Fibonacci Heap

Alessio Ghio
Eduardo Medina

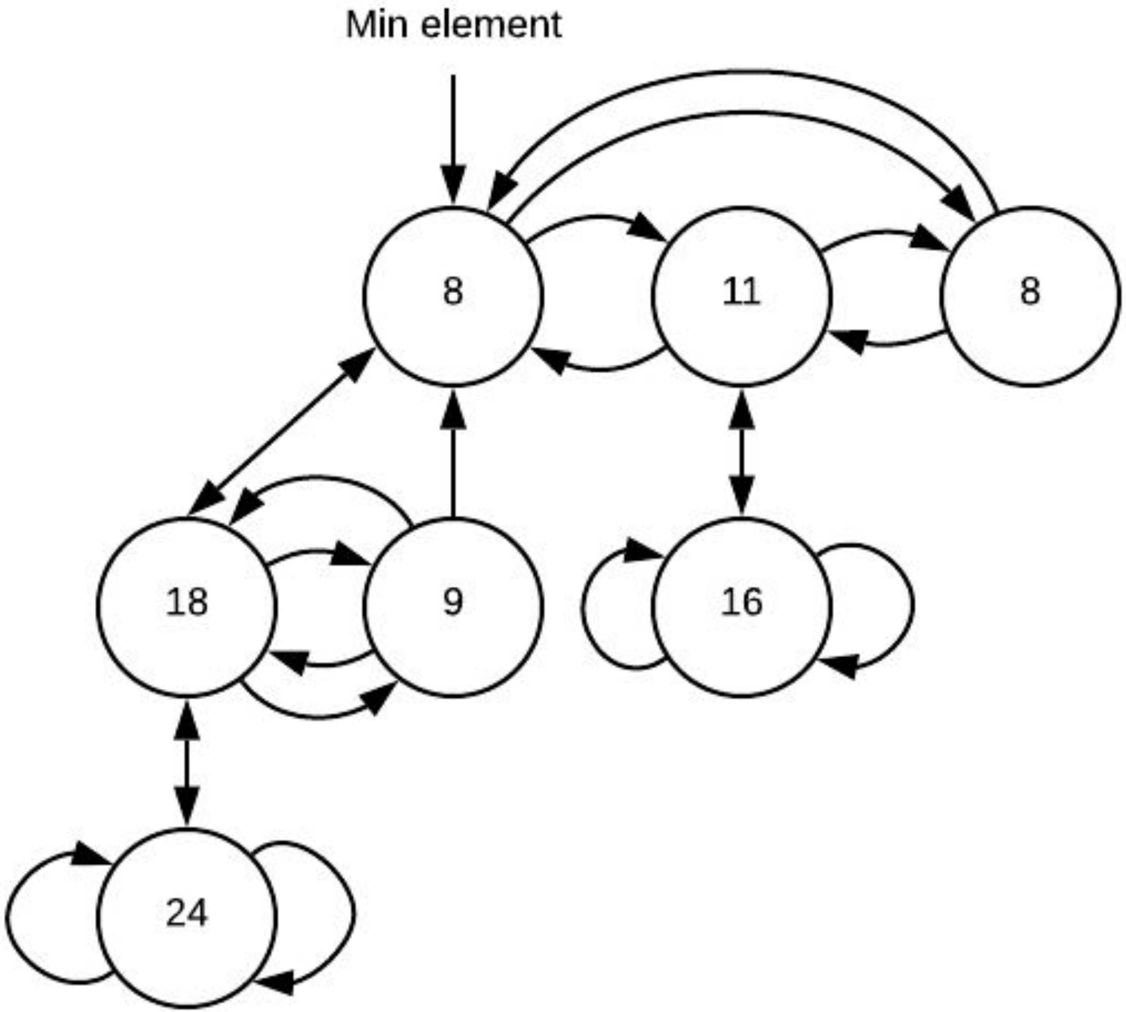
Outline

1. Structure
2. Methods
    2.1. Insert
    2.2. Decrease Key
    2.3. Delete
    2.4. Consolidate
    2.5. Merge
3. Advantages and Disadvantages
4. Applications

