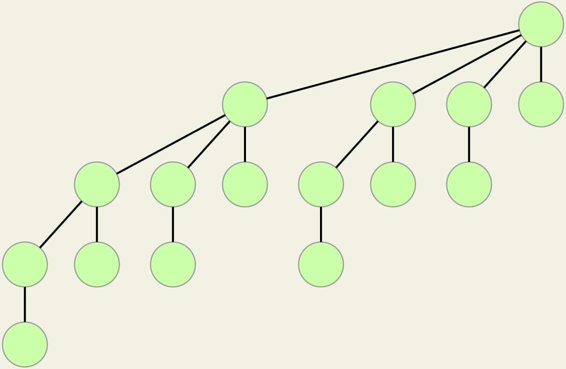


Example

B_3

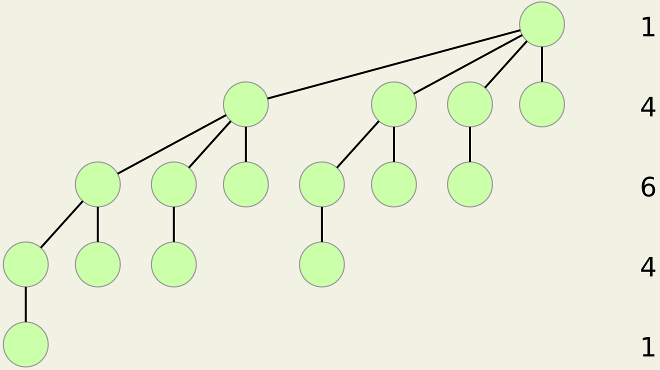


B_4



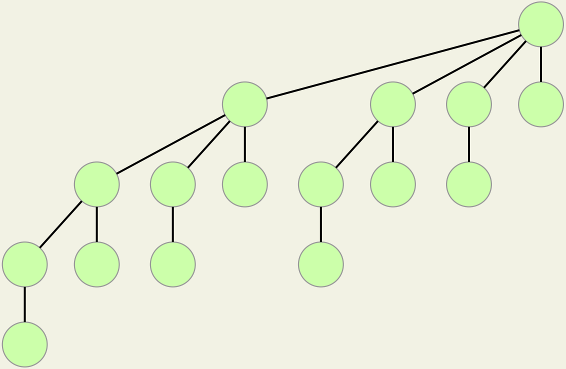
Example

B_4



Example

B_4



$$1 = \binom{4}{0}$$

$$4 = \binom{4}{1}$$

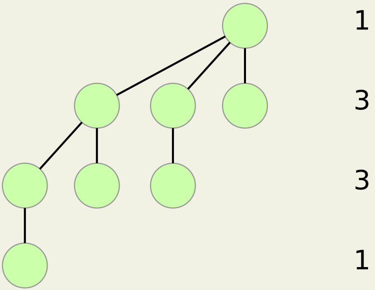
$$6 = \binom{4}{2}$$

$$4 = \binom{4}{3}$$

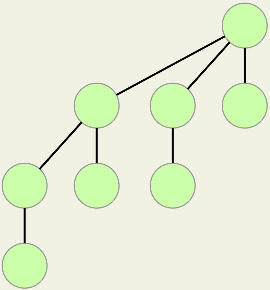
$$1 = \binom{4}{4}$$

Example

B_3



Example



$$1 = \binom{3}{0}$$

$$3 = \binom{3}{1}$$

$$3 = \binom{3}{2}$$

$$1 = \binom{3}{3}$$

Properties of a Binomial Tree

Theorem 1

The ℓ'th layer of the Binomial Tree B_n has $\binom{n}{\ell}$ nodes.

Here we are assuming the root is in layer $\ell = 0$.

Properties of a Binomial Tree

Theorem 1

The ℓ'th layer of the Binomial Tree B_n has $\binom{n}{\ell}$ nodes.

Here we are assuming the root is in layer $\ell = 0$.

Corollary 1

The Binomial Tree B_n has exactly 2^n nodes.

Properties of a Binomial Tree

Observation 1

Children of the root of B_k from right to left look like:
$B_0, B_1, \ldots, B_{k-1}$.

Binomial Heap

- A forest of Binomial Trees.
- Each Binomial Tree has the min-heap property.
- For any degree, at most one Binomial Tree of that degree.

