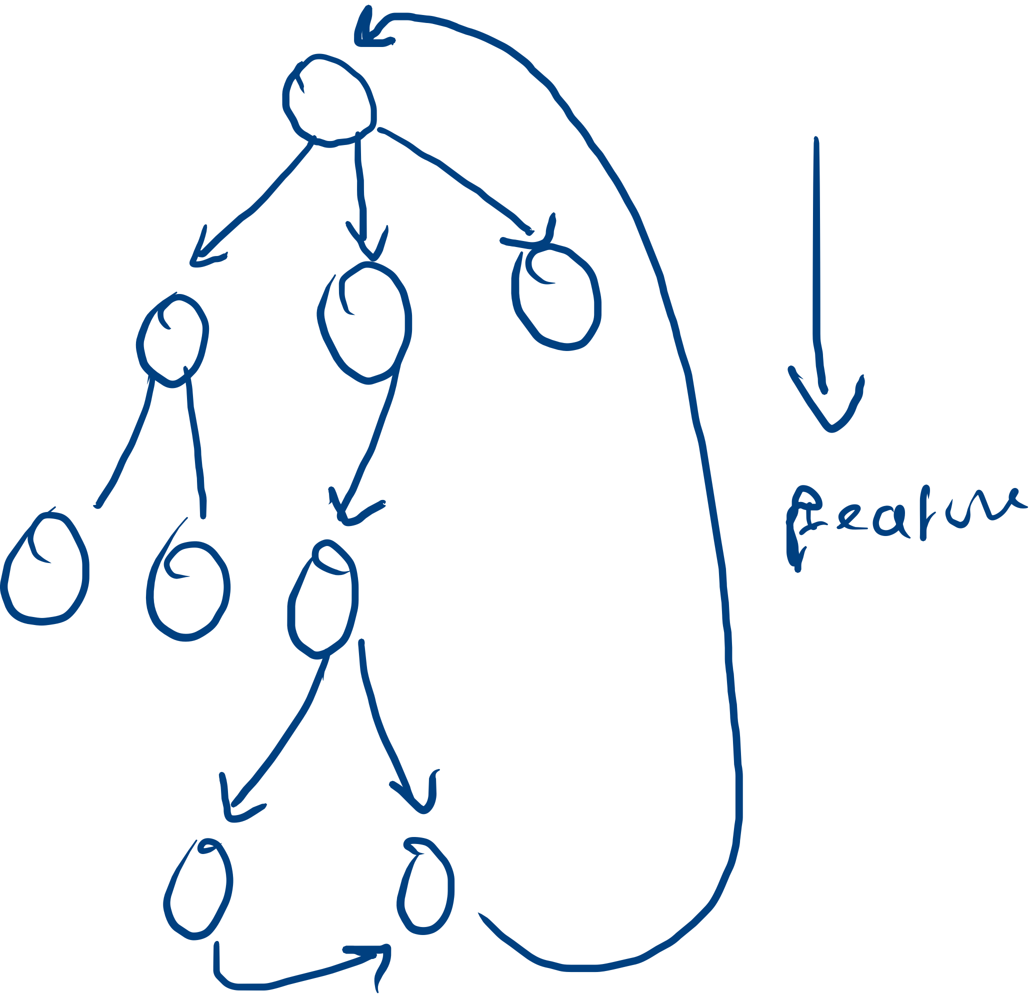
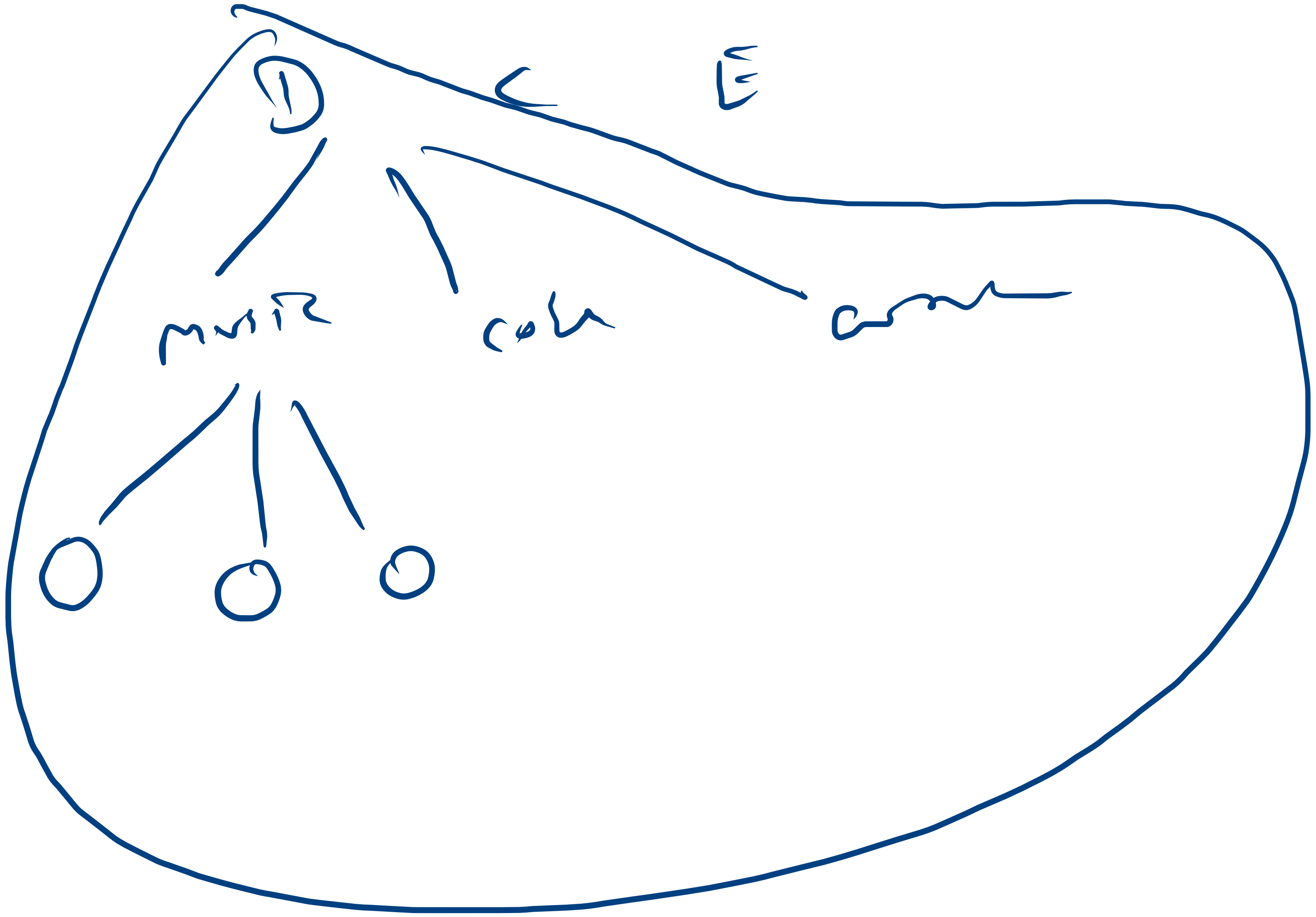
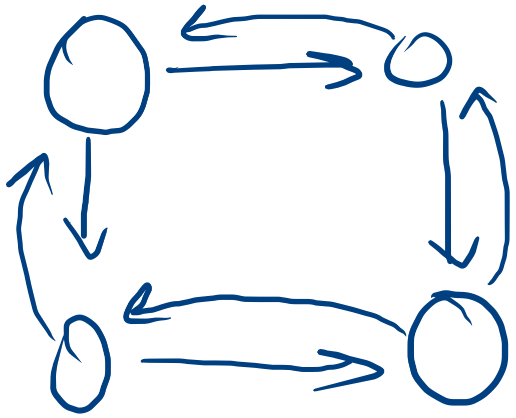