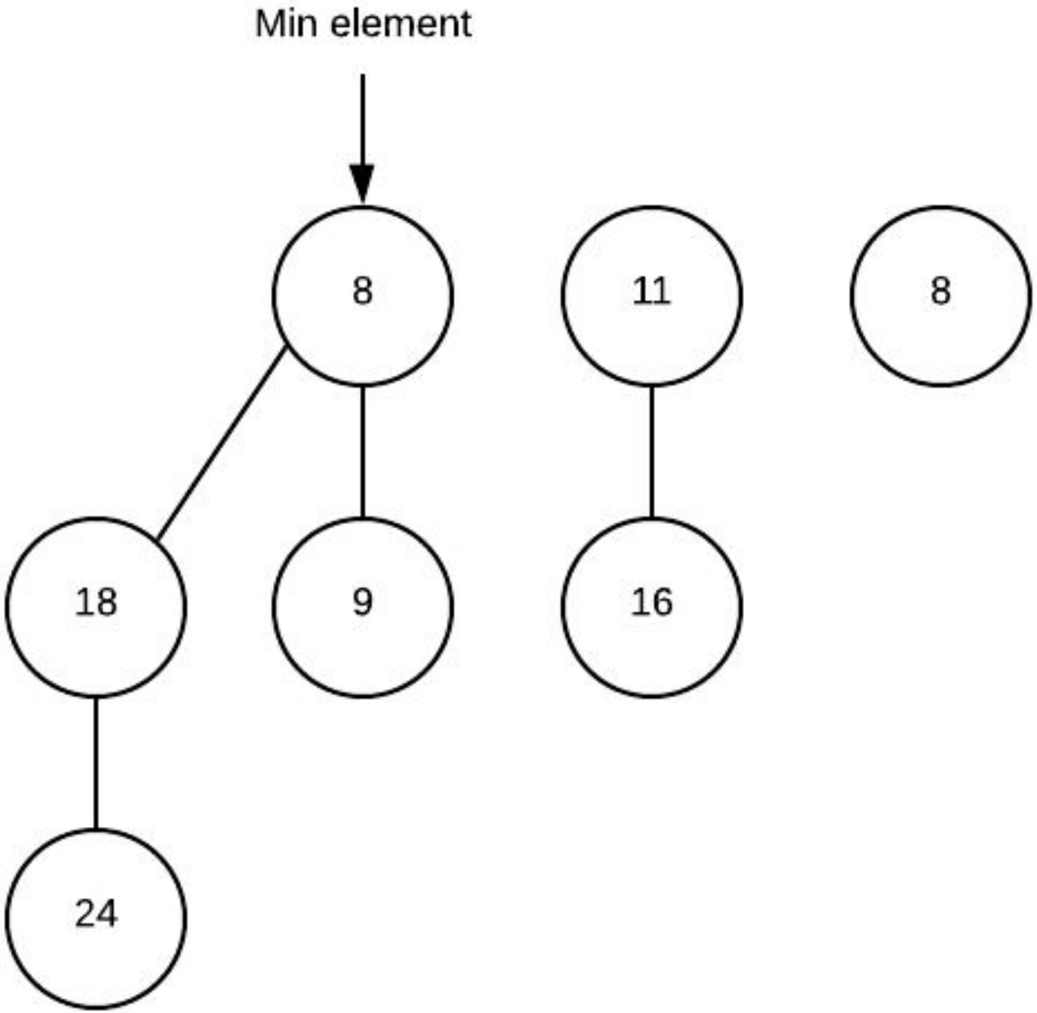
Structure

Structure

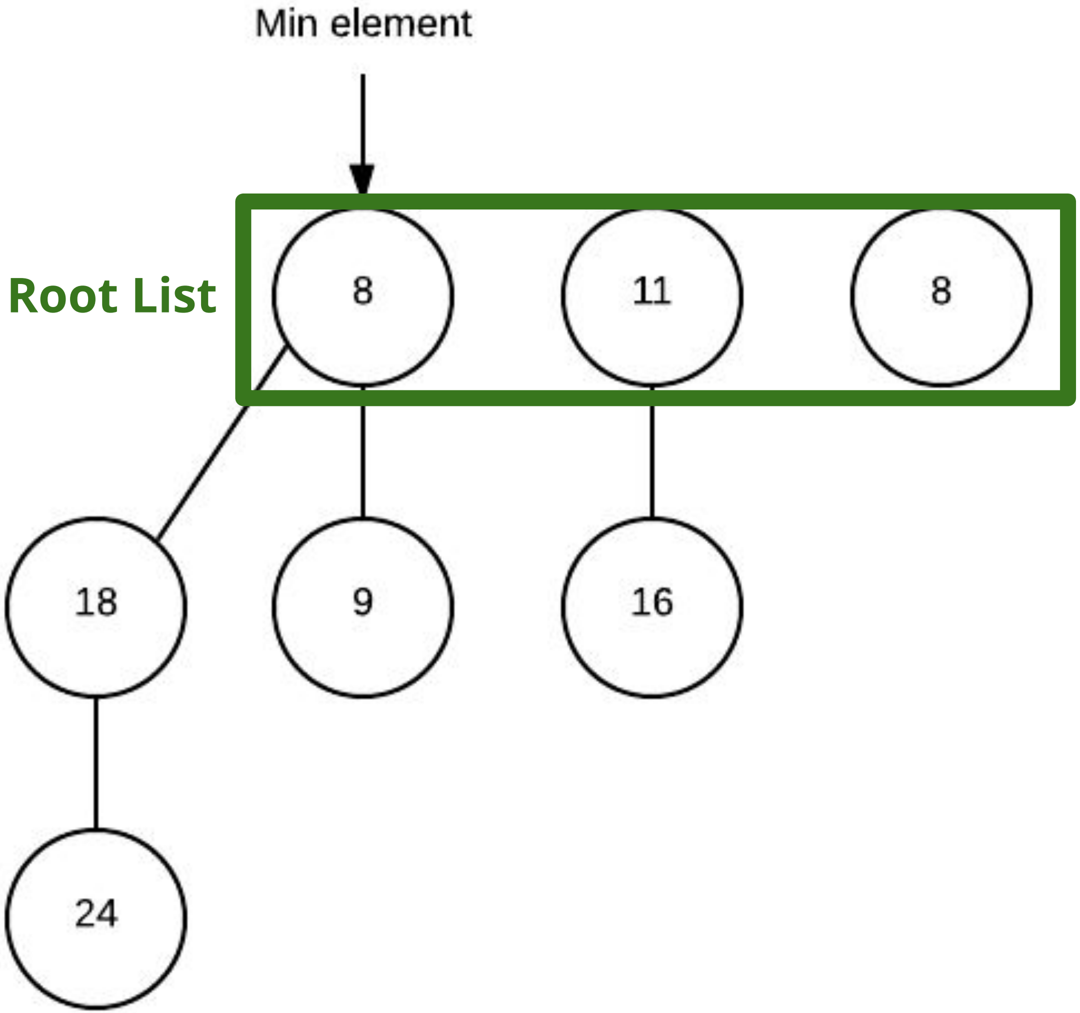
Node:
Left
Right
Child
Parent



Structure



Structure

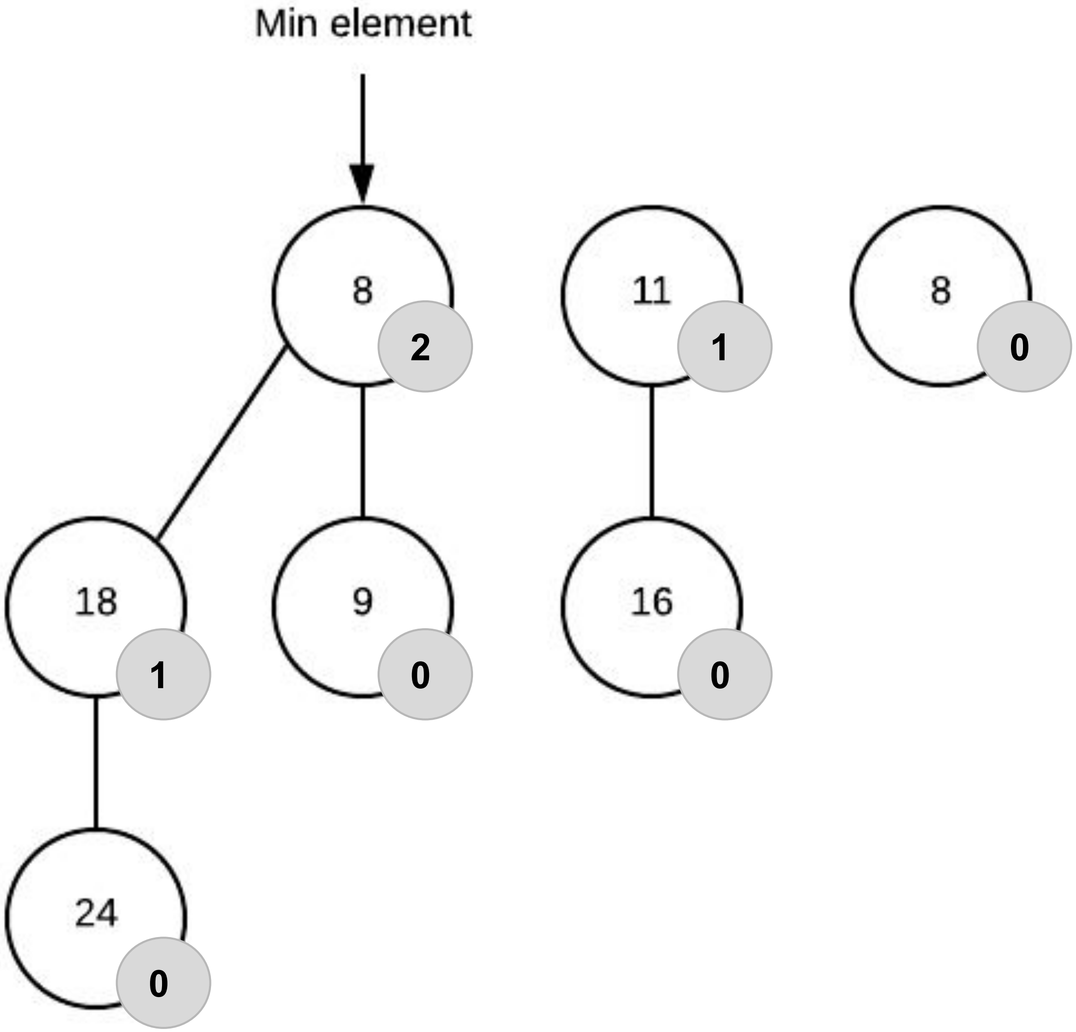


Structure

Rank/Degree

n

n = number of children



Methods

Function	Time complexity
Find Min	O(1)
Delete Min/Consolidate	O(log n)
Insert	O(1)
Decrease-Key	O(1)
Merge	O(1)

Table 1. Fibonacci Heap functions time complexity in big O notation [1]

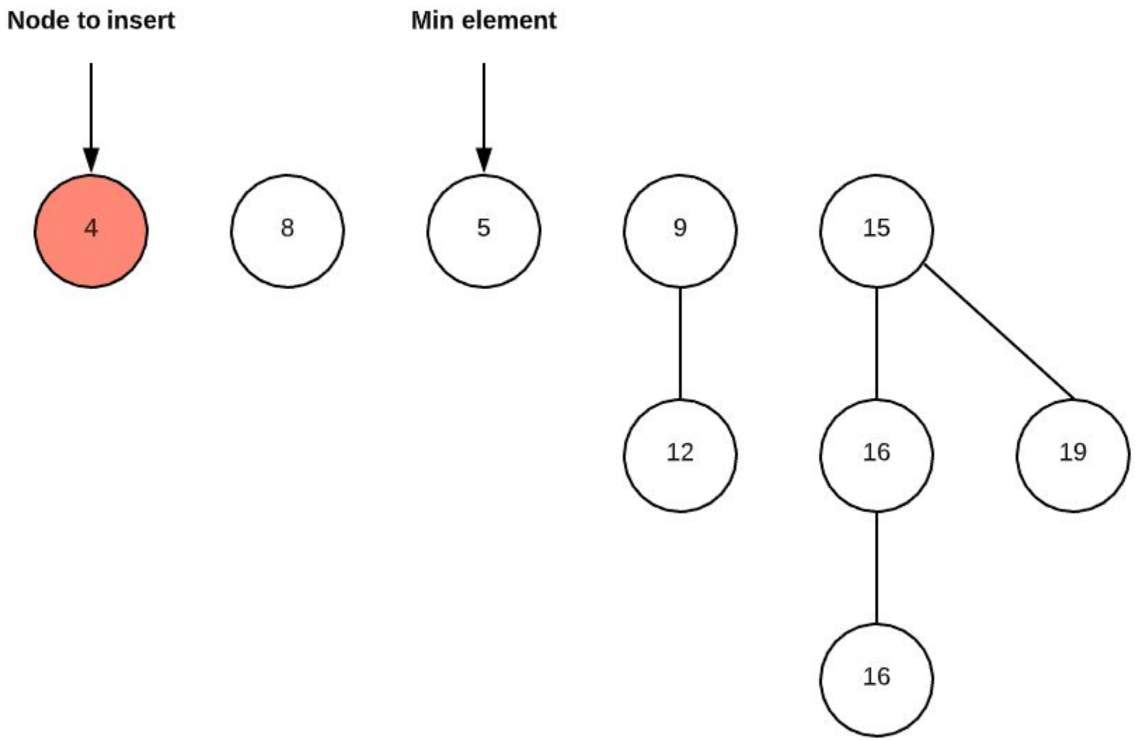
Insert

Algorithm 1: Insert
Source: Adapted from [3]

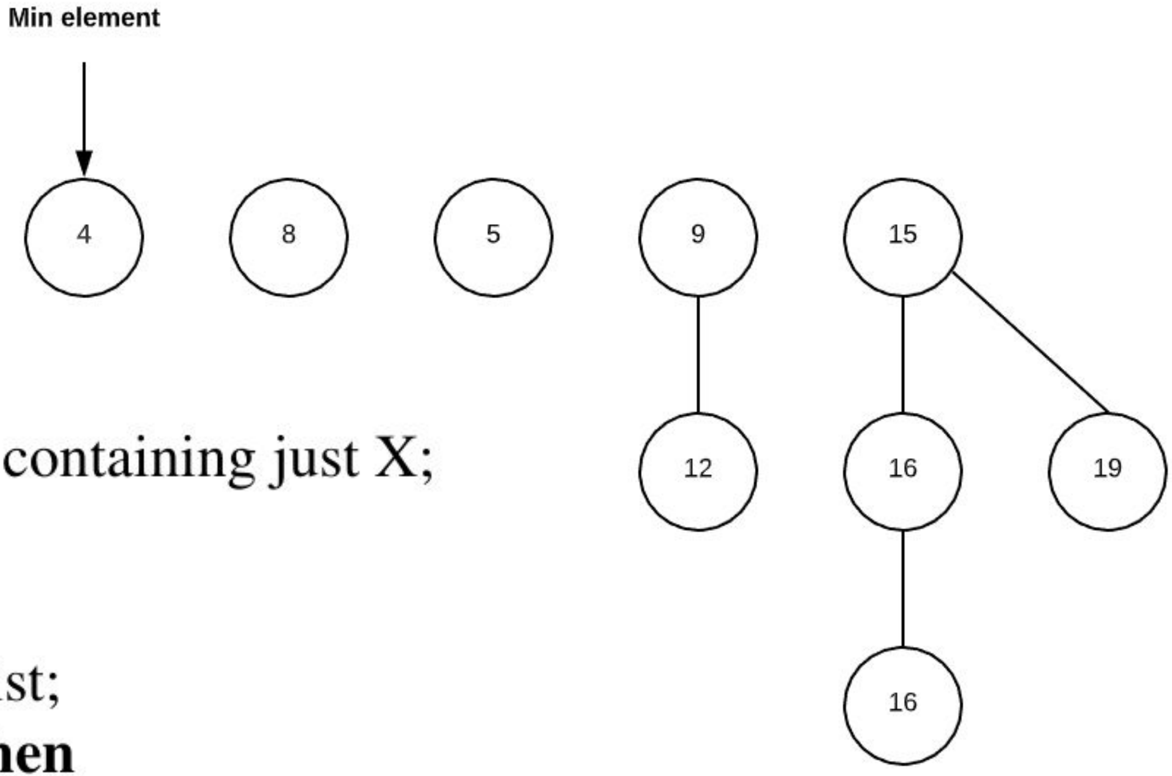
```
x.degree = 0;
x.p = NULL;
x.child = NULL;
x.mark = FALSE;
if H.min == NULL then
    create a root list for H containing just X;
    H.min = x;
else
    insert x into H's root list;
    if x.key < H.min.key then
        H.min = x;
H.n = H.n + 1
```

