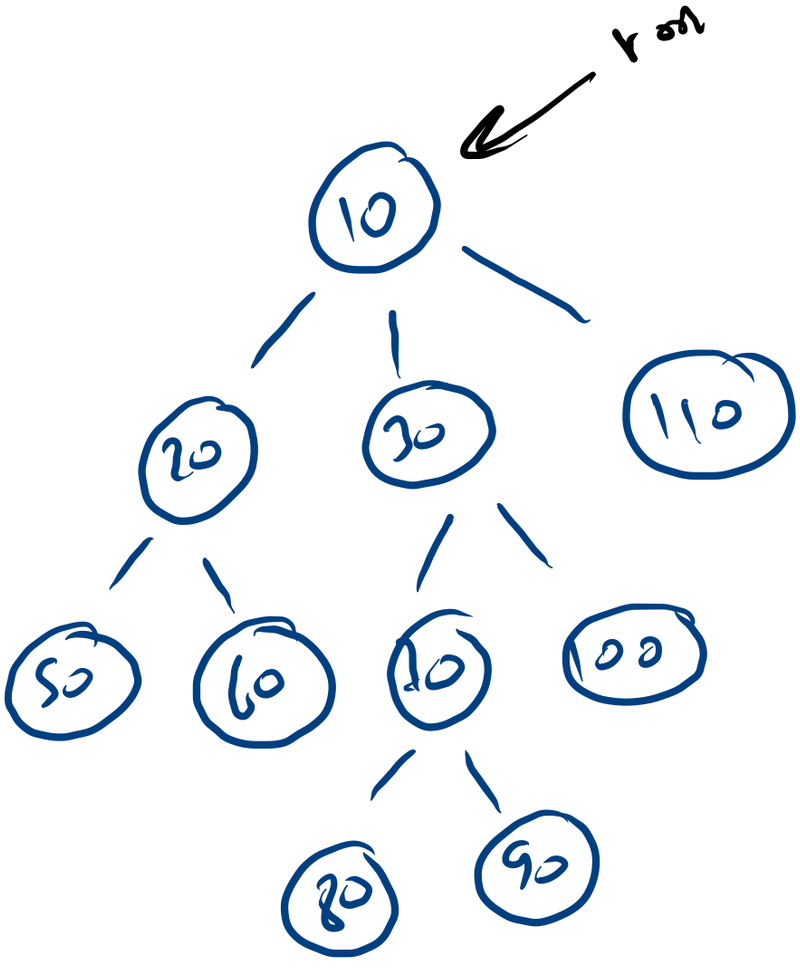
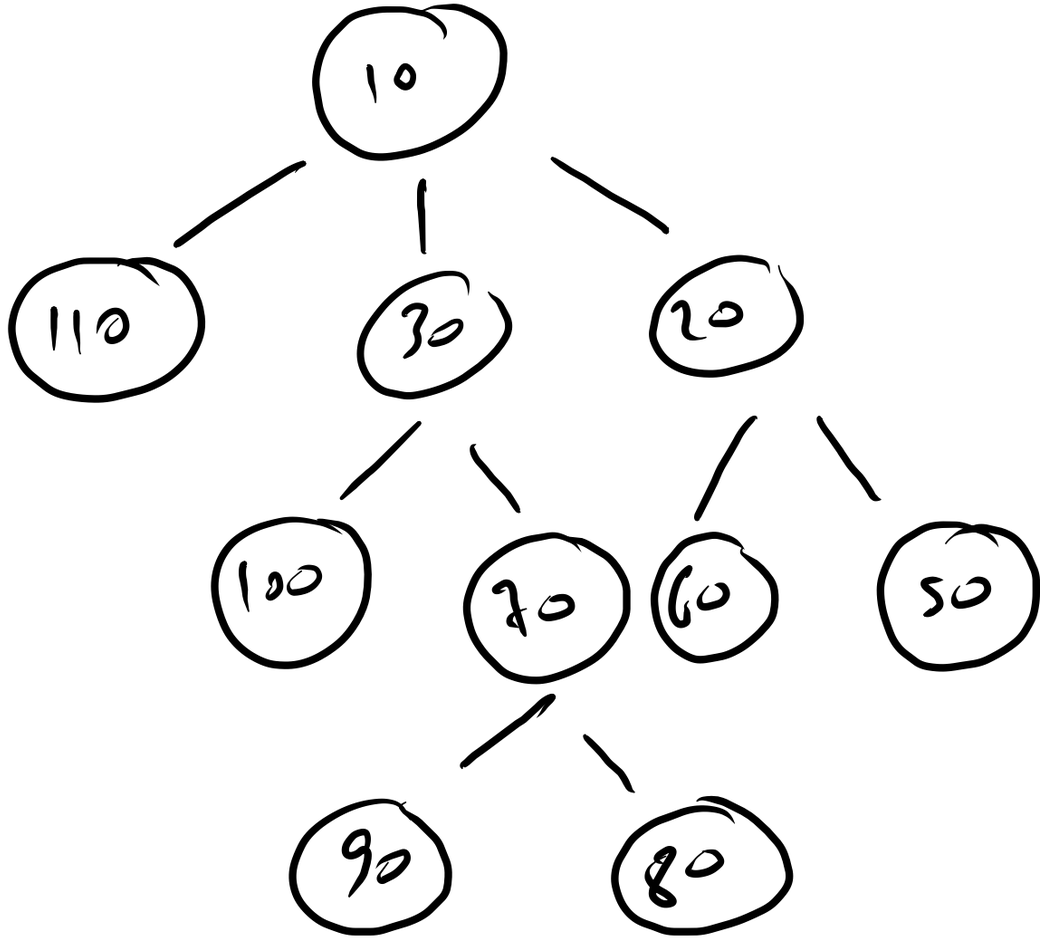
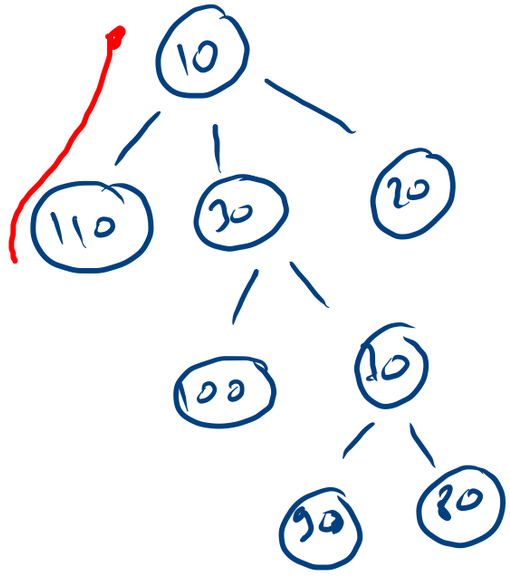
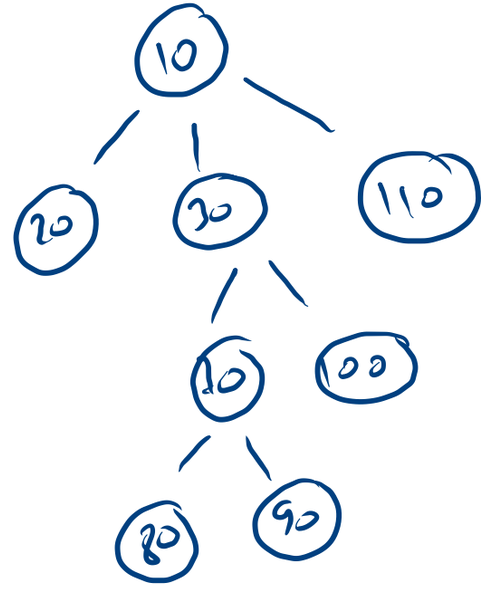
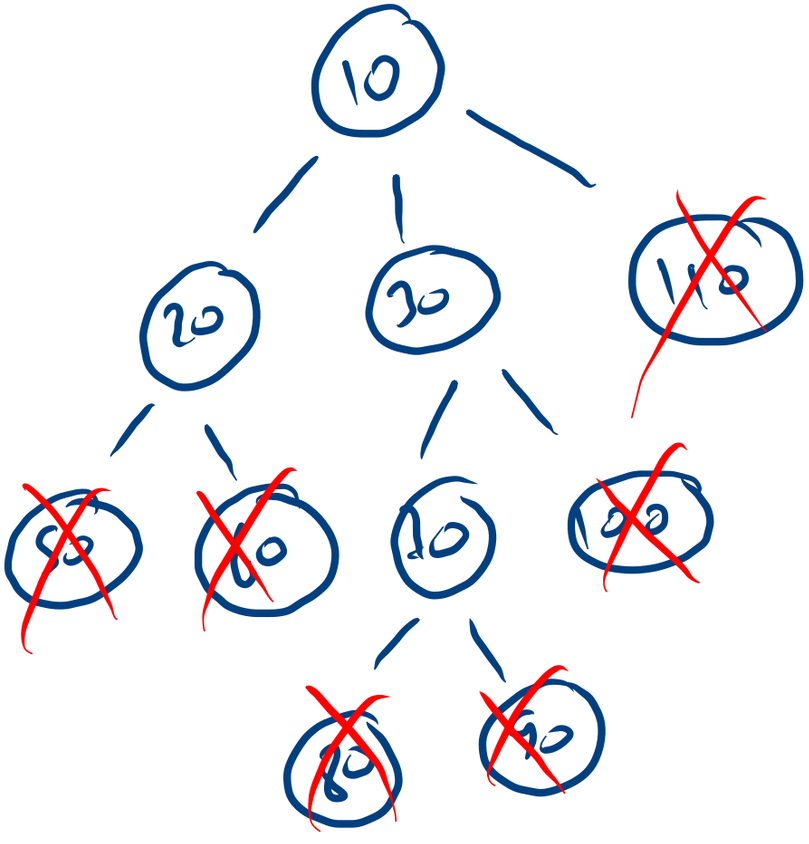

root
10
20
30
110
50
60
40
100
80
90

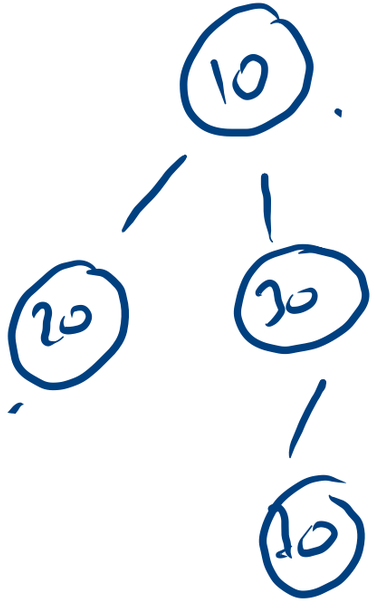

10
110
30
20
100
70
60
50
90
80

```java
public static void mirror(Node node){

    for(Node child: node.children){
        mirror(child);
    }

    int i=0;
    int j=node.children.size()-1;

    while(i<j){
        Node t = node.children.get(i);
        node.children.set(i, node.children.get(j));
        node.children.set(j, t);
        i++;
        j--;
    }

}
```