Binomial Heap

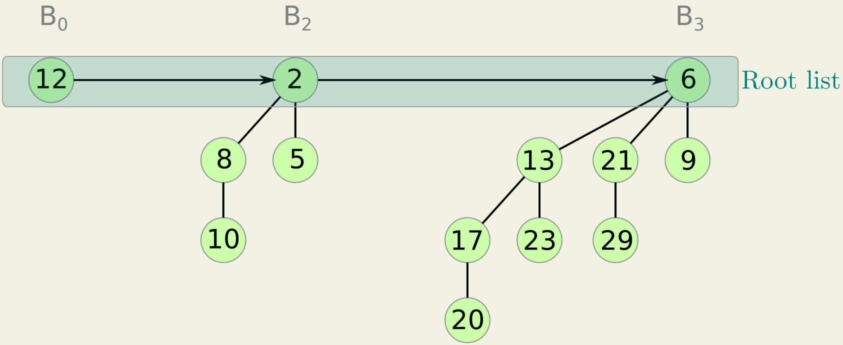
- A forest of Binomial Trees.
- Each Binomial Tree has the min-heap property.
- For any degree, at most one Binomial Tree of that degree.

Supports:

- Insert
- Decrease Key
- Return-Min
- Extract-Min
- Delete Key

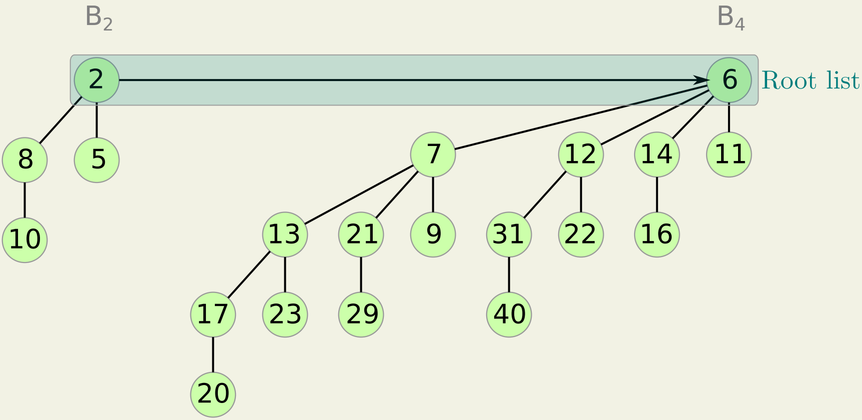
Example

Binomial Heap to store 13 nodes.



Example

Binomial Heap to store 20 nodes.



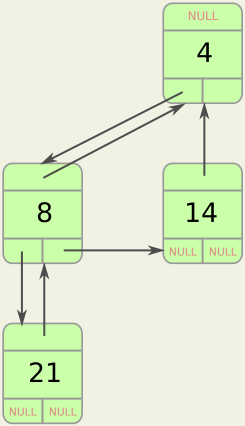
Implementation

- The roots of each Binomial Tree form a linked list
- Each Binomial Tree is stored in "left child - right sibling" representation.
- Maintain min-heap property in each Binomial Tree.

Implementation

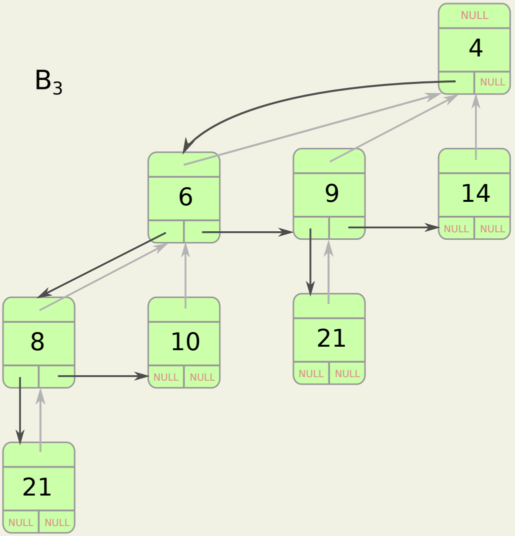
Left child - right sibling representation