Insert 4

```
x.degree = 0;
x.p = NULL;
x.child = NULL;
x.mark = FALSE;
```



Insert 4




```
if H.min == NULL then
    create a root list for H containing just X;
    H.min = x;
else
    insert x into H's root list;
    if x.key < H.min.key then
        H.min = x;
H.n = H.n + 1
```

Decrease Key

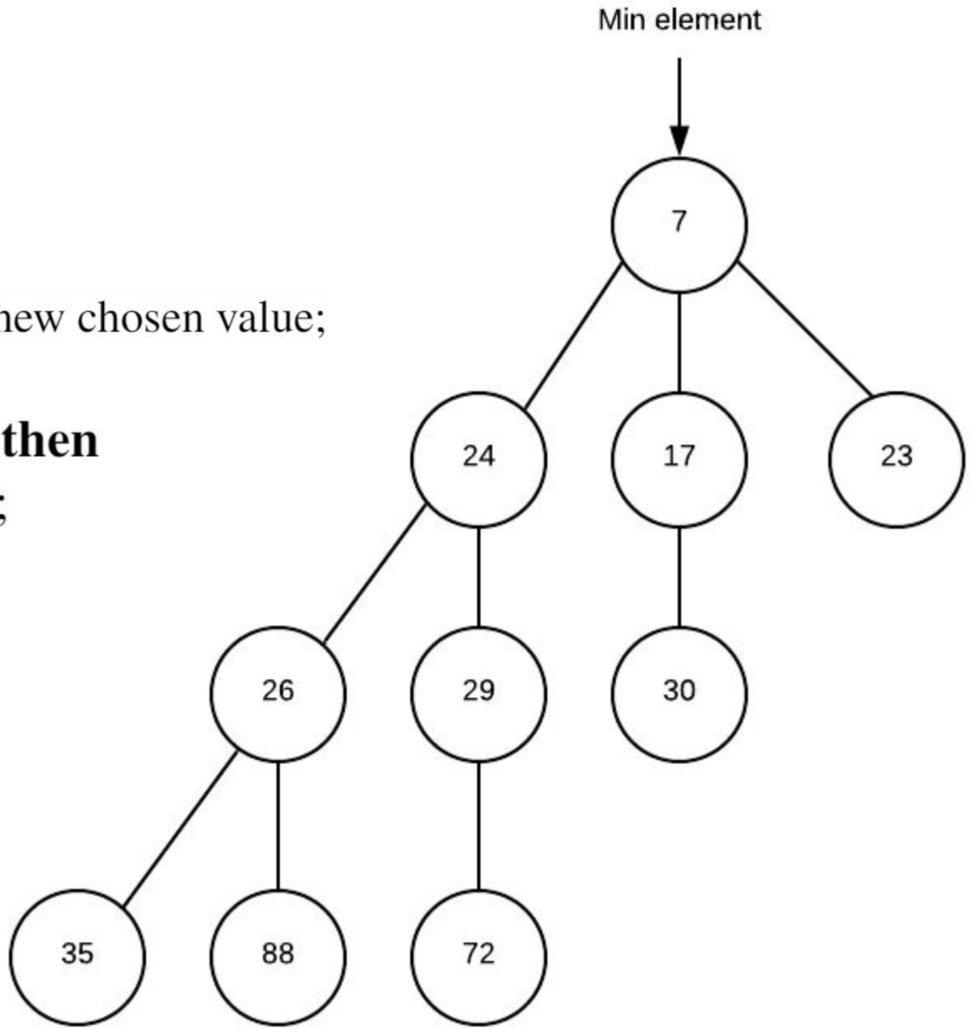
Algorithm 2: Decrease Key

Source: Adapted from [2]

```
1  Decrease the value of the node 'x' to the new chosen value;
2  p[x] = Parent of node 'x';
3  if min heap property is not violated then
4      Update min pointer if necessary;
5  else if min heap property is violated and p[x] is unmarked then
6      Cut off the link between 'x' and p[x];
7      Mark p[x];
8      Add 'x' and its children to the root list and update min pointer if necessary;
9  else if min heap property is violated and p[x] is marked then
10     Cut off the link between 'x' and p[x];
11     Add 'x' to the root list and update min pointer if necessary;
12     Cut off link between p[x] and p[p[x]];
13     Add p[x] to the root list and update min pointer if necessary;
14     if p[p[x]] is unmarked then
15         Mark p[p[x]];
16     else
17         cut off p[p[x]];
18         x = p[p[x]];
19         Jump to line 11;
```

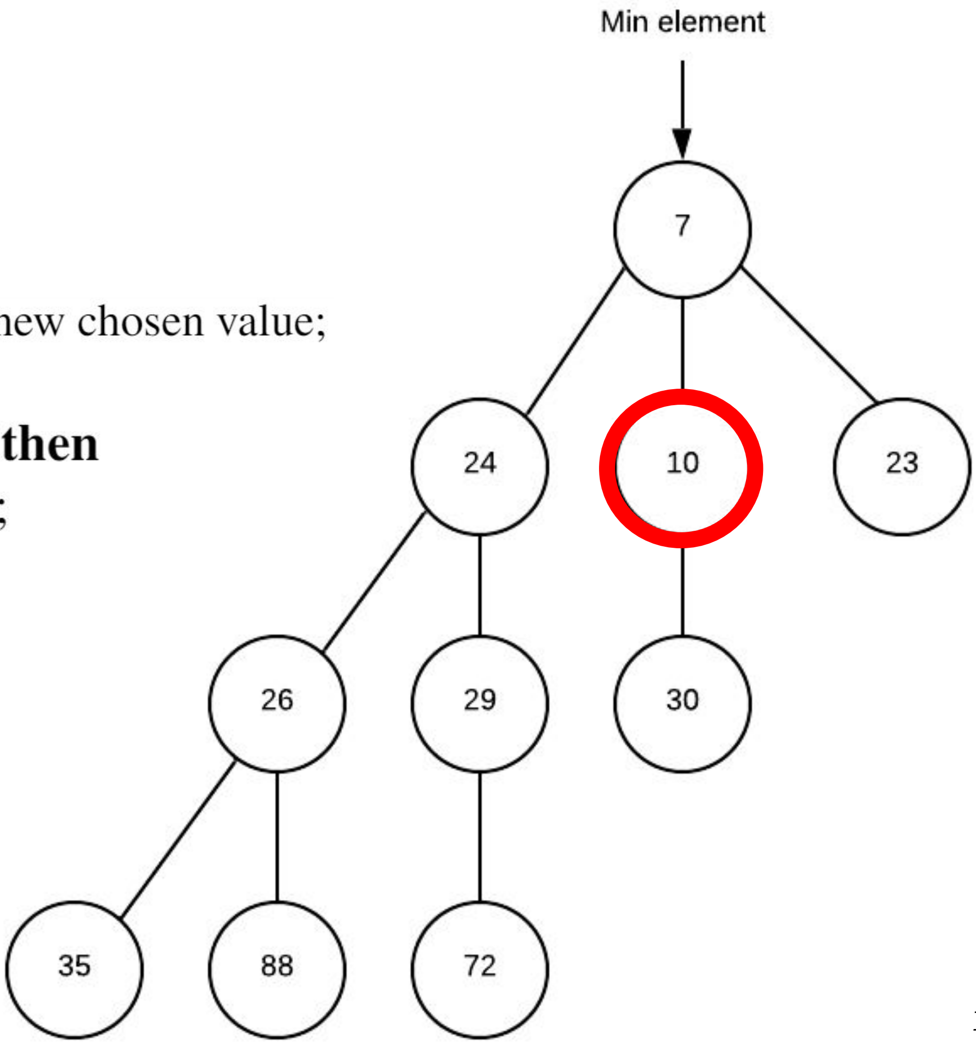
Decrease Key: 17->10

Decrease the value of the node ‘x’ to the new chosen value;
p[x] = Parent of node ’x’;
if *min heap property is not violated* **then**
    Update min pointer if necessary;



