Binomial Heaps

CLRS, Chapters 6, 19

Thanks to Kevin Wayne for the slides.

Priority Queues (heaps)

Supports the following operations.

- Insert element x.
- Return min element.
- Return and delete minimum element.
- Decrease key of element x to k.

Applications.

- Dijkstra's shortest path algorithm.
- Prim's MST algorithm.
- Event-driven simulation.
- Huffman encoding.
- Heapsort.
- . . .

Heaps in Action

Dijkstra's Shortest Path Algorithm

```
HeapInit() /*Initialized Priority Queue */
for each v ∈ V
   d[v] ← ∞ ;
   HeapInsert(v)

d[s] ← 0
while (!HeapIsEmpty())
   u = Heap_Extract_Min()
   for each v ∈ V(G) s.t (u,v) ∈ E(G)
      if d[v] > d[u] + w(u,v)
         Heap_Decrease(v, d[u] + w(v,w))
```


Priority Queues

	Heaps				
Operation	**Linked List**	**Binary**	**Binomial**	**Fibonacci ***	**Relaxed**
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

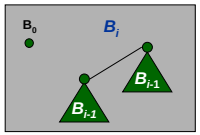
Dijkstra/Prim
1 make-heap
|V| insert
|V| delete-min
|E| decrease-key

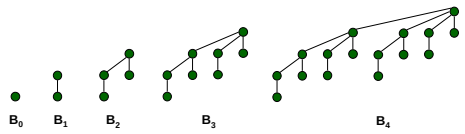
$O(|V|^2)$ $O(|E| \log |V|)$ $O(|E| + |V| \log |V|)$

Binomial Tree

Binomial tree - definition.

- Recursive definition:
- B_i is created from two B_{i-1} by connect the root of the larger key is a child of the second.
- So the degree could be rather large.

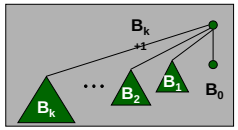




Binomial Tree

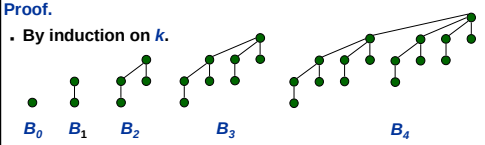
Useful properties of order k binomial tree B_k.

- Number of nodes = 2^k.
- Height = k.
- Degree of root = k.
- Deleting root yields binomial trees $B_{k-1}, \ldots, B_0$.



Proof.

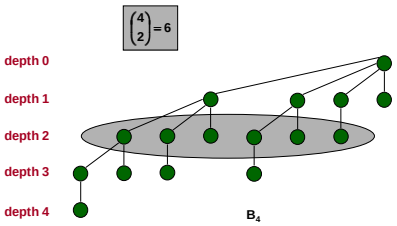
- By induction on k.



Binomial Tree

A property useful for naming the data structure.

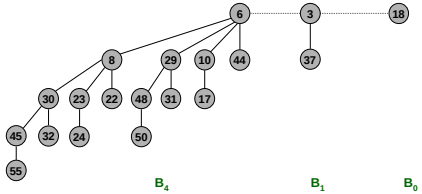
- **B_k has $\binom{k}{i}$ nodes at depth i.**



Binomial Heap

Binomial heap. Vuillemin, 1978.

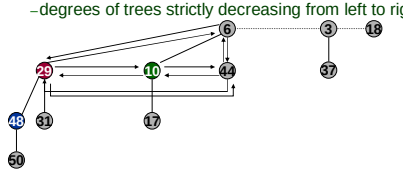
- **Sequence of binomial trees that satisfy binomial heap property.**
  - **each tree is min-heap ordered (parent ≤ each child)**
  - **0 or 1 binomial tree of order *k***



Binomial Heap: Implementation

Implementation.

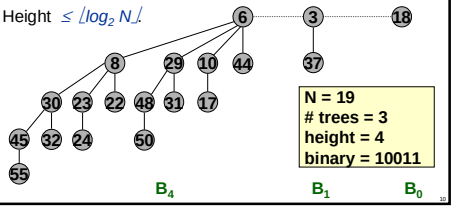
- Represent trees using pointers to
  - left-sibling, right-sibling, parent, a child.
    - (need a pointer only to one child)
    - All siblings are in a doubly connected list, called sibling list (so it is enough to point to only one of them)
- Roots of trees connected with linked list.
  - degrees of trees strictly decreasing from left to right



Binomial Heap: Properties

Properties of N-node binomial heap.