feature

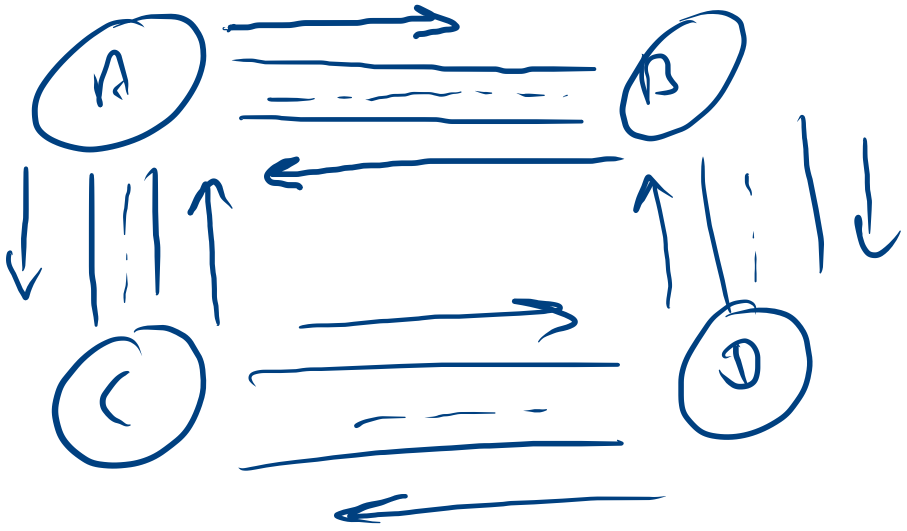

E
1
music
cola
cam

Graph

GT

N ←



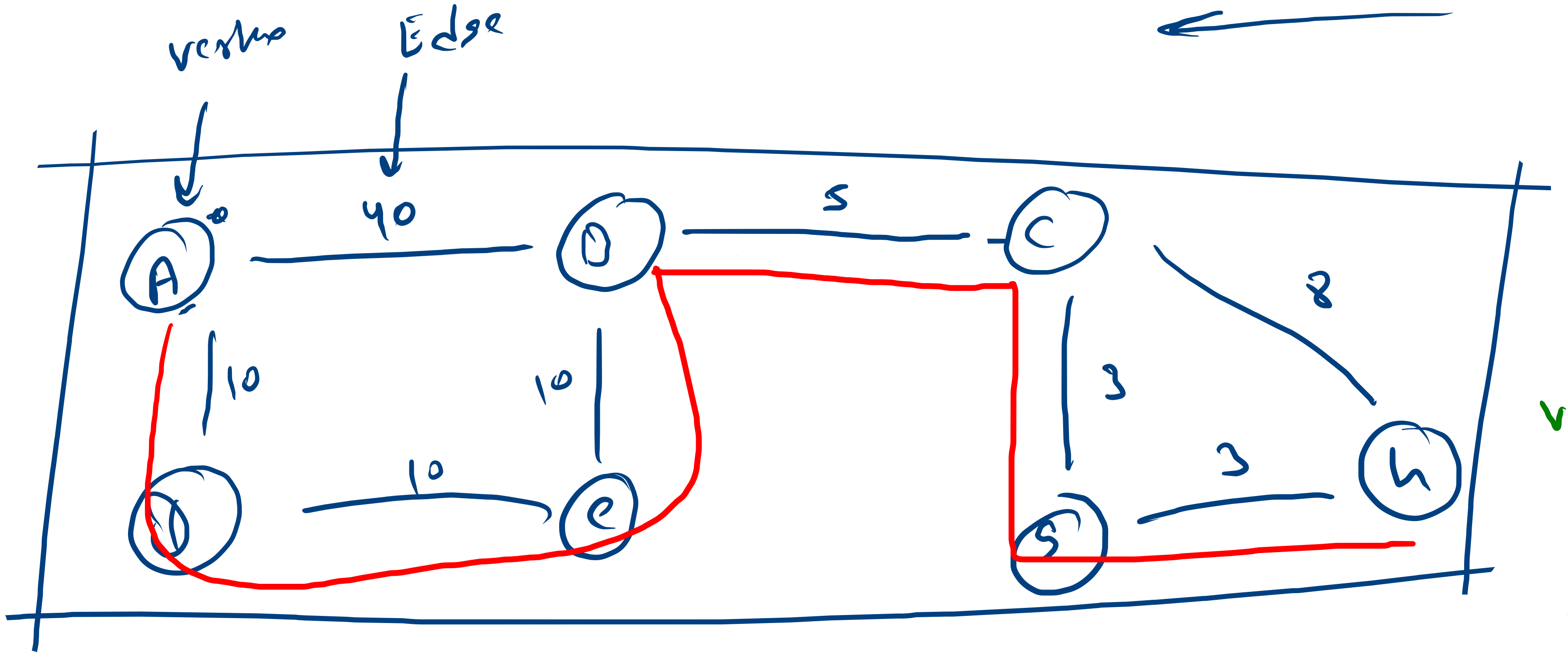


information network

Graph

vertex

Edge



hes path T / L

A-H all path

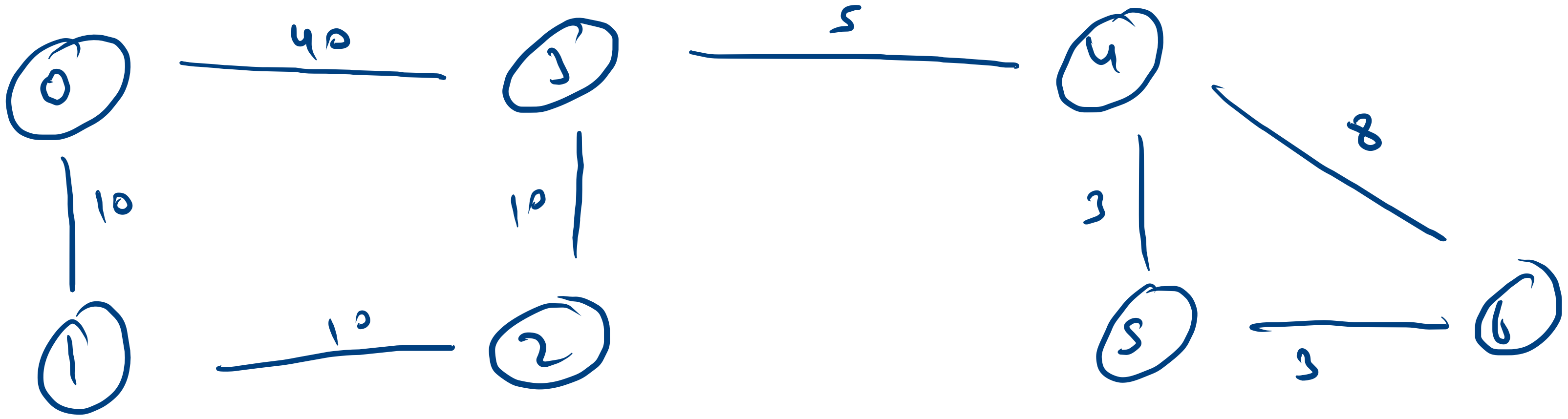
ver shortest

A B C H 4

A B C G H

A D E B C H

sh Edge 41 A D E B G H



1000

$\Rightarrow 10^6$

n^2

Adj.... matrix

	0	1	2	3	4	5	6
0	—	10	—	40			
1	10		10				
2		10		10			
3	40		10		5		
4				5		3	8
5					3		3
6					8	3	