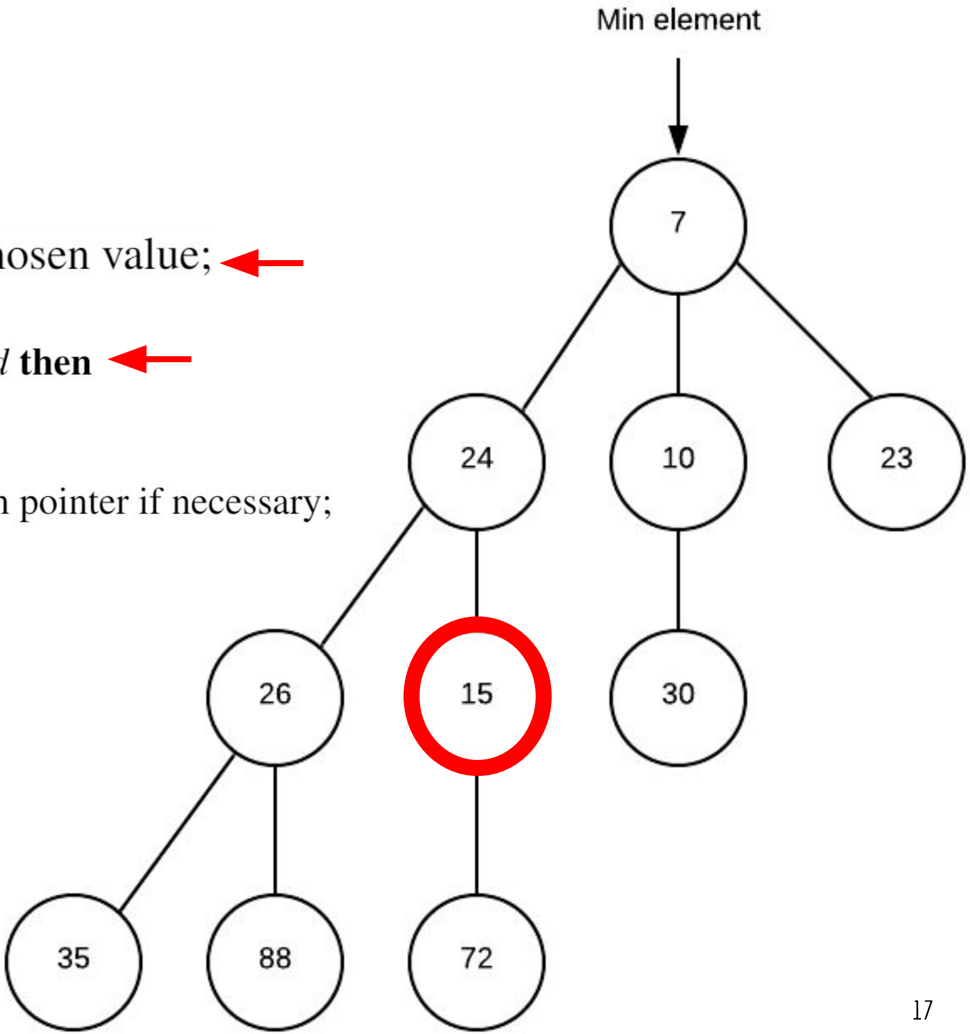
Decrease Key: 17->10

Decrease the value of the node ‘x’ to the new chosen value;
p[x] = Parent of node ’x’;
if *min heap property is not violated* **then**
    Update min pointer if necessary;



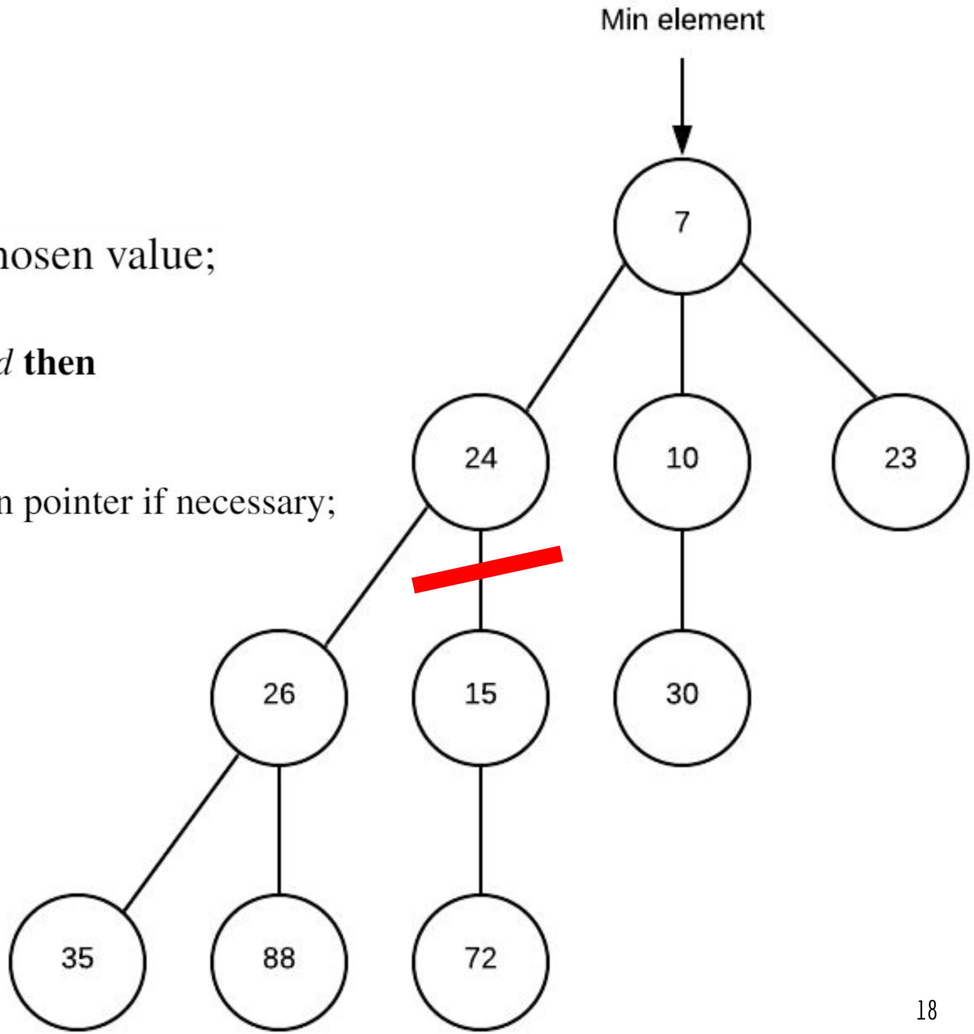
Decrease Key: 29->15

Decrease the value of the node ‘x’ to the new chosen value;
p[x] = Parent of node ’x’;
else if *min heap property is violated and p[x] is unmarked* **then**
    Cut off the link between ‘x’ and p[x];
    Mark p[x];
    Add ‘x’ and its children to the root list and update min pointer if necessary;



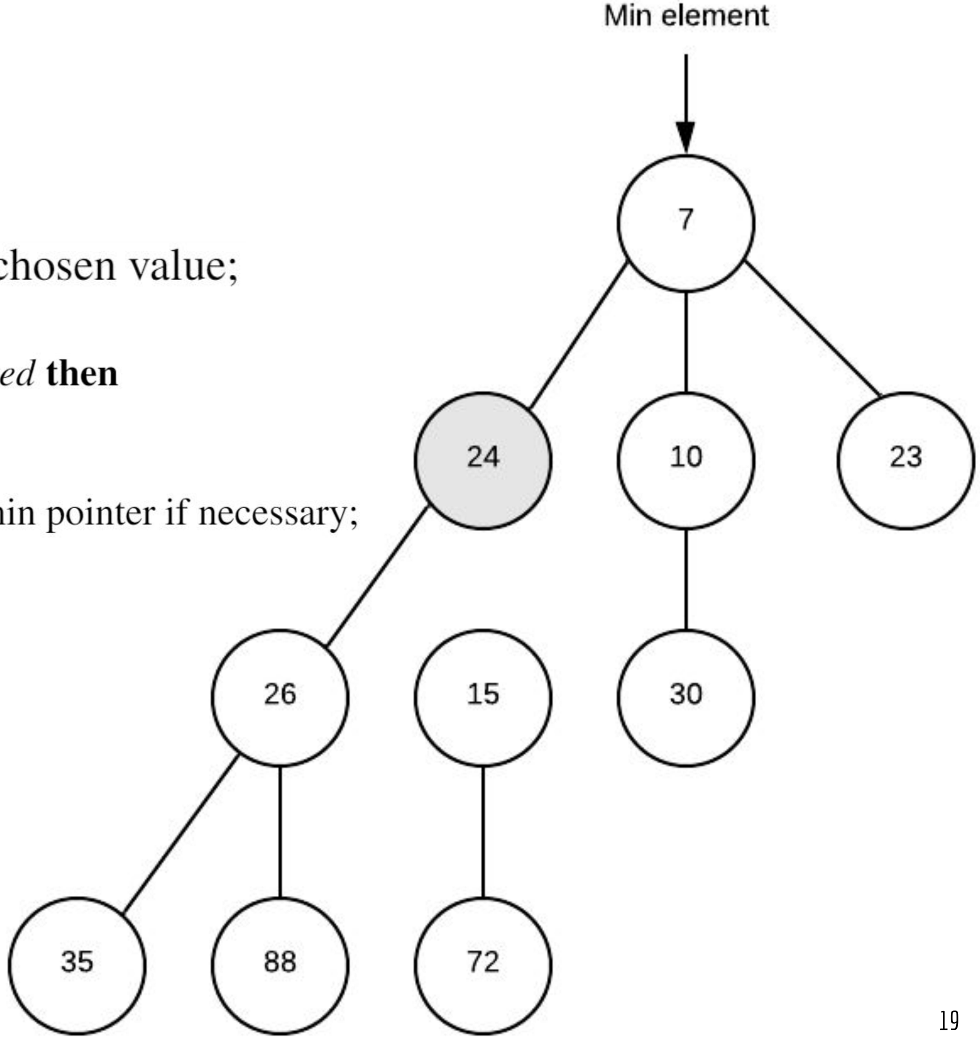
Decrease Key: 29->15

Decrease the value of the node ‘x’ to the new chosen value;
p[x] = Parent of node ’x’;
else if *min heap property is violated and p[x] is unmarked* **then**
    Cut off the link between ‘x’ and p[x];
    Mark p[x];
    Add ‘x’ and its children to the root list and update min pointer if necessary;



