COMP 250

Lecture 22

priority queue ADT
heaps 1

Oct. 31, 2016

Priority Queue

Assume a set of comparable elements. Sometimes one uses the term "keys" instead. This will make more sense once we discuss Maps a few lectures from now.

Like a queue (or stack), but we have a more general definition of which element to remove next.

e.g. hospital emergency room

several examples in COMP 251

Priority Queue ADT

- add(element)

- removeMin()

  “highest” priority = “number 1” priority

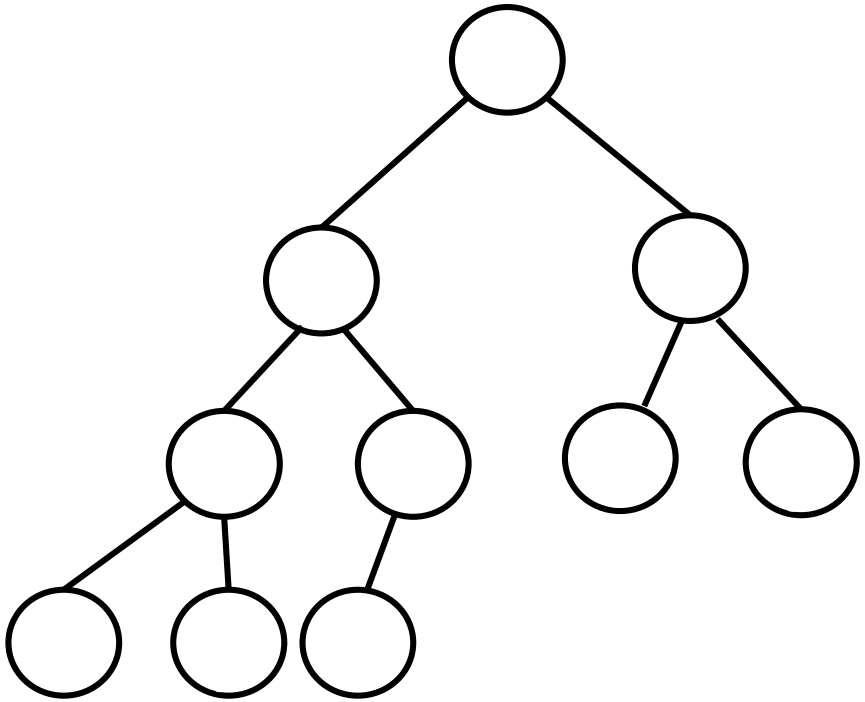
- peek()
- contains(element)
- remove(element)

How to implement a Priority Queue ?

- sorted list ?
- balanced binary search tree (COMP 251) ?
- heap (this week)

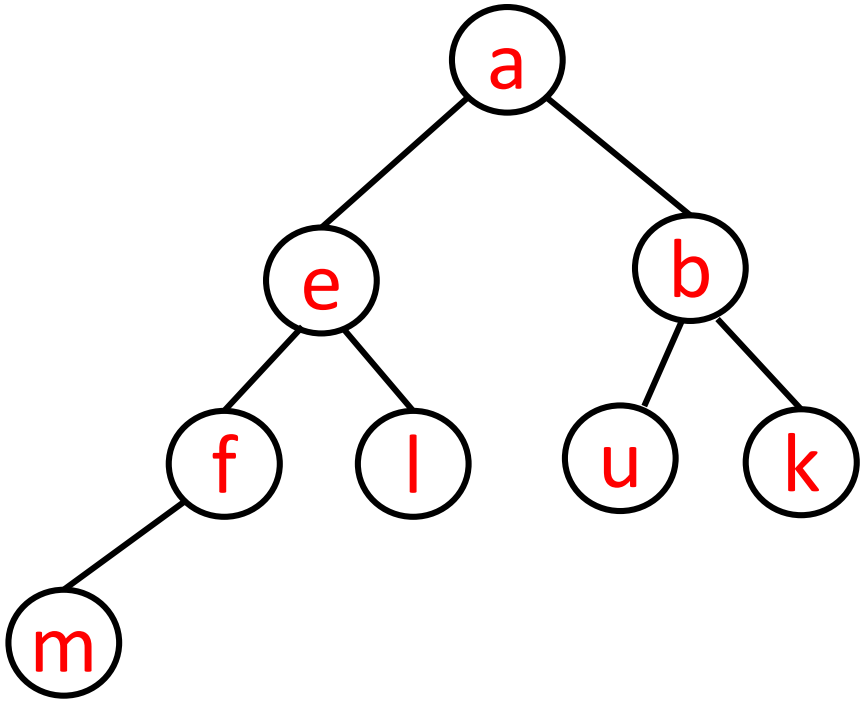
Not the same heap you hear about in COMP 206.

Complete Binary Tree (definition)



Binary tree of height h such that every level less than h is full, and all nodes at level h are as far to the left as possible

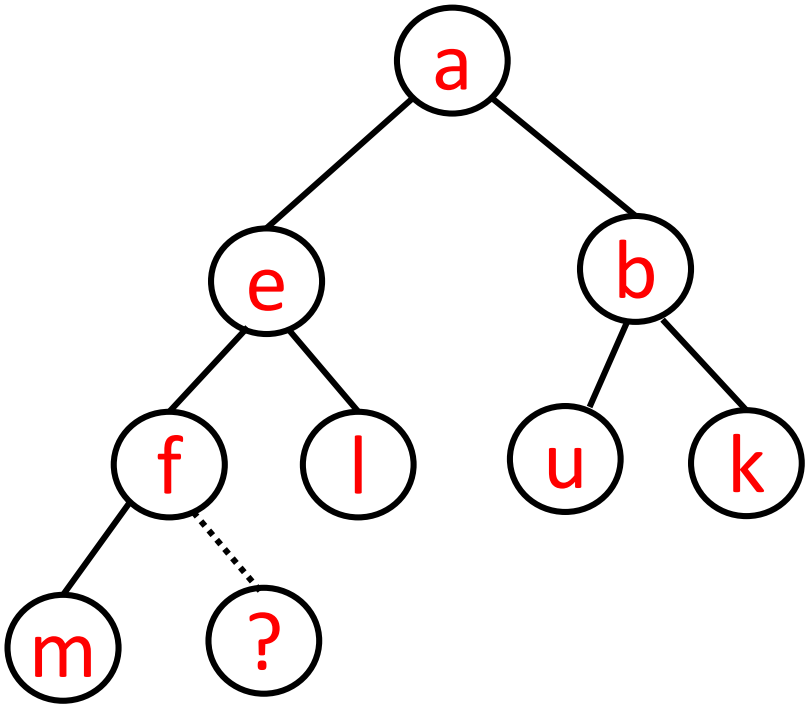
min Heap (definition)



Complete binary tree with (unique) comparable elements, such that each node's element is less than its children's element(s).

