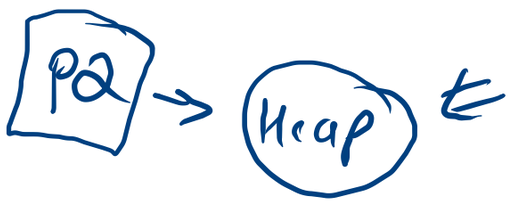
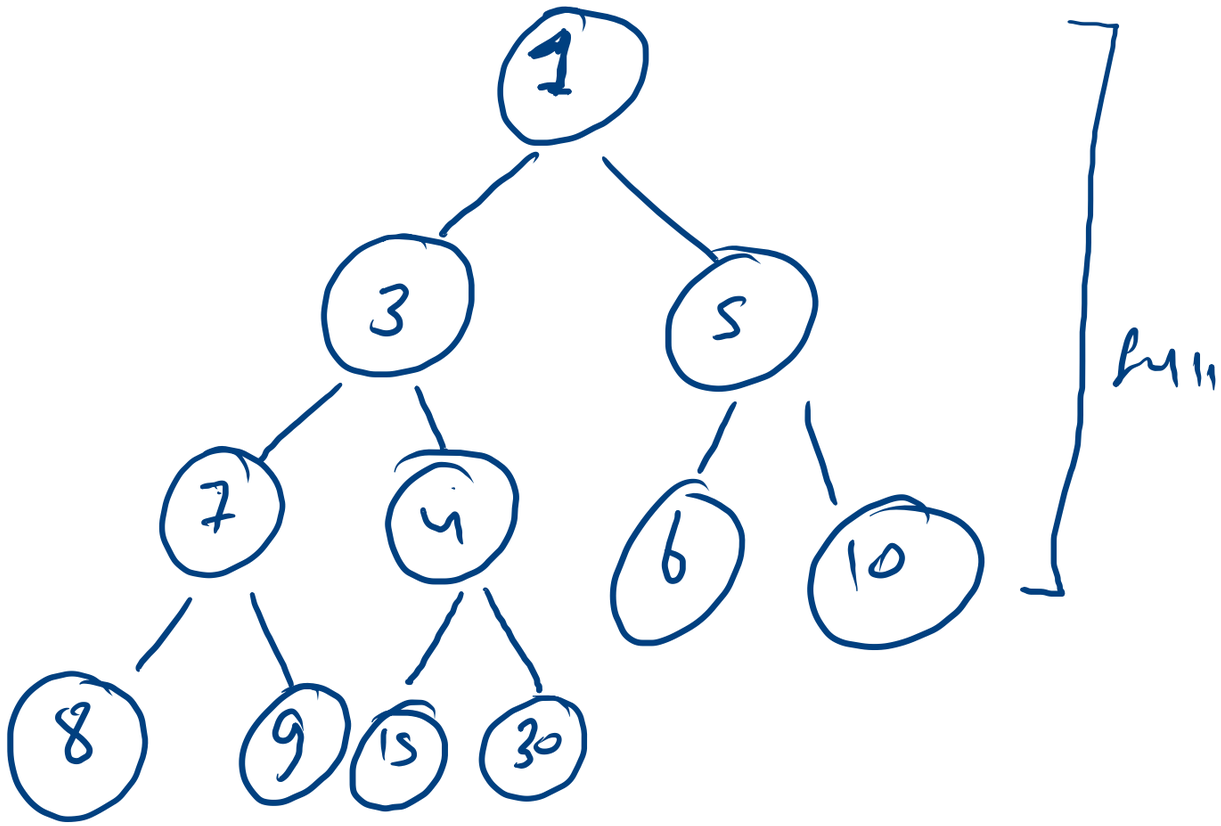


add O(log n)

remove O(log n)

peek O(1)



Heap order

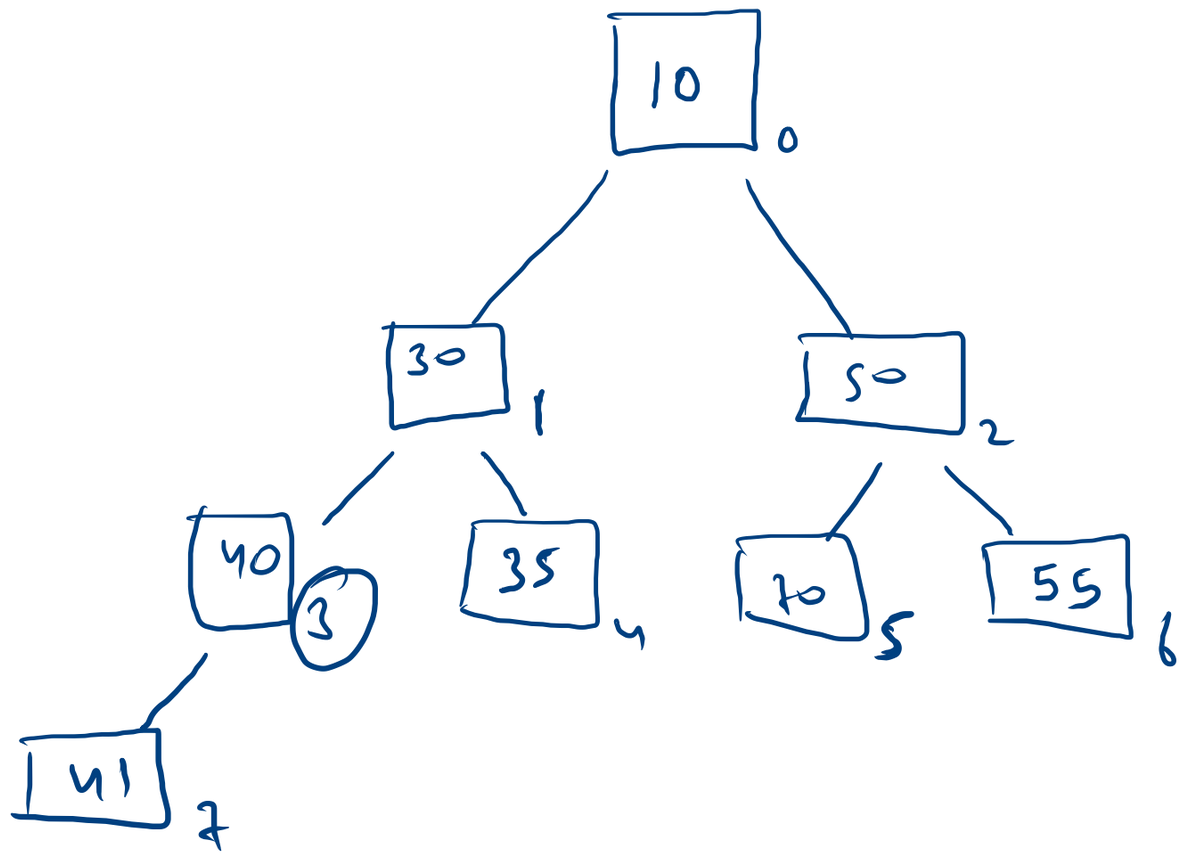
CBT

min heap

arr → [10 | 30 | 50 | 40 | 35 | 70 | 55 | 41]
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

add O(log n)
remove O(log n)
peek O(1)

