

Binomial Heaps

Devin Delfino

Comp 401: Senior Seminar

3/05/2014

Outline

1. Binomial Trees
2. Binomial Heaps
3. Implementation Structure
4. Standard Functions
5. Uses of Binomial Heaps

Outline

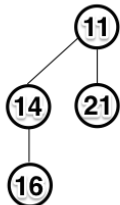
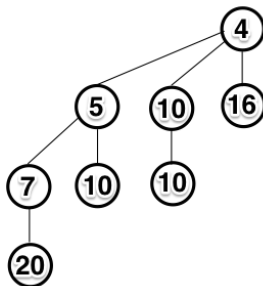
1. Binomial Trees
2. Binomial Heaps
3. Implementation Structure
4. Standard Functions
5. Uses of Binomial Heaps

Binomial Trees

- A **Binomial Tree** is a specific type of tree that includes the following specifications:
 1. The **order** or **rank** of the binomial tree is the number of children of the root node.
 2. A Binomial Tree of order 0 is a single node.
 3. A Binomial Tree of order k has k child nodes, all of which are the roots of binomial trees of orders $k - 1, k - 2, \dots, 2, 1, 0$ from left to right.

Binomial Trees: Examples

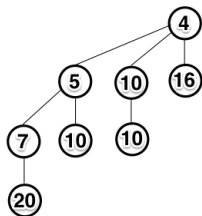
- If a binomial tree has order k , the orders of the k child nodes decrease from left to right from $k - 1$ to 0.

 $k = 2$  $k = 0$  $k = 3$  $k = 1$ 

Binomial Trees

- If a Binomial Tree has an order k :
 1. The tree has 2^k nodes.
 2. The height of the tree is k .
 3. There are $\binom{k}{d}$ nodes at depth d .
- $\binom{k}{d} = \frac{k!}{d!(k-d)!}$ is known as the Binomial Coefficient.

Example: $k = 3, d = 2$



$$\binom{k}{d} = \binom{3}{2} = \frac{3!}{2!(3-2)!} = \frac{6}{2 * 1} = \frac{6}{2} = 3$$

Outline

1. Binomial Trees
2. Binomial Heaps
3. Implementation Structure
4. Standard Functions
5. Uses of Binomial Heaps

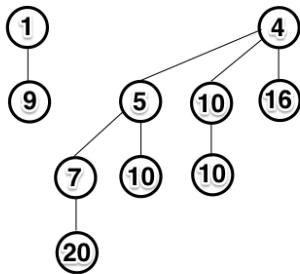
Binomial Heaps

- A **Binomial Heap** is a collection of binomial trees that satisfy the following two binomial heap properties:
 1. The key of any node is greater than or equal to the key of its parent (minimum-heap property).
 2. There cannot be two binomial trees of the same order.

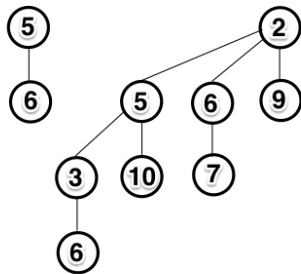
Binomial Heap: Property #1 (minimum-heap)

- The first property (minimum-heap) ensures that the root is the smallest key in each binomial tree.
- The smallest key of the entire heap is one of the roots.

Min-Heap Property ✓



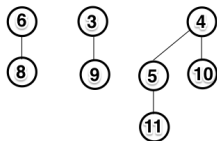
Min-Heap Property ✗



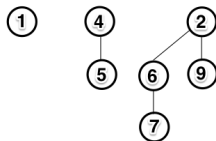
Binomial Heap: Property #2

- The order of each binomial tree must be unique.

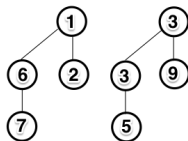
Property #2 ×



Property #2 ✓

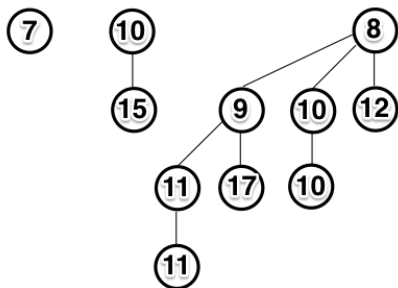


Property #2 ×



Binomial Heap: Property #2, Cont.

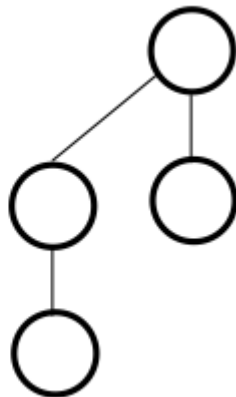
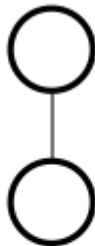
- The second property ensures that if a binomial heap has n nodes, then it will have at most $\lfloor \log n \rfloor + 1$ binomial trees.
- The total number of nodes can also be thought of as a binary string, where each binomial tree represents a bit.