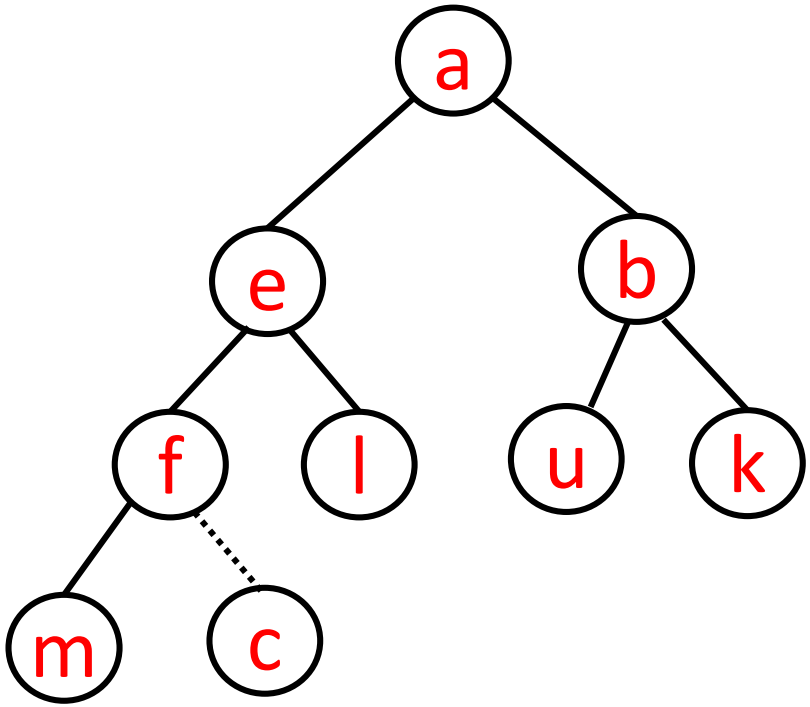
add(element)

e.g. add(c)



add(element)

e.g. add(c)

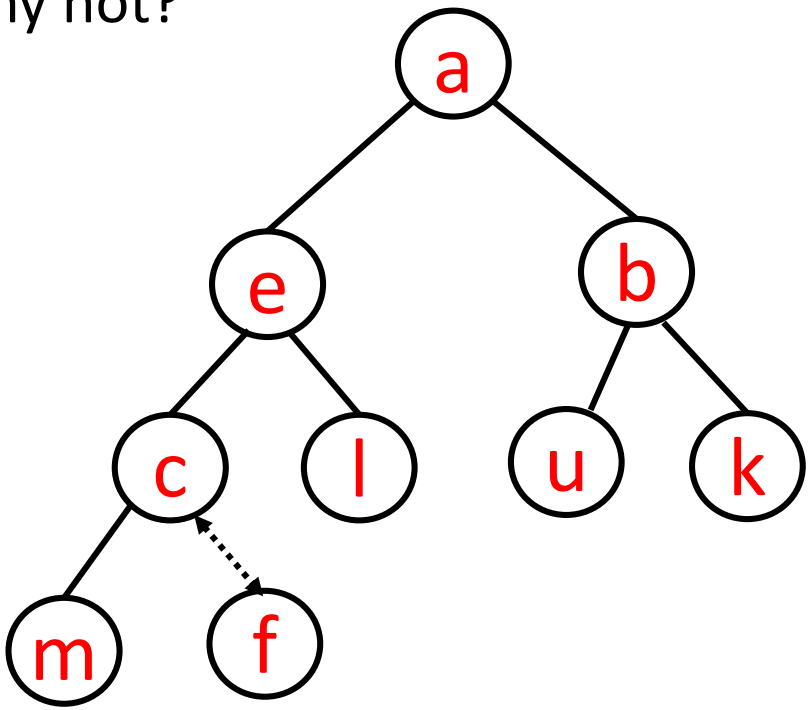


Problem is that adding at the next available slot might destroy the heap property.

We swap c with its parent f.

Q: Can this create a problem with c's former sibling, who is now c's child?

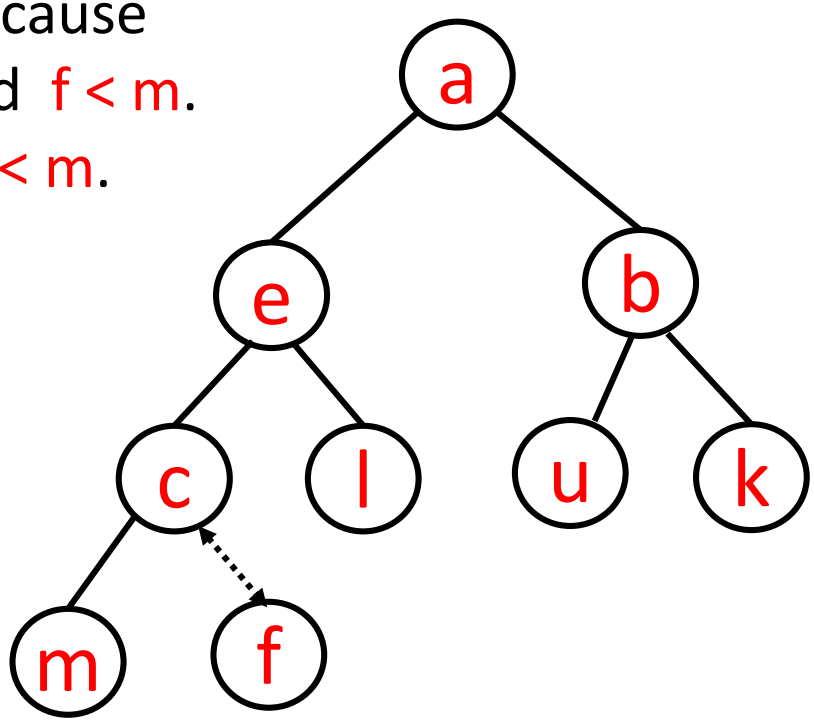
A: No. Why not?



We swap c with its parent f.

Q: Can this create a problem with c's former sibling, who is now c's child?

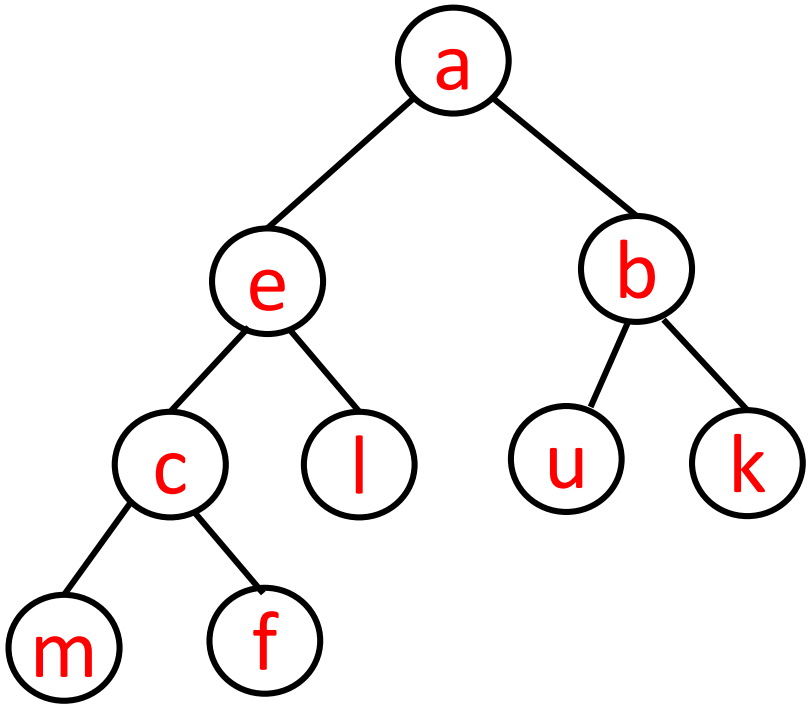
A: No. Because
c < f and f < m.
Thus, c < m.



We swap c with its parent f.

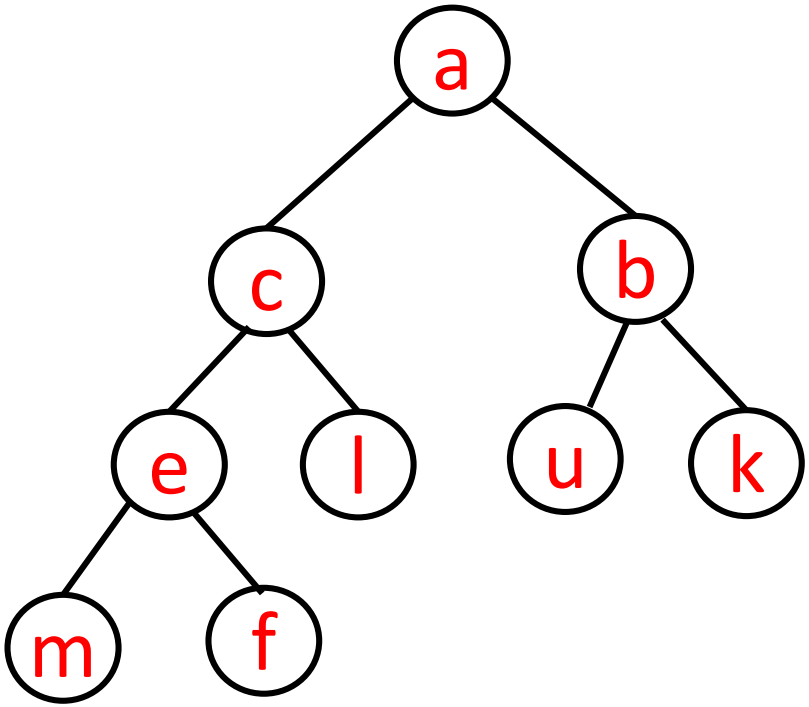
Q: Are we done ?