Min order



[L.C. → 2 x i + 1
R.C → 2 x i + 2

[P.i → (i-1)/2

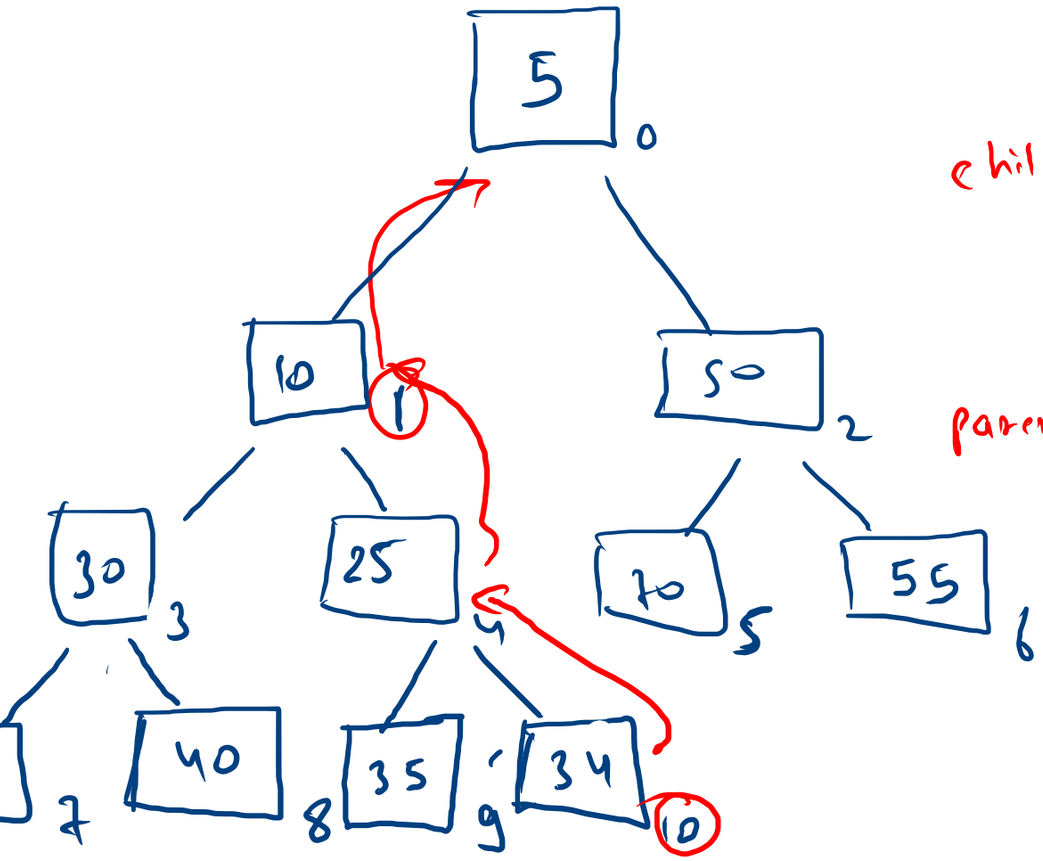
arr → [10 (0) | 25 50 (1 2) | 30 34 70 55 (3 4 5 6) | 41 (7) | 40 (8) | 35 (9) | 5 (10)]

add $O(\log_2 n)$

remove $O(\log n)$

peek $O(1)$

upheapify



child
- L.C → 2*i + 1
- R.C → 2*i + 2

parent
- P.i → (i-1)/2

25 →

comp

arr → [35 | 25 | 50 | 30 | 34 | 70 | 55 | 41 | 40 | ~~0~~]

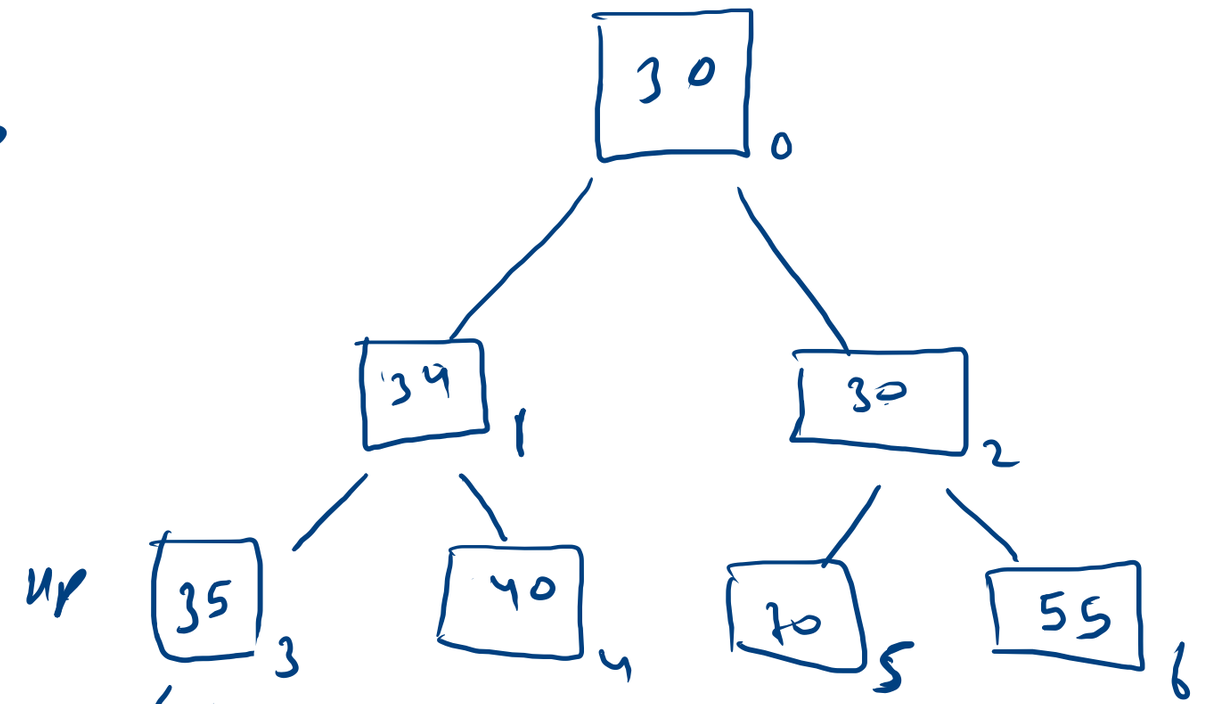
0 1 2 3 4 5 6 7 8 9

add $O(\log_2 n)$

remove $O(\log_2 n)$ ←

peek $O(1)$

downheapify •



5, 10, 25 →

L.C. → $2 \times i + 1$

R.C → $2 \times i + 2$

P.i → $(i-1)/2$

comp

```
data.get(li).compareTo(data.get(min)) < 0)
data.get(ri).compareTo(data.get(min)) < 0)
```

a ← b

li = 0
min = 1

a.compareTo(b)