B_2



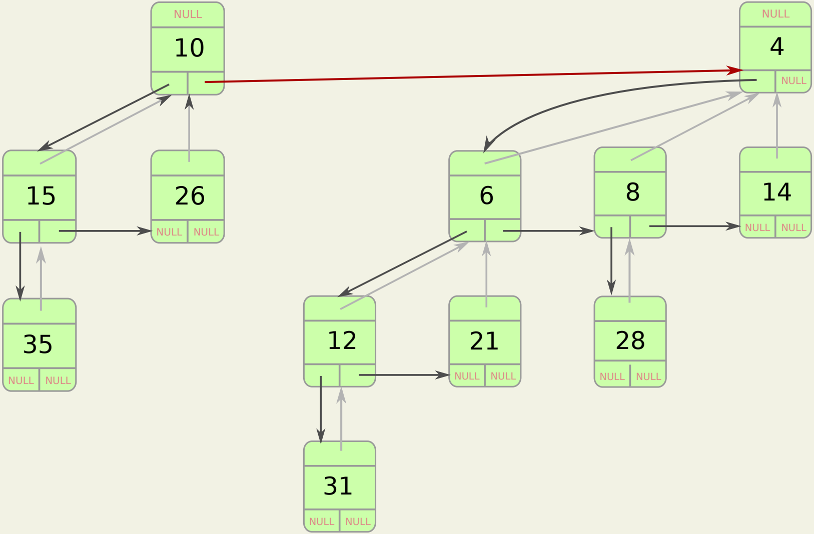
Implementation

Left child - right sibling representation



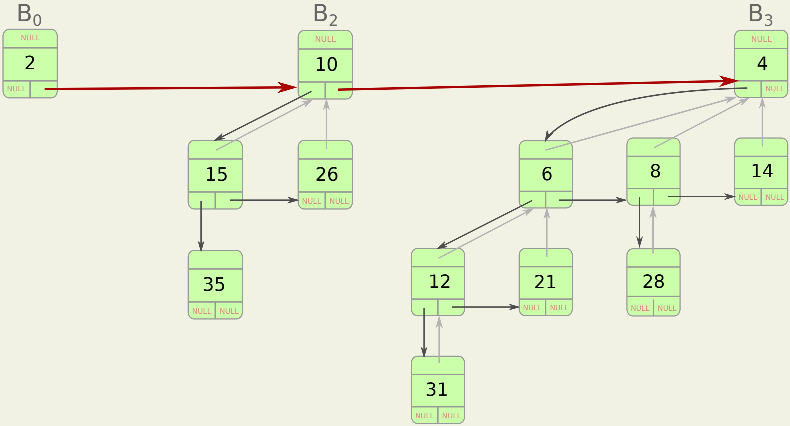
Implementation

Left child - right sibling representation with root list



Implementation

Left child - right sibling representation with root list



Properties of a Binomial Heap

Theorem 1

A Binomial Heap with n nodes has $O(\log n)$ many Binomial Trees.

Proof

Let the binary representation of n be

$$n = b_{\log n}\, b_{\log n-1} \cdots\; b_0$$

A BinHeap with n nodes has the tree B_k if and only if $b_k = 1$. Hence a Binomial Heap with n nodes has precisely as many Trees as the number of 1s in the binary representation of n.

Return-Min

To return the minimum element in a Binomial Heap:

- Walk through the root list.
- Return the minimum value seen.

Note: Since each Binomial Tree present is a min-heap, the roots contain the minimum element of their respective tree.

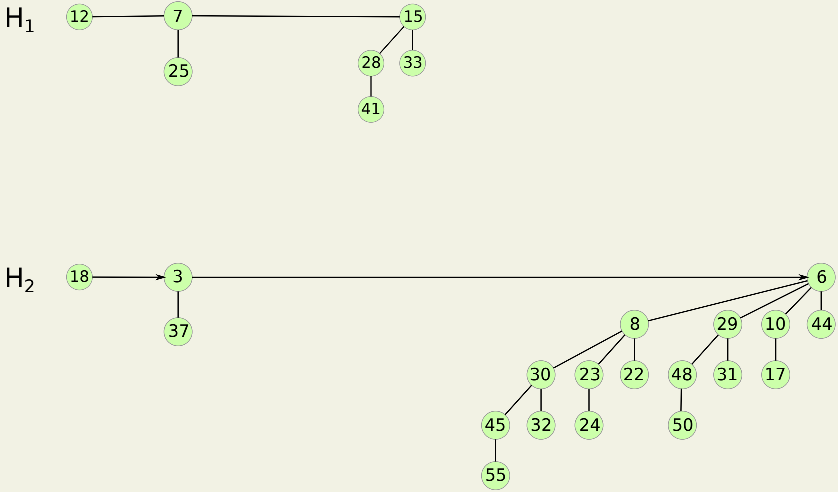
Union

Union of two Binomial Heaps H_1 and H_2 is the most important procedure of all. It works as follows:

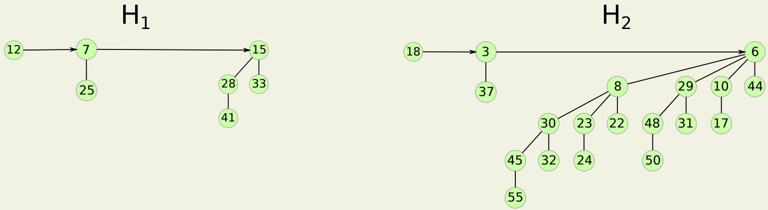
- Merge H_1 and H_2 based on degree of root.
- Fix the merged list to correct double instances of same degree.