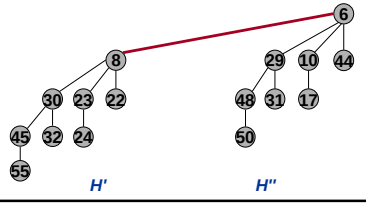
- Min key contained in root of $B_0, B_1, \ldots, B_k$.
- Contains binomial tree B_i iff $b_i = 1$ where $b_n \cdot b_2 b_1 b_0$ is binary representation of N.
- At most $\lfloor \log_2 N \rfloor + 1$ binomial trees.
- Height $\leq \lfloor \log_2 N \rfloor$.



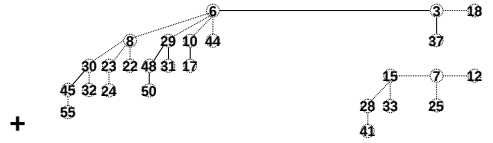
Binomial Heap: Union

Create heap *H* that is union of heaps *H'* and *H''*.

- **"Mergeable heaps."**
- **Easy if *H'* and *H''* are each order *k* binomial trees.**
  - **connect roots of *H'* and *H''***
  - **choose smaller key to be root of *H***



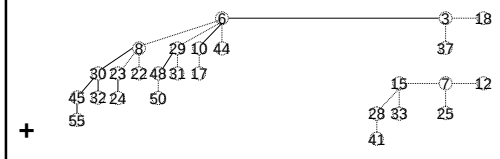
Binomial Heap: Union



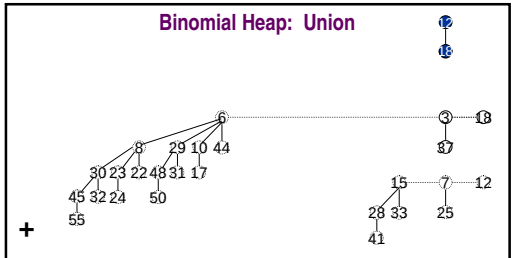
19 + 7 = 26

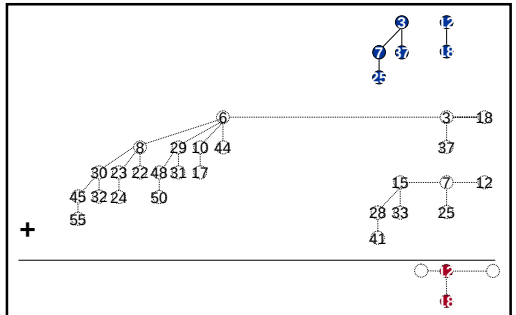
		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