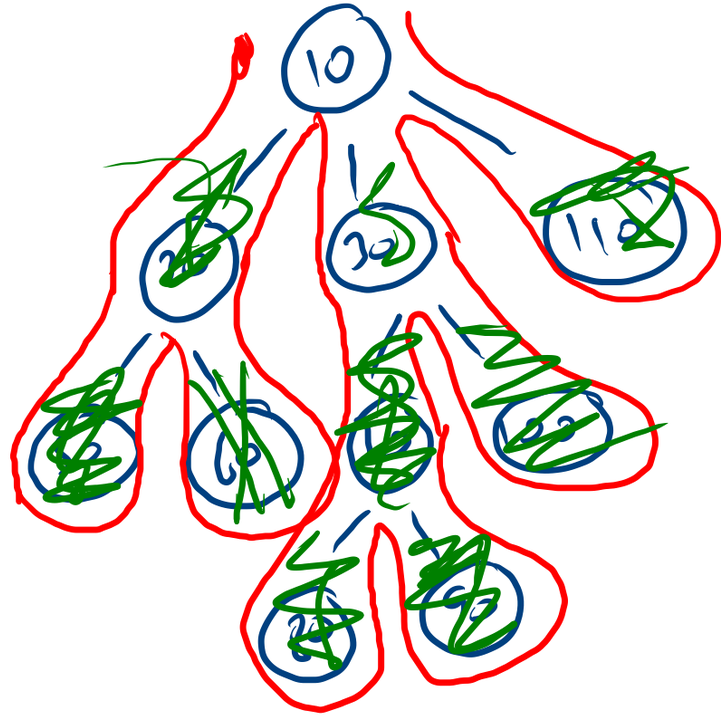




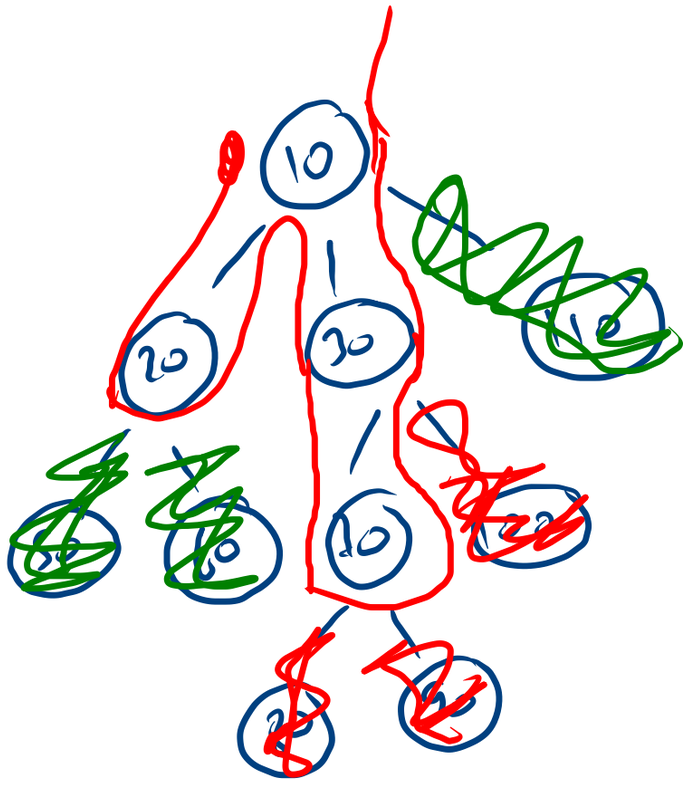

10
20
30
110
90
60
10
90
60
90

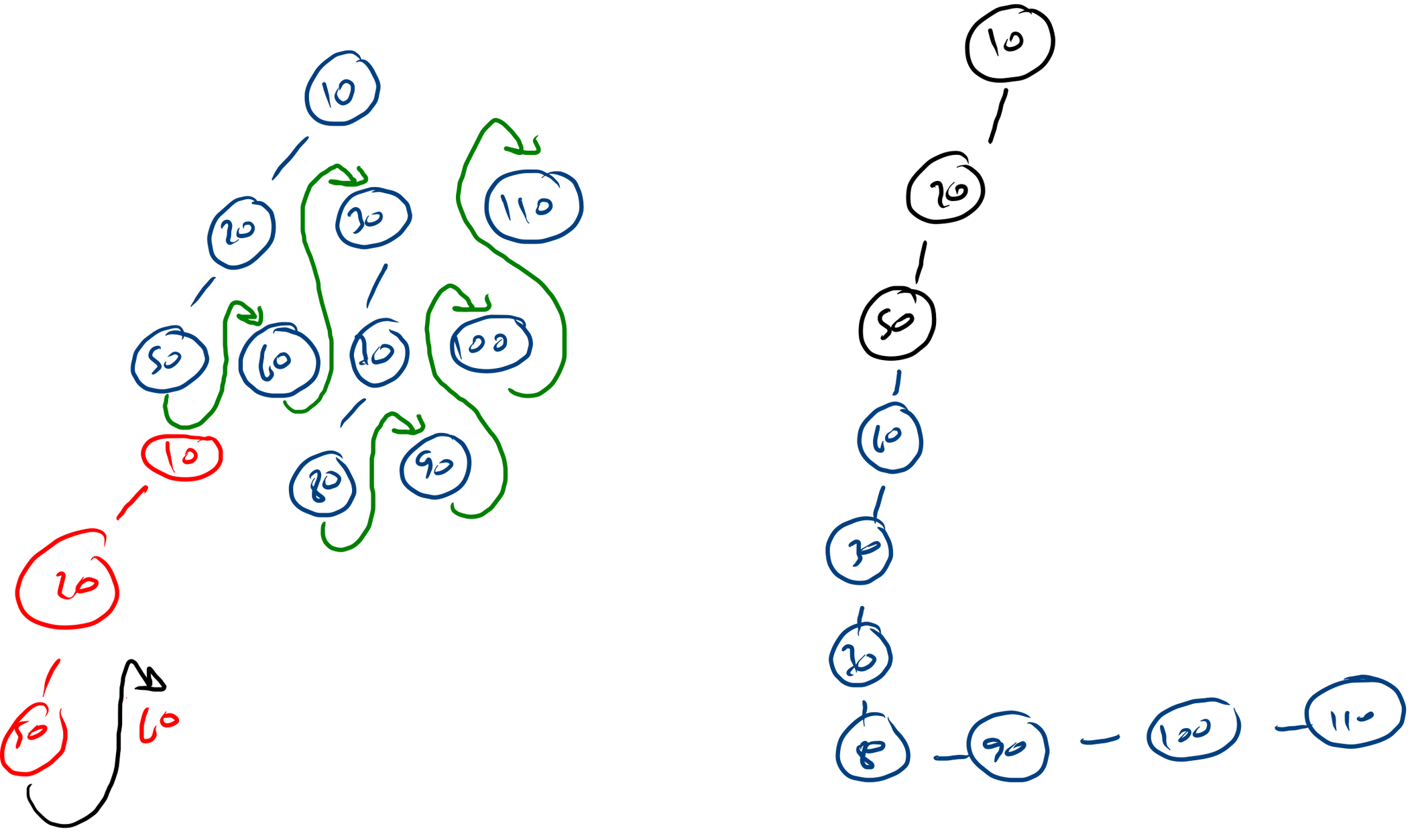

10
20
30
10

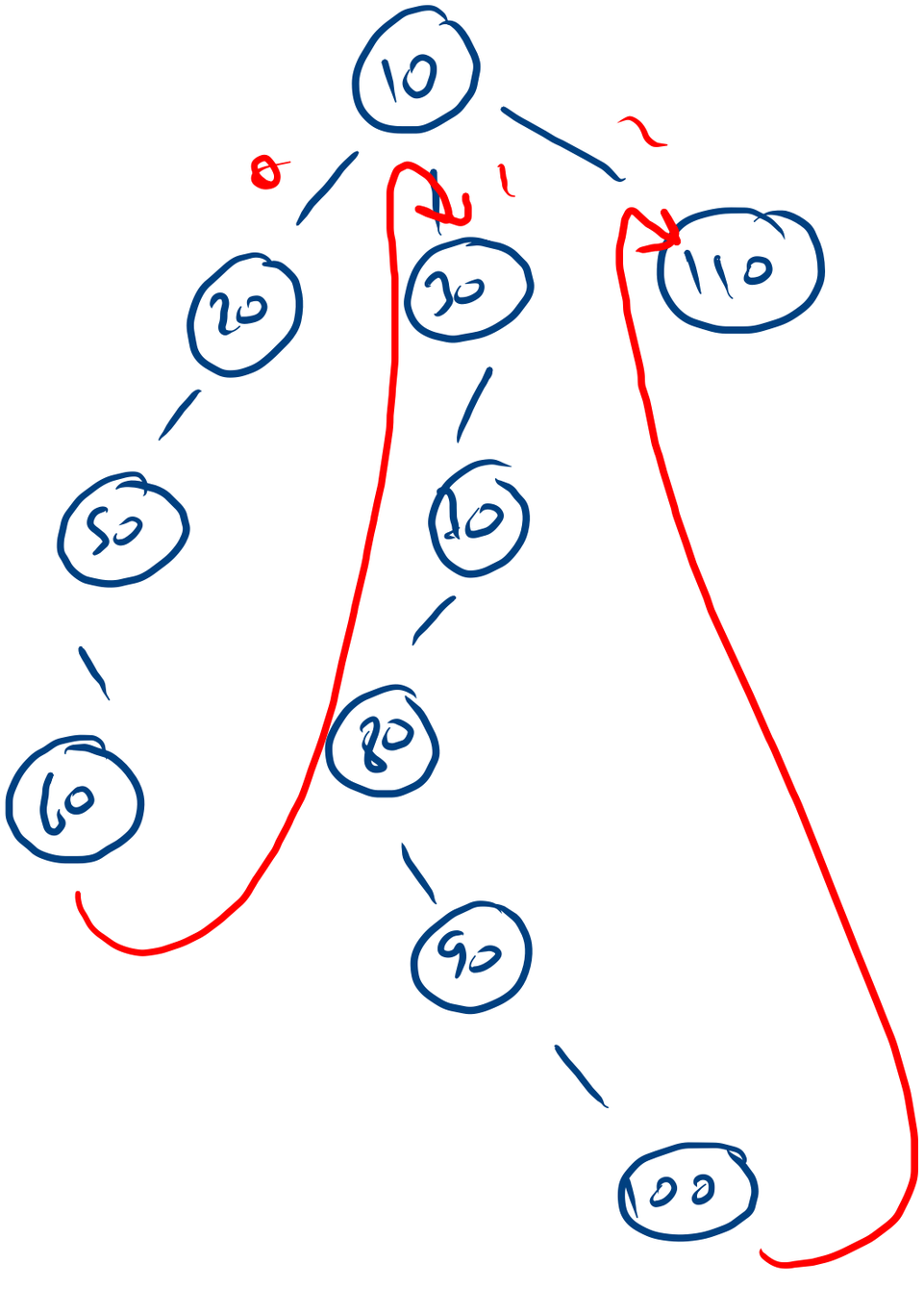
```java
public static void removeLeaves(Node node) {


    for (Node child : node.children) {
        removeLeaves(child);
    }

    for(int i=node.children.size()-1;i>=0;i--){
        Node child = node.children.get(i);

        if(child.children.size() == 0){
            node.children.remove(i);
        }
    }

}
```



```java
public static void removeLeaves(Node node) {



    for(int i=node.children.size()-1;i>=0;i--){
        Node child = node.children.get(i);

        if(child.children.size() == 0){
            node.children.remove(i);
        }
    }

    for (Node child : node.children) {
        removeLeaves(child);
    }
}
```

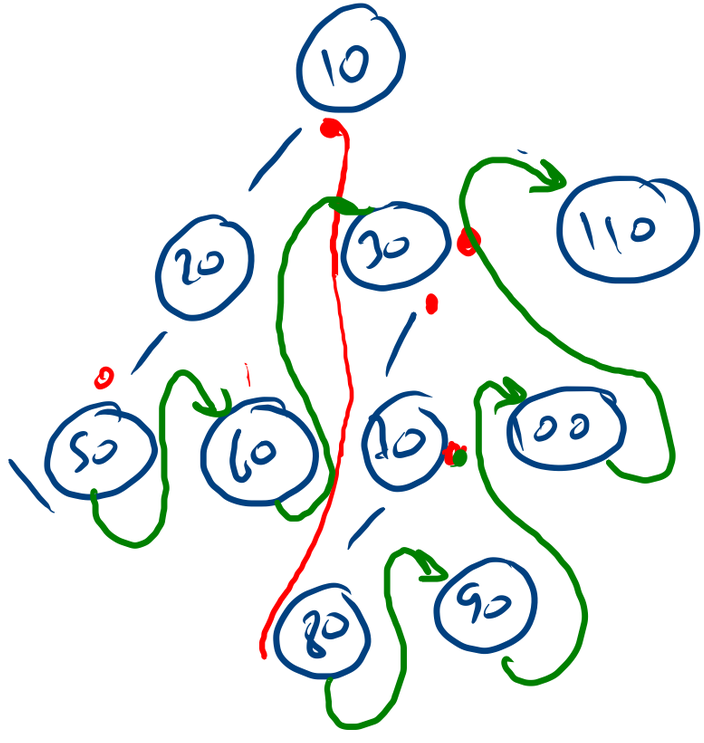


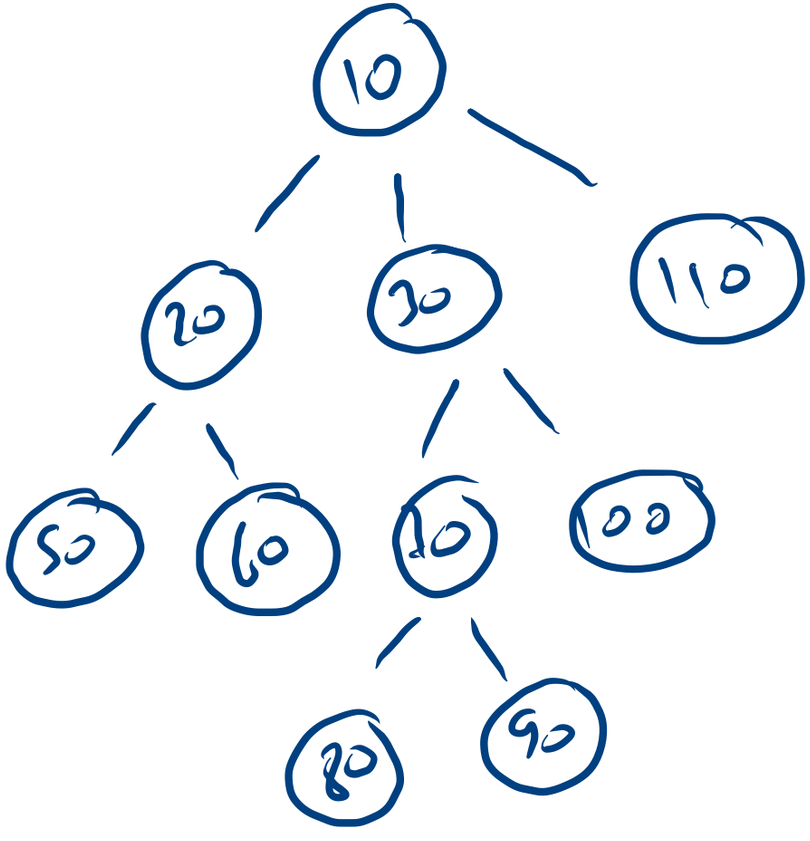

10
20
30
110
50
60
70
100
80
90
10
20
50
60
70
80
90
100
110
50
60


10
0
1
1
20
30
110
50
40
60
80
90
100

```
private static Node getLeaf(Node node){
    while(node.children.size() >0){
        node = node.children.get(0);
    }
    return node;
}
```

```
public static void linearize(Node node){


    for(Node child: node.children){
        linearize(child);
    }

    for(int i=0;i<node.children.size()-1;i++){
        Node child = node.children.get(i);
        Node nextChild = node.children.get(i+1);


        Node leaf = getLeaf(child);
        leaf.children.add(nextChild);
    }

    for(int i=node.children.size()-1;i>=1;i--){
        node.children.remove(i);
    }

}
```



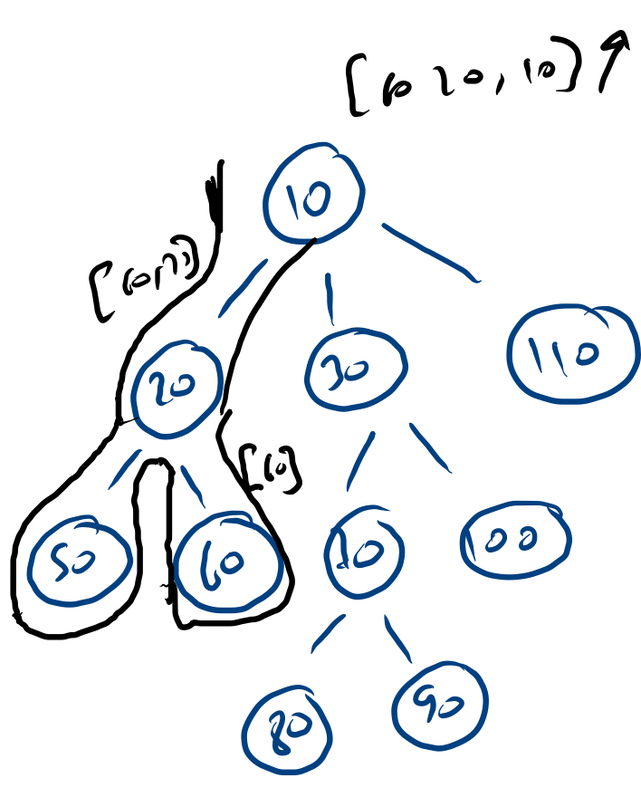


data → 70

[70 70 10]

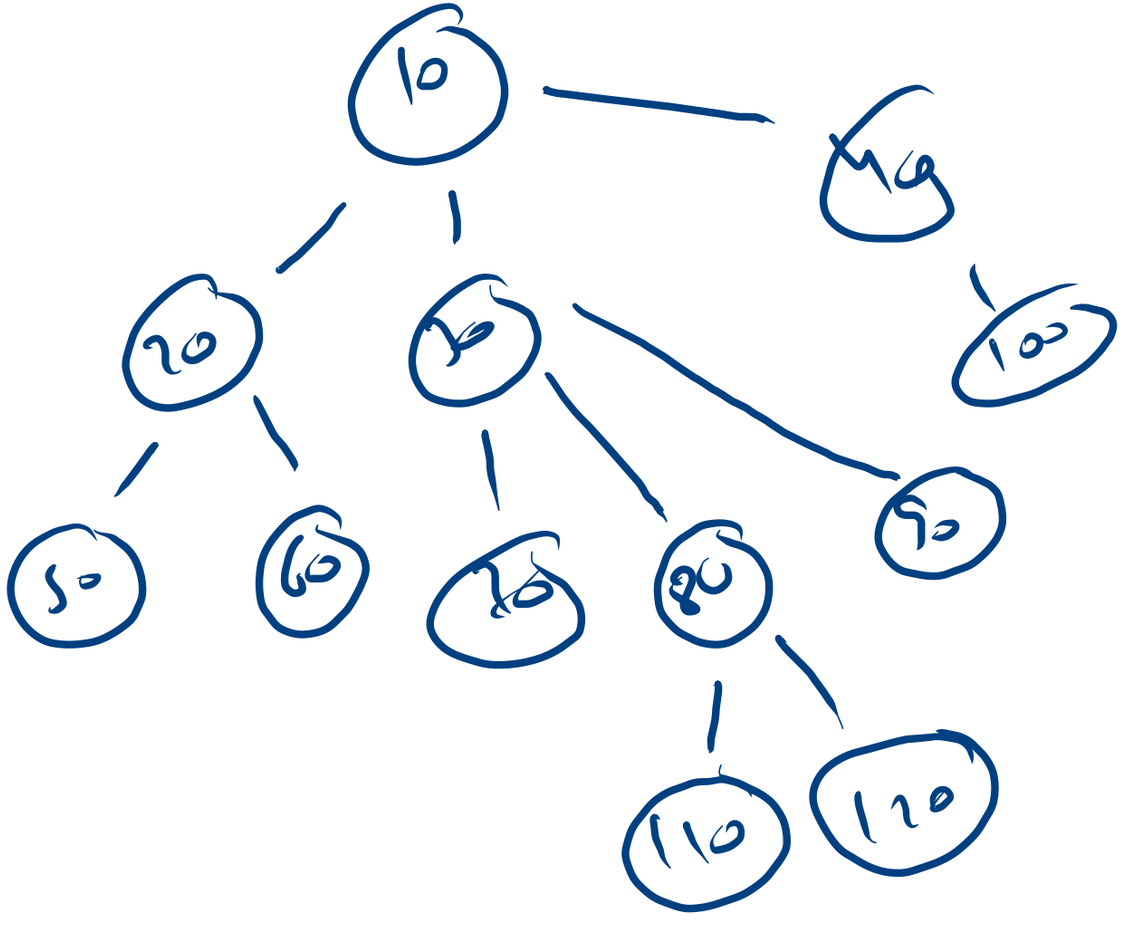
data → 10000

```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data){

    if(node.data == data){
        ArrayList<Integer> ans =new ArrayList<Integer>();

        ans.add(node.data);
        return ans;
    }

    for(Node child: node.children){
        ArrayList<Integer> ans = nodeToRootPath(child, data);

        if(ans.size() > 0){
            ans.add(node.data);
            return ans;
        }
    }

    return new ArrayList<Integer>();
```

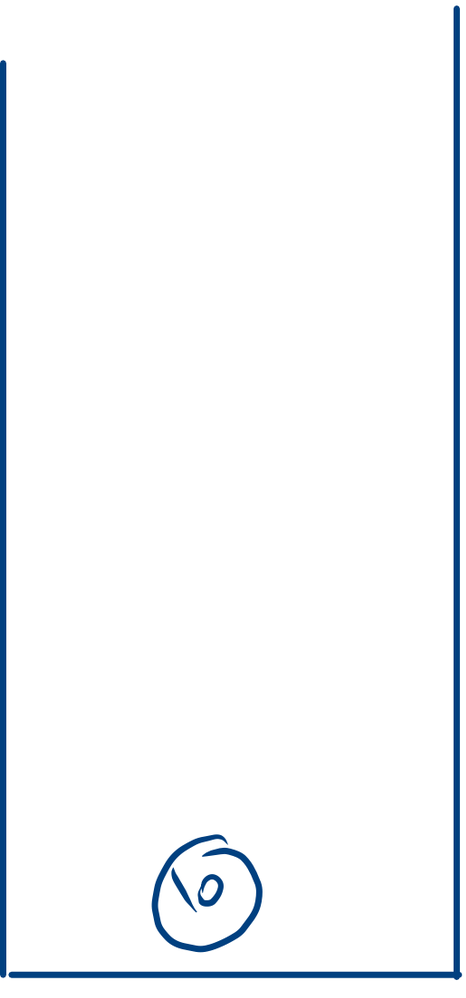


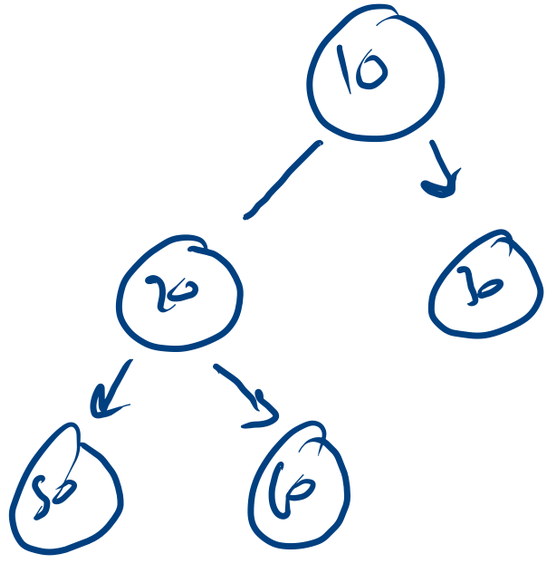
dth = 60

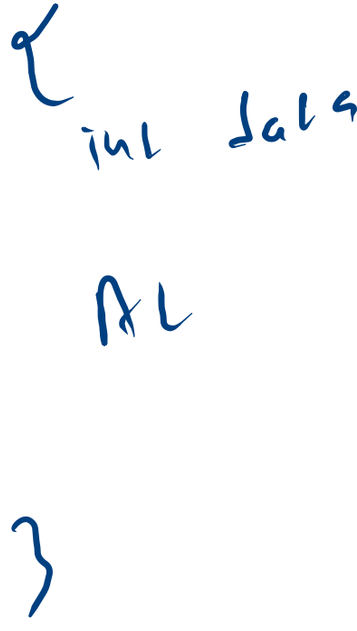
10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1