Union

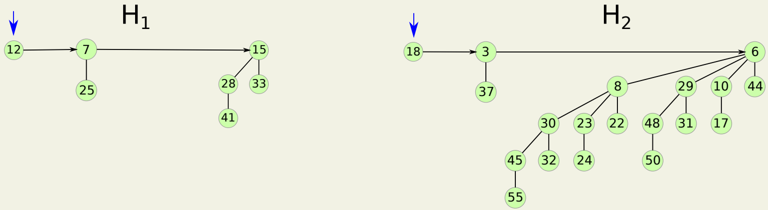
Heap H_1 has 7 nodes and H_2 has 19 nodes.



Union

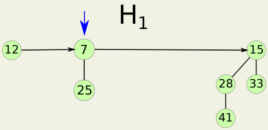

H1
12
7
15
25
28
33
41
H2
18
3
6
37
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55

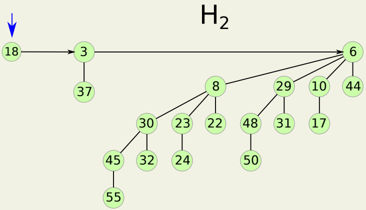
Union



Merge by degree:

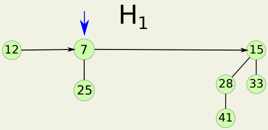
Union

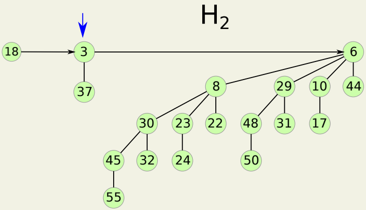




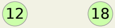
Merge by degree:

Union

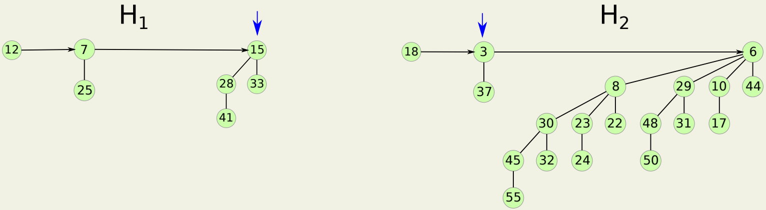




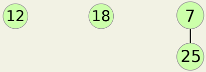
Merge by degree:



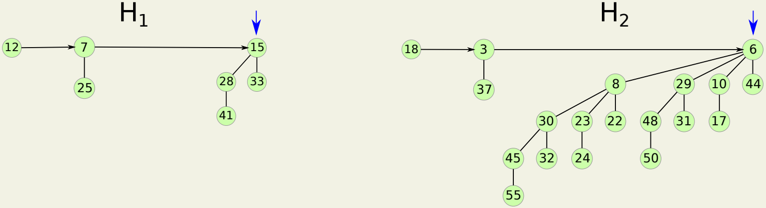
Union



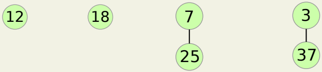
Merge by degree:



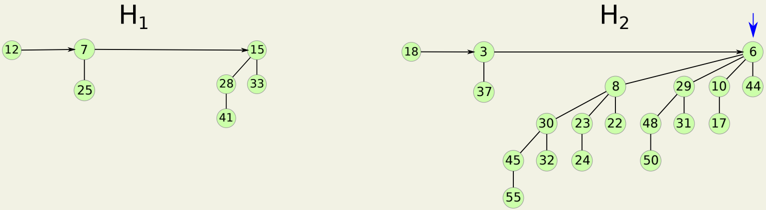
Union



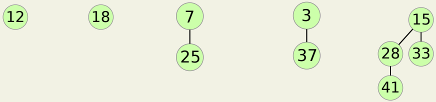
Merge by degree:



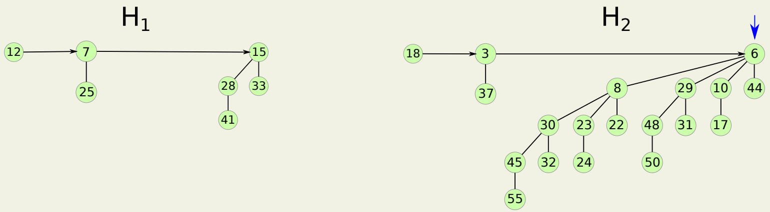
Union



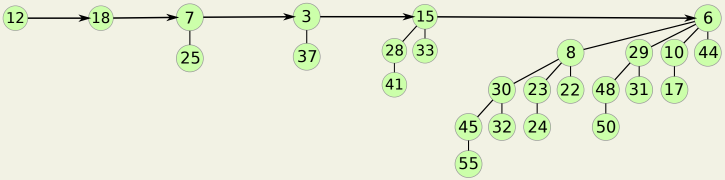
Merge by degree:



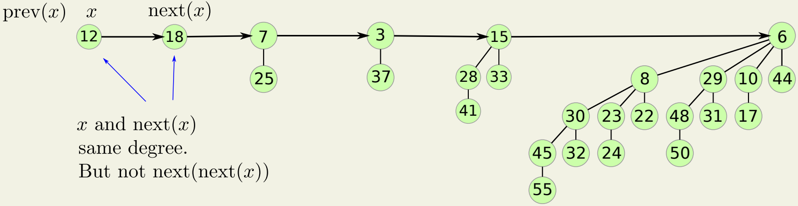
Union



Merge by degree:

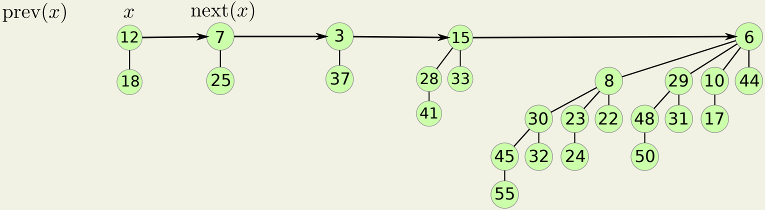


Union

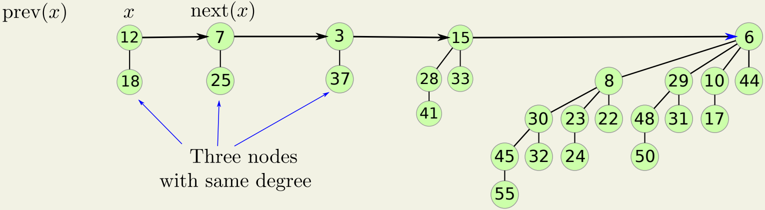


prev(x)
x
next(x)
12
18
7
25
3
37
15
28
33
41
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
x and next(x)
same degree.
But not next(next(x))

Union

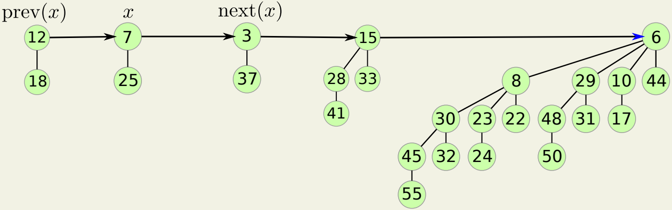

prev(x)
x
next(x)
12
18
7
25
3
37
15
28
33
41
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55

Union

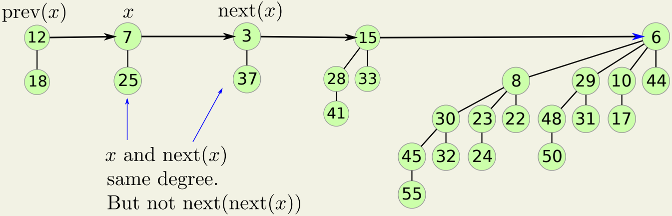


prev(x)
x
next(x)
12
7
3
15
6
18
25
37
28
33
41
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
Three nodes
with same degree

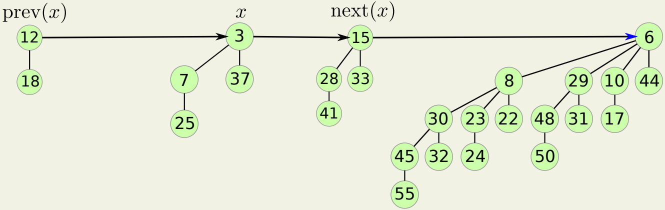
Union


prev(x)
x
next(x)
12
7
3
15
6
18
25
37
28
33
8
29
10
44
41
30
23
22
48
31
17
45
32
24
50
55

Union

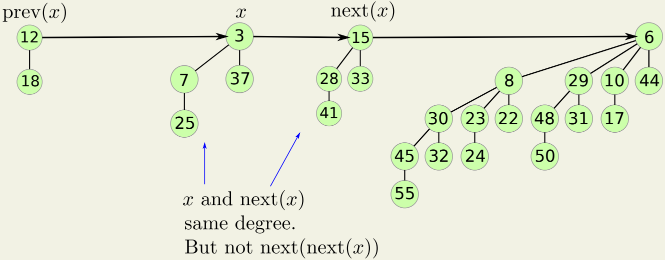

prev(x)
x
next(x)
12
7
3
15
6
18
25
37
28
33
8
29
10
44
41
30
23
22
48
31
17
45
32
24
50
55
x and next(x)
same degree.
But not next(next(x))