this

```
public int compareTo(Student b){
    return score - b.score;
}
```

this: 78 (90) 10

b: 87 → -V
80 → +V
10 → 0

yk sk

a < b = -V

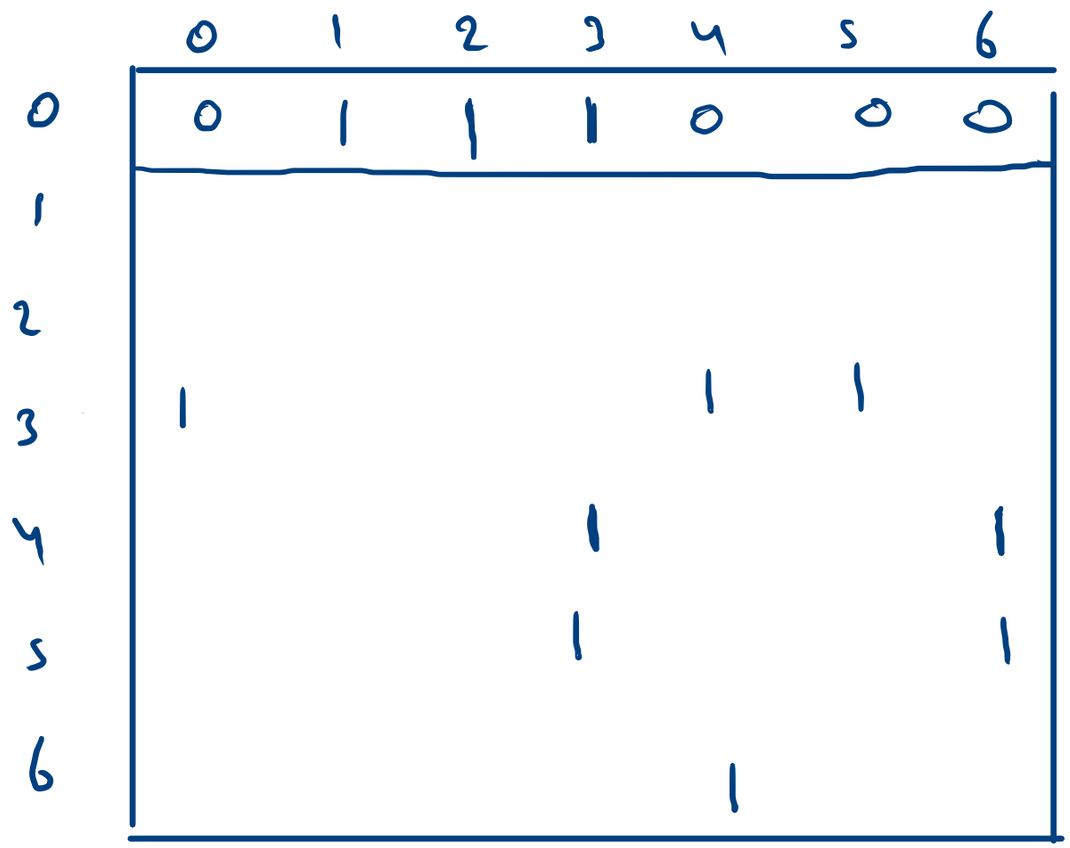
a > b = +V

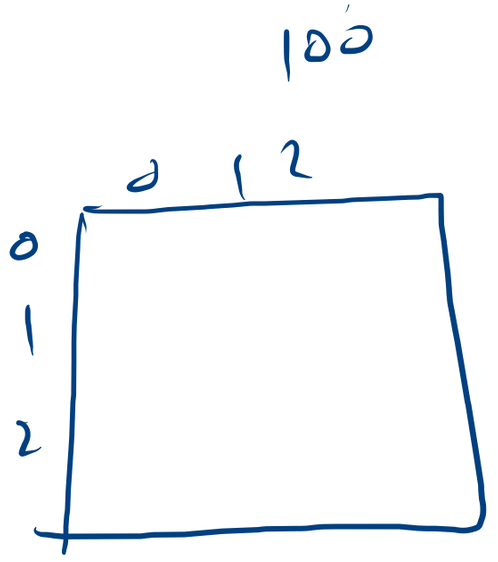
a = b = 0

0 1

[{ roll 3, nam 'A', score 78 } , { roll 7, nam 'B', score 87 }]