10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1

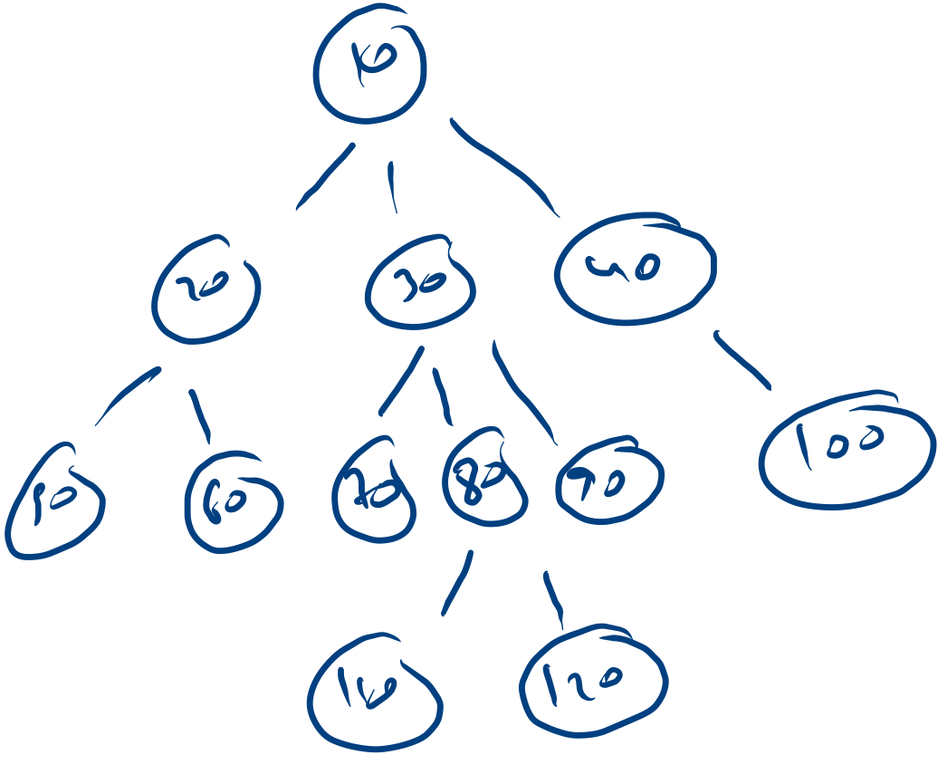


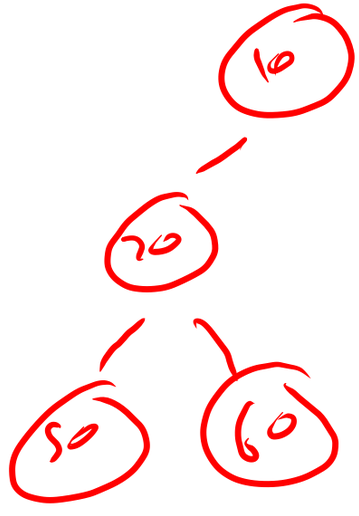


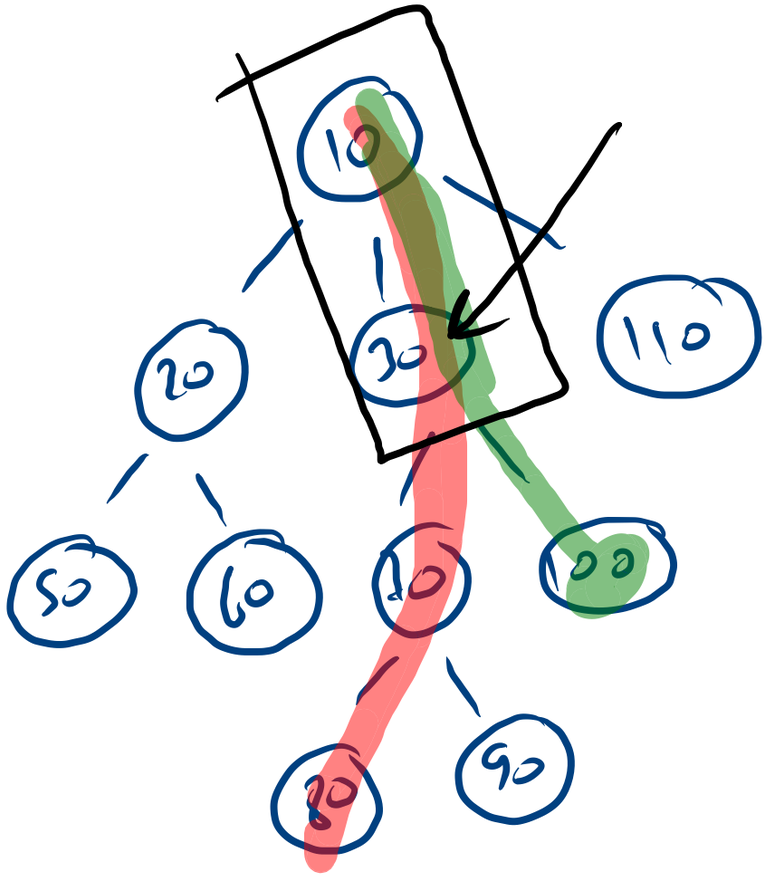


10 20 50 -1 60 -1 -1 30 70 -1 80 110 -1 120 -1 -1 90 -1 -1 40 100 -1 -1 -1

```
10 -> 20, 30, 40, .
20 -> 50, 60, .
50 -> .
60 -> .
30 -> 70, 80, 90, .
70 -> .
80 -> 110, 120, .
110 -> .
120 -> .
90 -> .
40 -> 100, .
100 -> .
```







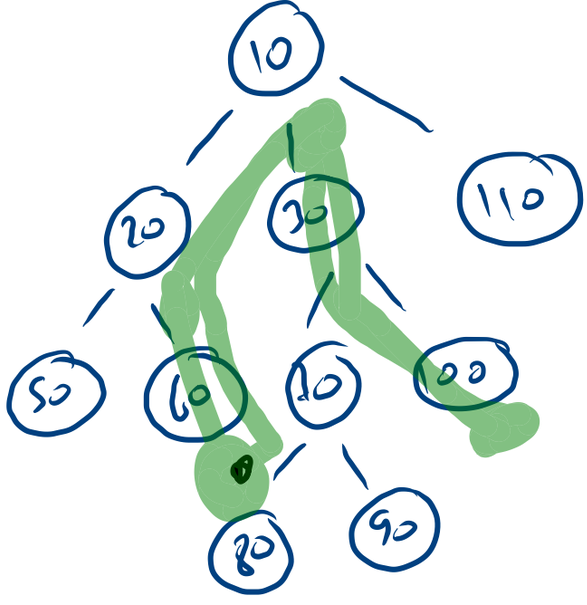
LCA

d1 → 80

d2 → 100

80 → [80 70 30 10]

100 → [100 30 10]



4

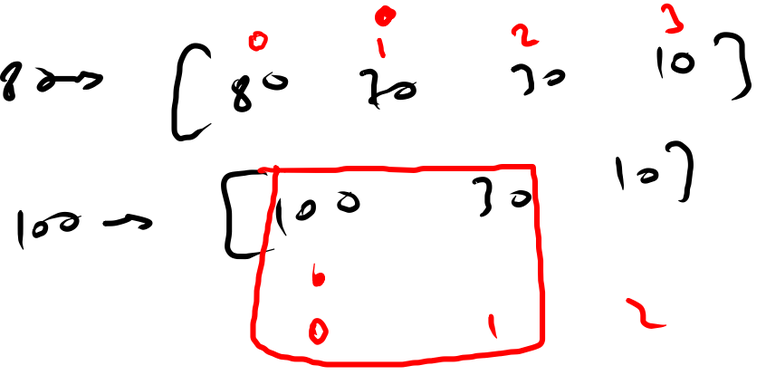
dl → 60

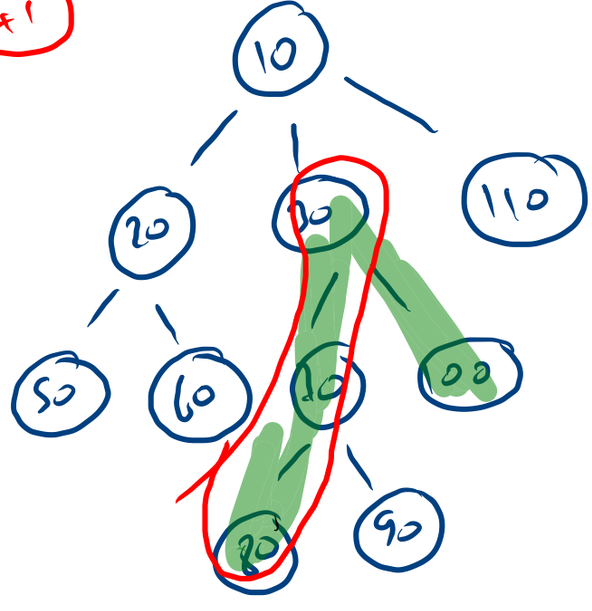
dr → 100

if!

if!

if!