Union

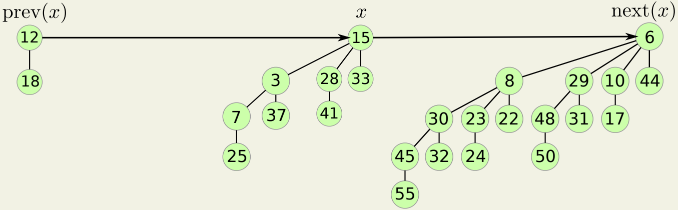


prev(x)
x
next(x)
12
18
3
7
37
25
15
28
33
41
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55

Union



Union



prev(x)
x
next(x)
12
18
15
3
28
33
7
37
41
25
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55

Union procedure

Primarily three cases:

1. Node x and next(x) have different degree:
   Move pointers down the list.
2. Nodes x, next(x) and next(next(x)) have same degree:
   Move pointers down the list.
3. Nodes x, next(x) have same degree, but not next(next(x)):
   Join* trees rooted at x and next(x) to get a bigger Binomial Tree.

Note: Join the bigger key as child of smaller key.

Insert into Binomial Heap

To insert x into a Binomial heap H.

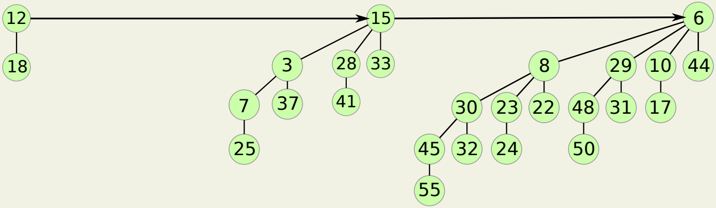
- Create new Binomial heap H' with just x
- Call Union on H and H'.

Extract-Min

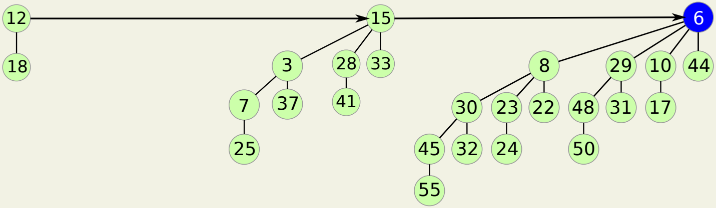
Extract-Min from a Binomial heap H is as follows:

1. Find the tree T that contains the minimum root in the root list.
2. Disconnect T from H.
3. Remove the root.
4. Create new heap from the children:
   - From right to left, link nodes to create the new root list.
5. Call Union on H and T.