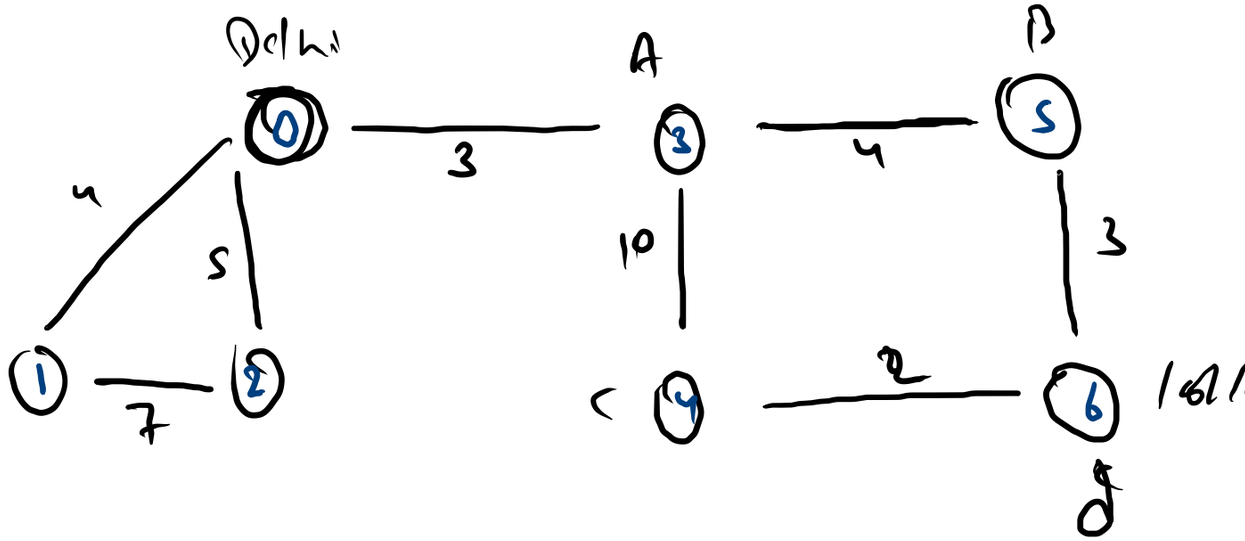
yk 5k

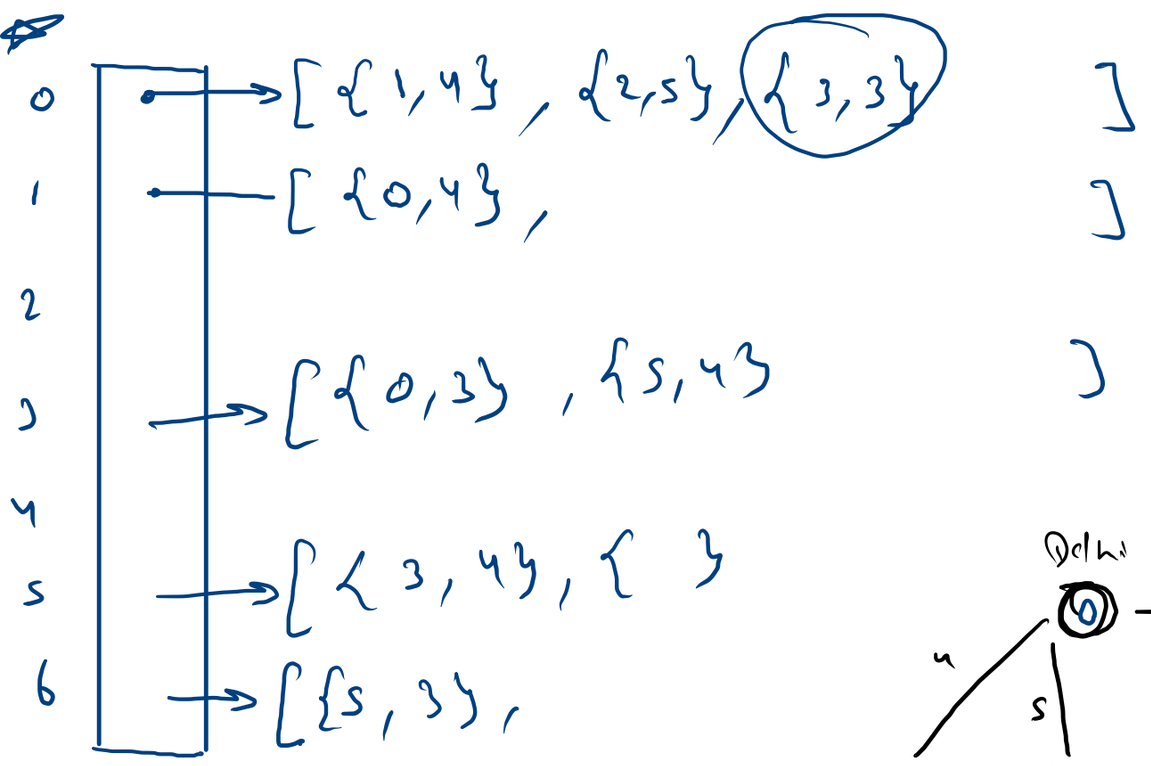

0 1 2 3 4 5 6
0 0 1 1 1 0 0 0
1
2
3 1 1 1
4 1 1
5 1 1
6 1

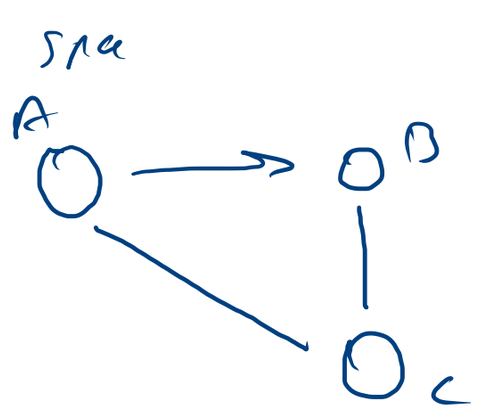

100
0 1 2
0
1
2

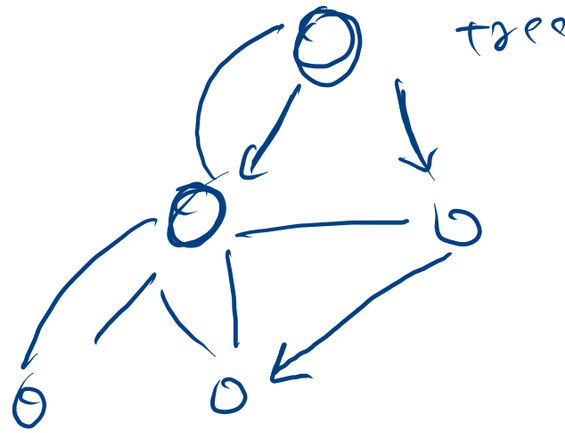

edges
0 1 2
0 1 2
0 →
1 →
2 →

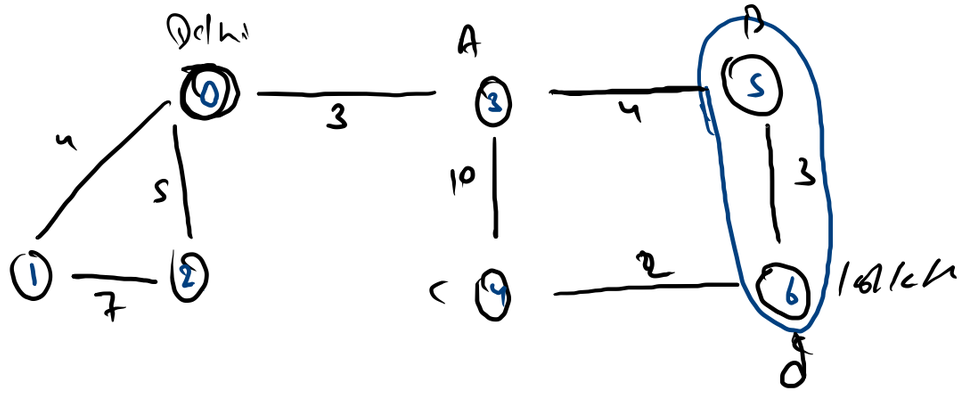

Searching
Time
1 0


Delhi
A
B
0
3
5
3
4
5
10
3
1
7
2
4
2
6
Kolkata


0
1
2
3
4
5
6
[{1,4}, {2,5}, {3,3}]
[{0,4},]
[{0,3}, {5,4}]
[{3,4}, {}
[{5,3},


A
B
C




3
4
5
10
2
7
3

```java
static class Edge {
    int src;
    int nbr;
    int wt;

    Edge(int src, int nbr, int wt){
        this.src = src;
        this.nbr = nbr;
        this.wt = wt;
    }
}
```

0 1 10
src nbr

.src
.nbr

```java
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int vtces = Integer.parseInt(br.readLine());
ArrayList<Edge>[] graph = new ArrayList[vtces];
for(int i = 0; i < vtces; i++){
    graph[i] = new ArrayList<>();
}

int edges = Integer.parseInt(br.readLine());
for(int i = 0; i < edges; i++){
    String[] parts = br.readLine().split(" ");
    int v1 = Integer.parseInt(parts[0]);
    int v2 = Integer.parseInt(parts[1]);
    int wt = Integer.parseInt(parts[2]);
    graph[v1].add(new Edge(v1, v2, wt));
    graph[v2].add(new Edge(v2, v1, wt));
}
```

nodes

7 8

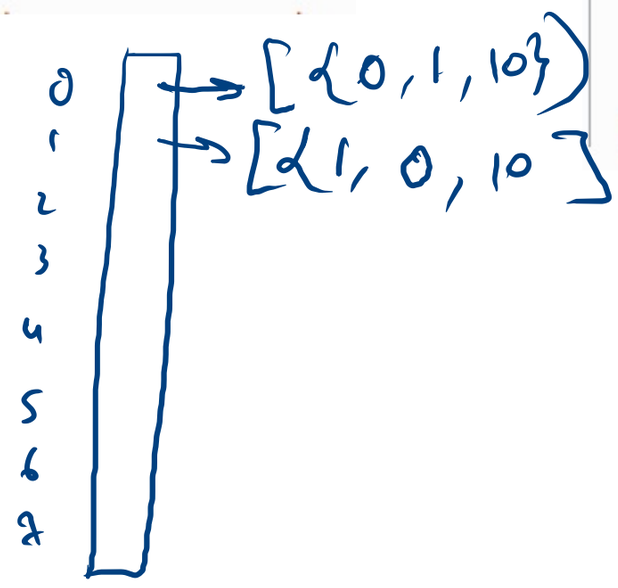
0 1 10

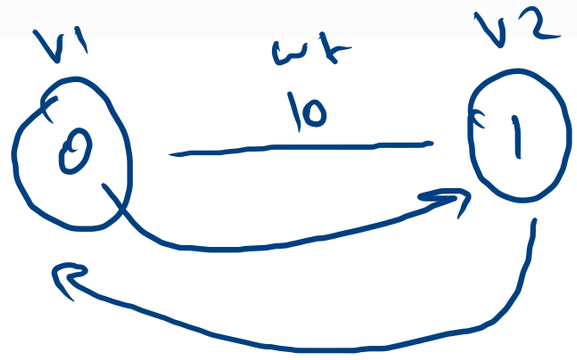
1 2 10, 2 3 10,

0 3 10, 3 4 10,

4 5 10 ,5 6 10,

4 6 10





src

0 → [{0,1,10}, {0,3,10}]