Outline

1. Binomial Trees
2. Binomial Heaps
3. Implementation Structure
4. Standard Functions
5. Uses of Binomial Heaps

Implementation Structure



Outline

1. Binomial Trees
2. Binomial Heaps
3. Implementation Structure
- 4. Standard Functions**
5. Uses of Binomial Heaps

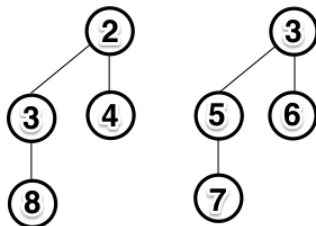
Standard Functions: Merge

- Two k ordered binomial trees are merged into one $k + 1$ ordered binomial tree.

Merge

Input: Node* root_A, Node* root_B

```
if root_A.key ≤ root_B.key then
    root_B.right_sibling ← root_A.child
    root_A.child ← root_B
    root_B.parent ← root_A
    return root_A
else
    root_A.right_sibling ← root_B.child
    root_B.child ← root_A
    root_A.parent ← root_B
    return root_B
end if
```



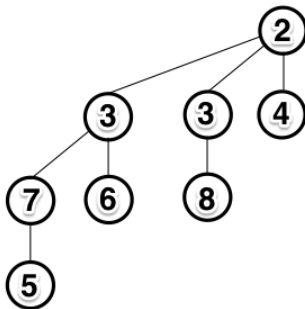
Standard Functions: Merge

- Two k ordered binomial trees are merged into one $k + 1$ ordered binomial tree.

Merge

Input: Node* root_A, Node* root_B

```
if root_A.key ≤ root_B.key then
    root_B.right_sibling ← root_A.child
    root_A.child ← root_B
    root_B.parent ← root_A
    return root_A
else
    root_A.right_sibling ← root_B.child
    root_B.child ← root_A
    root_A.parent ← root_B
    return root_B
end if
```

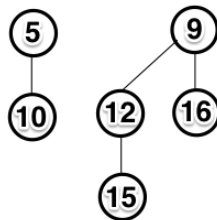
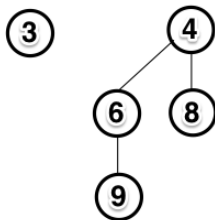


Standard Functions: Join

- Joins one Binomial Heap into another Binomial Heap.

BinomialHeap : Join

Input: Binomialheap merge_heap
 for each *binomialtree* in merge_heap **do**
 this_heap.insert(binomialtree.root)
 end for



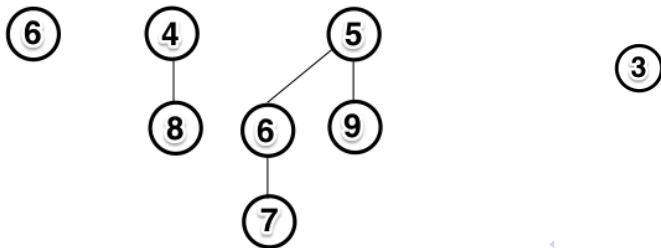
Standard Functions: Insert

- Inserts either an integer or an existing node into a Binomial Heap.

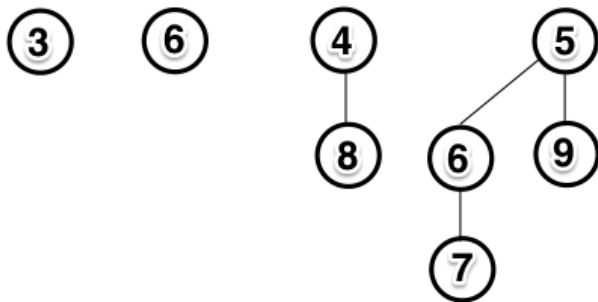
BinomialHeap : Insert

Input: int key

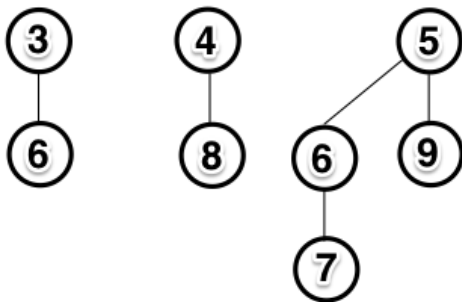
```
BinomialHeap temp_heap ← new BinomialHeap(key)  
this_heap.join(temp_heap)
```