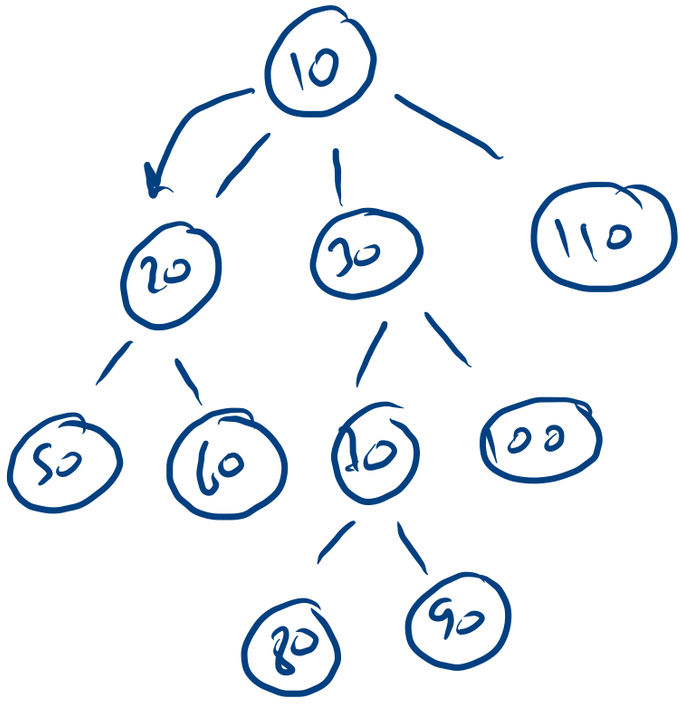


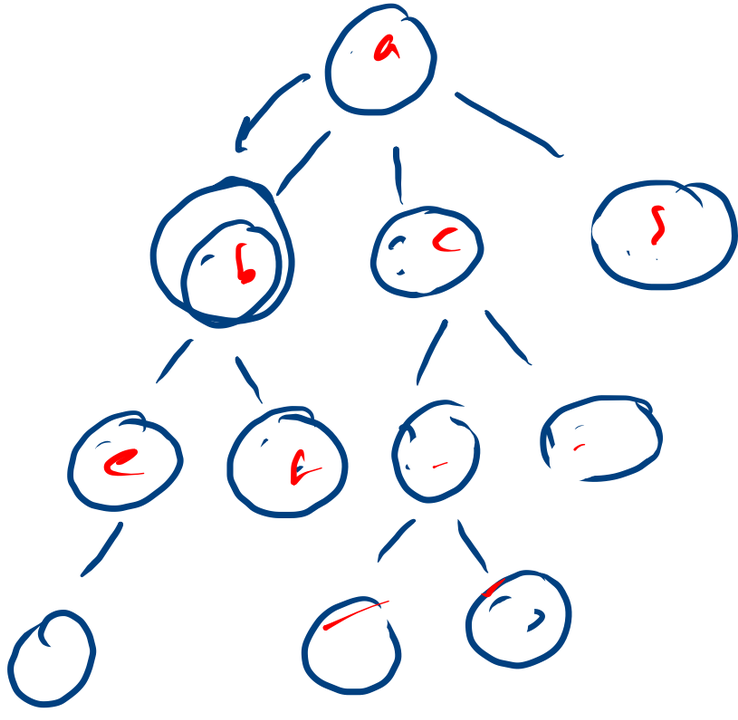
dl → 80

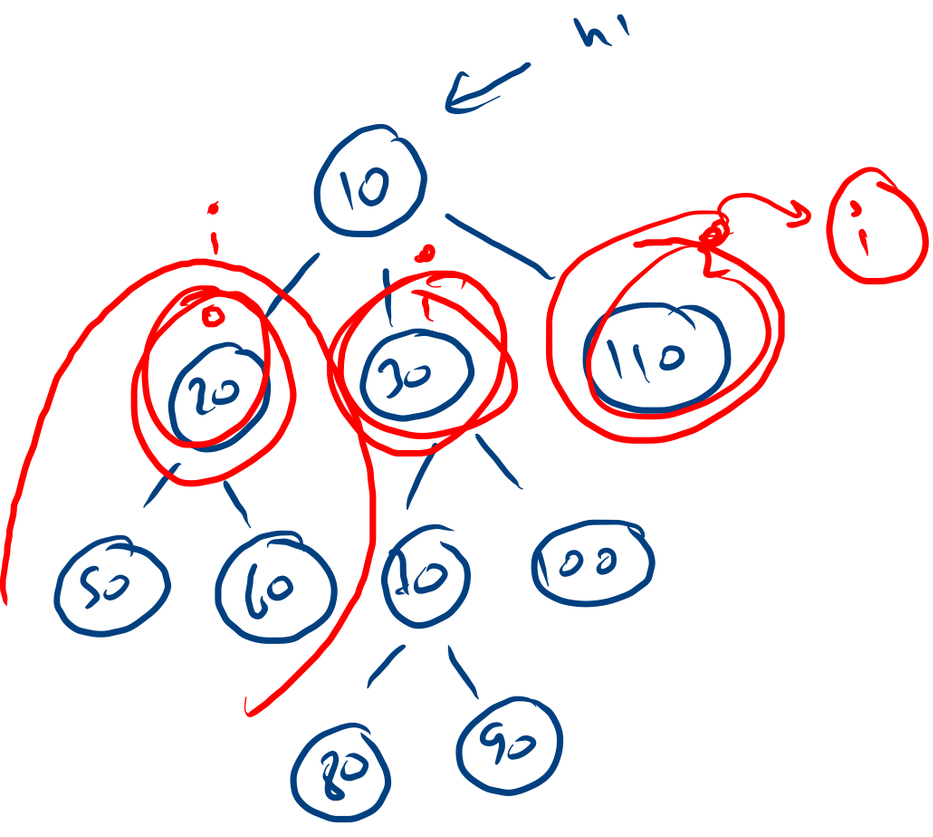
dr → 100

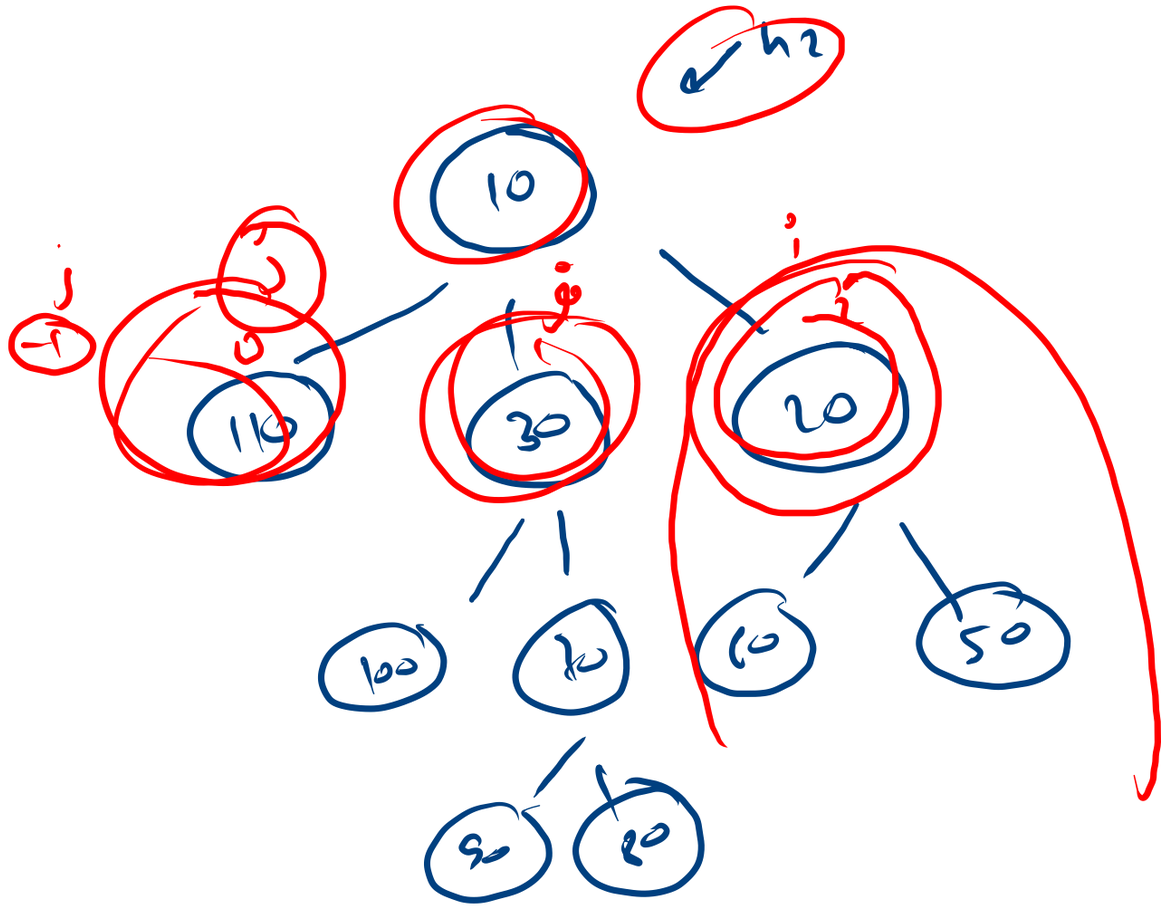
3

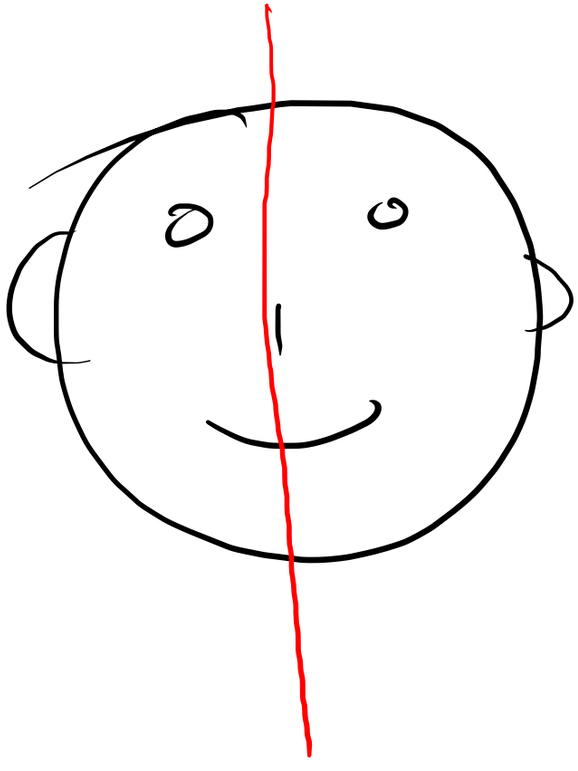
Similar

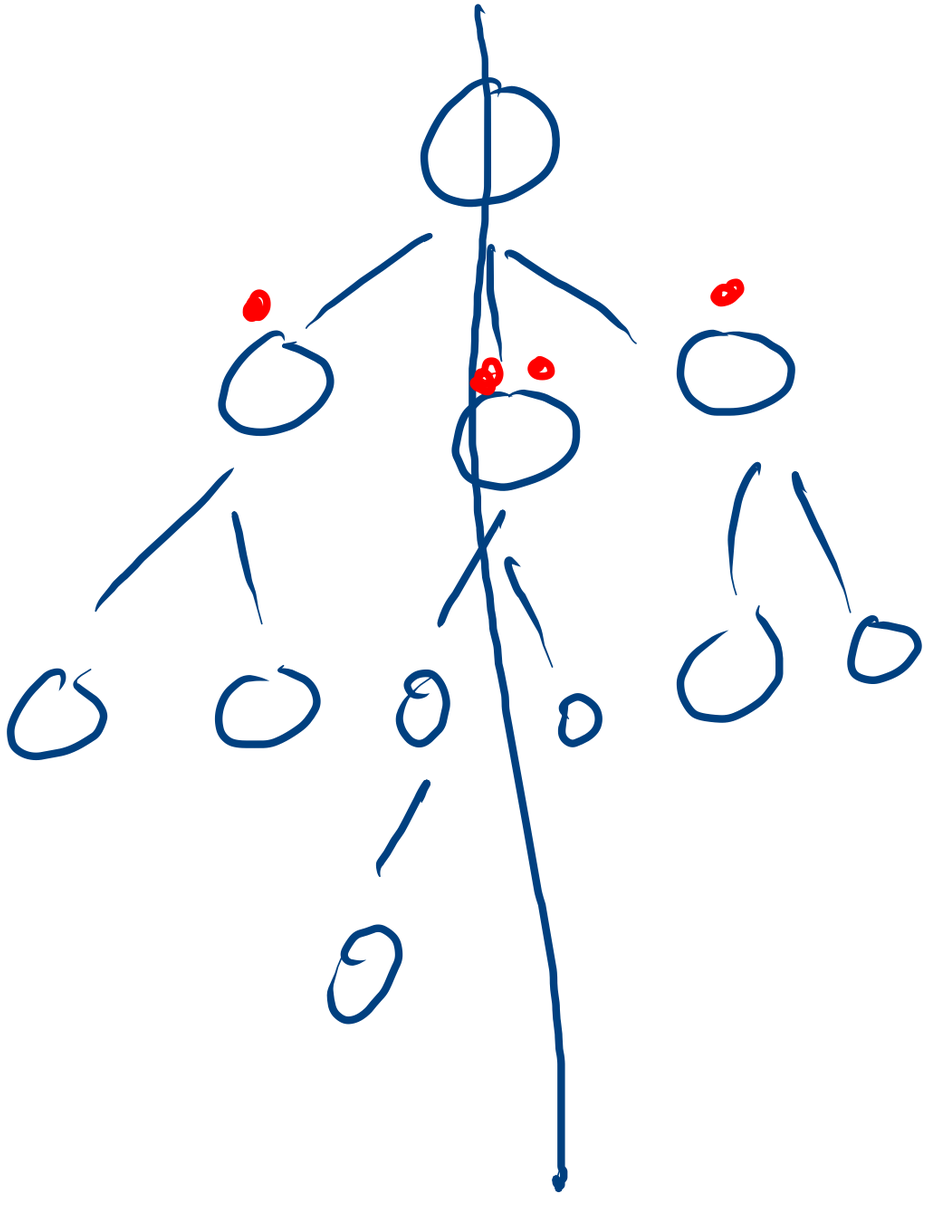


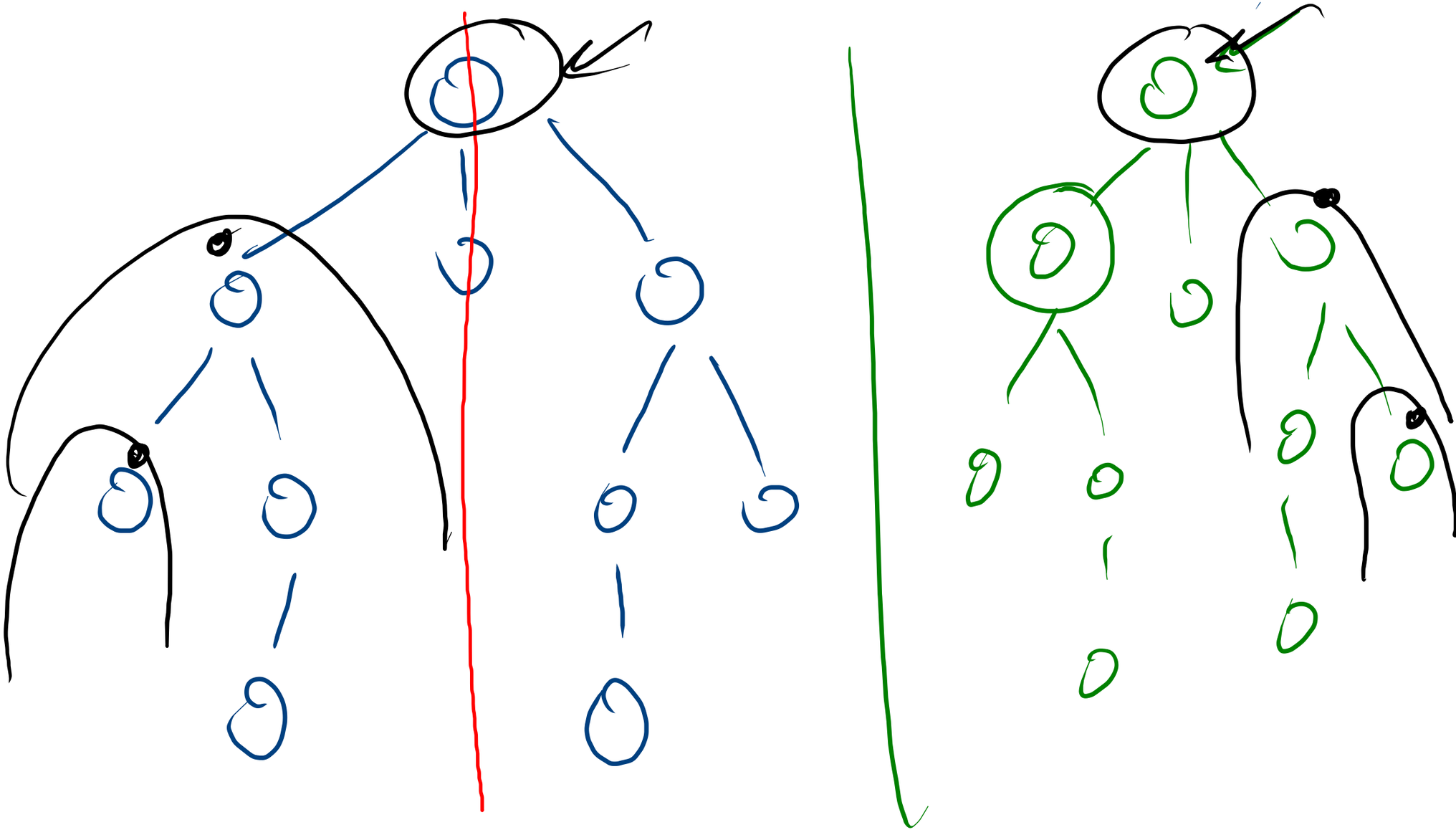



h1
10
20
30
110
1
50
60
10
100
80
90

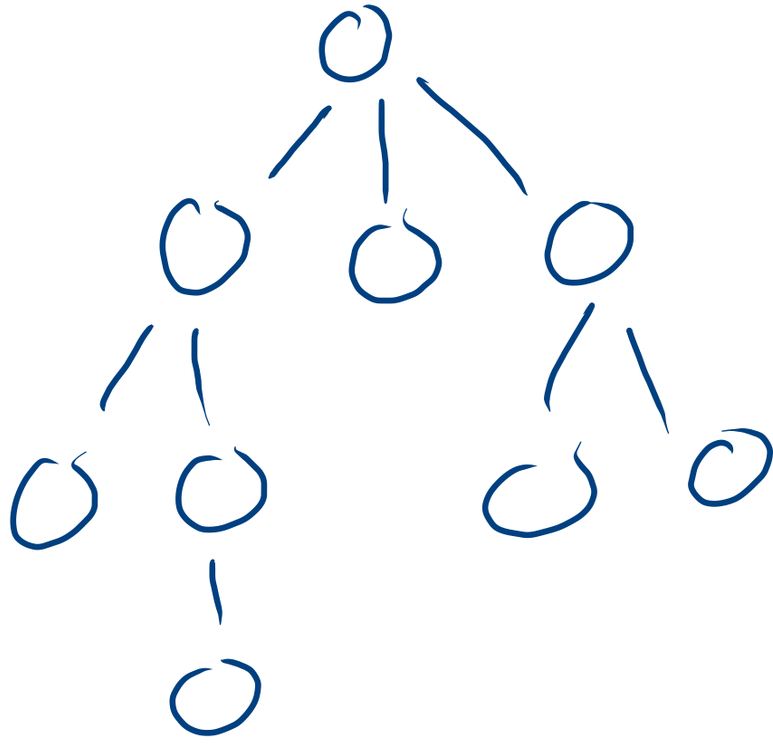

h2
10
110
30
20
100
10
10
50
90
10







```java
public static boolean areMirror(Node n1, Node n2) {

    if(n1.children.size() != n2.children.size())return false;

    int i=0;
    int j = n2.children.size()-1;

    while(j>=0){
        Node c1 = n1.children.get(i);
        Node c2 = n2.children.get(j);

        if(false == areMirror(c1,c2)){
            return false;
        }
        i++;
        j--;
    }

    return true;
}

public static boolean IsSymmetric(Node node) {
    return areMirror(node, node);
}
```