1 → [{1,0,10}, {1,2,10}

2 → [{2,1,10}, {2,3,10}

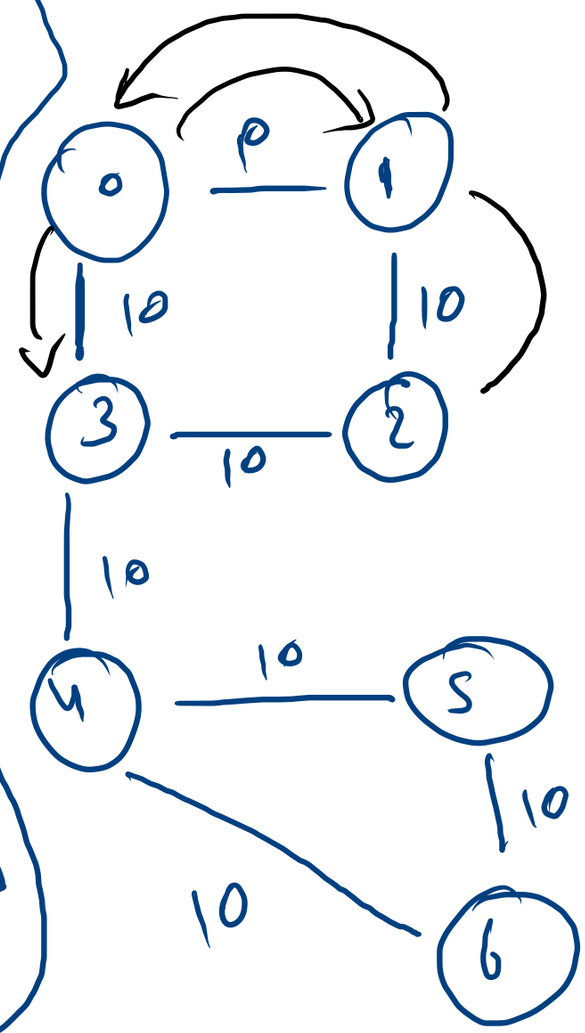
3 → [{3,2,10}, {3,0,10}
{3,4,10}

4 → [{4,3,10}, {4,5,10}
{4,6,10}

5 → [{5,4,10}, {5,6,10}

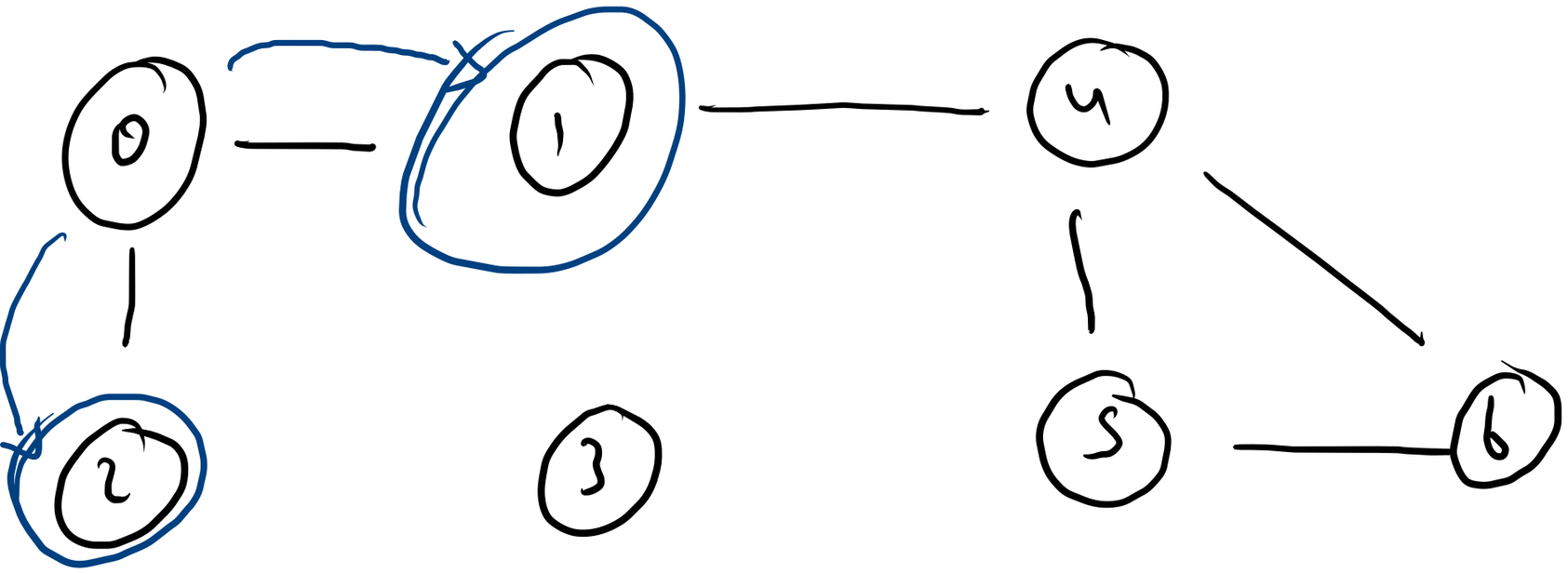
6 → [{6,5,10}, {6,4,10}]

graph[src]