A: Not necessarily. What about c's parent?



We swap c with its (new) parent e.

Now we are done because c is greater than its parent a



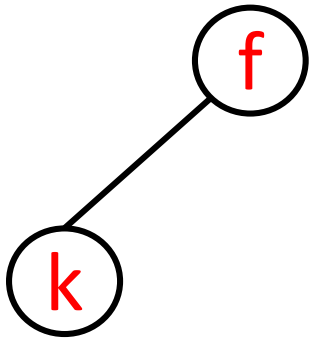
```
add( element ){
    cur = new node at next leaf position
    cur.element = element
    while (cur != root) and (cur.element < cur.parent.element){
       swapElement(cur, parent)
       cur = cur.parent
    }
}
```

How to build a heap?

add(k)
add(f)

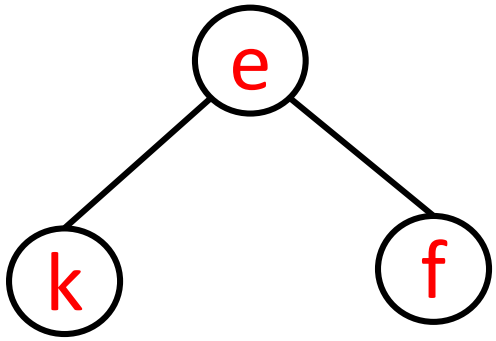
How to build a heap?

add(k)
add(f)
add(e)



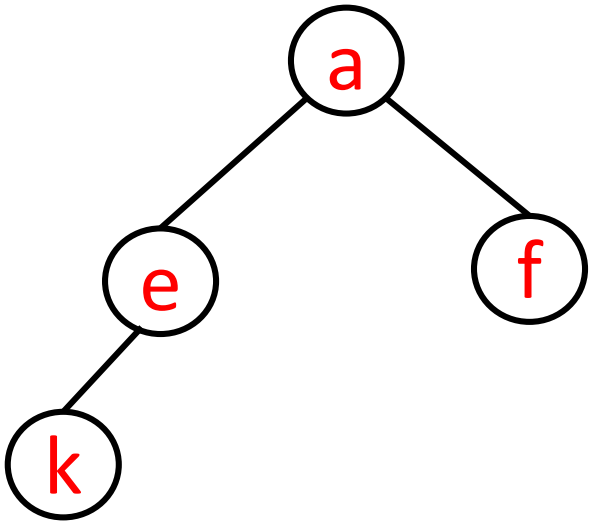
How to build a heap?

add(k)
add(f)
add(e)
add(a)



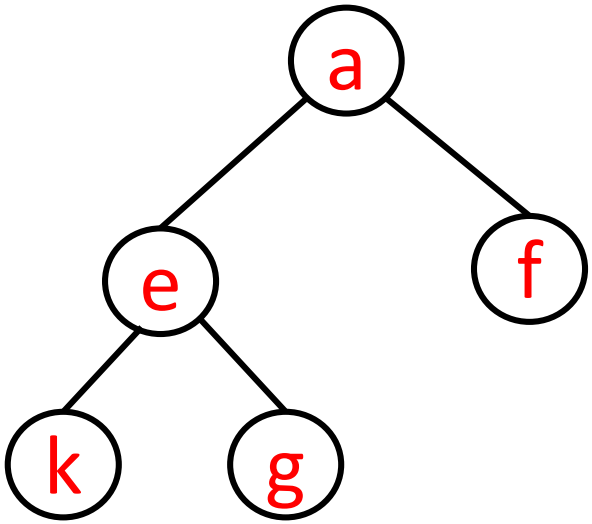
How to build a heap?

add(k)
add(f)
add(e)
add(a)
add(g)

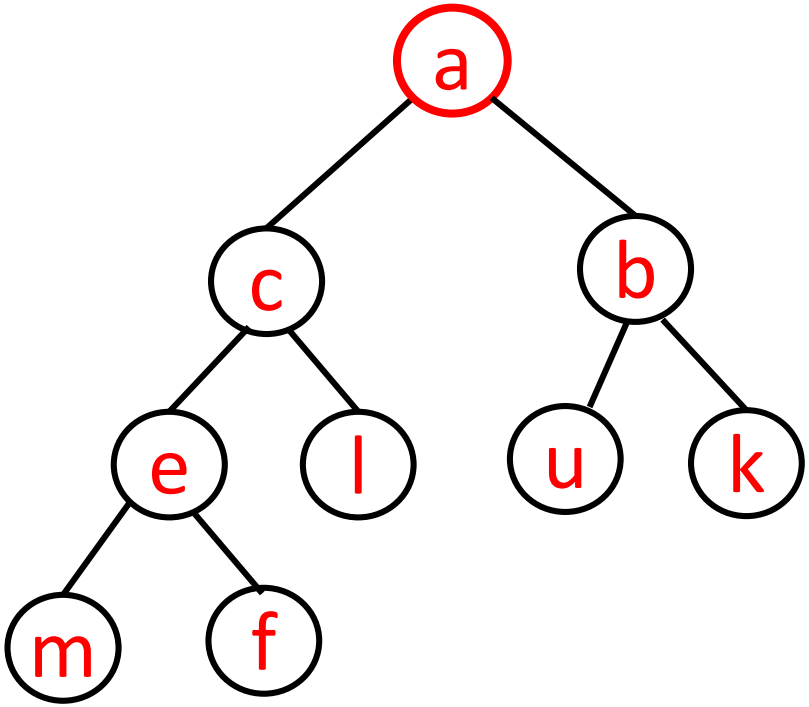


How to build a heap?

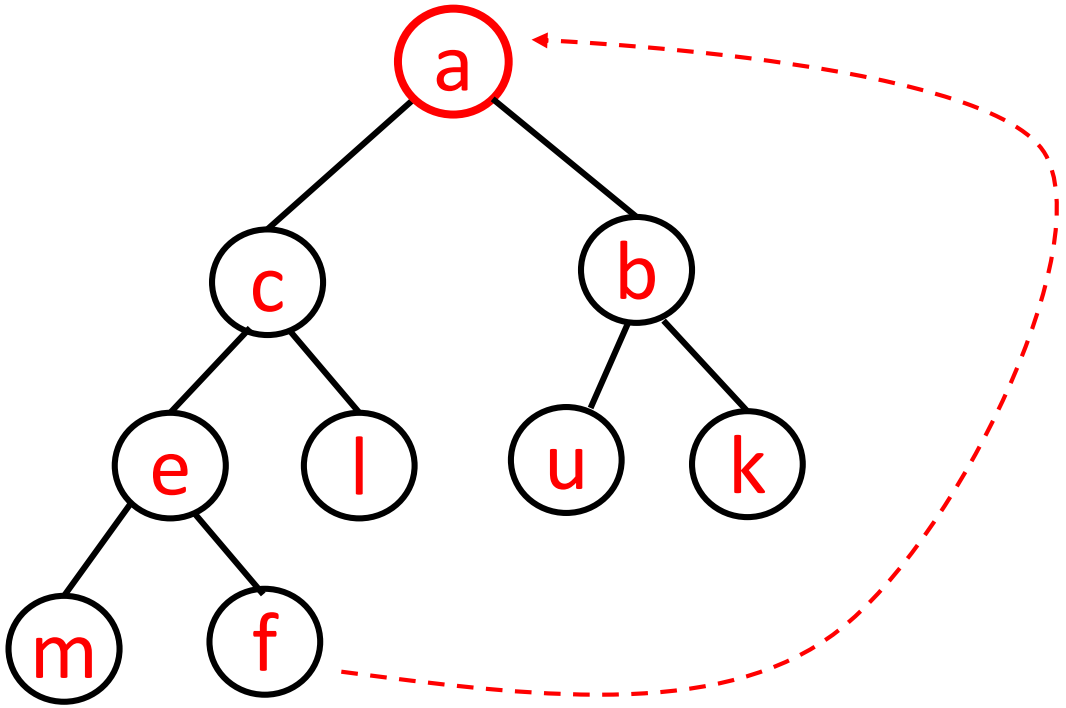
add(k)
add(f)
add(e)
add(a)
add(g)



removeMin()