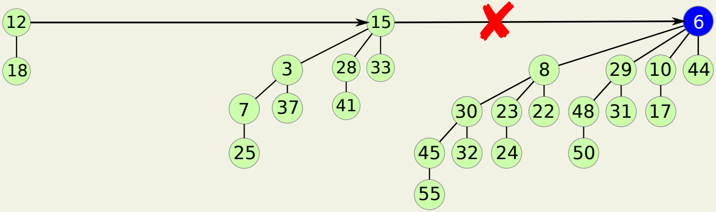
Extract-Min



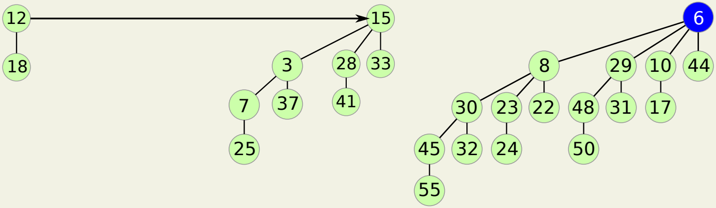
Extract-Min



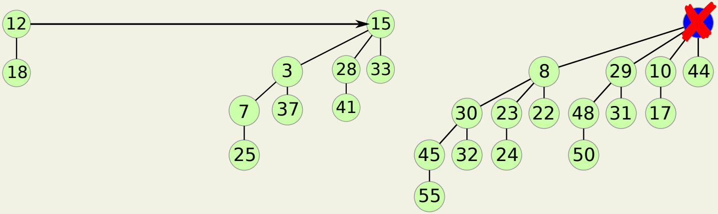
Extract-Min



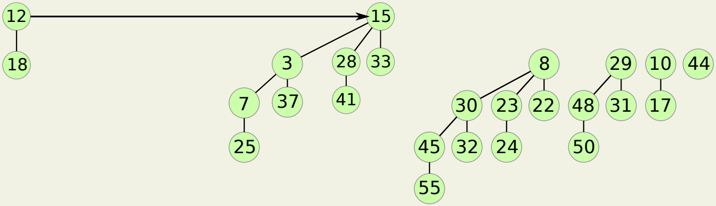
Extract-Min



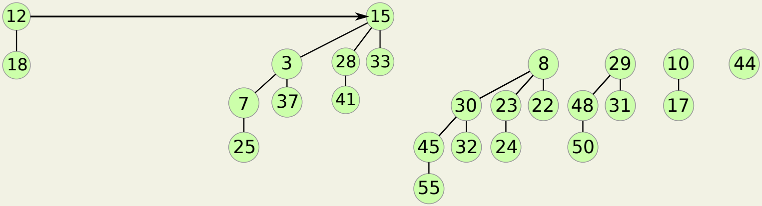
Extract-Min



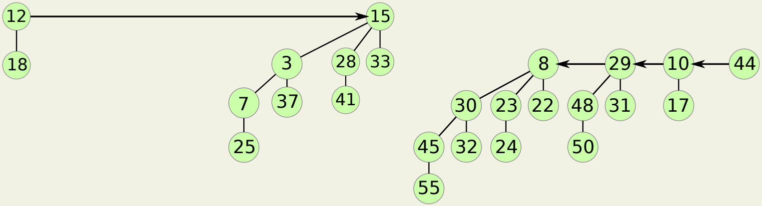
Extract-Min



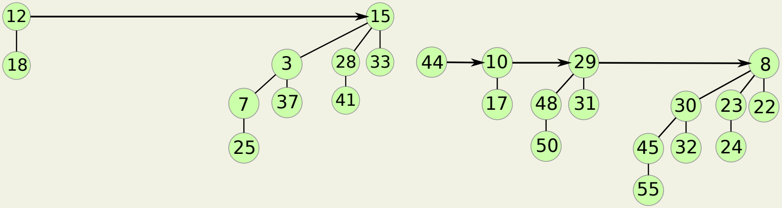
Extract-Min



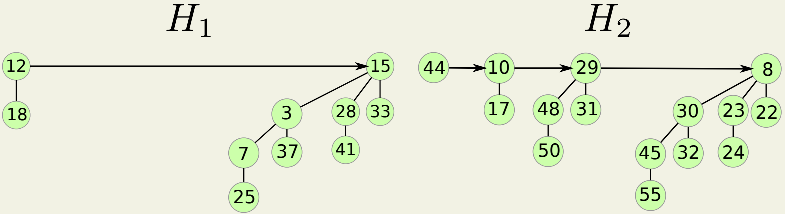
Extract-Min



Extract-Min

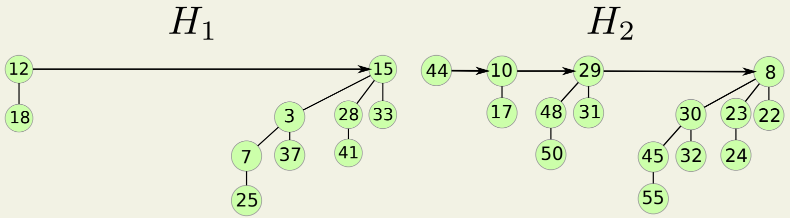


Extract-Min



H_1
H_2
12
18
15
3
28
33
7
37
41
25
44
10
17
29
48
31
50
8
30
23
22
45
32
24
55

Extract-Min



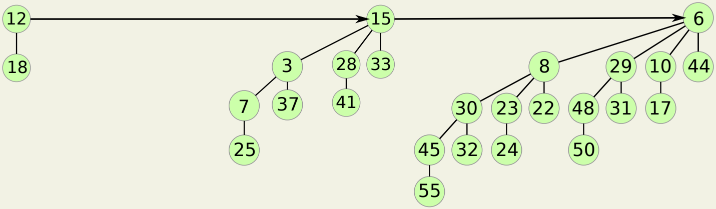
Call $\texttt{Union}(H_1, H_2)$

Decrease-Key from Binomial Heap

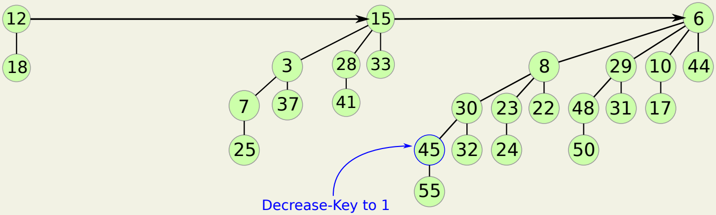
Decrease-Key(x, v) decreases the key of node x to new value v:

- If v is larger than key in x, return error.
- Else, assign new value v in node x
- Check if key in parent of x is smaller.
  - If yes, stop.
  - Else, swap keys between x and its parent.
  - Recurse.