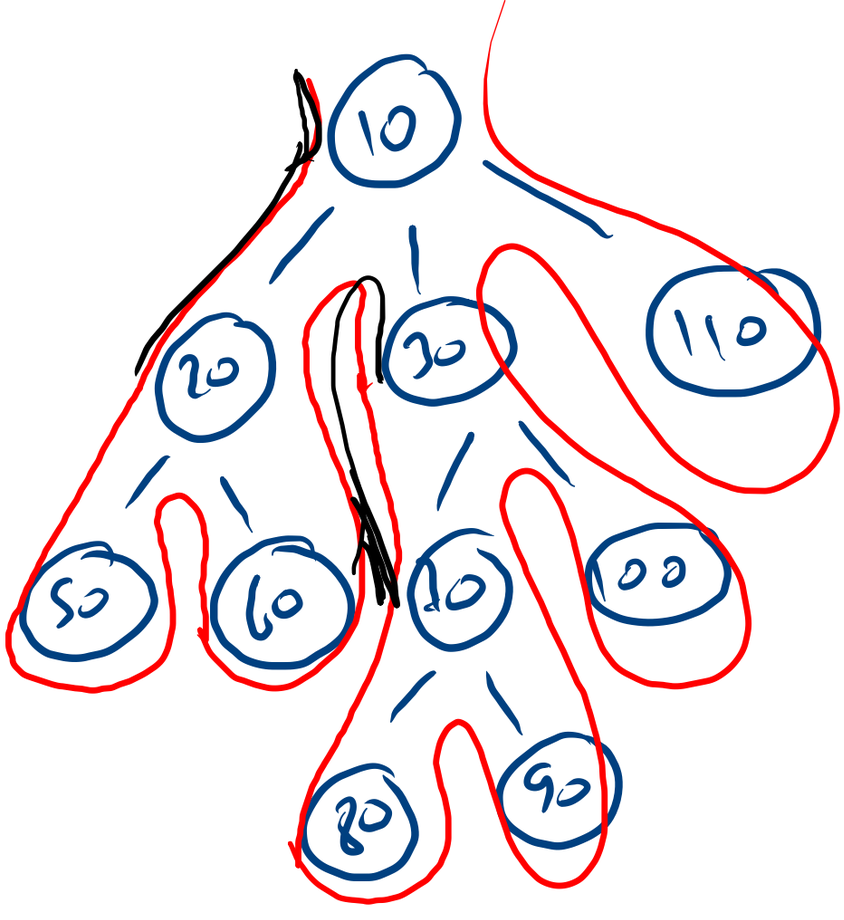
9 2 8 9 0 1 5 1 7 7

aakash.duhan@pepcoding.com

vanshika.sethi@pepcoding.com

shushami.rastogi@pepcoding.com

→ 7 0 1 5 2 8 4 3 9 2



70
30
10
10

di → 70
10

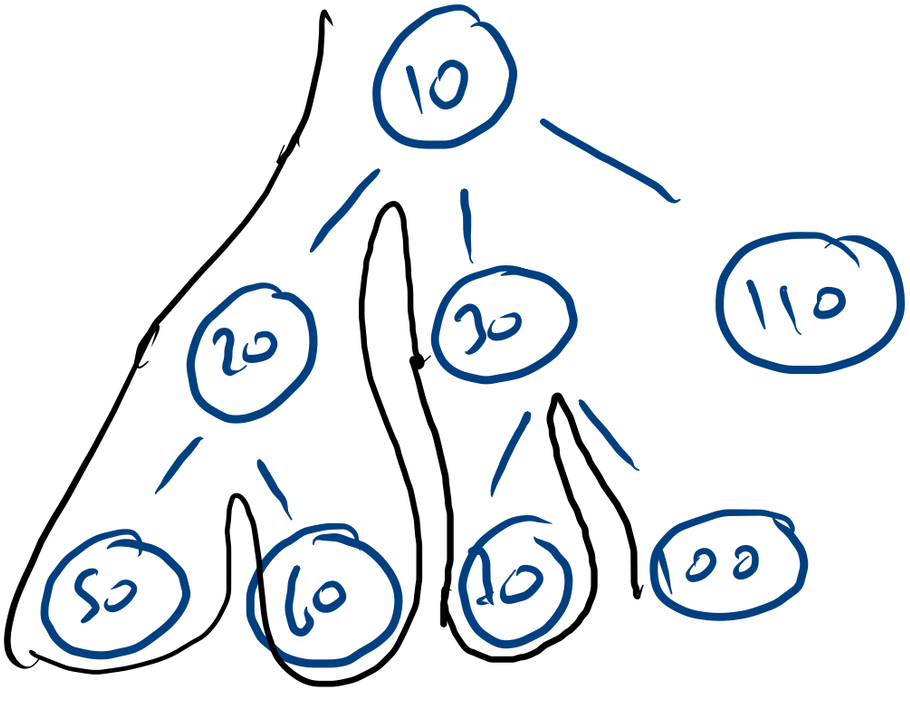
predecessor → 30 null

succor → 80 null

10

prede 50

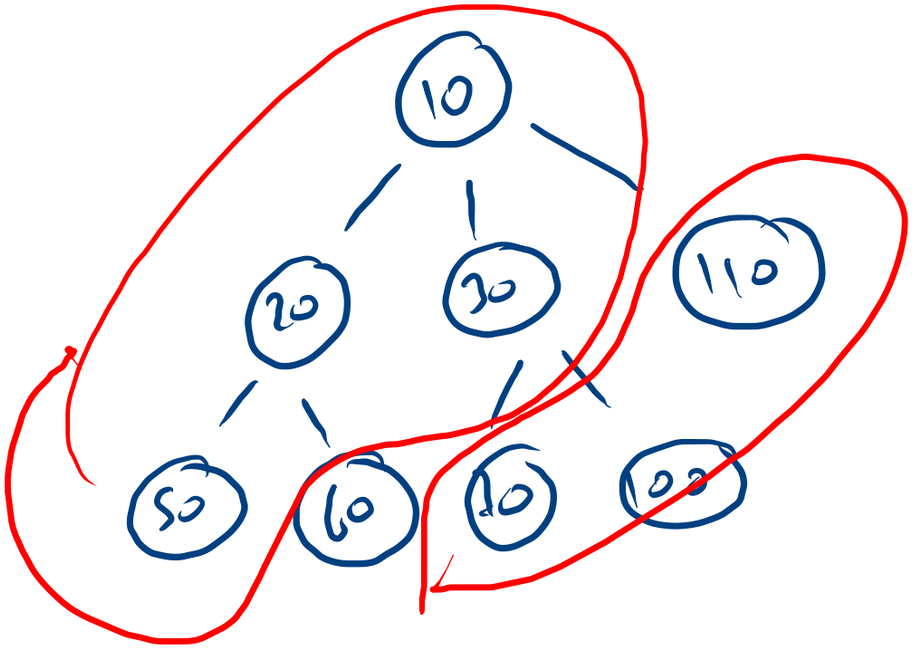
succ → 30



if data == target → flag = 1

else if flag = 0 → perform bob

else if flag = 1 → success block flag = 2



target → 60

ceil → 70

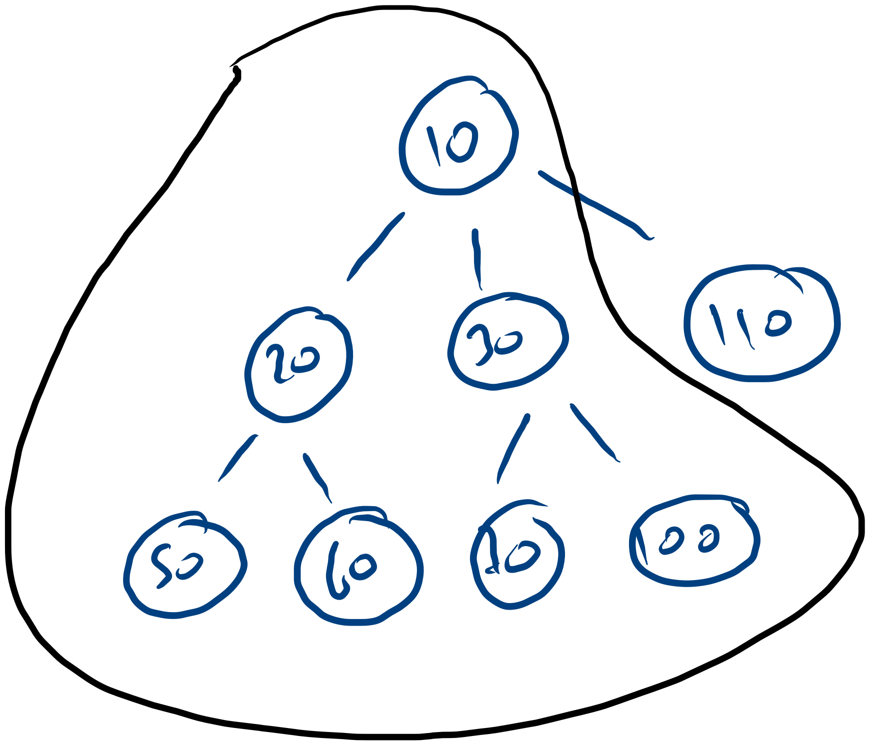
floor → 50

max

min

floor ~~60 50~~ 50 ~~60~~ 0

ceil 8 70



k	larr
✓ 1	→ 110
✓ 2	→ 100
3	→ 70

data = ∞

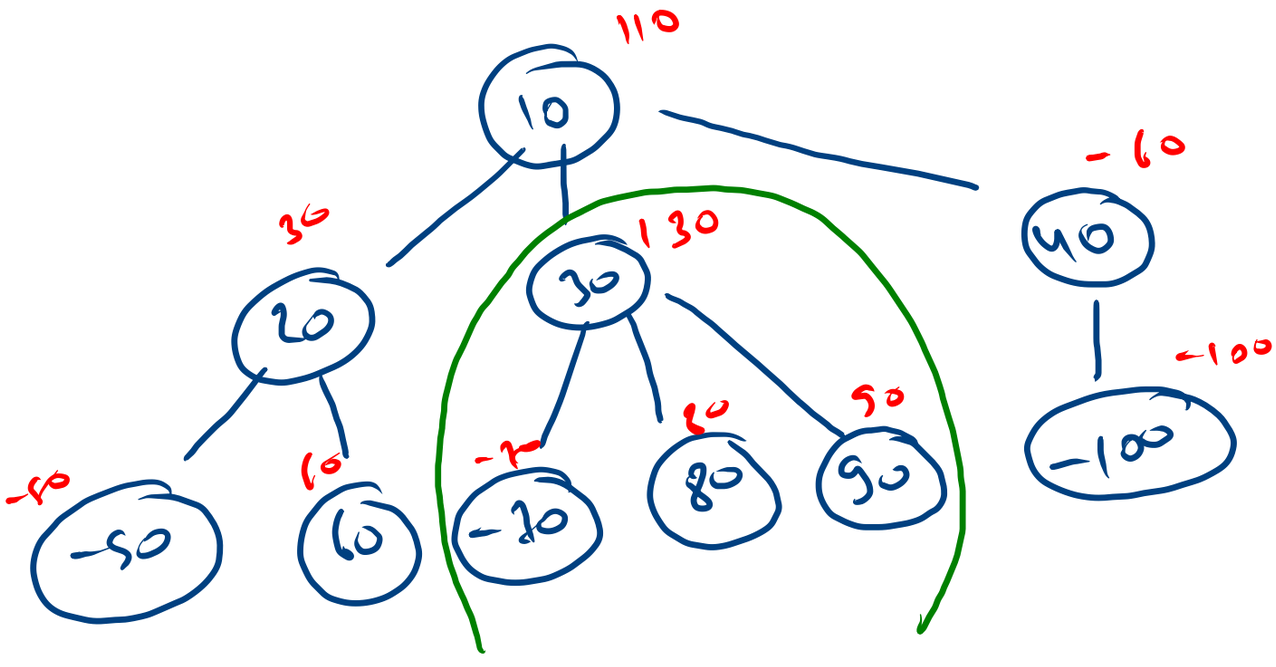
110

ceil

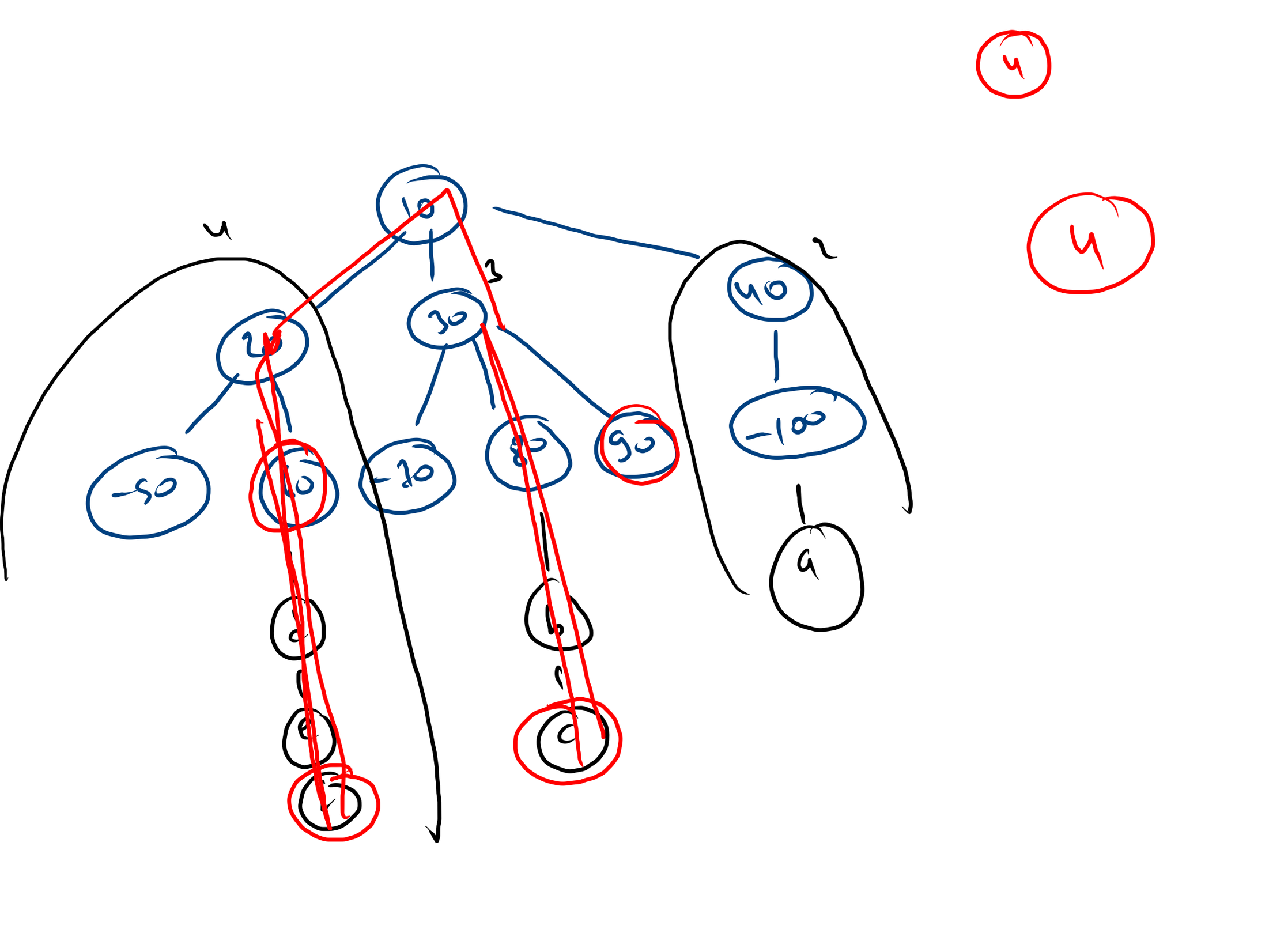
max floor → 110

floor 100

10 20 -50 -1 60
-1 -1 30 -70 -1 80
-1 90 -1 -1 40
-100 -1 -1 -1



30 @ 170

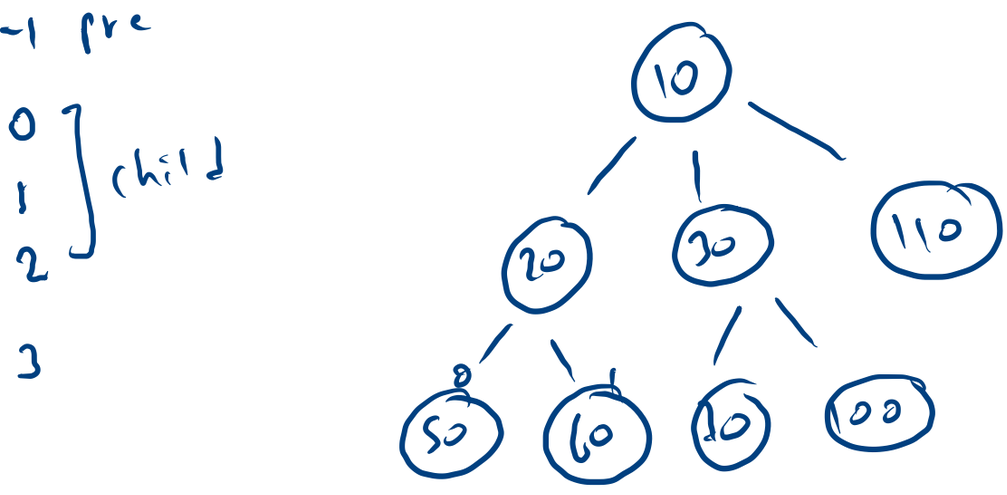

4
4
4
10
4
3
30
2
40
2
-50
20
-20
80
90
-100
9

10 20 50

50

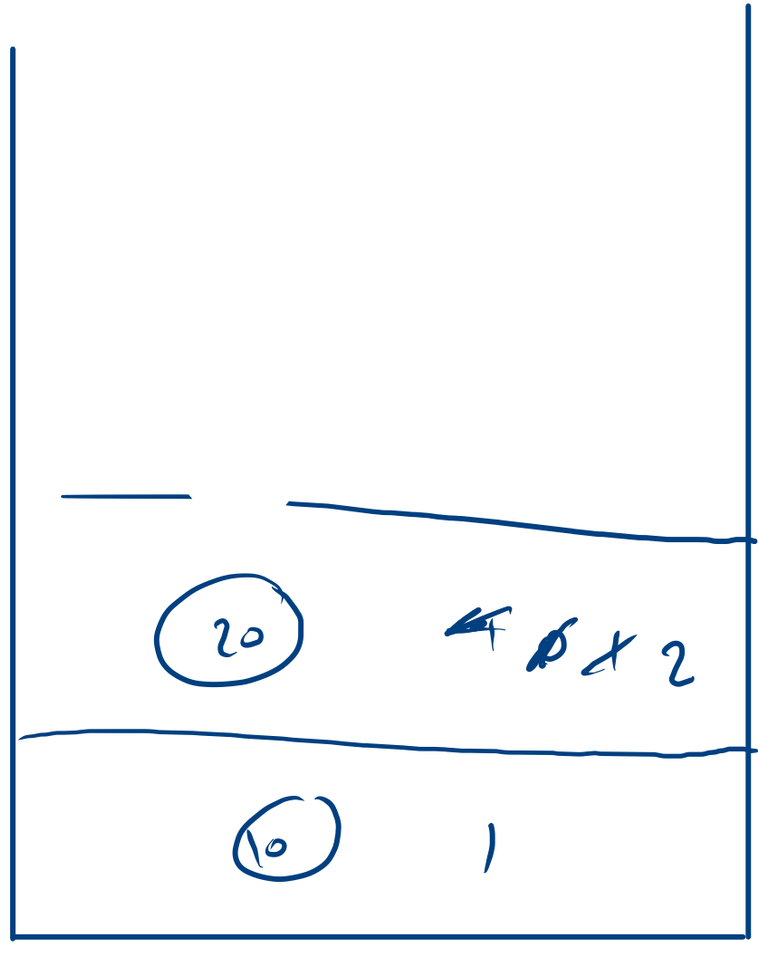
pre 10 20 50 60 30 70 10

post 50 60 20 70 100



pre 10 20 50 60

post 50 60



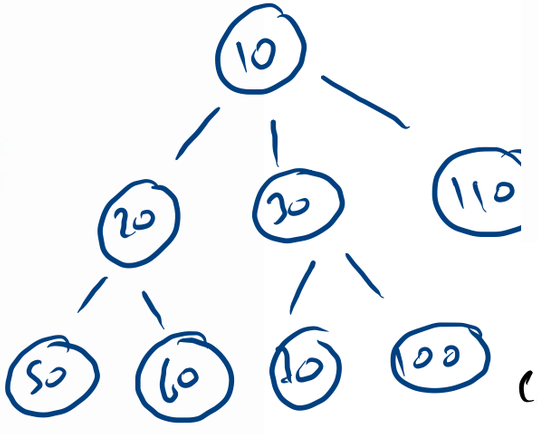
```java
public static void IterativePreandPostOrder(Node node) {

    Stack<Pair> st = new Stack<>();
    st.push(new Pair(node, -1));

    StringBuilder pre = new StringBuilder();
    StringBuilder post = new StringBuilder();

    while(st.size() > 0){
        Pair cp = st.peek();

        if(cp.status == -1){
            pre.append(cp.node.data+" ");
            cp.status++;
        }else if(cp.status==cp.node.children.size()){
            post.append(cp.node.data+" ");
            cp.status++;
            st.pop();
        }else{
            Node child = cp.node.children.get(cp.status);
            st.push(new Pair(child, -1));
            cp.status++;
        }
    }

    System.out.println(pre);
    System.out.println(post);

}
```