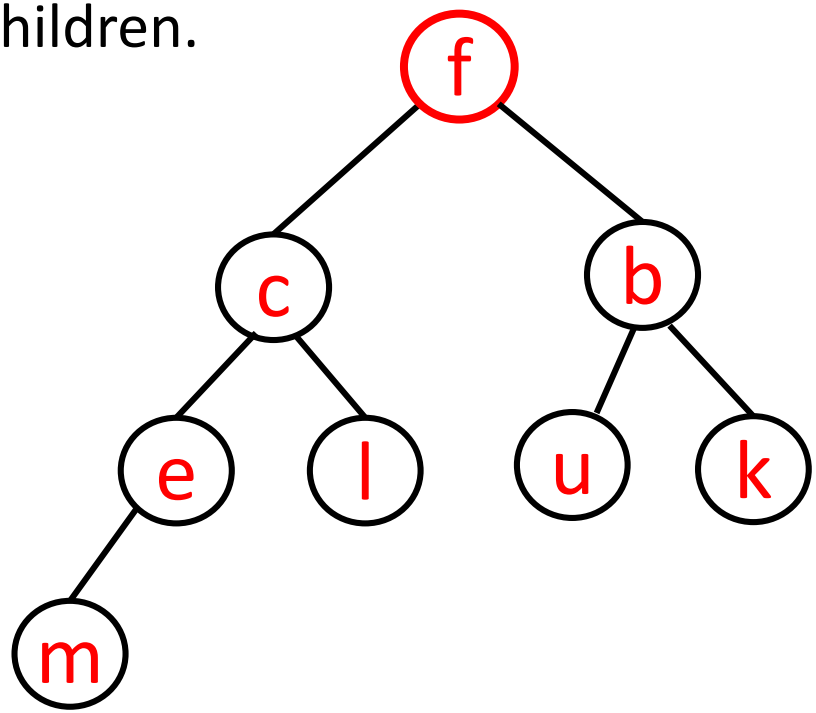
removeMin()



removeMin()

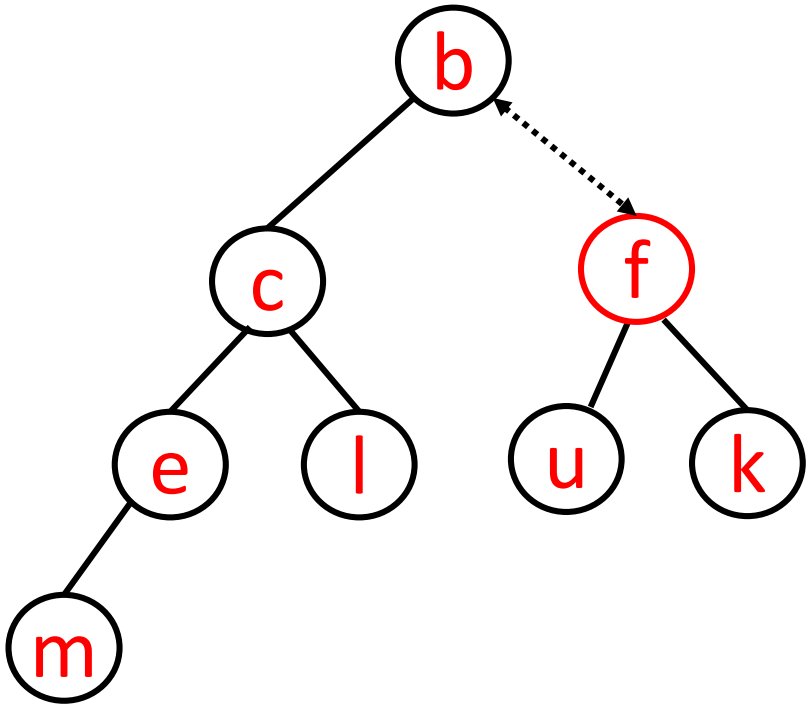
Problem is that the root will be greater than at least one of its children.

Why?



removeMin()

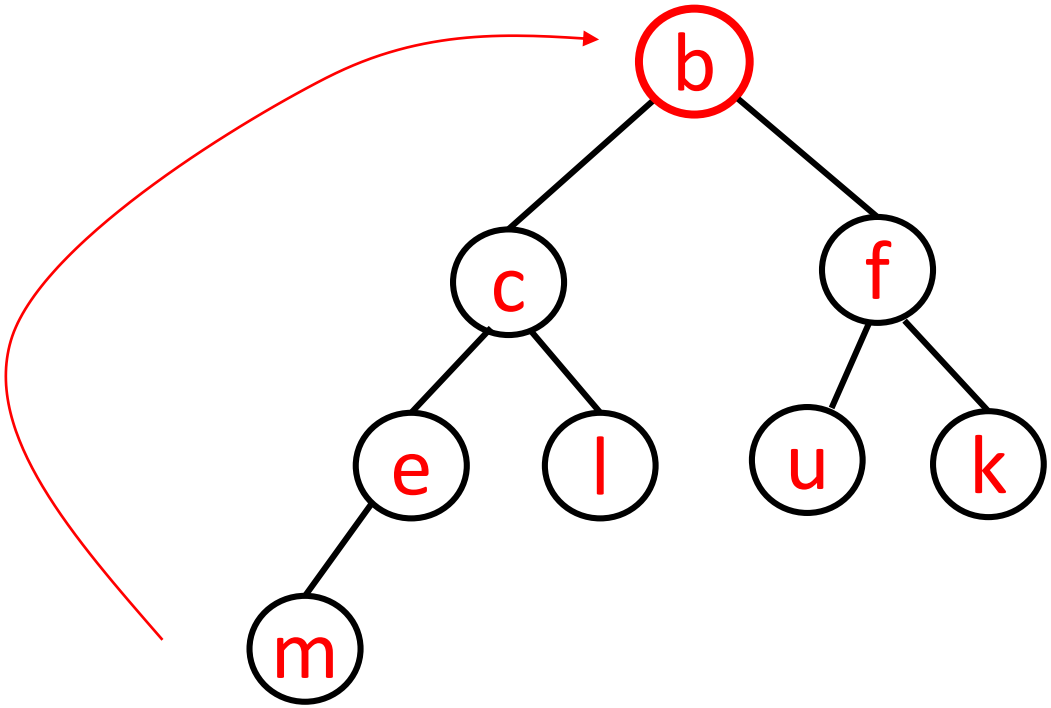
Swap elements with smaller child.



Keep swapping with smaller child, if necessary.