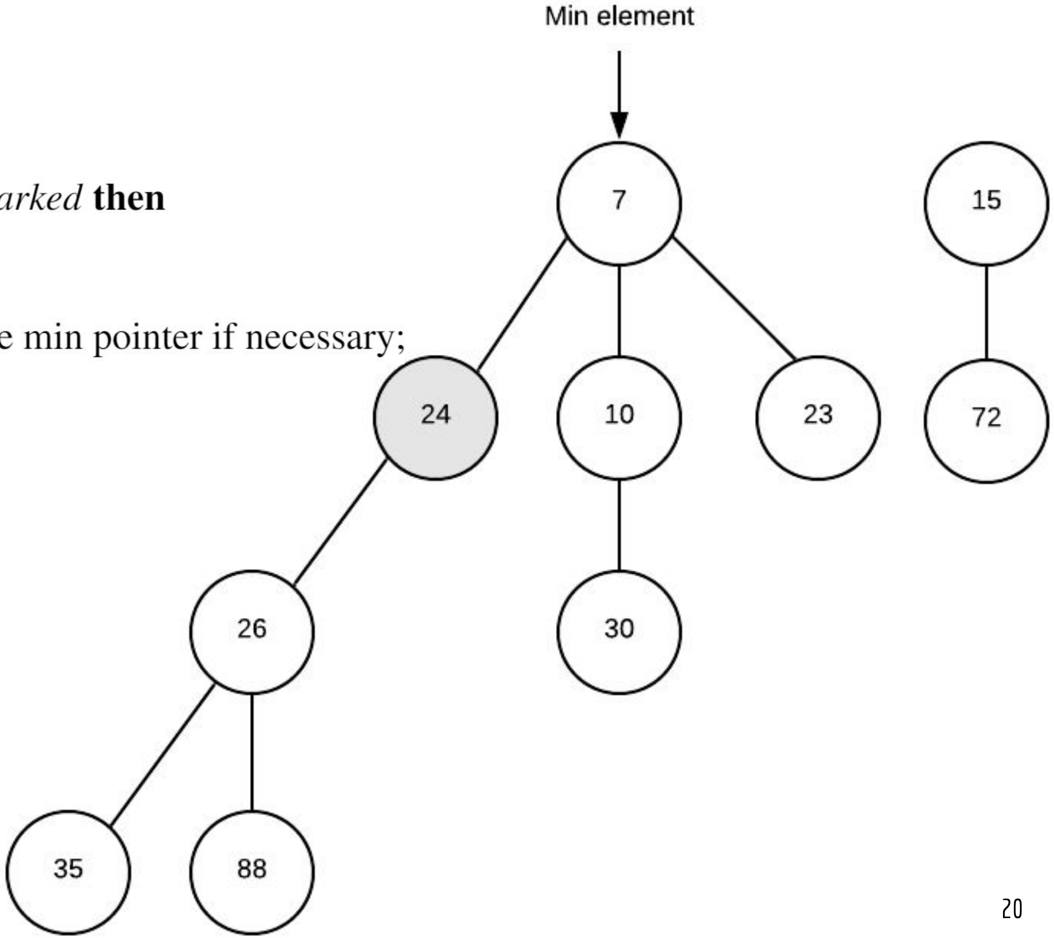
Decrease Key: 29->15

Decrease the value of the node ‘x’ to the new chosen value;
p[x] = Parent of node ’x’;
else if *min heap property is violated and p[x] is unmarked* **then**
    Cut off the link between ‘x’ and p[x];
    Mark p[x];
    Add ‘x’ and its children to the root list and update min pointer if necessary;



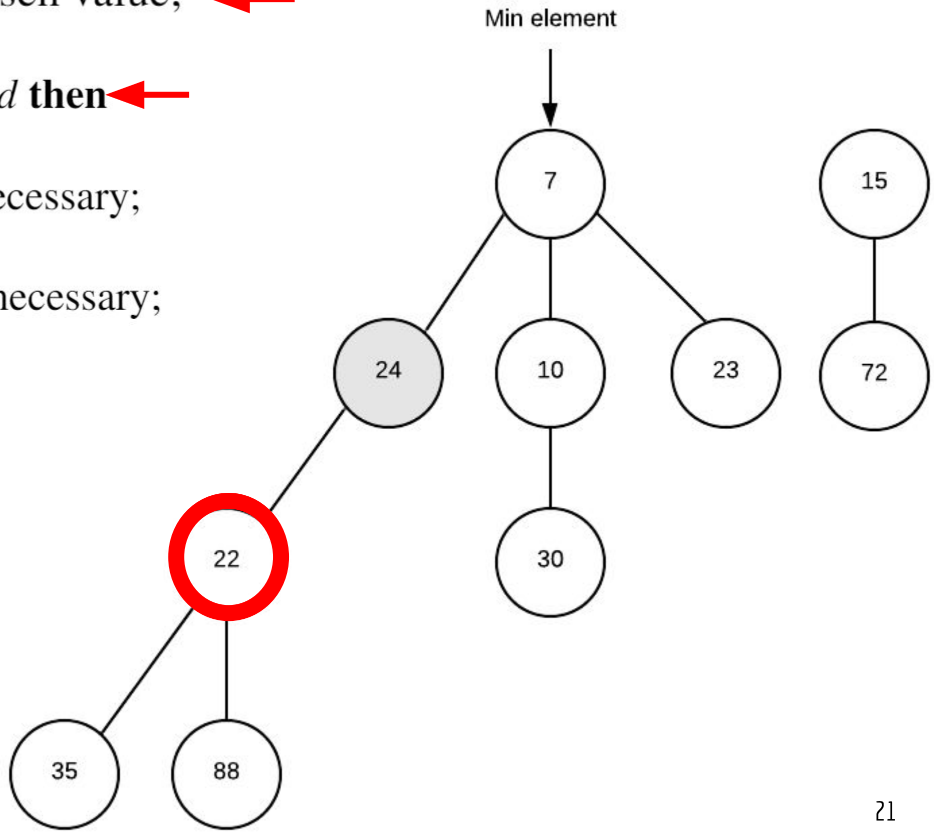
Decrease Key: 29->15

```
Decrease the value of the node 'x' to the ne
p[x] = Parent of node 'x';
else if min heap property is violated and p[x] is unmarked then
    Cut off the link between 'x' and p[x];
    Mark p[x];
    Add 'x' and its children to the root list and update min pointer if necessary;
```



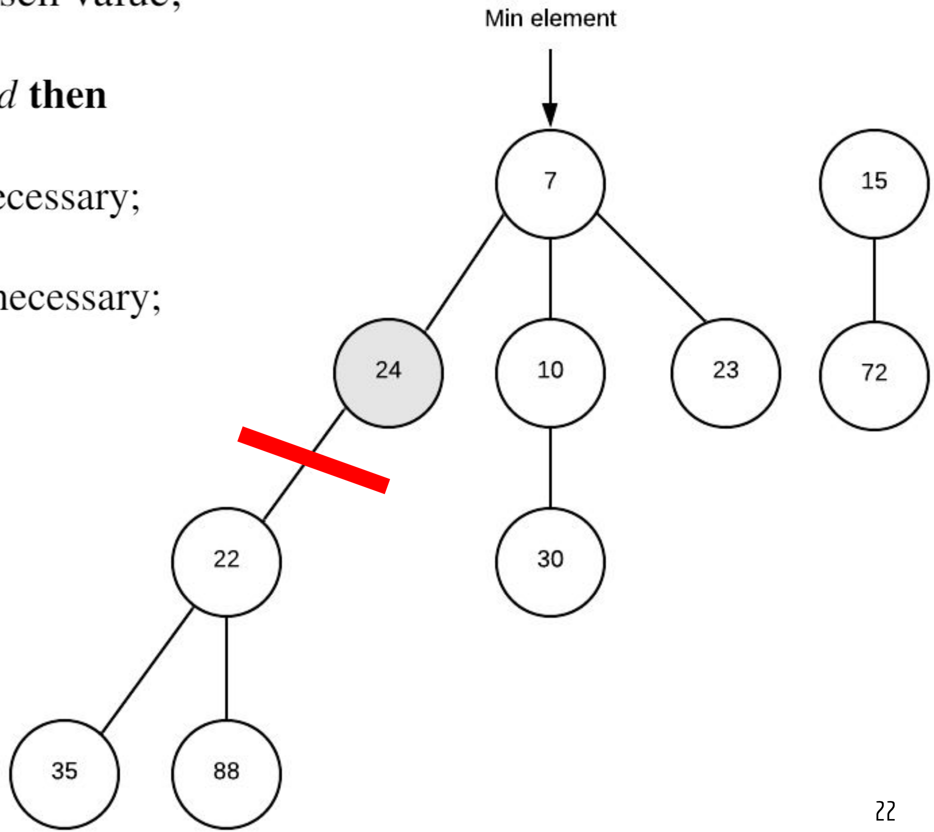
Decrease Key: 26->22

```
1  Decrease the value of the node 'x' to the new chosen value;
2  p[x] = Parent of node 'x';
9  else if min heap property is violated and p[x] is marked then
10 |   Cut off the link between 'x' and p[x];
11 |   Add 'x' to the root list and update min pointer if necessary;
12 |   Cut off link between p[x] and p[p[x]];
13 |   Add p[x] to the root list and update min pointer if necessary;
14 |   if p[p[x]] is unmarked then
15 |   |   Mark p[p[x]];
16 |   else
17 |   |   cut off p[p[x]];
18 |   |   x = p[p[x]];
19 |   |   Jump to line 11;
```