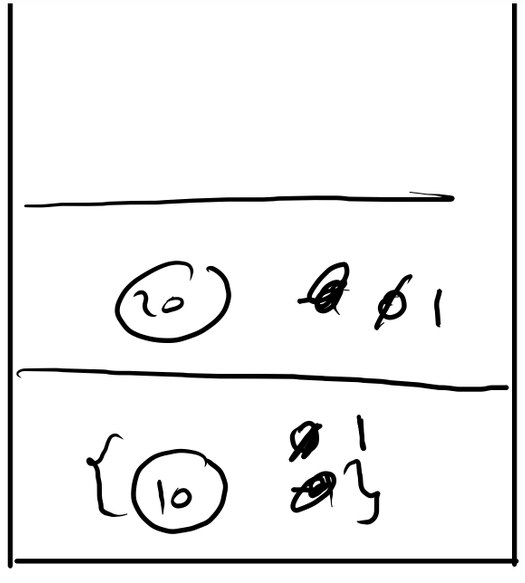
not required

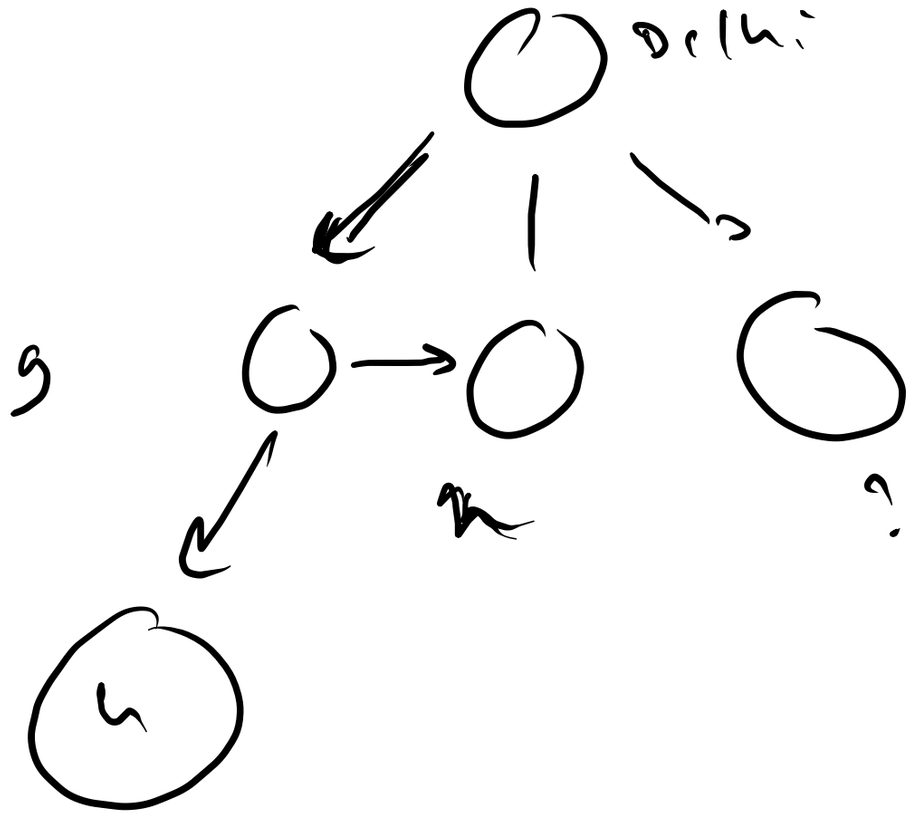
pre 10 20 50

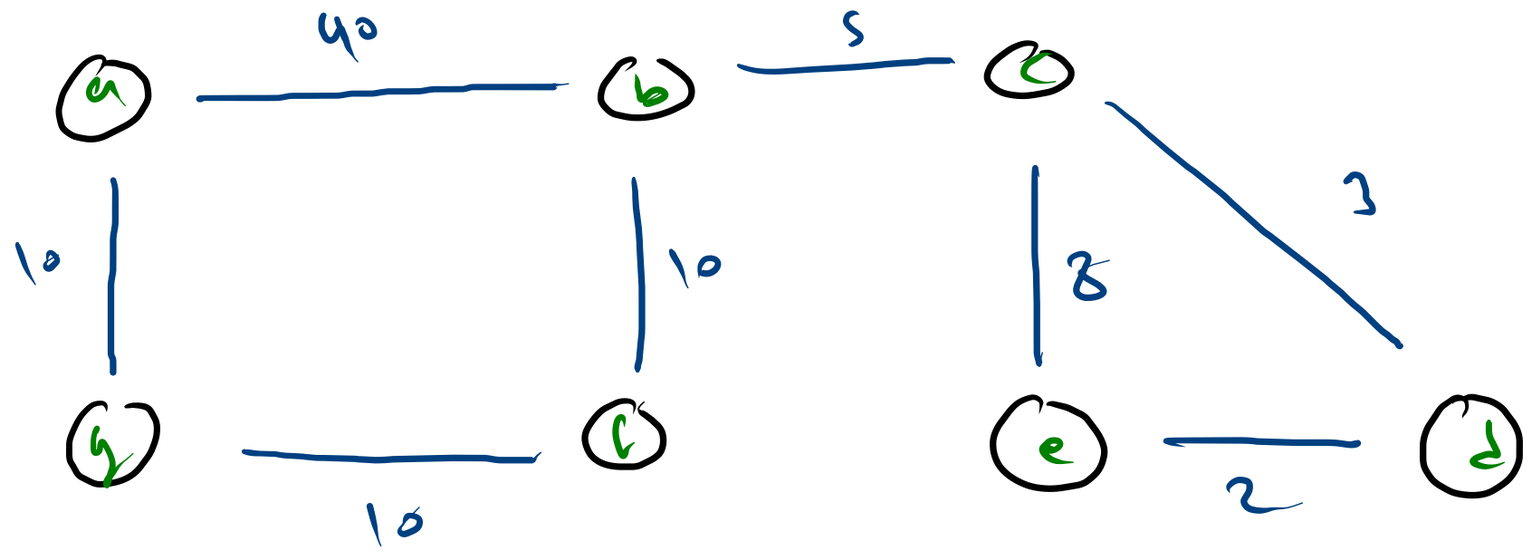
post 50

```java
static class Pair{
    int status;
    Node node;
    Pair(Node n, int s){
        node = n;
        status = s;
    }
}
```






Delhi
9
?



MST

a – d

a b c d → 48 ✓

a b c e d → 53

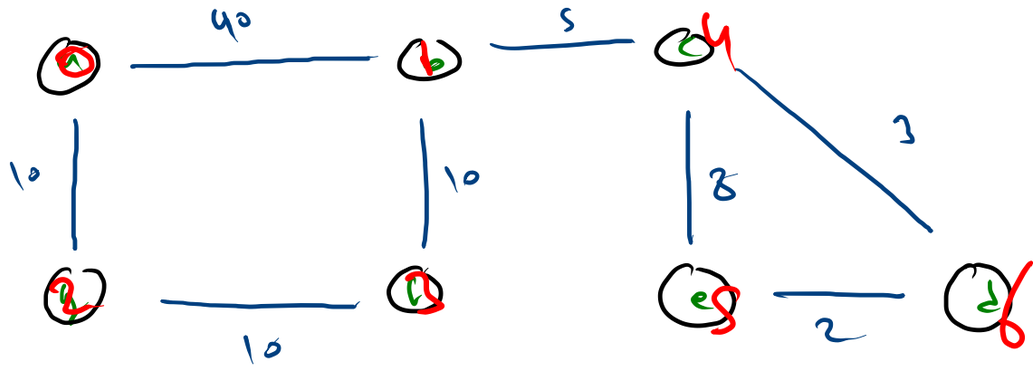
a g f b c d → 38 ✓

a g f b c e d → 45

a → b O(1)

a → . O(V) + [a|b

4



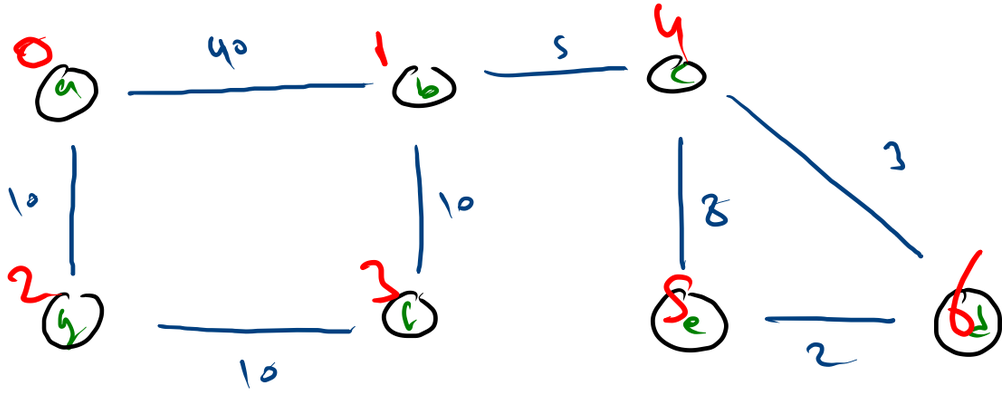
	0	1	2	3	4	5	6	7
0		40	10					
1	40							
2	10			10				
3			10		10			
4								
5								
6								
7								

4 10^3

10^6

a → b O(n)

a → . O(n)



edge

0 → [{0,1,40} , {0,2,10}]

1 → [{1,0,40}]

2 → [{2,0,10}]

3

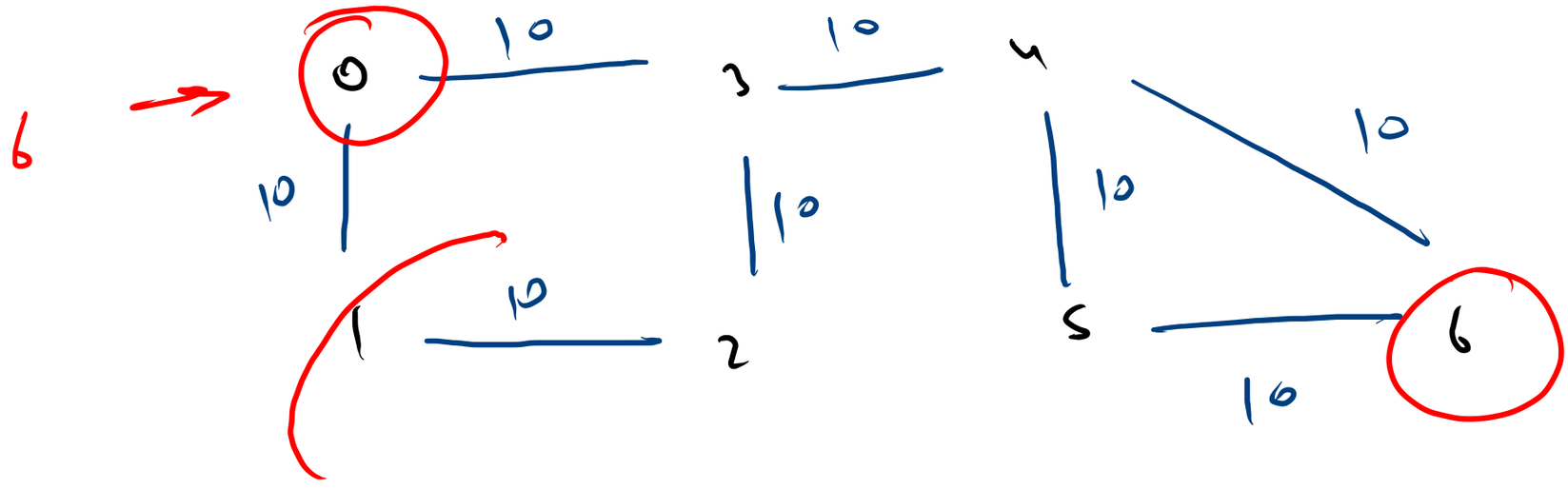
4

5

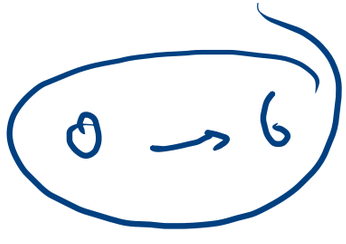
6

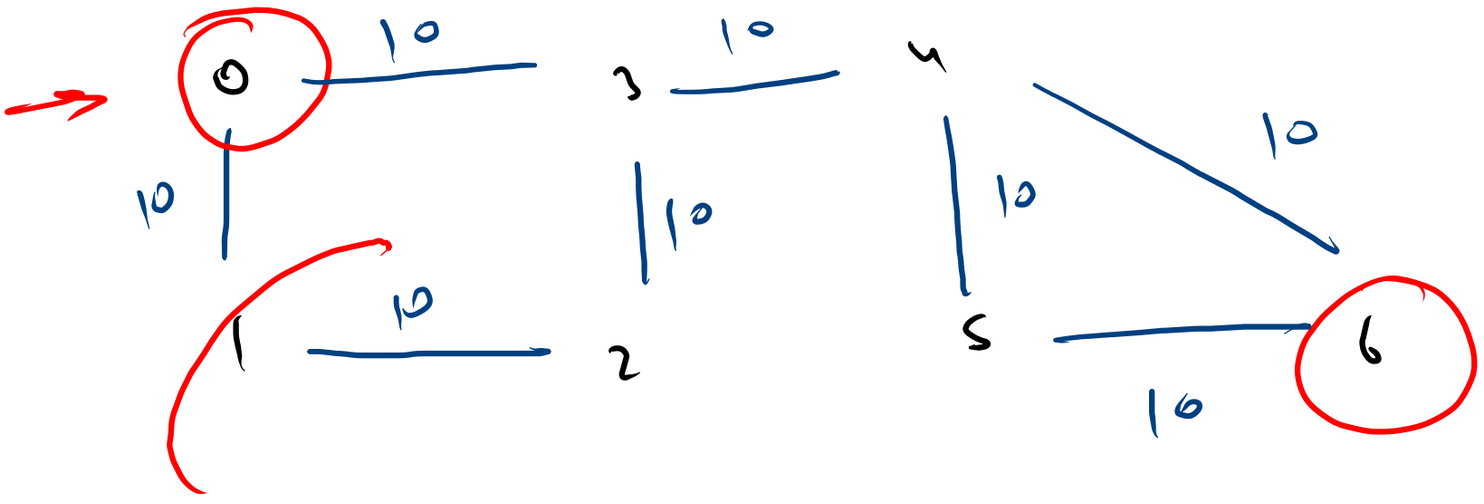
7

AL < Edge > [ ]

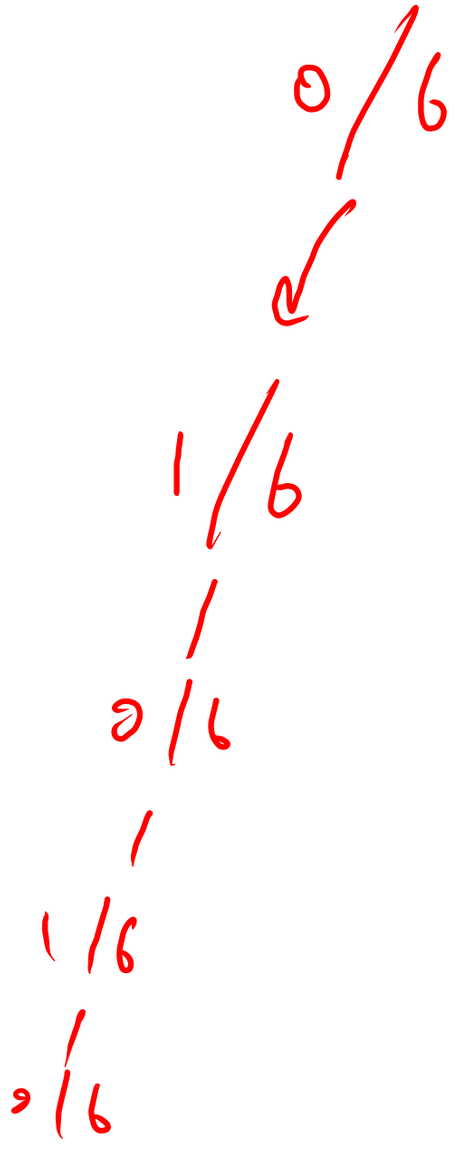


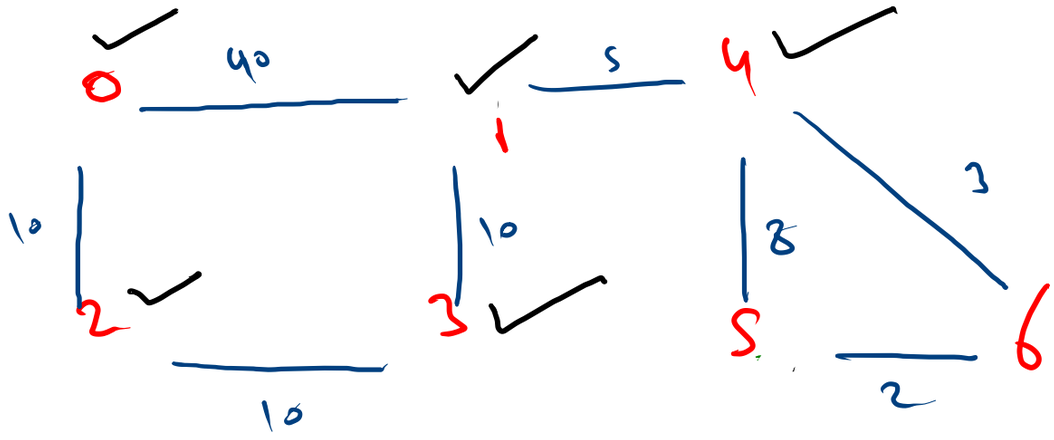
7
8
0 1 10
1 2 10
2 3 10
0 3 10
3 4 10
4 5 10
5 6 10
4 6 10