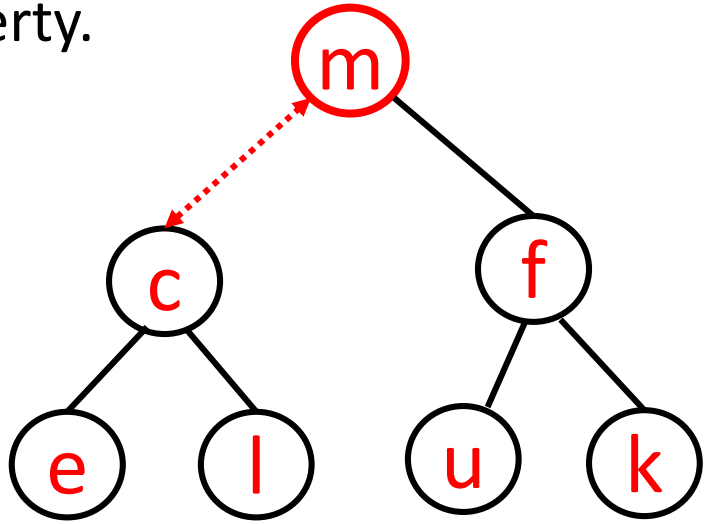
removeMin()

Let's do it again.



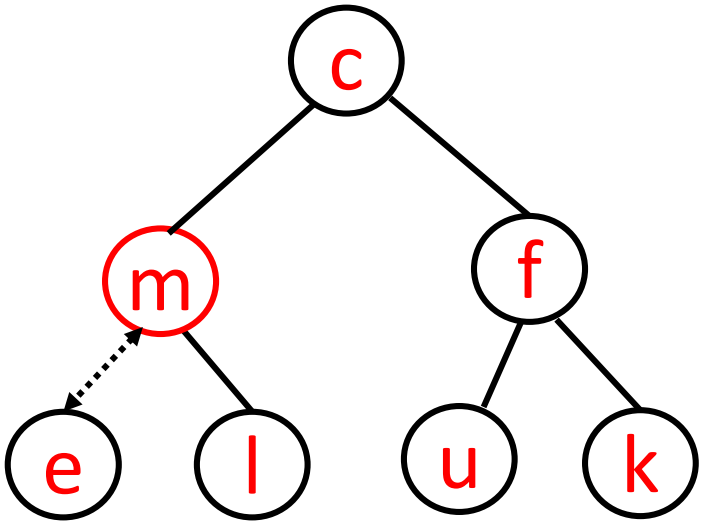
removeMin()

Now swap with smaller child, if necessarily, to preserve heap property.

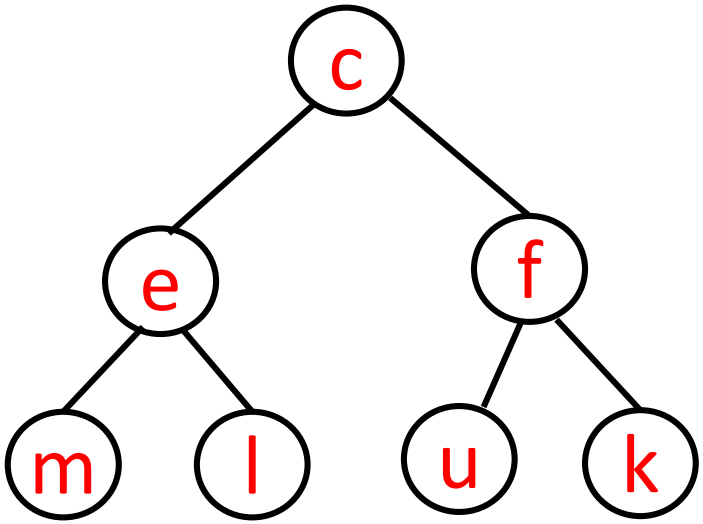


removeMin()

Keep swapping with smaller child, if necessary.

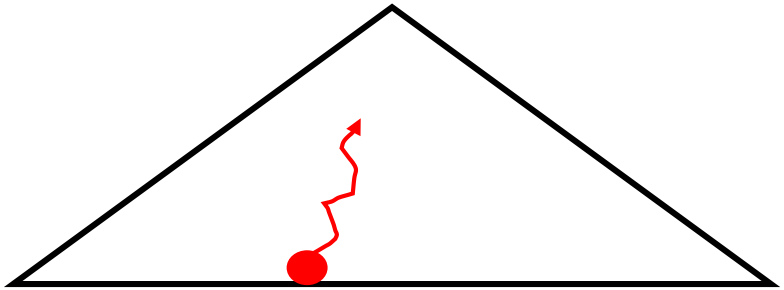


removeMin()



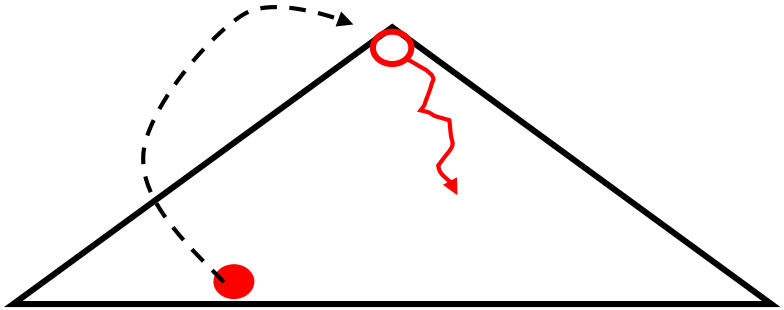
```
removeMin(){
    remove last leaf node and put its element into the root
    node = root
    while ((node has at least one child) and
            ( (node.element > left.element) or
              (node has right child and node.element > right.element)) )
       minChild = child with the smaller element
       swapElement(node, minChild)
       node = minChild
    }
}
```

add(element)



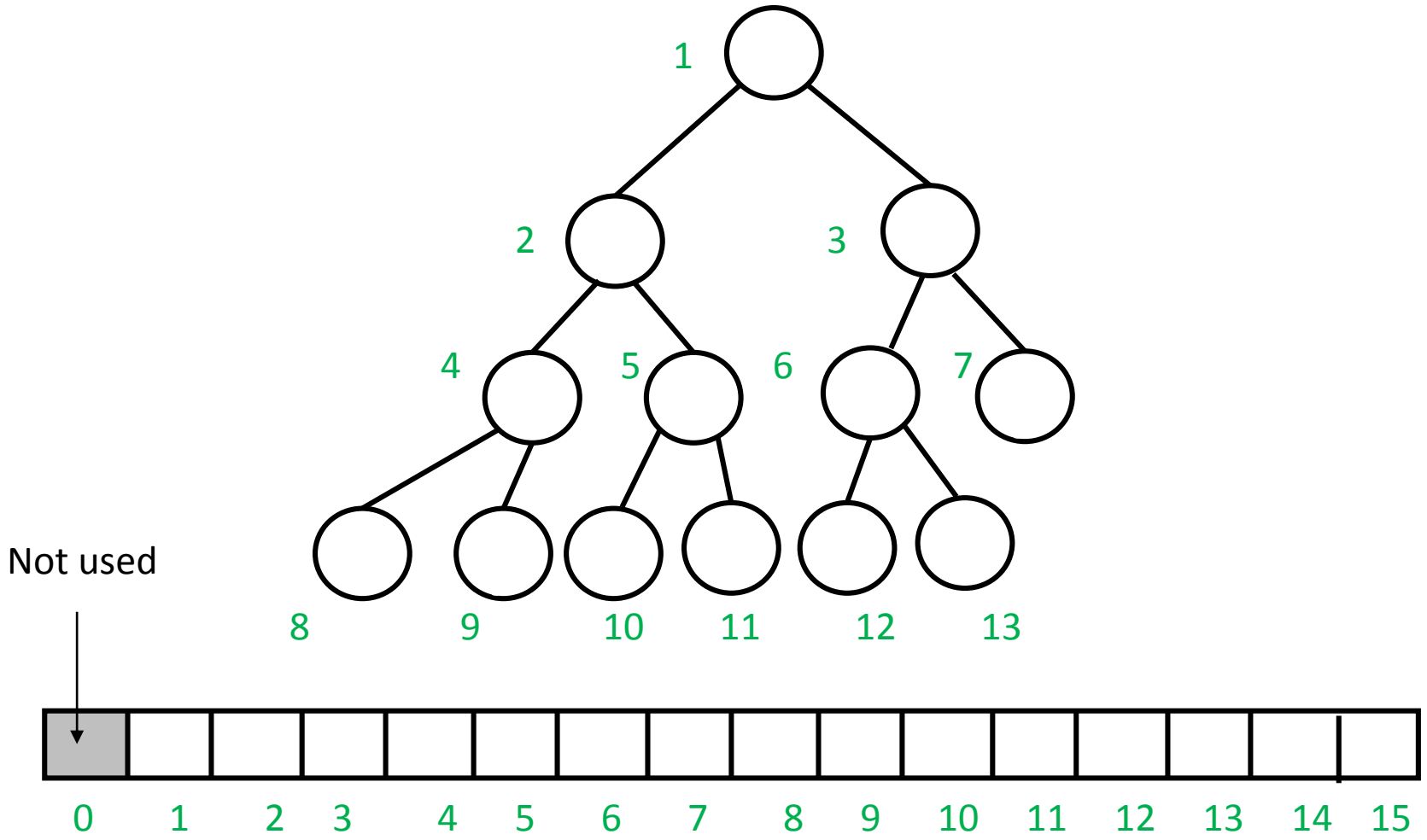
“upHeap”