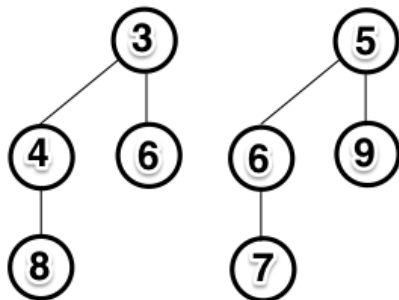
Standard Functions: Insert



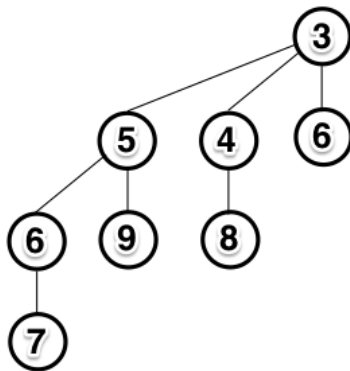
Standard Functions: Insert



Standard Functions: Insert



Standard Functions: Insert



Standard Functions: DeleteMinimum

- Deletes the node with the smallest key from the Binomial Heap.

BinomialHeap : DeleteMinimum

Input: None

```
Node* deleted_node  $\leftarrow$  this_heap.FindMinimum()  
BinomialHeap temp_heap  $\leftarrow$  new BinomialHeap()  
for each child in delete_node do  
    temp_heap.insert(child)  
end for  
this_heap.join(temp_heap)
```

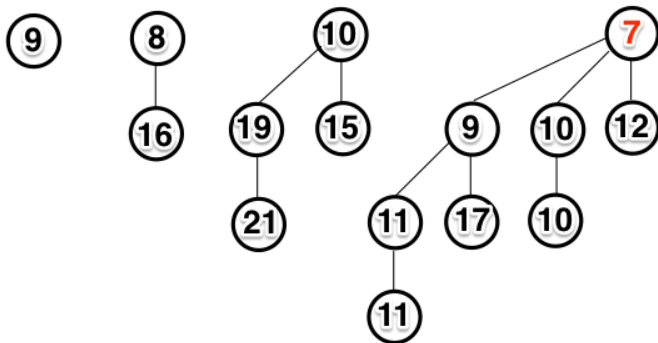
Standard Functions: DeleteMinimum

BinomialHeap : DeleteMinimum

Input: None

Node* deleted_node \leftarrow this_heap.FindMinimum()

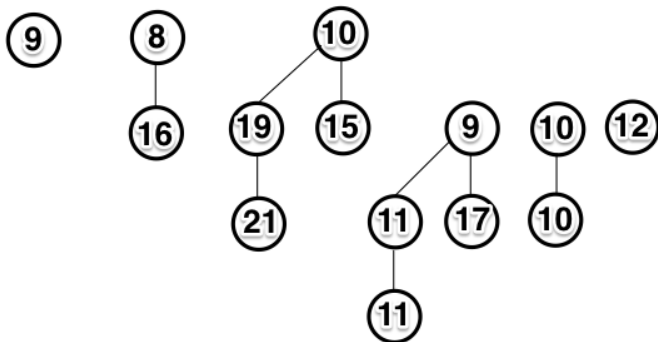
BinomialHeap temp_heap \leftarrow new BinomialHeap() ...



Standard Functions: DeleteMinimum

BinomialHeap : DeleteMinimum, cont.

```
for each child in delete_node do  
    temp_heap.insert(child)  
end for...
```

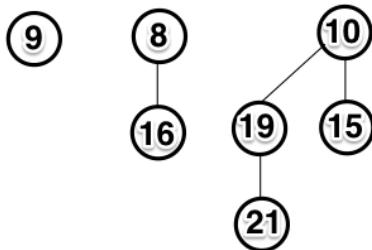


Standard Functions: DeleteMinimum, cont.

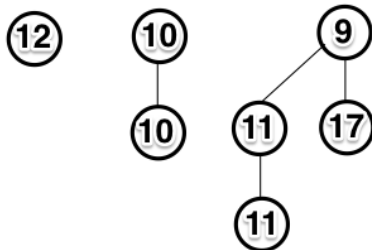
BinomialHeap : DeleteMinimum

`this_heap.join(temp_heap)`

this_heap



temp_heap

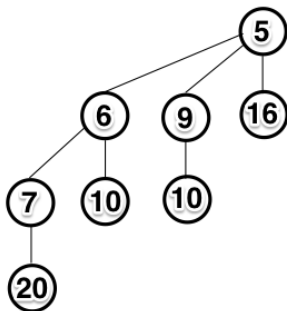


Standard Functions: DecreaseKey

- Decreases the key of a given node in the Binomial Heap

BinomialHeap : DecreaseKey

Input: Node* node, int new_key
decrease_node.key \leftarrow new_key
Node* temp \leftarrow decrease_node
while temp.parent \neq NULL **do**
 if temp.key < temp.parent.key
 then
 swap(temp.key, temp.parent.key)
 else
 break
 end if
 temp \leftarrow temp.parent
end while