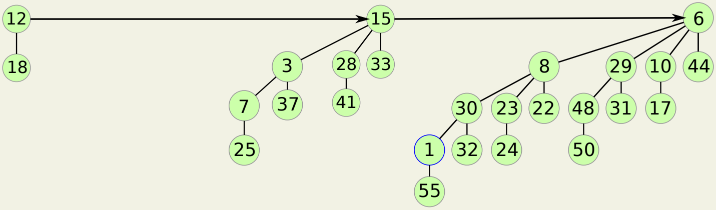
Decrease-Key from Binomial Heap



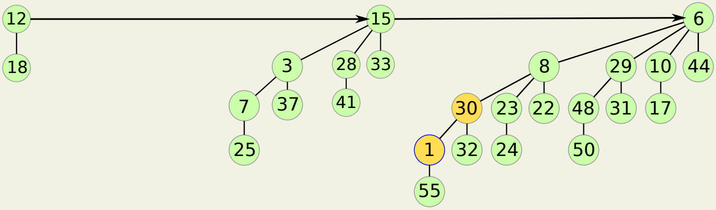
Decrease-Key from Binomial Heap



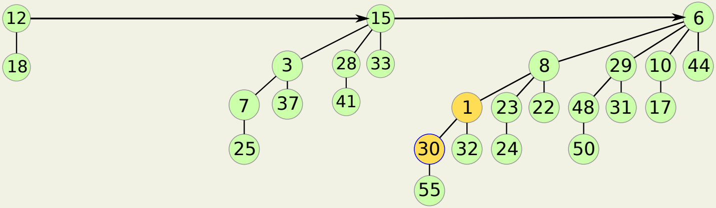
Decrease-Key from Binomial Heap



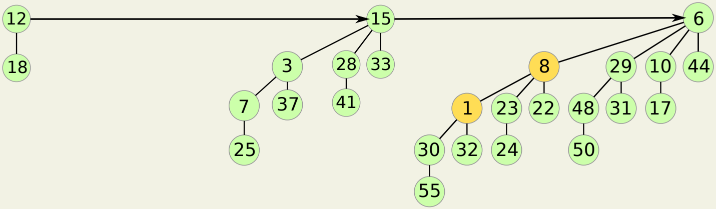
Decrease-Key from Binomial Heap



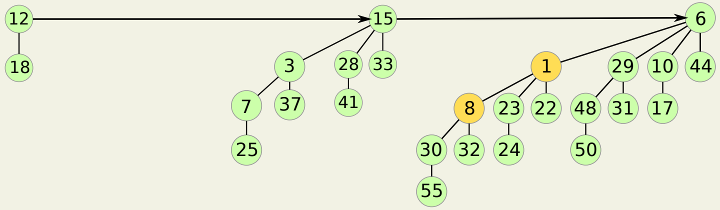
Decrease-Key from Binomial Heap



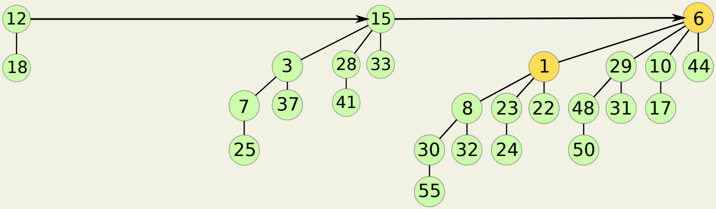
Decrease-Key from Binomial Heap



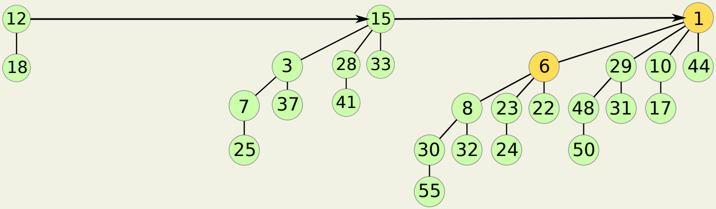
Decrease-Key from Binomial Heap



Decrease-Key from Binomial Heap



Decrease-Key from Binomial Heap



Delete from Binomial Heap

To delete a node x from a Binomial heap H.

- Decrease-Key of x to $-\infty$.
- Call Extract-Min.

Exercises

- Study pseudocode of all procedures from CLRS (2nd ed)
- Study running time of all procedures.

Thank You