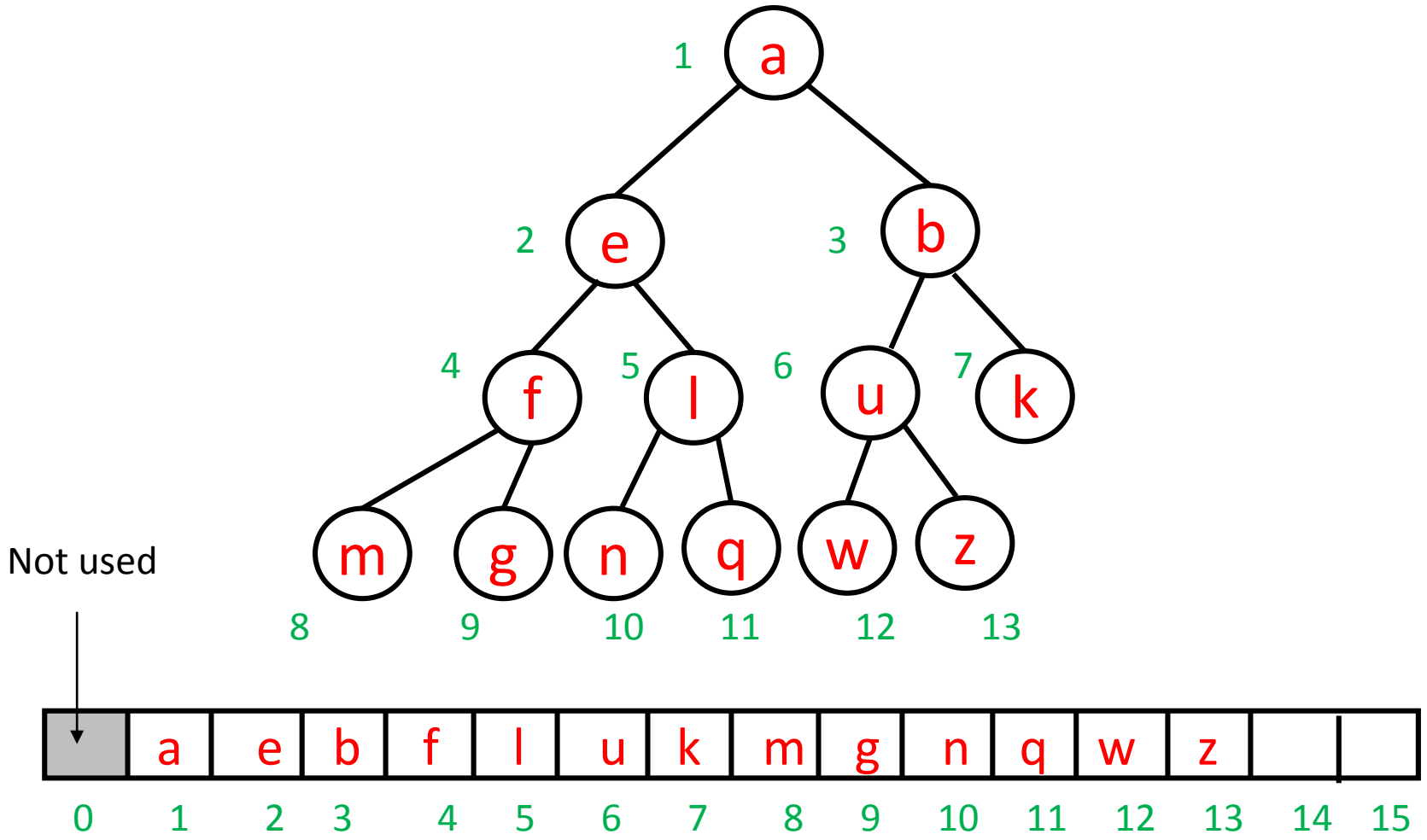
removeMin()



“downHeap”

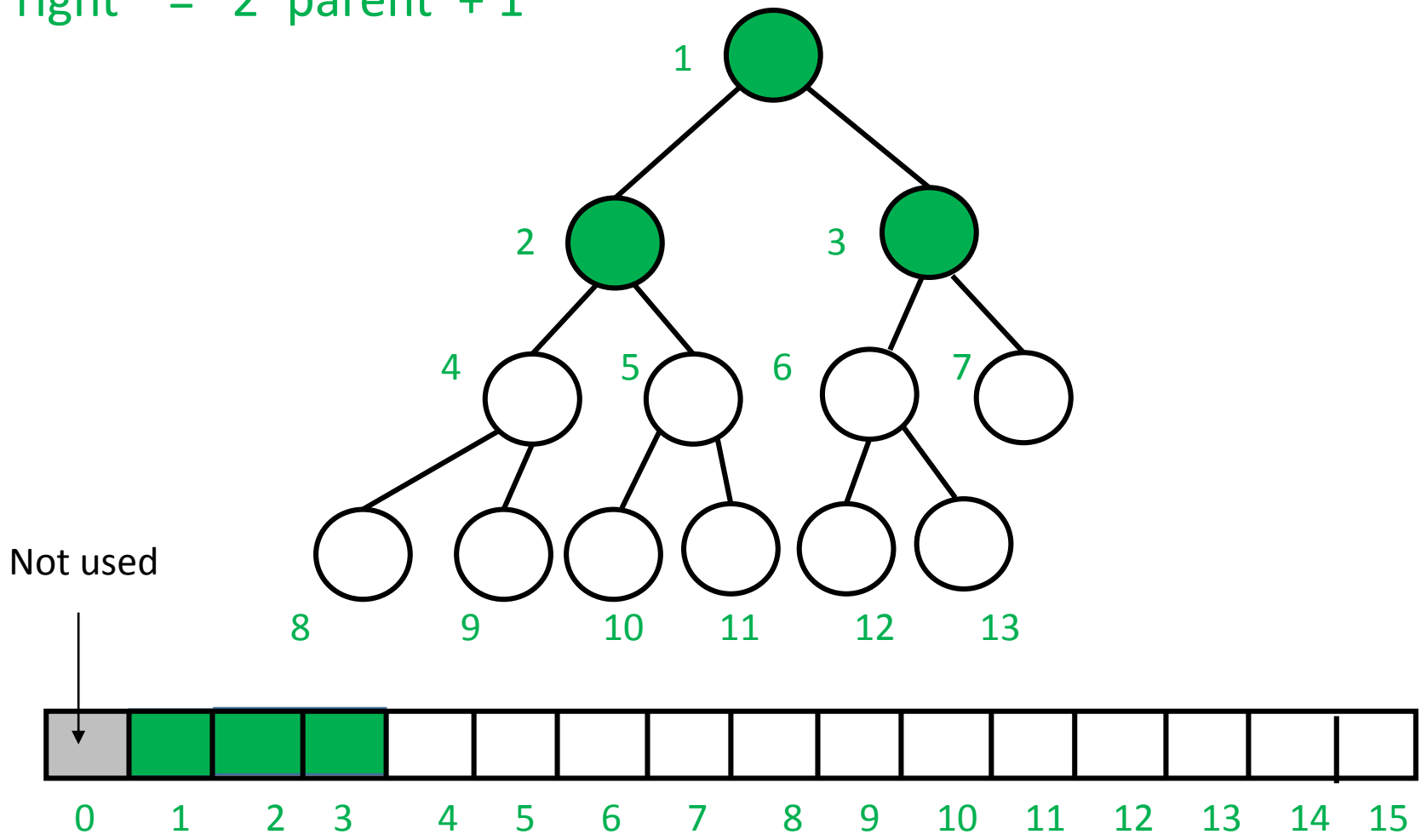
Heap (array implementation)