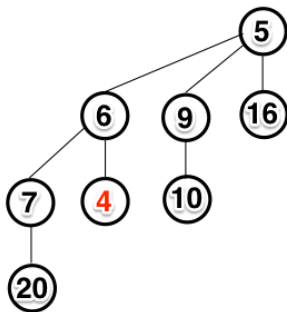


Standard Functions: DecreaseKey

- Decreases the key of a given node in the Binomial Heap

BinomialHeap : DecreaseKey

Input: Node* node, int new_key
decrease_node.key \leftarrow new_key
Node* temp \leftarrow decrease_node
while temp.parent \neq NULL **do**
 if temp.key < temp.parent.key
 then
 swap(temp.key, temp.parent.key)
 else
 break
 end if
 temp \leftarrow temp.parent
end while

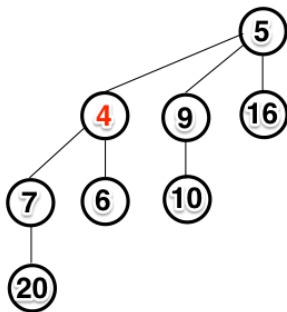


Standard Functions: DecreaseKey

- Decreases the key of a given node in the Binomial Heap

BinomialHeap : DecreaseKey

Input: Node* node, int new_key
decrease_node.key \leftarrow new_key
Node* temp \leftarrow decrease_node
while temp.parent \neq NULL **do**
 if temp.key < temp.parent.key
 then
 swap(temp.key, temp.parent.key)
 else
 break
 end if
 temp \leftarrow temp.parent
end while

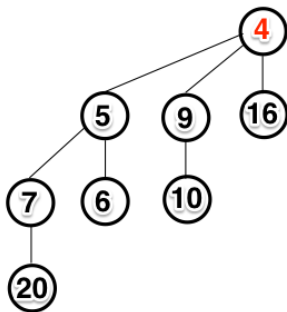


Standard Functions: DecreaseKey

- Decreases the key of a given node in the Binomial Heap

BinomialHeap : DecreaseKey

Input: Node* node, int new_key
decrease_node.key \leftarrow new_key
Node* temp \leftarrow decrease_node
while temp.parent \neq NULL **do**
 if temp.key < temp.parent.key
 then
 swap(temp.key, temp.parent.key)
 else
 break
 end if
 temp \leftarrow temp.parent
end while



Standard Functions: Delete

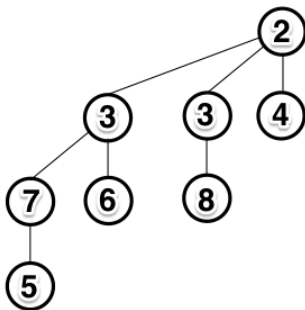
- Deletes a given node from the Binomial Heap

BinomialHeap : Delete

Input: Node* delete_node

BinomialHeap.DecreaseKey(delete_node, -9999)

BinomialHeap.DeleteMinimum()



Outline

1. Binomial Trees
2. Binomial Heaps
3. Implementation Structure
4. Standard Functions
5. Uses of Binomial Heaps

Uses of Binomial Heaps

- Binomial Heaps are used to implement priority queues
- Priority queues can be used in applications that manage network bandwidth or manage events in a discrete even simulation.
- Priority queues can also be used in various algorithms including heapsort, Dijkstra's shortest-path algorithm, Prim's minimum spanning tree algorithm, and Huffman Encoding.

Operation	Linked List	Binary	d-ary	Binomial
FindMinimum	$O(n)$	$O(1)$	$O(1)$	$O(\log n)$
Insert	$O(\log n)$	$O(\log n)$	$O(\log_d n)$	$O(\log n)$
Join	$O(1)$	$O(n)$	$O(n)$	$O(\log n)$
DeleteMinimum	$O(n)$	$O(\log n)$	$O(d \log_d n)$	$O(\log n)$
DecreaseKey	$O(1)$	$O(\log n)$	$O(\log_d n)$	$O(\log n)$
Delete	$O(1)$	$O(\log n)$	$O(d \log_d n)$	$O(\log n)$

References

- ▶ KEVIN WAYNE.
Priority Queues.
<http://www.cs.princeton.edu/wayne/kleinberg-tardos/pdf/BinomialHeaps.pdf>,
2013.
- ▶ UNKNOWN.
Binomial Heap.
http://en.wikipedia.org/wiki/Binomial_heap, 2015.