0 ... 6 (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

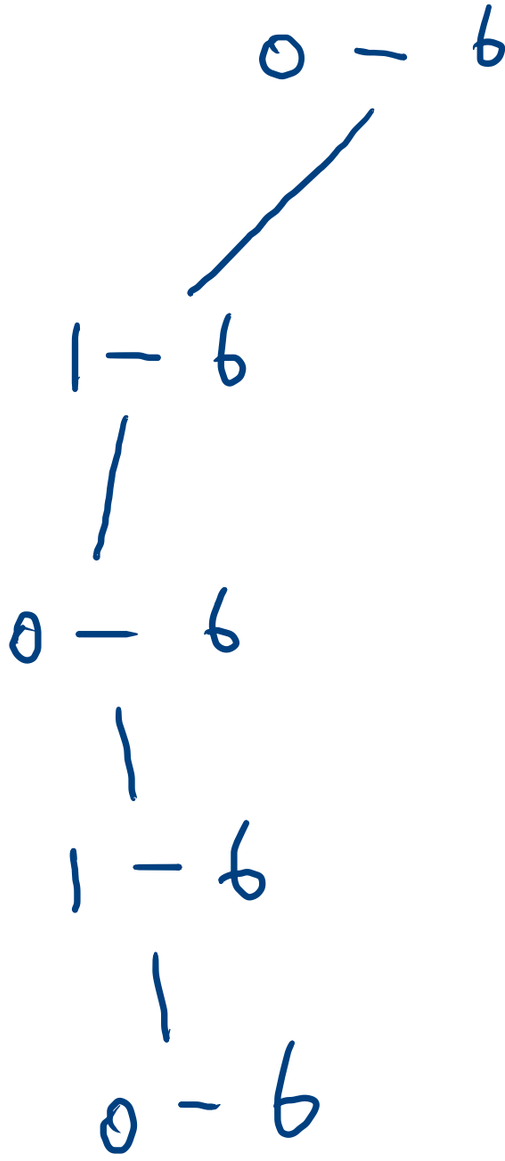


high Level

1 level

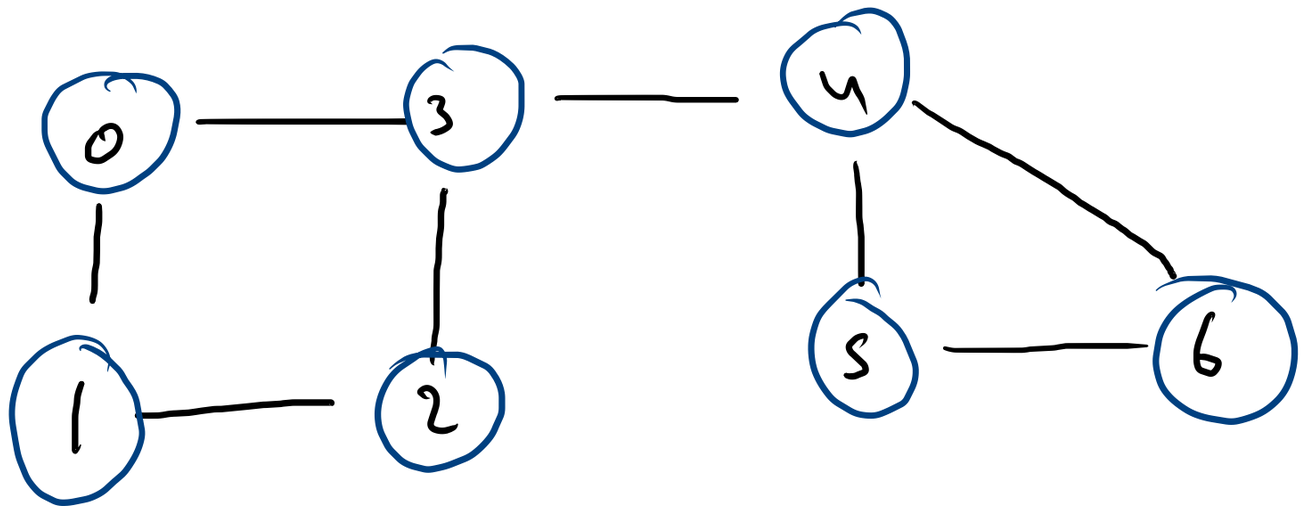
src	dst	
0	6	true
0	3	False

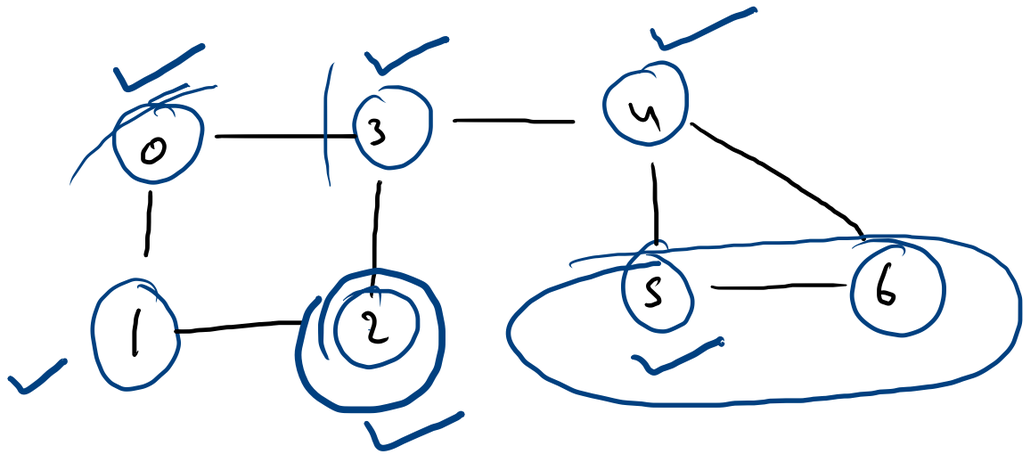
```java
public static boolean haspath(ArrayList<Edge>[] graph, int src, int dest){
    if(src == dest)return true;

    for(Edge edge: graph[src]){// src, nbr, wt
        if(haspath(graph, edge.nbr, dest)){
            return true;
        }
    }

    return false;
}
```



Src -> 0

dest -> 6

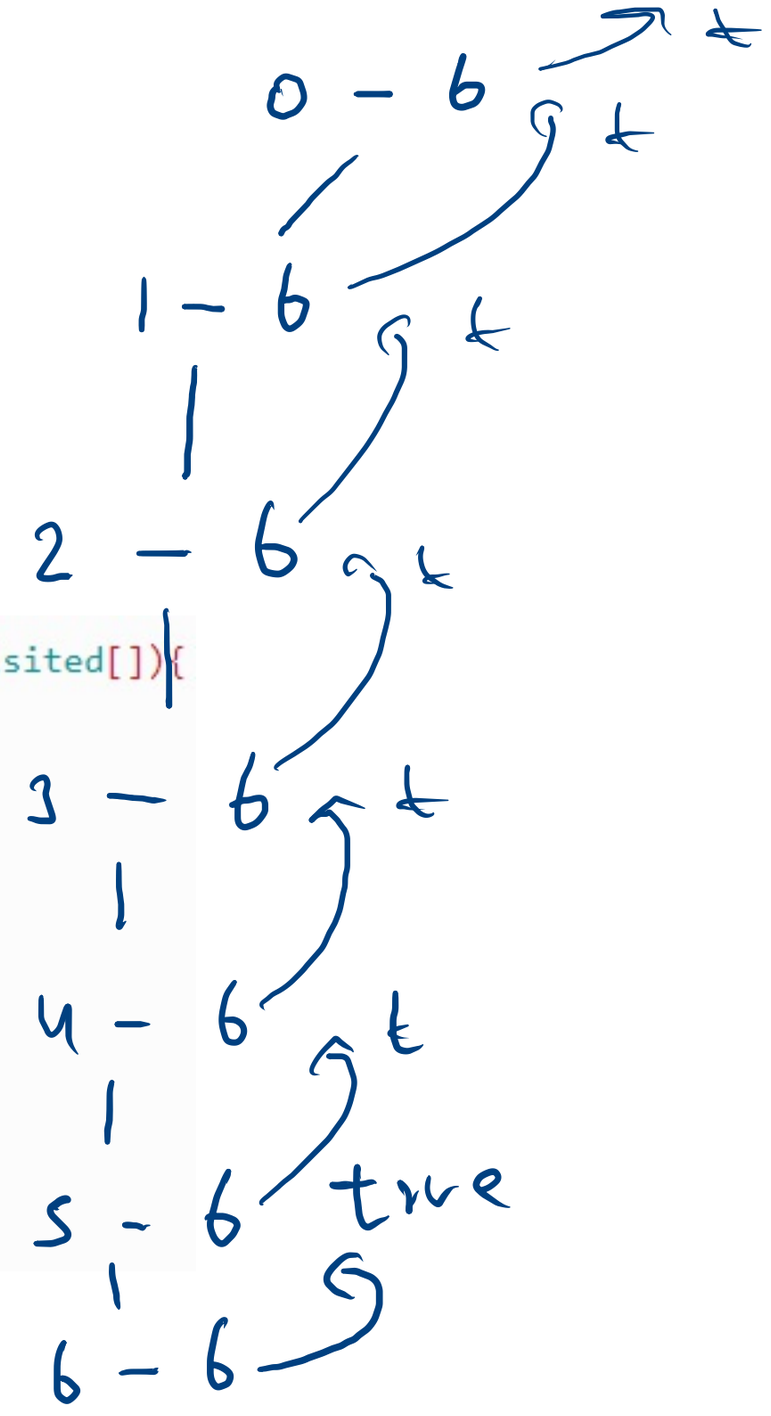


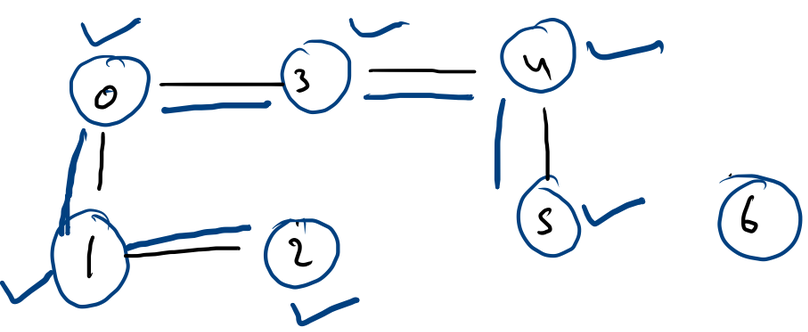



```java
public static boolean haspath(ArrayList<Edge>[] graph, int src, int dest, boolean visited[]){
    if(src == dest)return true;

    visited[src] = true;
    for(Edge edge: graph[src]){// src, nbr, wt
        if(visited[edge.nbr] == false){
            if(haspath(graph, edge.nbr, dest, visited)){
                return true;
            }
        }
    }

    return false;
}
```