1
a
2
e
3
b
4
f
5
l
6
u
7
k
m
g
n
q
w
z
8
9
10
11
12
13
Not used
a
e
b
f
l
u
k
m
g
n
q
w
z
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

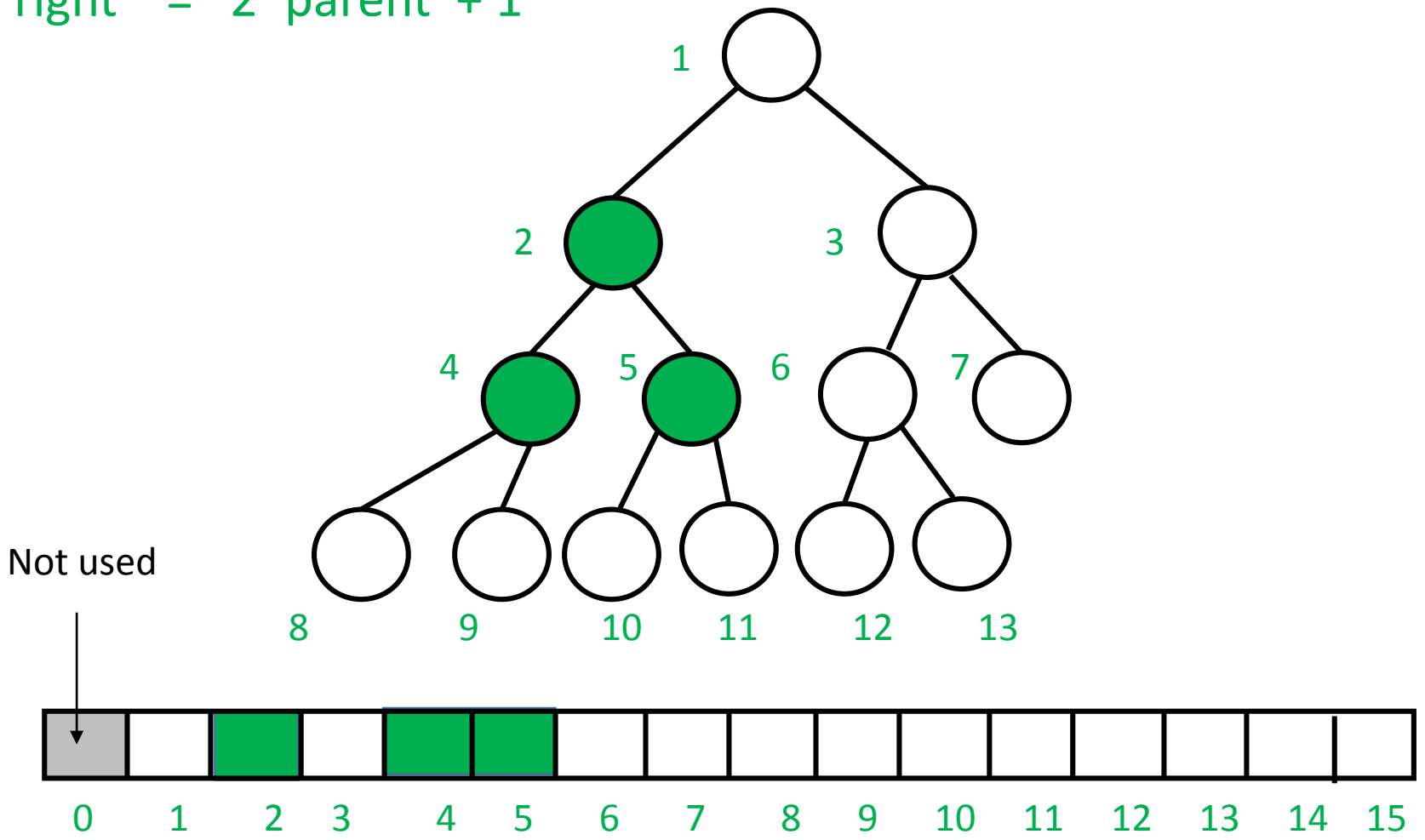
Heap index relations

parent = child / 2
left = 2*parent
right = 2*parent + 1



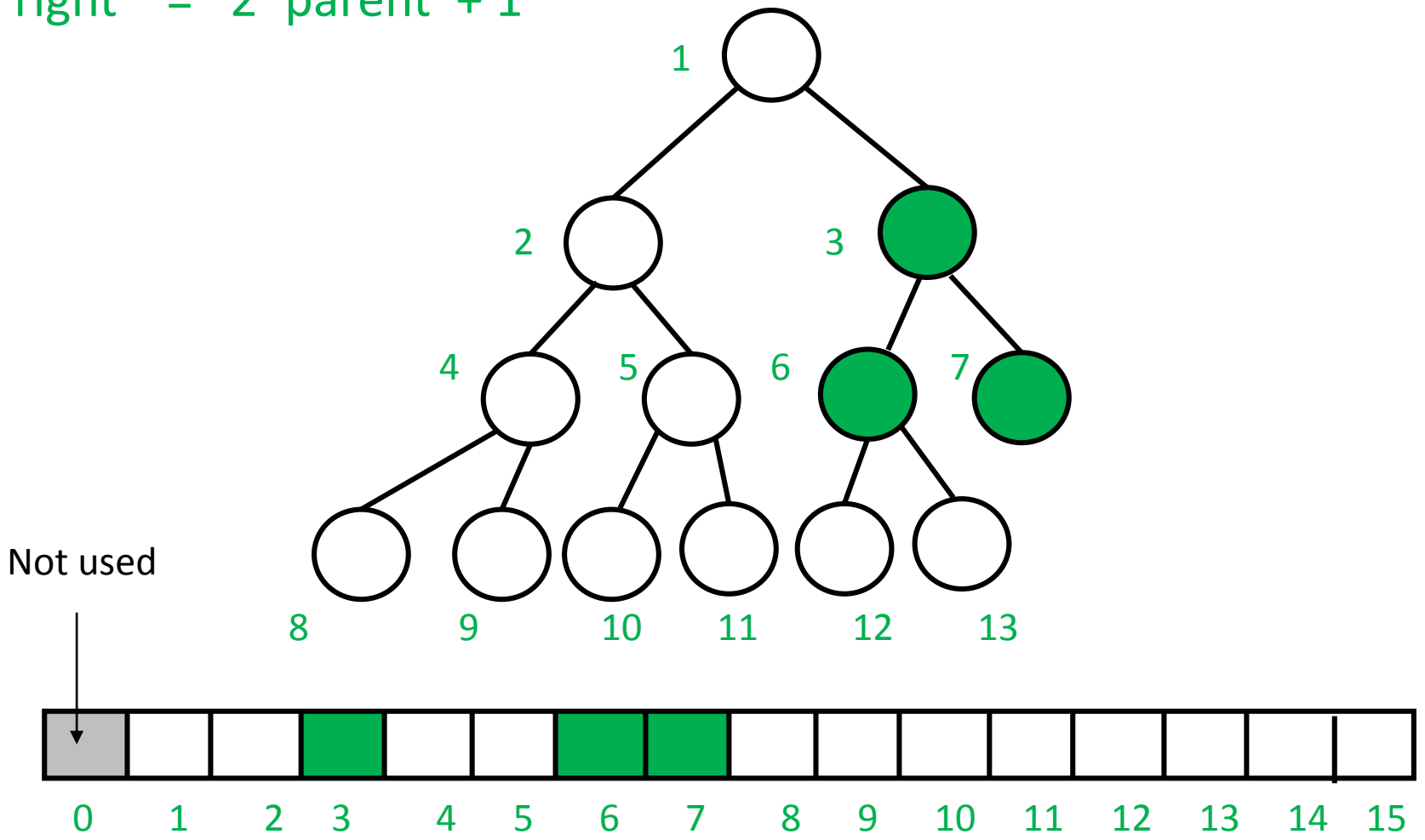
Heap index relations

parent = child / 2
left = 2*parent
right = 2*parent + 1



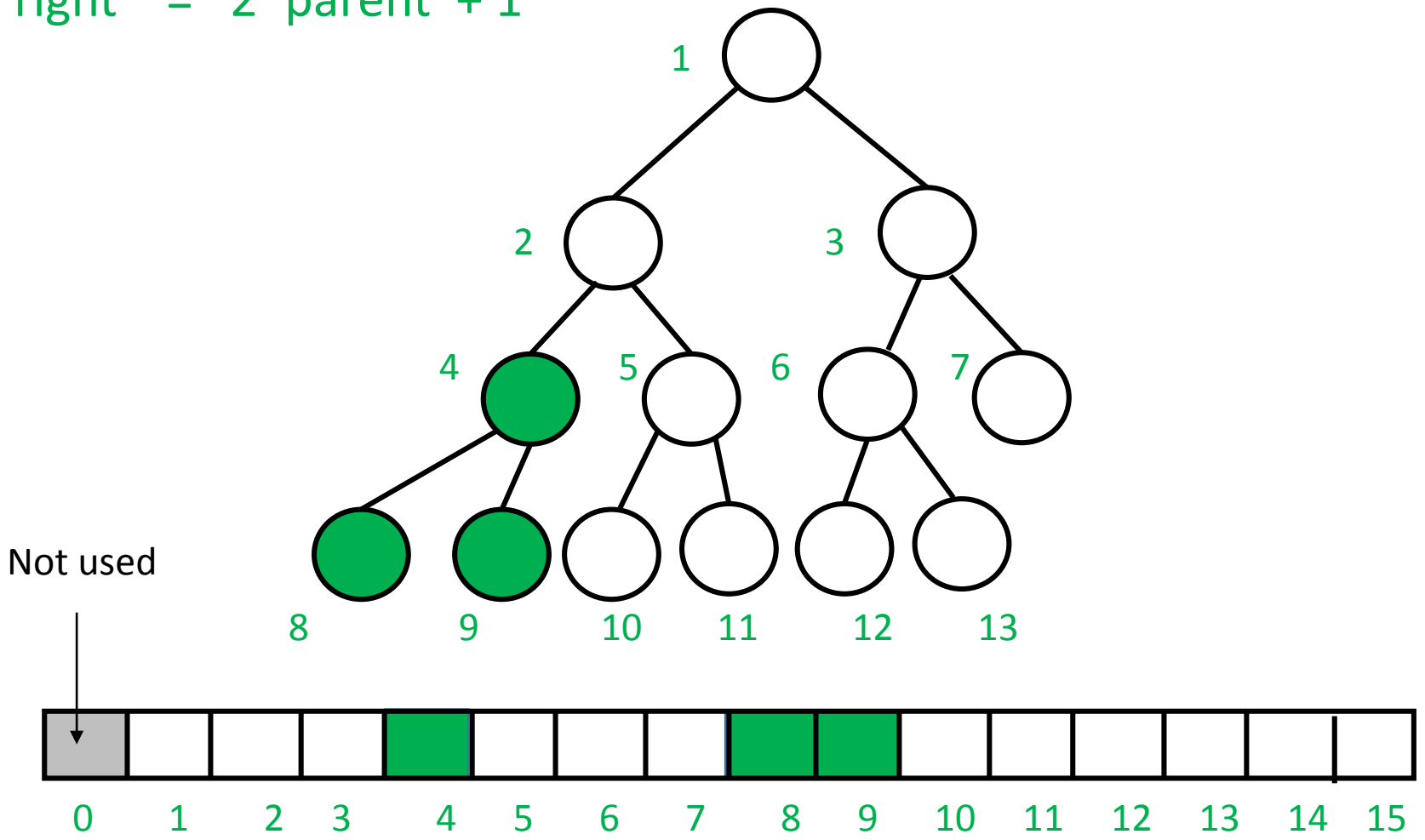
Heap index relations

parent = child / 2