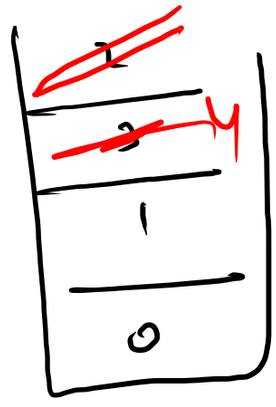


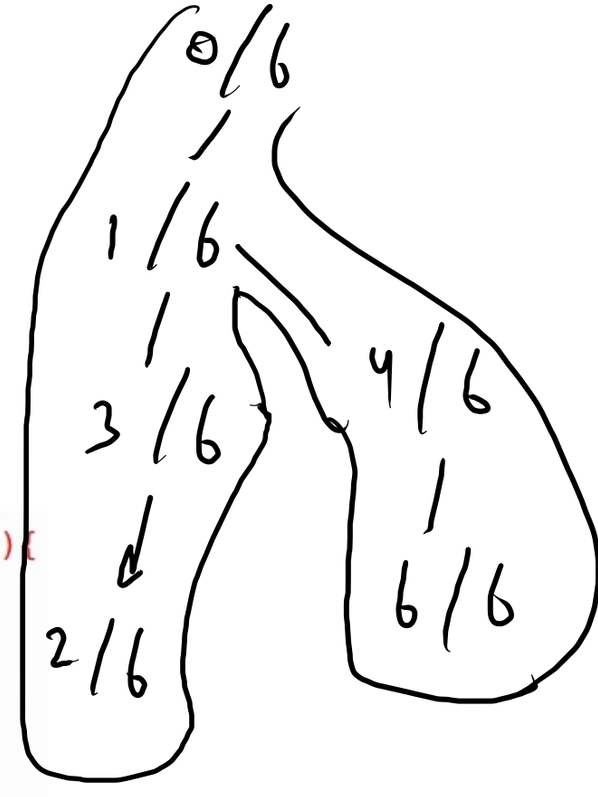



```java
static boolean hasPath(ArrayList<Edge>[]graph, int src, int des){
    if(src == des)return true;

    for(Edge edg: graph[src]){

        if(hasPath(graph, edg.nbr, des)){
            return true;
        }

    }
    return false;
}
```

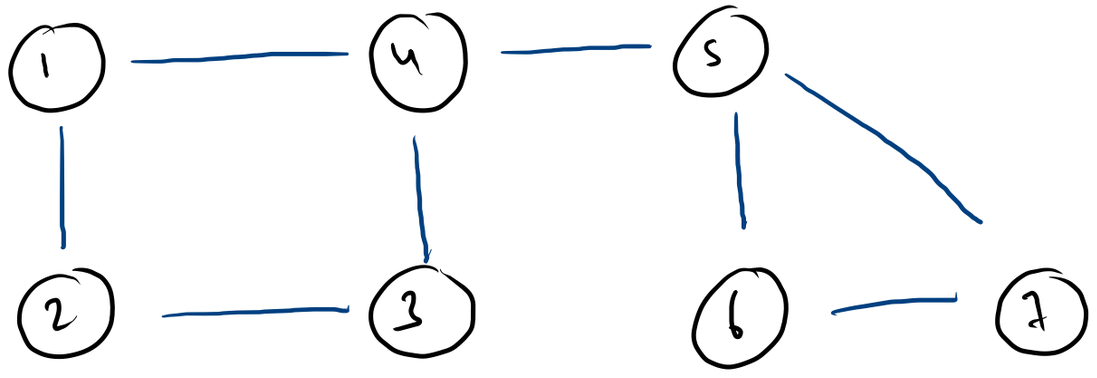









```java
static boolean hasPath(ArrayList<Edge>[]graph, int src, int des, boolean[]visited){
    if(src == des)return true;

    visited[src] = true;

    for(Edge edg: graph[src]){

        if(visited[edg.nbr] == false && hasPath(graph, edg.nbr, des, visited)){
            return true;
        }

    }
    return false;
}
```



1 → 7

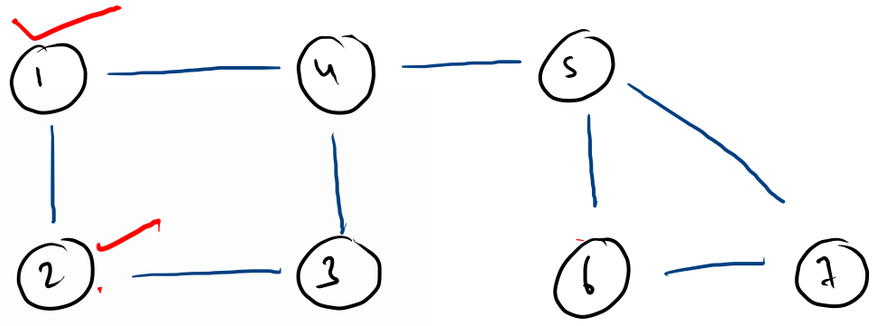
1 4 5 7

1 4 5 6 7

1 2 3 4 5 7

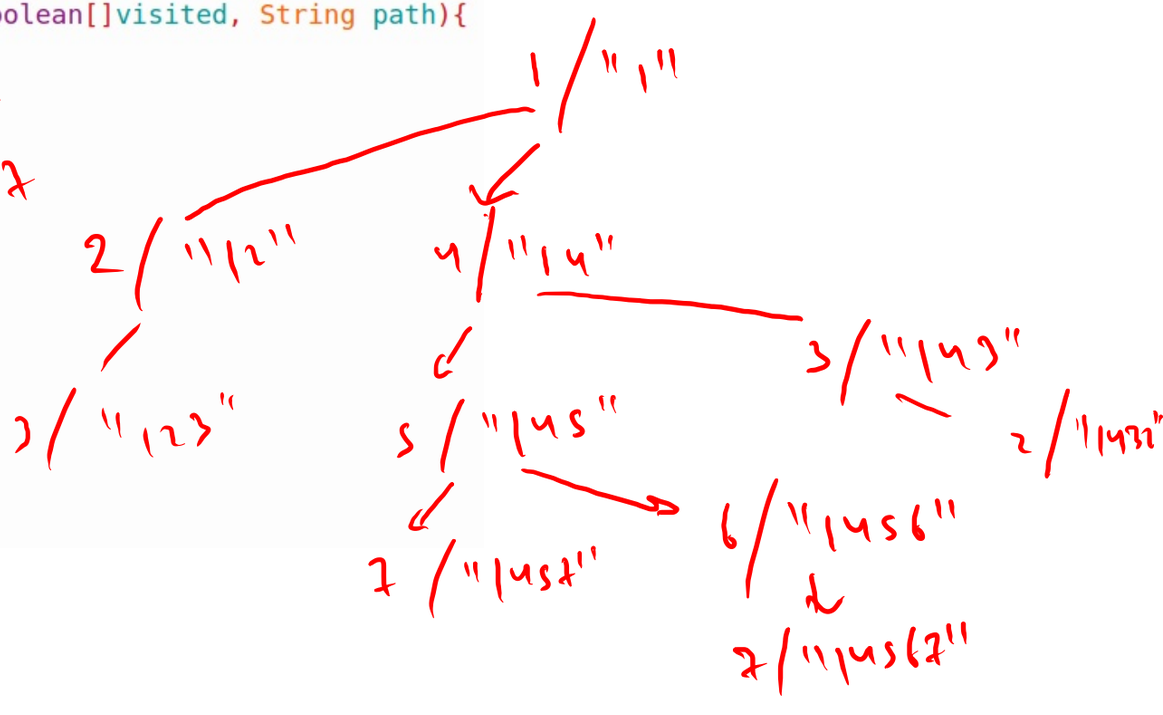
1 2 3 4 6 7

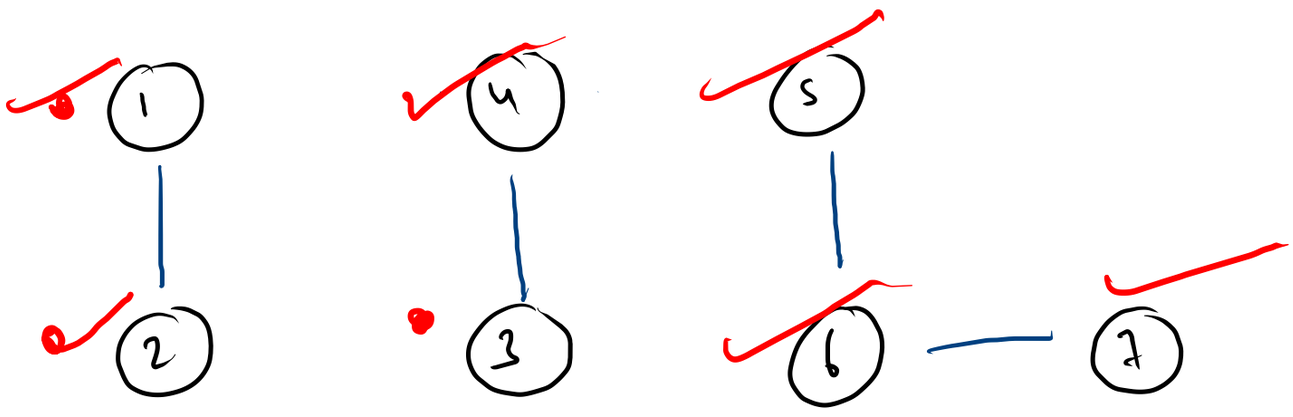
```
    int src = scn.nextInt();
    int des = scn.nextInt();

    hasPath(graph, src, des, new boolean[n], src+"");

    // System.out.println(hasPath);
}

static void hasPath(ArrayList<Edge>[]graph, int src, int des, boolean[]visited, String path){
    if(src == des){
        System.out.println(path);
        return;
    }

    visited[src] = true;

    for(Edge edg: graph[src]){

        if(visited[edg.nbr]){
            continue;
        }
        hasPath(graph, edg.nbr, des, visited, path+edg.nbr);

    }

    visited[src] = false;
```



1 4 5 7

1 4 5 6 7





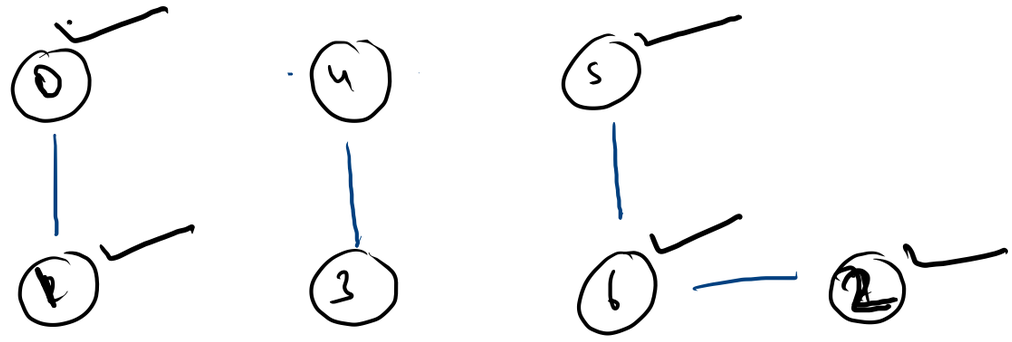
[3, 4]

[5 6 7]

[1, 2]

[[1, 2], [3, 4], [5, 6, 7]]

```
    ArrayList<ArrayList<Integer>> comps = new ArrayList<>();

    boolean visited[] = new boolean[vtces];
    for(int i=0;i<vtces;i++){
        if(visited[i] == false){
            ArrayList<Integer> comp = new ArrayList<>();
            dfs(graph, i, visited, comp);
            comps.add(comp);
        }
    }

    System.out.println(comps);
}

static void dfs(ArrayList<Edge>[]graph, int src, boolean[]vi

    visited[src] = true;

    comp.add(src);

    for(Edge ed: graph[src]){
        if(visited[ed.nbr] == true)continue;
        dfs(graph, ed.nbr, visited, comp);
    }
}
```



comps [

comp →

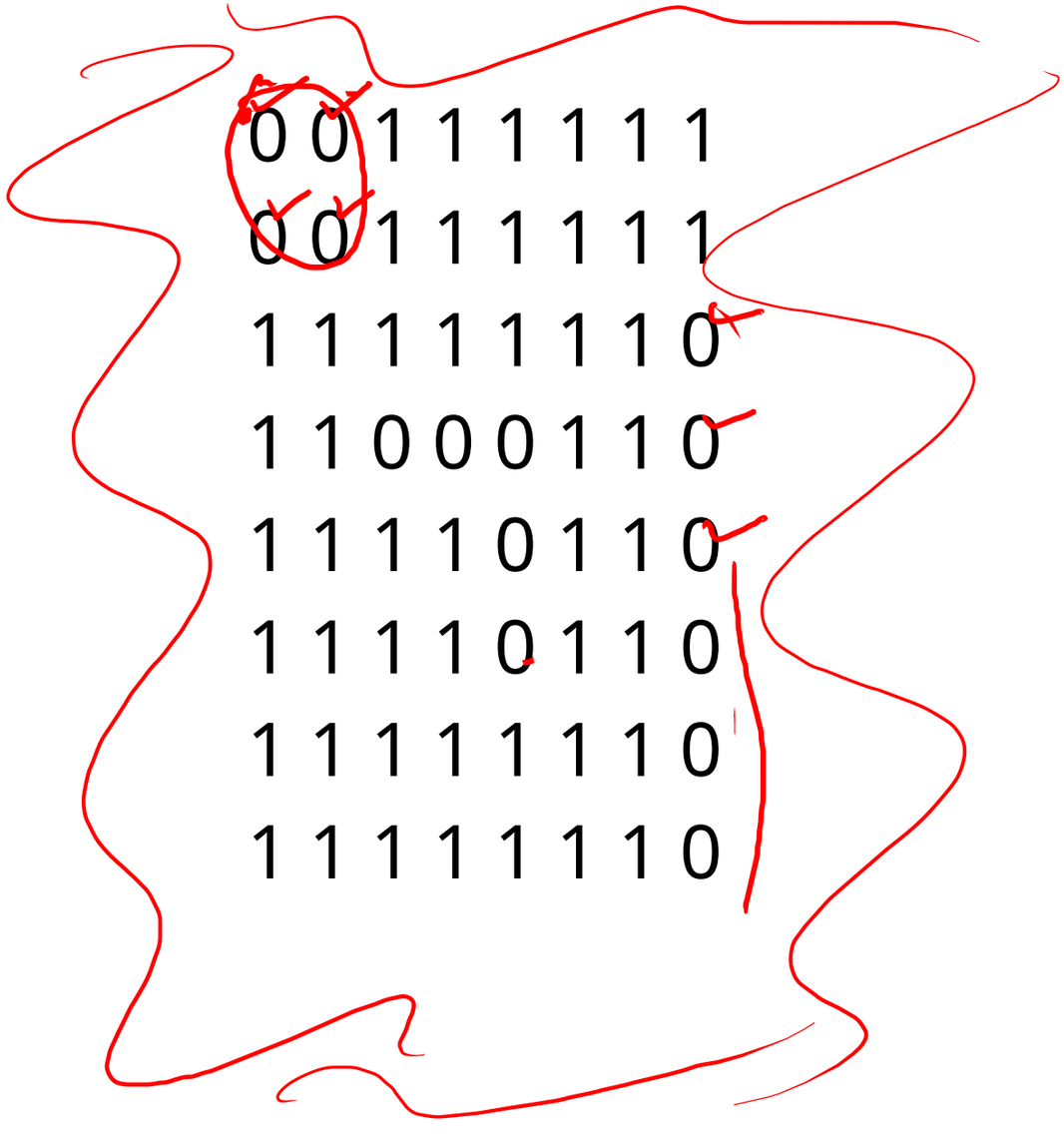
[0 , 1]

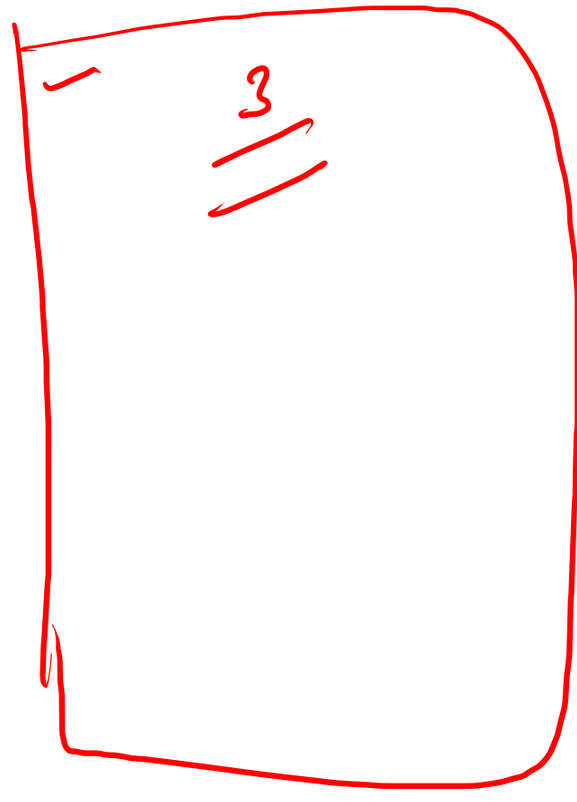
[2 , 6 , 5]