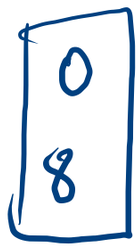
6 6

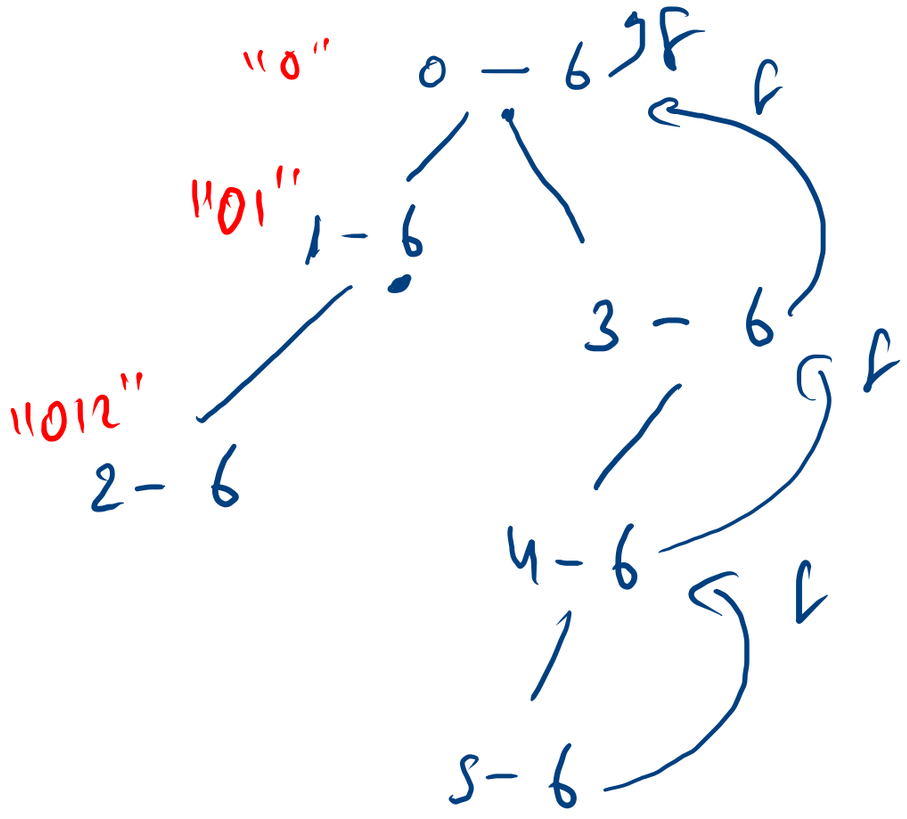
src 0
dest 6

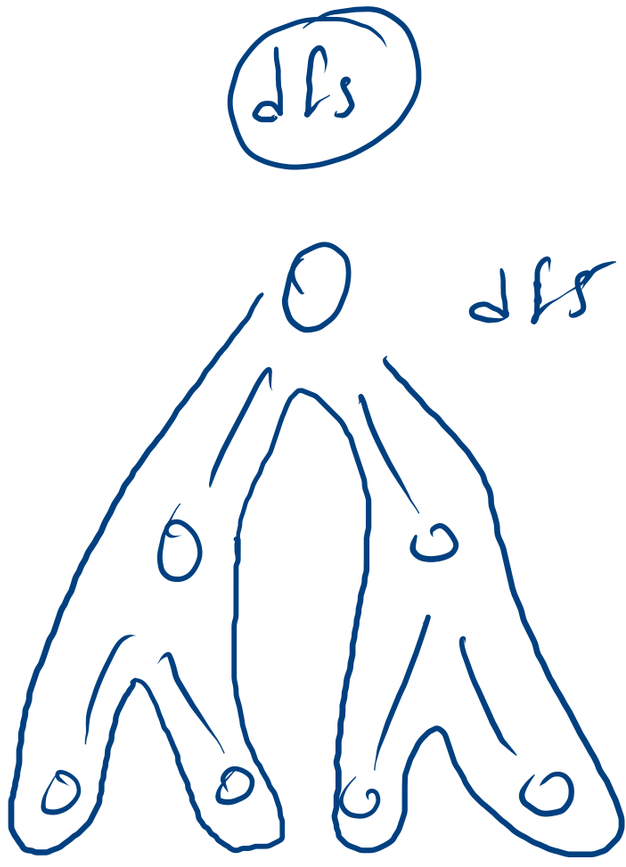


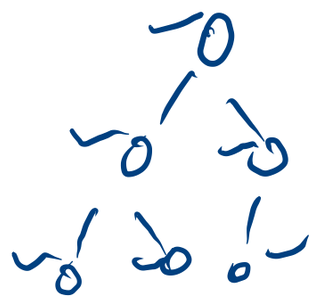


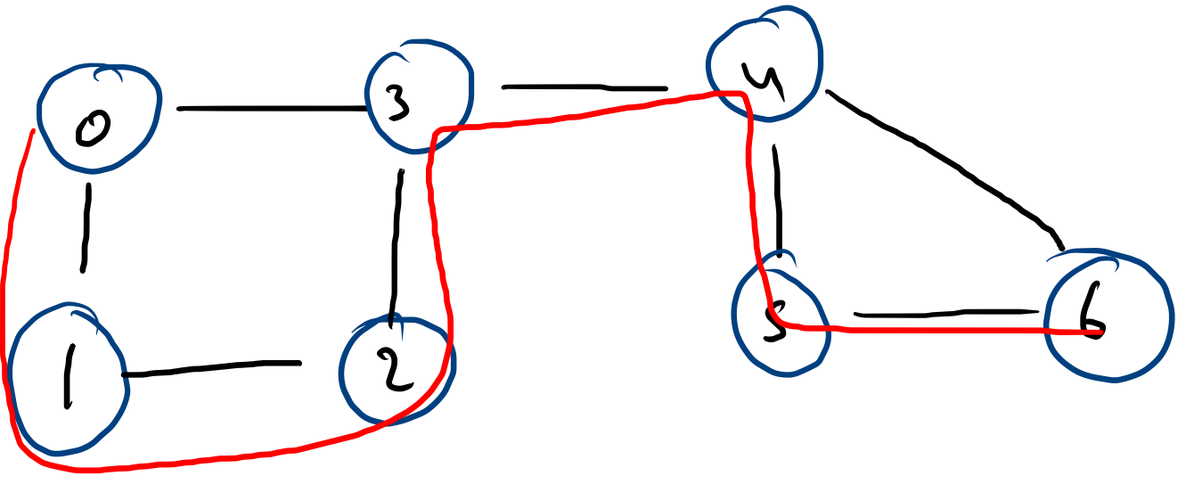
V
E



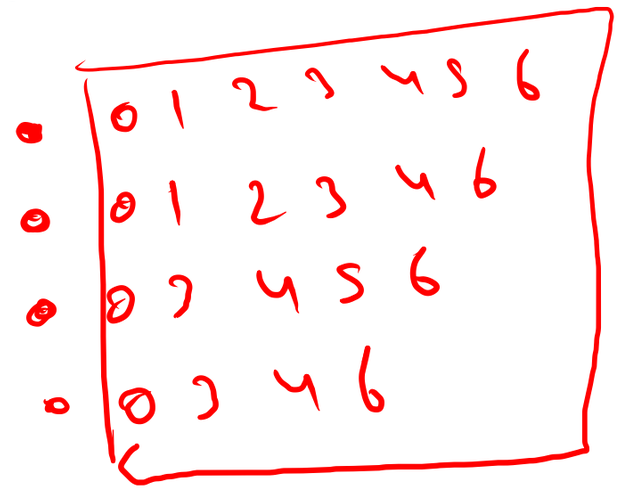








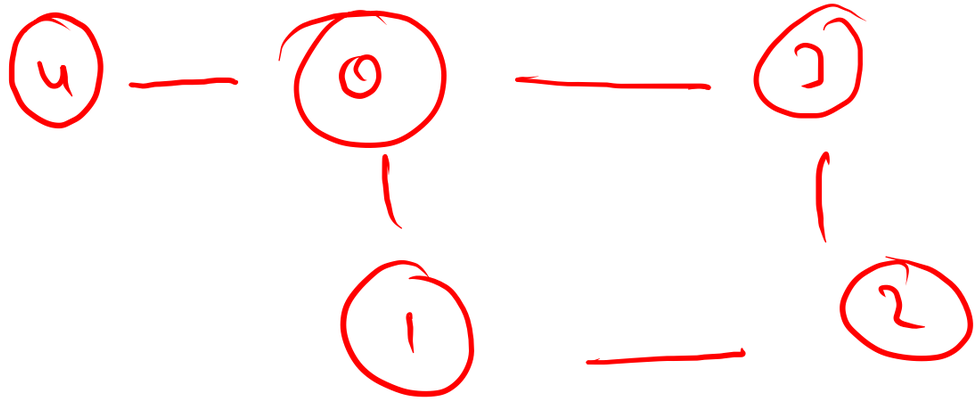
0 — 6

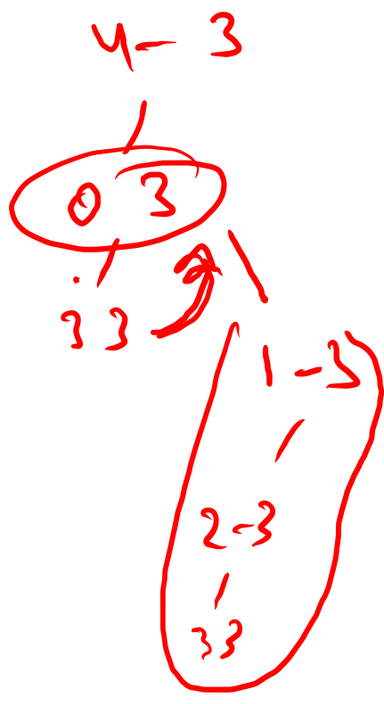


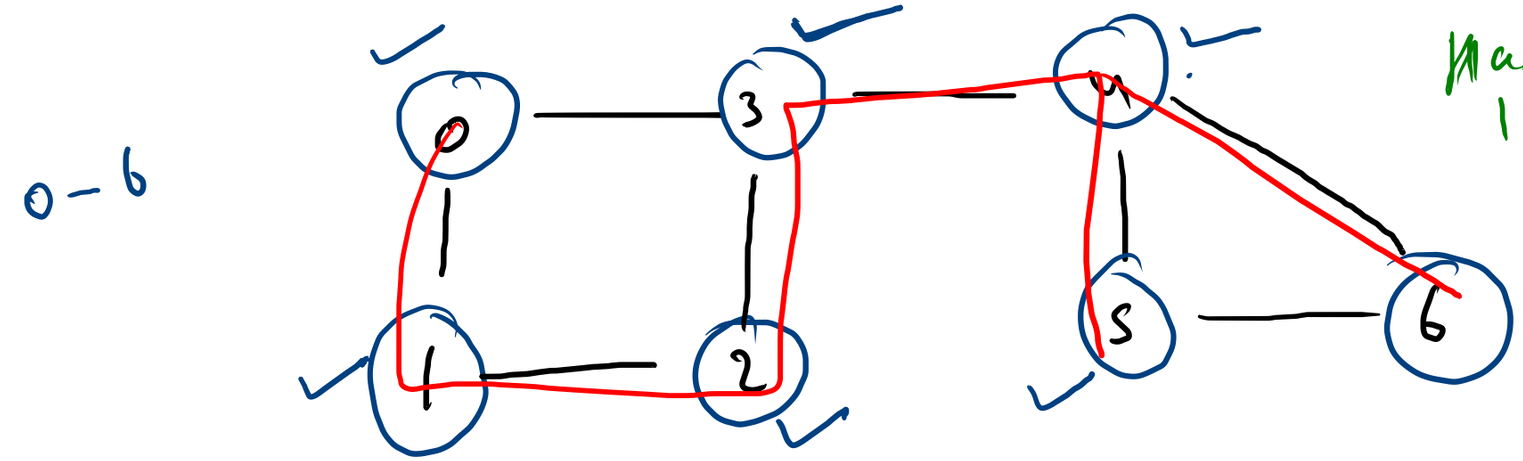
["","","",""]

smaller ... big

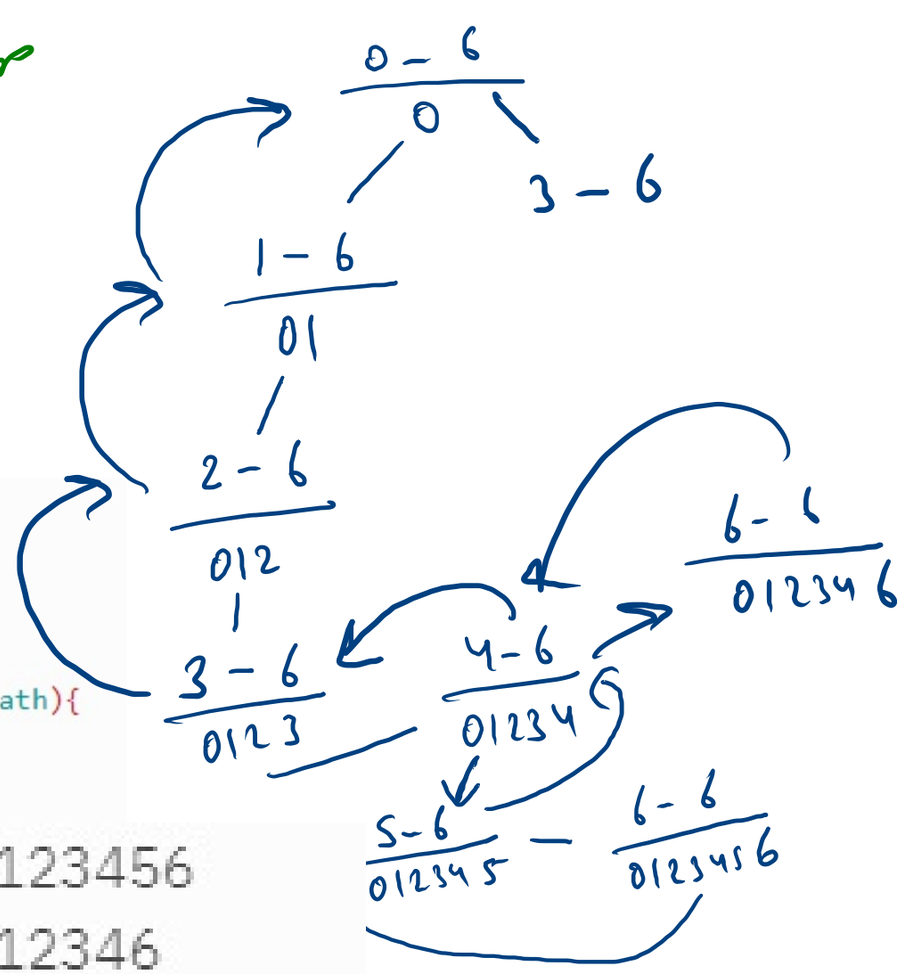
```
0123456
012346
03456
0346
```







Mark map
1.5




```java
  boolean visited[] = new boolean[vtces];
  haspath(graph, src, dest, visited, src+"");
  System.out.println(found);
}

public static void haspath(ArrayList<Edge>[] graph, int src, int dest, boolean visited[], String path){
    if(src == dest)return true;

    visited[src] = true;
  for(Edge edge: graph[src]){// src, nbr, wt
      if(visited[edge.nbr] == false){
          if(haspath(graph, edge.nbr, dest, visited, path+edge.nbr)){
              return true;
          }
      }
  }
  false
  return false;
}
```

- 0123456
- 012346
- 03456
- 0346

- LL BT BST Min Heap

easy ←
10' ←

Pair