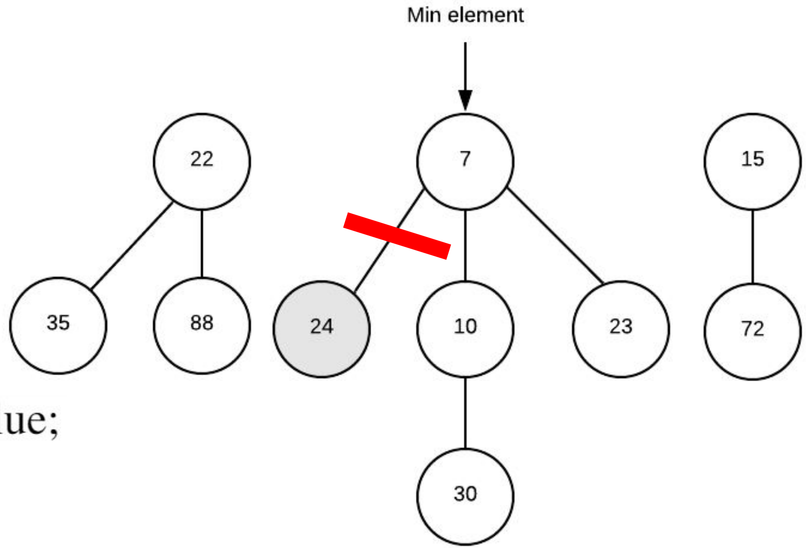


Decrease Key: 26->22

```
1  Decrease the value of the node ‘x’ to the new chosen value;
2  p[x] = Parent of node ’x’;
9  else if min heap property is violated and p[x] is marked then
10 |   Cut off the link between ‘x’ and p[x];
11 |   Add ‘x’ to the root list and update min pointer if necessary;
12 |   Cut off link between p[x] and p[p[x]];
13 |   Add p[x] to the root list and update min pointer if necessary;
14 |   if p[p[x]] is unmarked then
15 |   |   Mark p[p[x]];
16 |   else
17 |   |   cut off p[p[x]];
18 |   |   x = p[p[x]];
19 |   |   Jump to line 11;
```

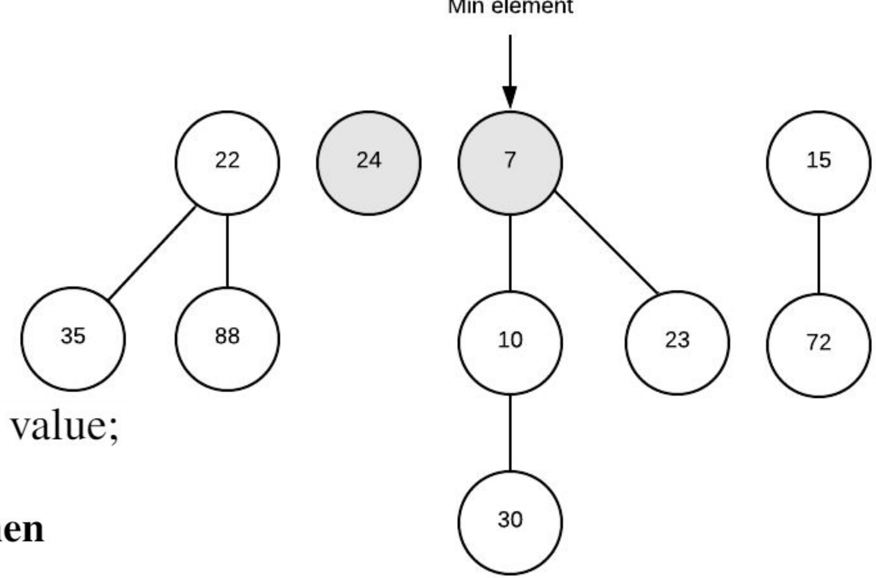


Decrease Key: 26->22




```
1  Decrease the value of the node 'x' to the new chosen value;
2  p[x] = Parent of node 'x';
9  else if min heap property is violated and p[x] is marked then
10 |    Cut off the link between 'x' and p[x];
11 |    Add 'x' to the root list and update min pointer if necessary;
12 |    Cut off link between p[x] and p[p[x]];
13 |    Add p[x] to the root list and update min pointer if necessary;
14 |    if p[p[x]] is unmarked then
15 |    |    Mark p[p[x]];
16 |    else
17 |    |    cut off p[p[x]];
18 |    |    x = p[p[x]];
19 |    |    Jump to line 11;
```

Decrease Key: 26->22




```
1  Decrease the value of the node 'x' to the new chosen value;
2  p[x] = Parent of node 'x';
9  else if min heap property is violated and p[x] is marked then
10     Cut off the link between 'x' and p[x];
11     Add 'x' to the root list and update min pointer if necessary;
12     Cut off link between p[x] and p[p[x]];
13     Add p[x] to the root list and update min pointer if necessary;
14     if p[p[x]] is unmarked then
15         Mark p[p[x]];
16     else
17         cut off p[p[x]];
18         x = p[p[x]];
19         Jump to line 11;
```

Delete Min

Algorithm 3: Delete
Source: Adapted from [4]

```
z = H.min;
if z != NULL then
    for each child x of z do
        add x to root list of H;
        x.p = NULL;
    end
    remove z from the root list of H;
    if z == z.right then
        H.min = NULL;
    else
        H.min = z.right;
        CONSOLIDATE(H);
    H.n = H.n-1;
return z;
```

Consolidate and Merge

Algorithm 4: Consolidate

Source: Adapted from [4]

```
let A[0...D(H.n)] = NULL, .., NULL;
for each node w in the root list of H do
    x = w;
    d = x.degree;
    while A[d] do
        y = A[d];
        if x.key > y.key then
            swap(x.key, y.key);
        MERGE(H,y,x);
        A[d] = NULL;
        d = d + 1;
    end
    A[d] = x;
end
H.min = NULL;
for i = 0 to D(H.n) do
    if A[i] then
        if !H.min then
            create a root list for H containing just A[i];
            H.min = A[i];
        else
            insert A[i] into H's root list;
            if A[i].key < H.min.key then
                H.min = A[i]
end
```

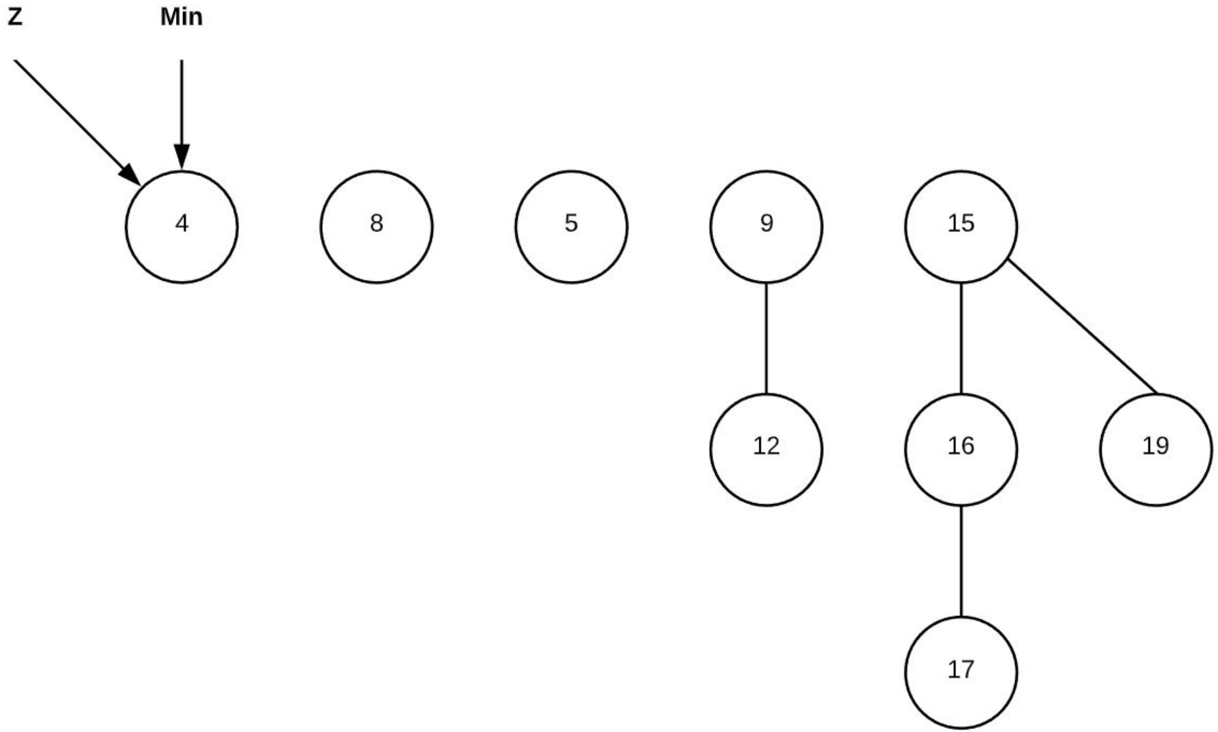
Merge

Algorithm 5: Merge
Source: Adapted from [4]

```
remove y from the root of list H;
make y a child of x, incrementing x.degree;
y.mark = FALSE;
```

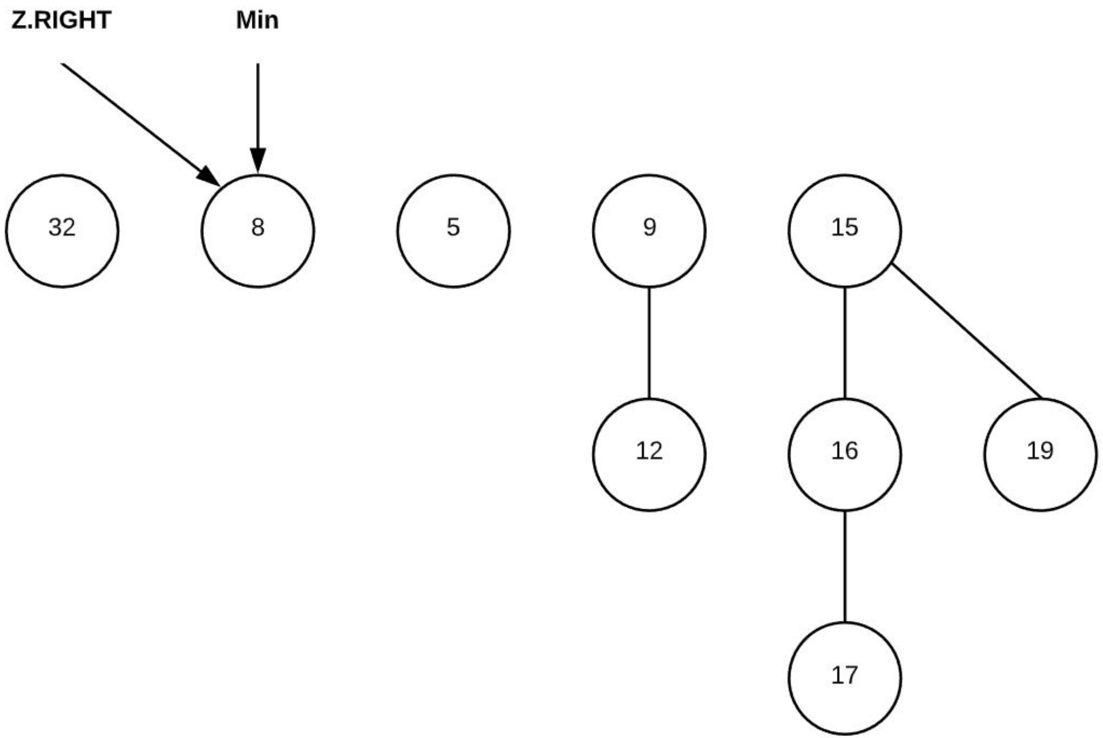
Delete Min

1 z = H.min;