left = 2*parent
right = 2*parent + 1

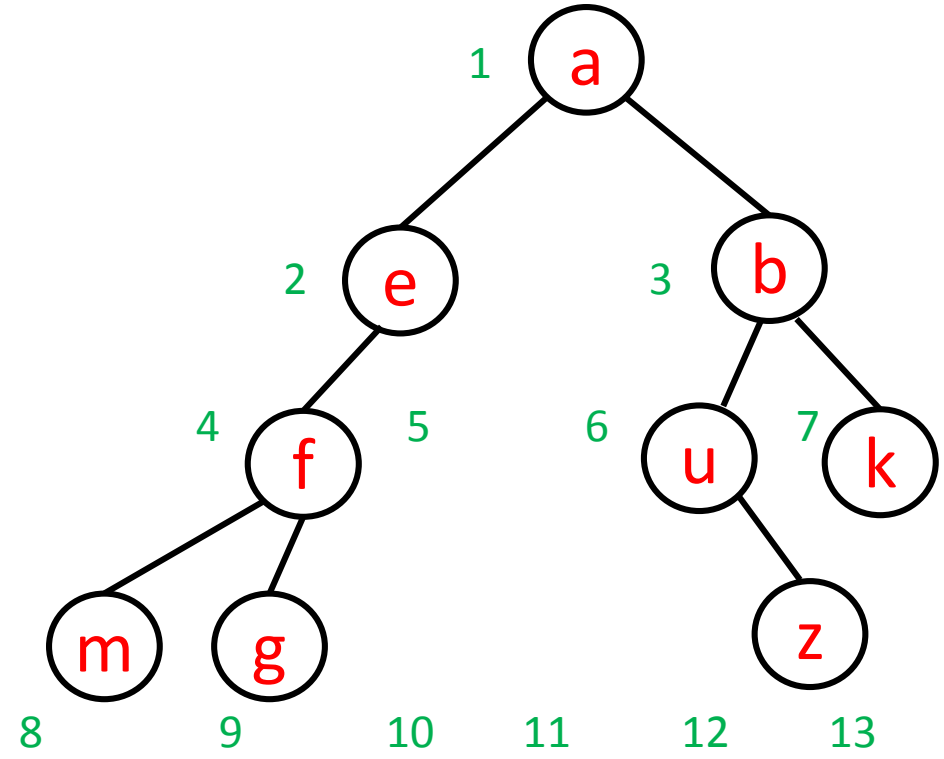


Heap index relations

parent = child / 2
left = 2*parent
right = 2*parent + 1



ASIDE: an array data structure can be used for any binary tree. But this is uncommon.



	a	e	b			u	k	m	g				z		
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Next lecture

- write add(element) and removeMin() using array indices
- best and worst case
- faster algorithm for building a heap