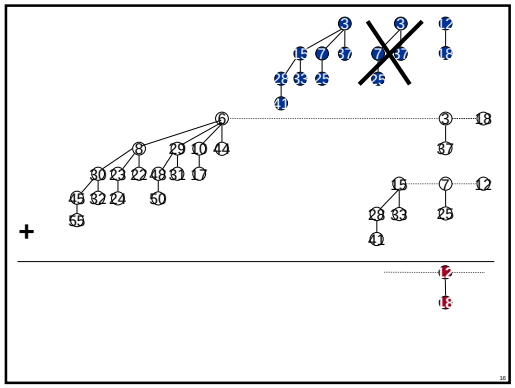
Binomial Heap: Union

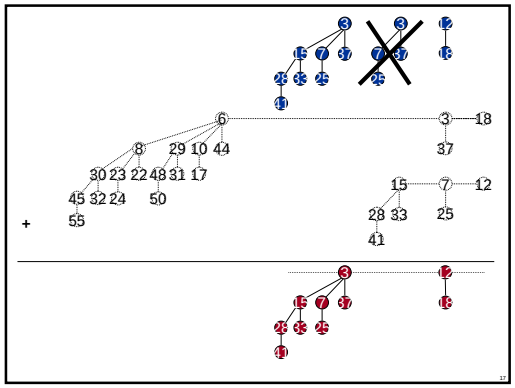


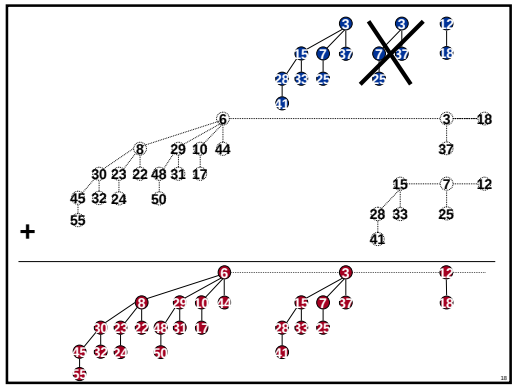
Binomial Heap: Union











Binomial Heap: Union

Create heap H that is union of heaps *H'* and *H''*.

- **Analogous to binary addition.**

Running time. O(log *N*)

- **Proportional to number of trees in root lists ≤ 2(⌊log$_2$ *N*⌋ + 1).**

19 + 7 = 26

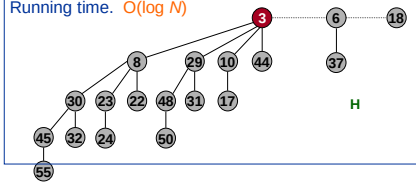
		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

Binomial Heap: Delete Min

Delete node with minimum key in binomial heap *H*.

- Find root *x* with min key in root list of *H*, and delete
- *H'* ← broken binomial trees
- *H* ← Union(*H'*, *H*)

Running time. O(log *N*)

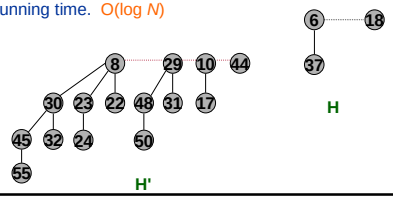


Binomial Heap: Delete Min

Delete node with minimum key in binomial heap *H*.

- Find root *x* with min key in root list of *H*, and delete
- *H'* ← broken binomial trees
- *H* ← Union(*H'*, *H*)

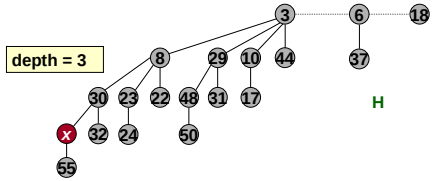
Running time. O(log *N*)



Binomial Heap: Decrease Key

Decrease key of node x in binomial heap *H*.

- Suppose *x* is in binomial tree B_k.
- Bubble node *x* up the tree if *x* is too small.
- Running time: O(log *N*)
- Proportional to depth of node *x* ≤ ⌊log$_2$ *N*⌋.



Binomial Heap: Delete

Delete node x in binomial heap H.

- **Decrease key of x to -∞.**
- **Delete min.**

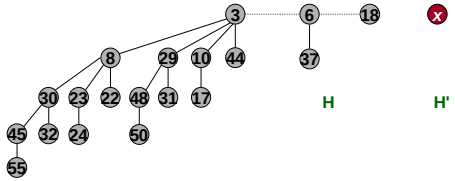
Running time. O(log N)

Binomial Heap: Insert

Insert a new node x into binomial heap *H*.

- ***H'*** ← **MakeHeap(*x*)**
- ***H*** ← **Union(*H'*, *H*)**

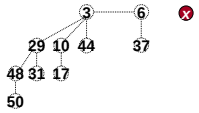
Running time. O(log N)



Binomial Heap: Sequence of Inserts

Insert a new node x into binomial heap H.

- If N = `.......0`, then only 1 steps.
- If N = `......01`, then only 2 steps.
- If N = `.....011`, then only 3 steps.
- If N = `....0111`, then only 4 steps.



Inserting 1 item can take $\Omega(\log N)$ time.

- If N = `11...111`, then $\log_2 N$ steps.

But, inserting sequence of N items takes $O(N)$ time!

- $(N/2)(1) + (N/4)(2) + (N/8)(3) + \ldots \leq 2N$
- Amortized analysis.
- Basis for getting most operations down to constant time.

$$\sum_{n=1}^{N} \frac{n}{2^n} = 2 - \frac{N}{2^N} - \frac{1}{2^{N-1}} \leq 2$$

Priority Queues

		Heaps			
Operation	**Linked List**	**Binary**	**Binomial**	**Fibonacci ***	**Relaxed**
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

just did this