Delete Min

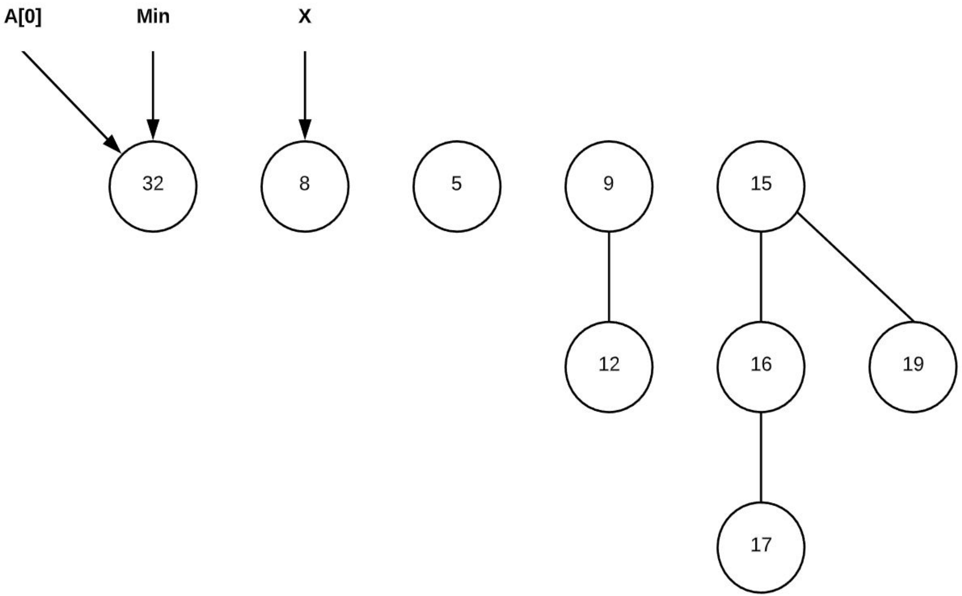
```
if z != NULL then
    for each child x of z do
        add x to root list of H;
        x.p = NULL;
    end
    remove z from the root list of H;
    if z == z.right then
        H.min = NULL;
    else
        H.min = z.right;
        CONSOLIDATE(H);
    H.n = H.n-1;
return z;
```



Consolidate / Merge

```
let A[0...D(H.n)] = NULL, .., NULL;
for each node w in the root list of H do
    x = w;
    d = x.degree;
    while A[d] do
        y = A[d];
        if x.key > y.key then
            swap(x.key, y.key);
        MERGE(H,y,x);
        A[d] = NULL;
        d = d + 1;
    end
    A[d] = x;
```

Consolidate/Merge



Root List = [32, 8, 5, 9, 15]

First Iteration:

d = 0

A[0] = 32

Second iteration:

x = 8

d = 0

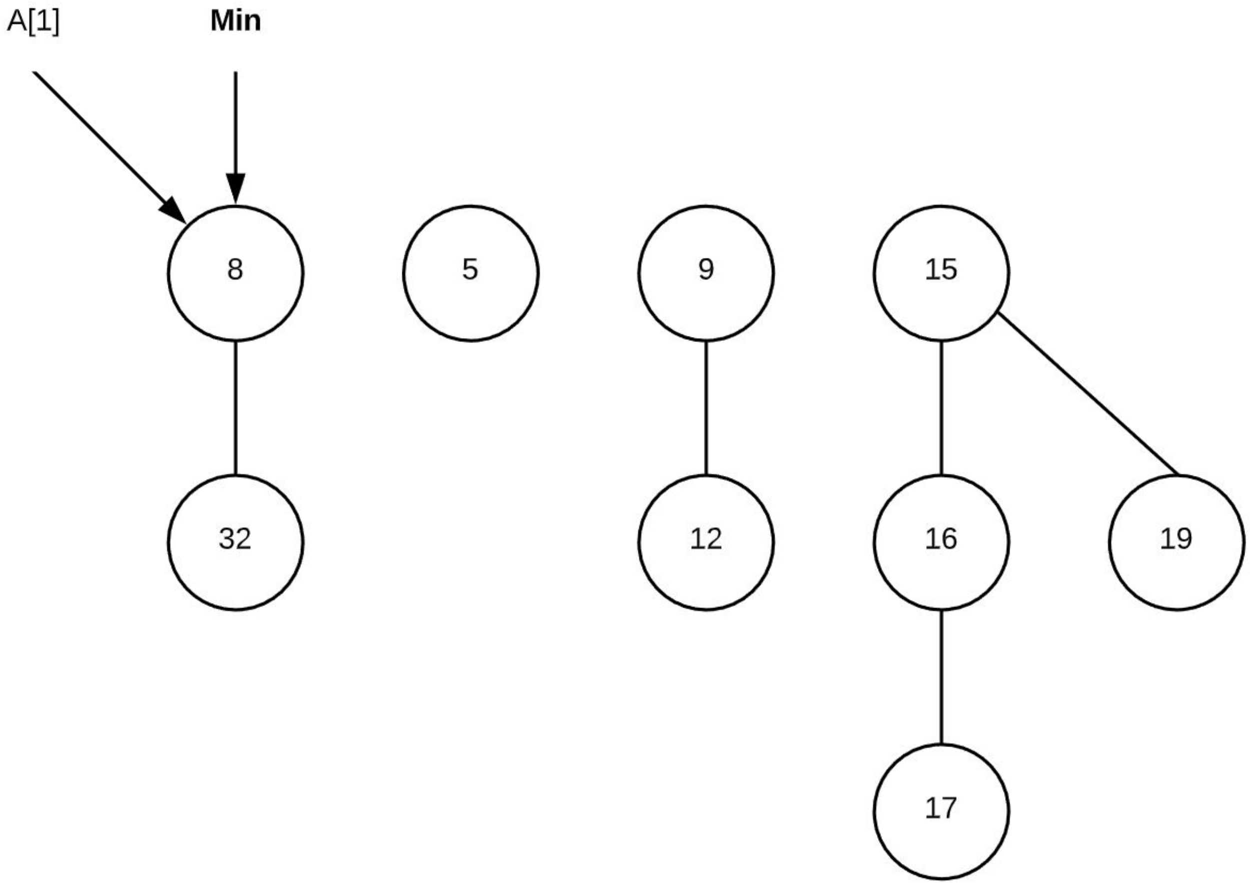
A[0] exists -> while loop:

Y = A[0] / Swap if necessary

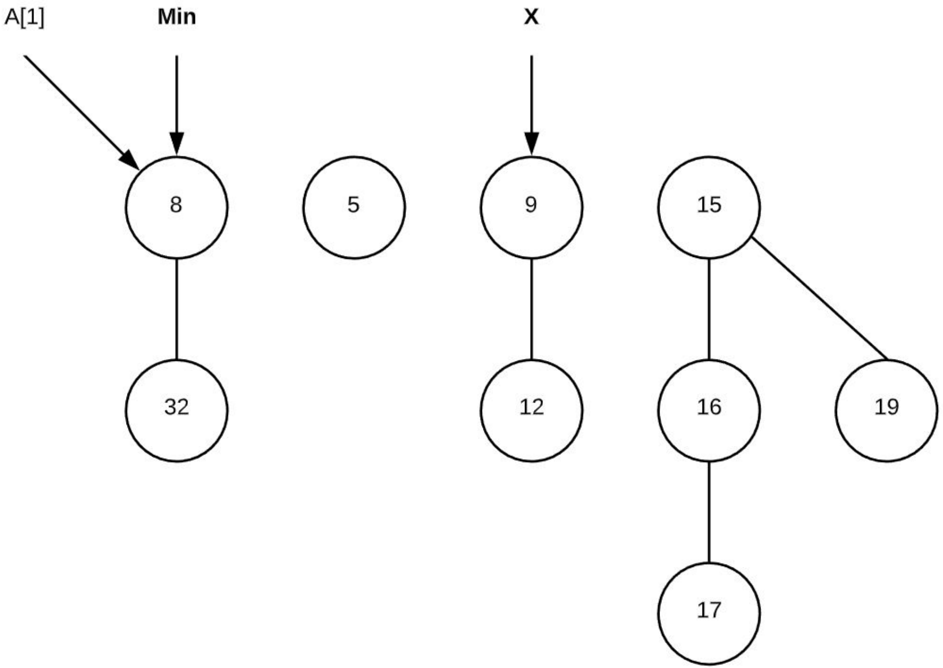
Call MERGE

A[1] = 8 and A[0] = NULL

Consolidate/Merge



Consolidate/Merge



Root List = [8, 5, 9, 15]

Third iteration:

A[0] = 5

Fourth iteration:

x = 9

d = 1

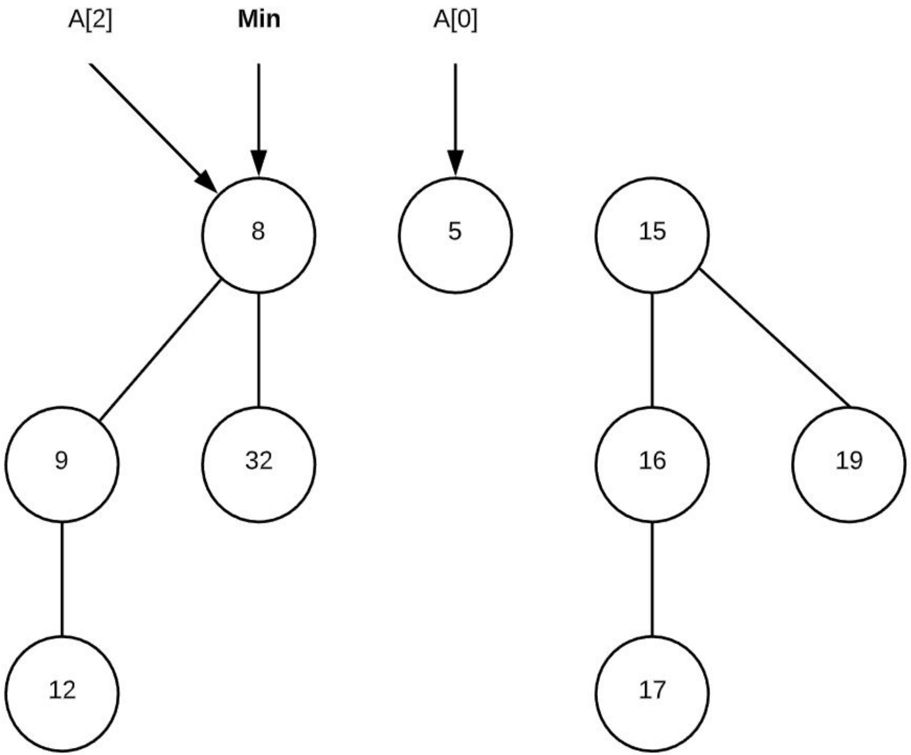
A[1] exists -> while loop:

Y = A[1] / Swap if necessary

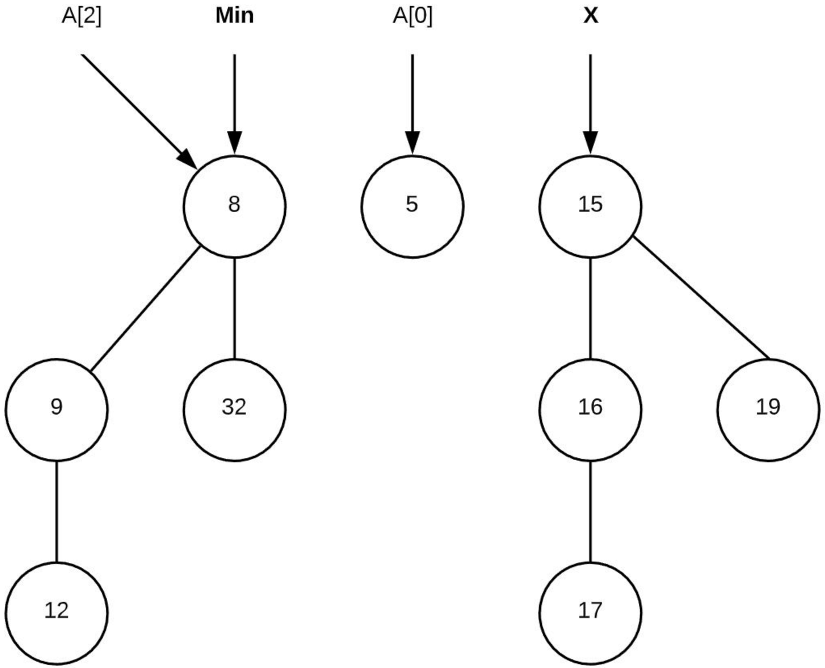
Call MERGE

A[2] = 8 and A[1] = NULL

Consolidate/Merge



Consolidate/Merge



Root List = [8, 5, 15]

Fifth iteration:

x = 15

d = 2

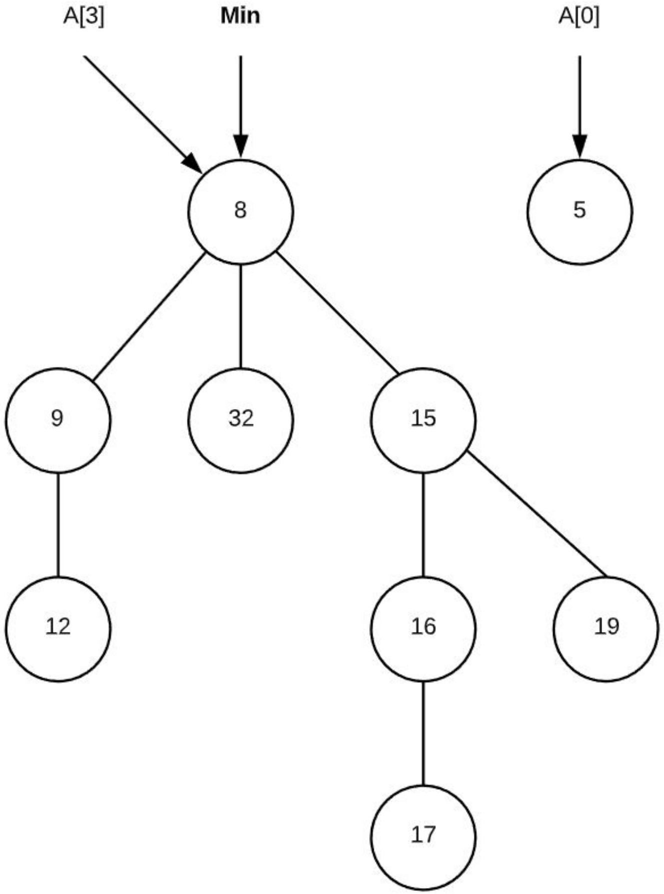
A[2] exists -> while loop:

Y = A[2] / Swap if necessary

CALL MERGE

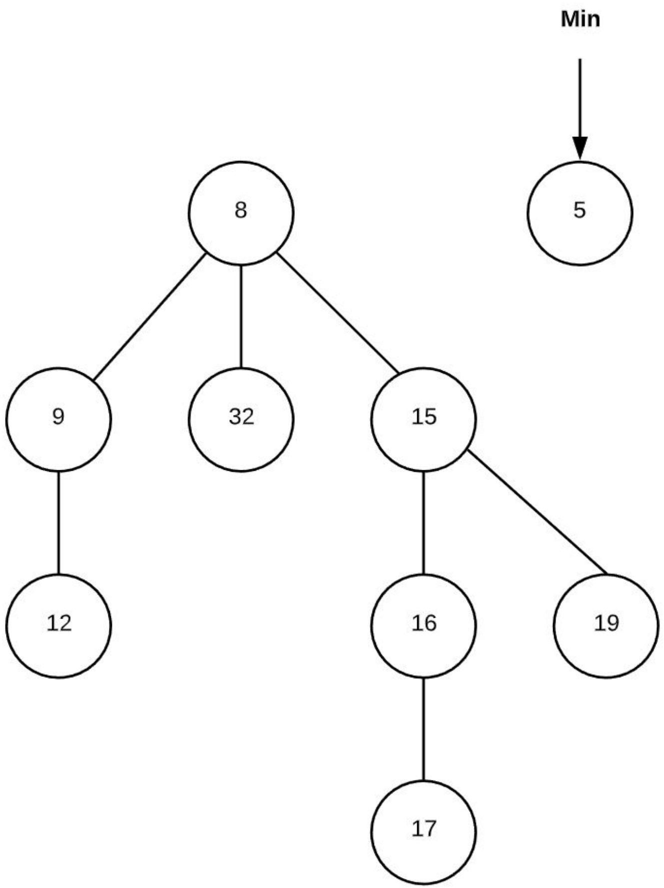
A[3] = 8 and A[2] = NULL

Consolidate/Merge



Consolidate

```
H.min = NULL;
for i = 0 to D(H.n) do
    if A[i] then
        if !H.min then
            create a root list for H containing just A[i];
            H.min = A[i];
        else
            insert A[i] into H's root list;
            if A[i].key < H.min.key then
                H.min = A[i]
end
```



Advantages and Disadvantages

Advantages

- “It is theoretically optimal” [5] (In theory, it has the best performance compared to other trees and heaps)
- Improves asymptotic running time for algorithms that require minimum values

Disadvantages

- Delete worst case has a time complexity of O(n)
- Difficult to implement (4 pointers per node must be manipulated)
- In practice, it is outperformed by simpler heaps and trees, for some applications [5]

Applications

Applications

- Used as the priority queue of Dijkstra Algorithm [1]

- Speed up the scaling algorithm of Edmonds and Karp, for minimum-cost network flow [6]

- Find shortest pairs of disjoint paths [6]

Thanks!
Questions?

Sources

[1] GeeksforGeeks. 2020. Fibonacci Heap | Set 1 (Introduction) - Geeksforgeeks. [online] Available at: <https://www.geeksforgeeks.org/fibonacci-heap-set-1-introduction/> [Accessed 5 July 2020].

[2] GeeksforGeeks. 2020. Fibonacci Heap - Deletion, Extract Min And Decrease Key - Geeksforgeeks. [online] Available at: <https://www.geeksforgeeks.org/fibonacci-heap-deletion-extract-min-and-decrease-key/?ref=rp> [Accessed 5 July 2020].

[3] CORMEN, T., LEISERSON, C., RIVEST, R., &STEIN, C.. (2009). Advanced data structures. En Introduction to Algorithms(pp.510-511). England: THE MIT PRESS.

[4] CORMEN, T., LEISERSON, C., RIVEST, R., &STEIN, C.. (2009). Advanced data structures. En Introduction to Algorithms(pp.512-517). England: THE MIT PRESS.

[5] People.ksp.sk. 2020. Fibonacci Heap | Gnarley Trees. [online] Available at: <https://people.ksp.sk/~kuko/gnarley-trees/?page_id=320> [Accessed 13 July 2020].

[6] Fredman, Michael L., and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms." Journal of the ACM (JACM) 34.3 (1987): 596-615.