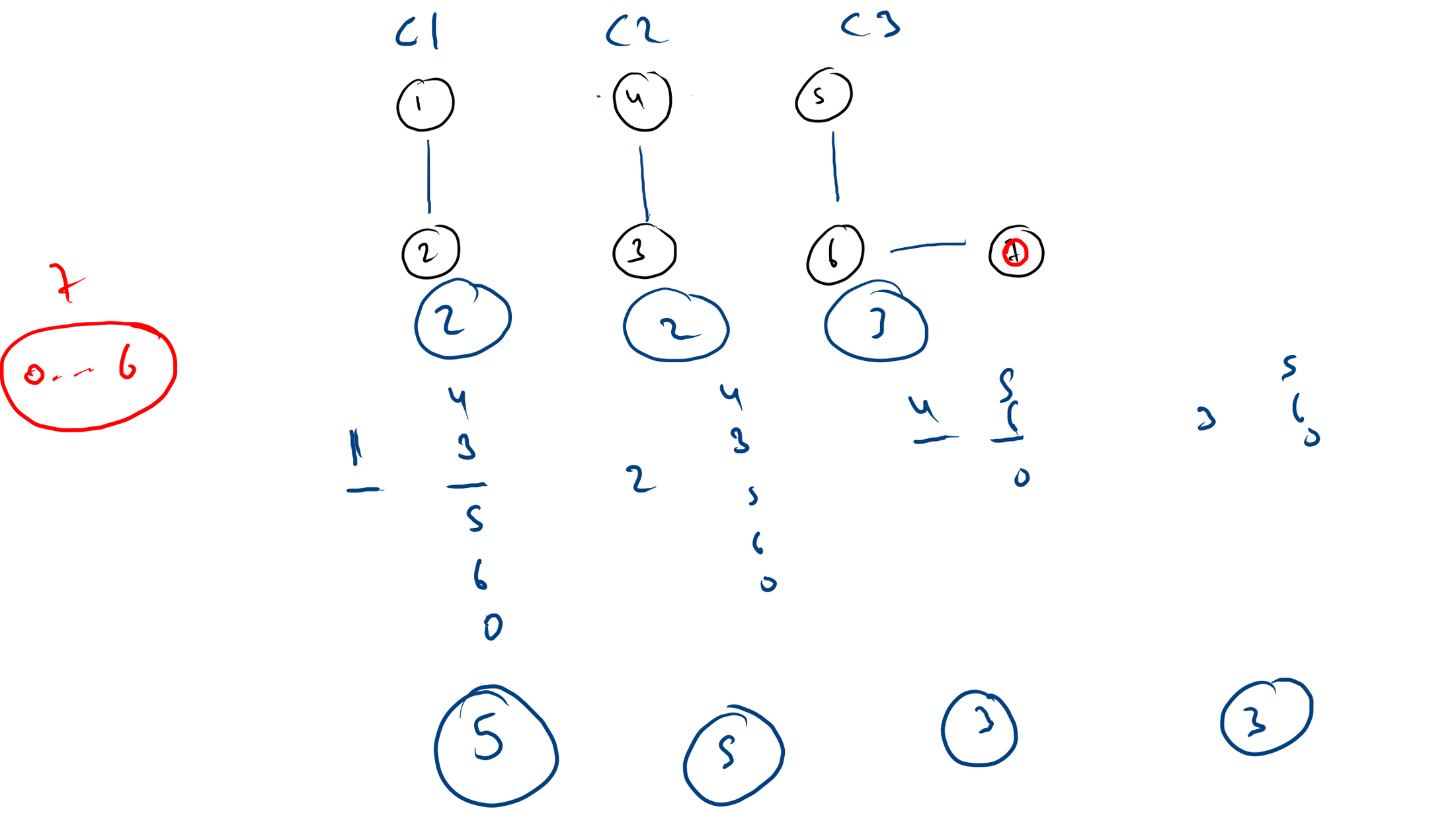
]

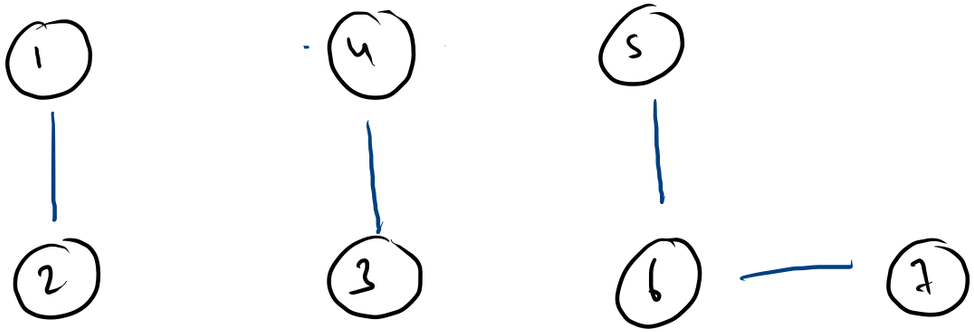
0 → land

1 → sea


0 0 1 1 1 1 1 1
0 0 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 0 0 0 1 1 0
1 1 1 1 0 1 1 0
1 1 1 1 0 1 1 0
1 1 1 1 1 1 1 0
1 1 1 1 1 1 1 0


3


C1
C2
C3
7
0 -- 6
1
4
5
2
3
6
0
5
5
3
3



2 2 3

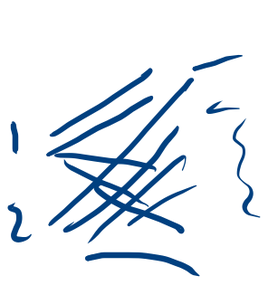
[2, 2, 3]

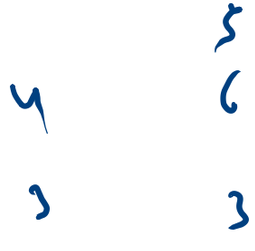
7

2x(7-1)

2x5

2x5





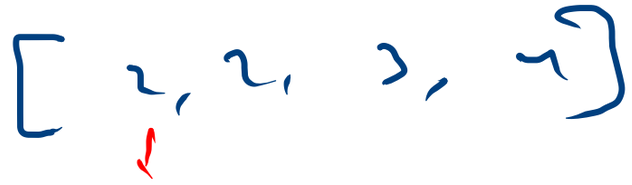
2x5

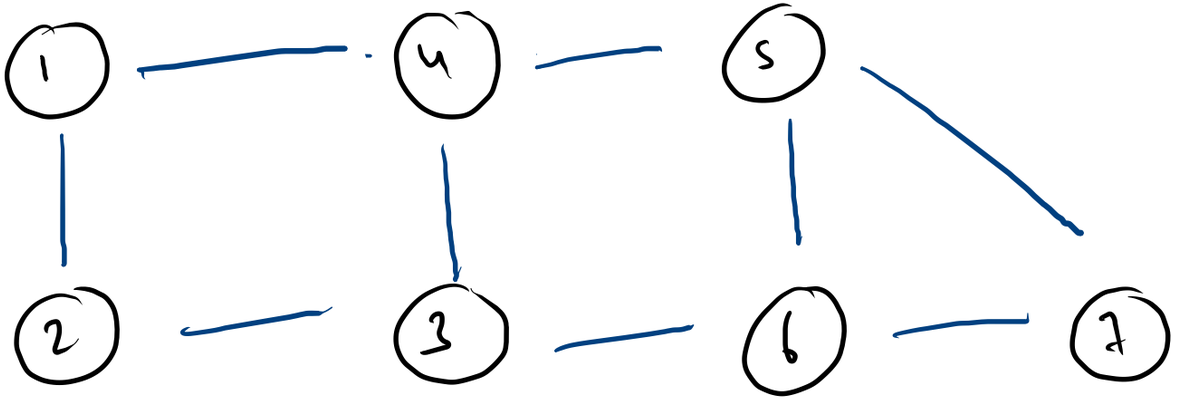
2x3

2x(7-4)

2x 3

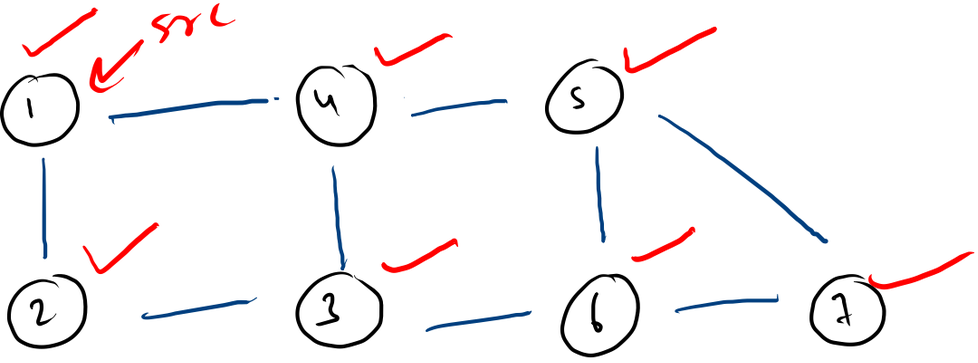
3x(7-4-3





HP path $\begin{cases} 1\,2\,3\,4\,5\,6\,7. \\ 1\,2\,3\,4\,5\,7\,6. \end{cases}$

cycle $\begin{cases} 1\,4\,5\,7\,6\,3\,2\,1 \\ 1\,2\,3\,6\,7\,5\,4\,1 \end{cases}$



1/"1"/0

2/"12"/1

3/"123"/2

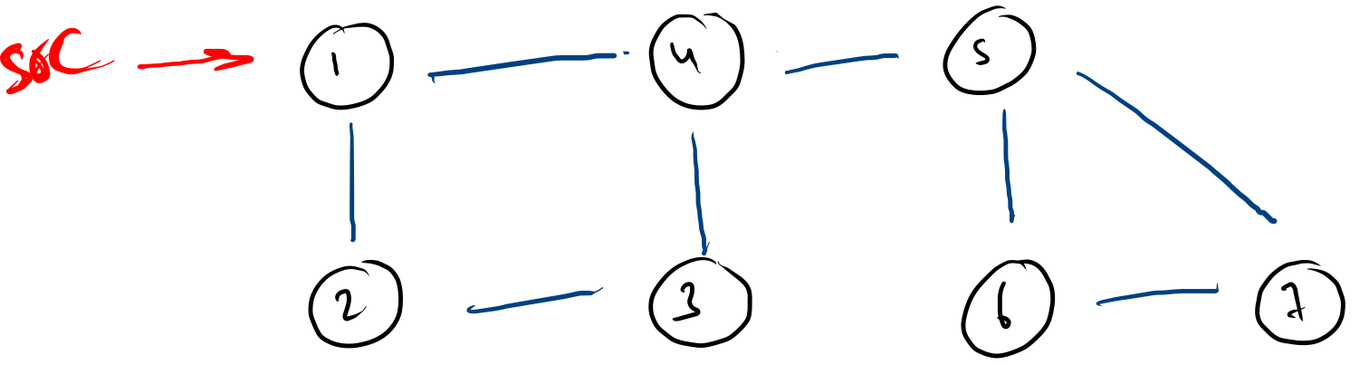
4/"1234"/3

5/"12345"/4

6/"123456"/5

7/"1234567"/6

src



1 @ 1

4 @ 14

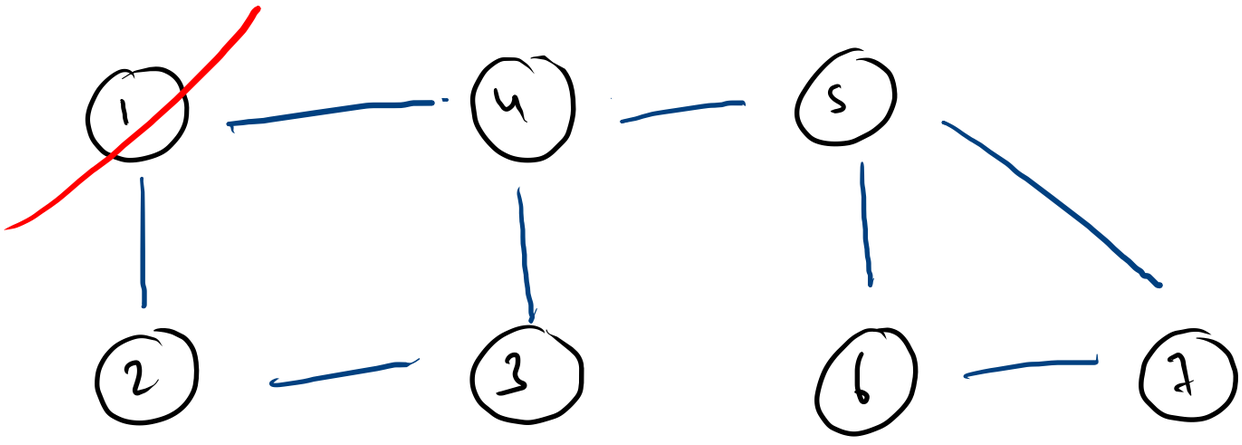
2 @ 12

3 @ 123

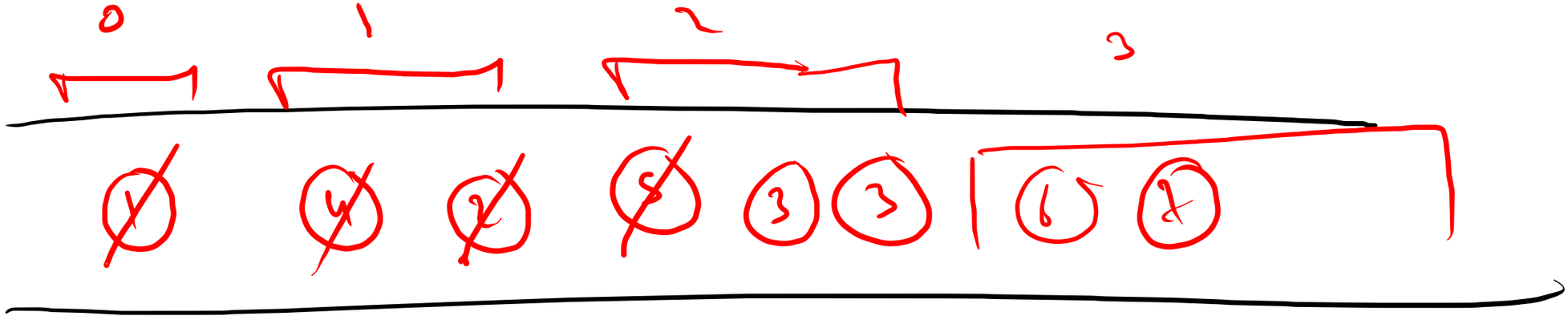
5 @ 145

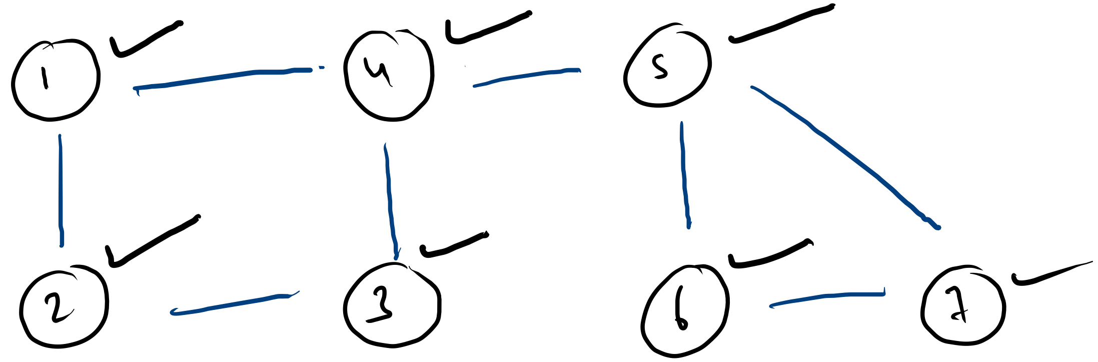
6 @ 1456

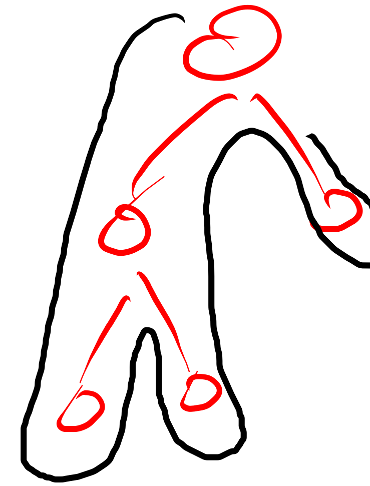
7 @ 1457



BFS



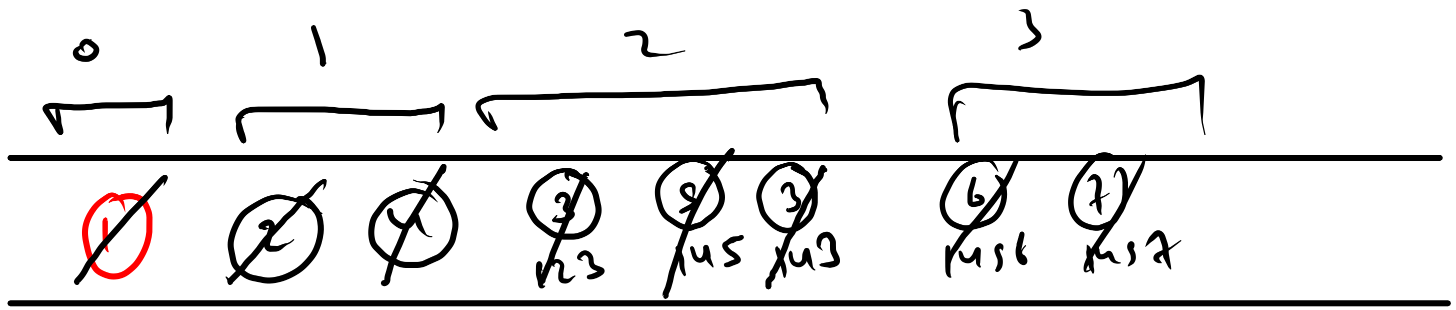




γ marked done nbr

↓ ↓

push



1 @ 1

2 @ 12

4 @ 14

3 @ 123

5 @ 145