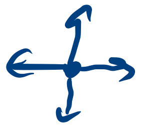
✓ real

direct connection $O(1)$

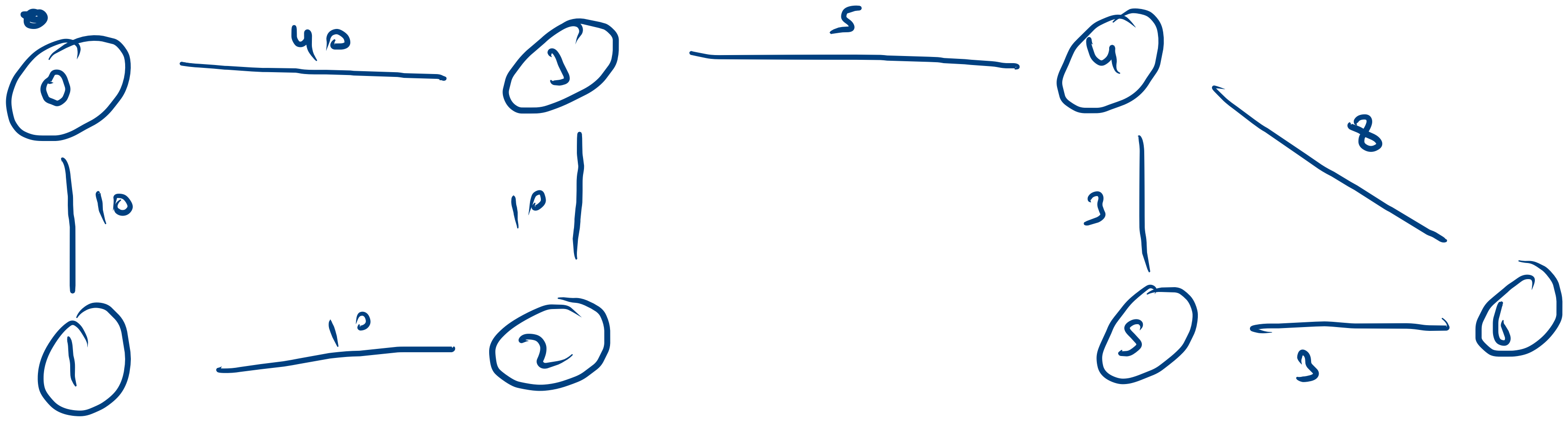
✗ 0 < all neighbors

✗ large memory

recursion flood fill



```
class Edge {
  int src
  int nbr
  int weight
}
```



1000

0 — 1

0 → [0 → 3 : 40 , 0 → 1 : 10]

1 → 1 → 0 : 10 , 1 → 2 : 10

2 → 2 → 1 : 10 2 → 3 : 10

3 → 3 → 4 : 5 , 3 → 0 : 40 , 3 → 2 : 10

4 →

5

6

✓ space opti

7 ← nuber of vertex

8 ← nub- edges

0 1 10

1 2 10

2 3 10

0 3 10

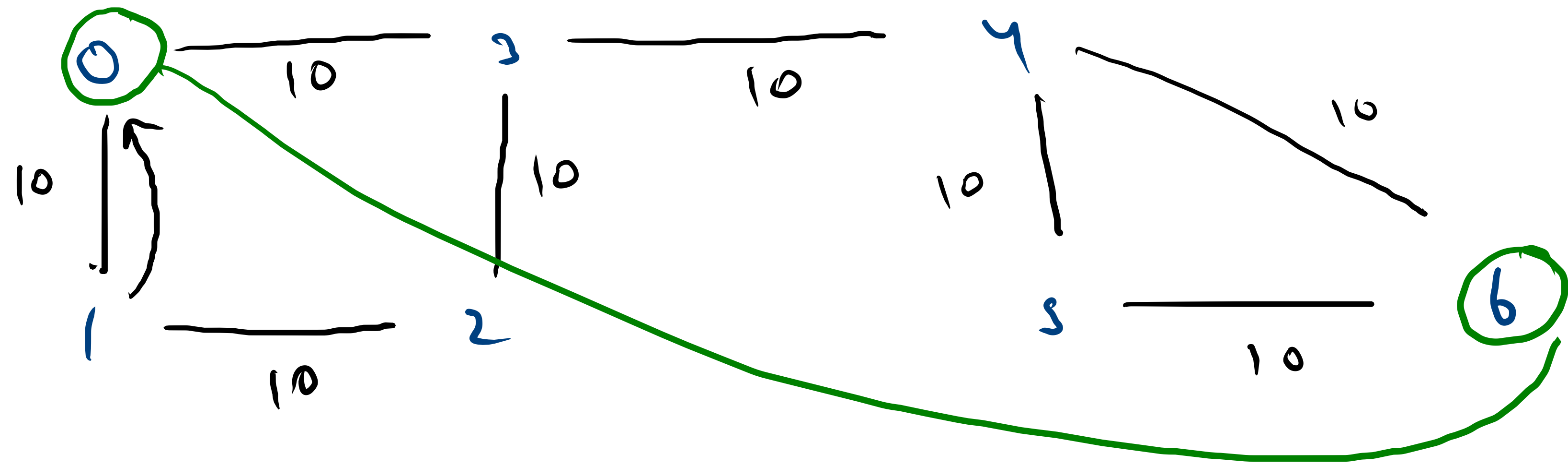
3 4 10

4 5 10

5 6 10

4 6 10

bi d.



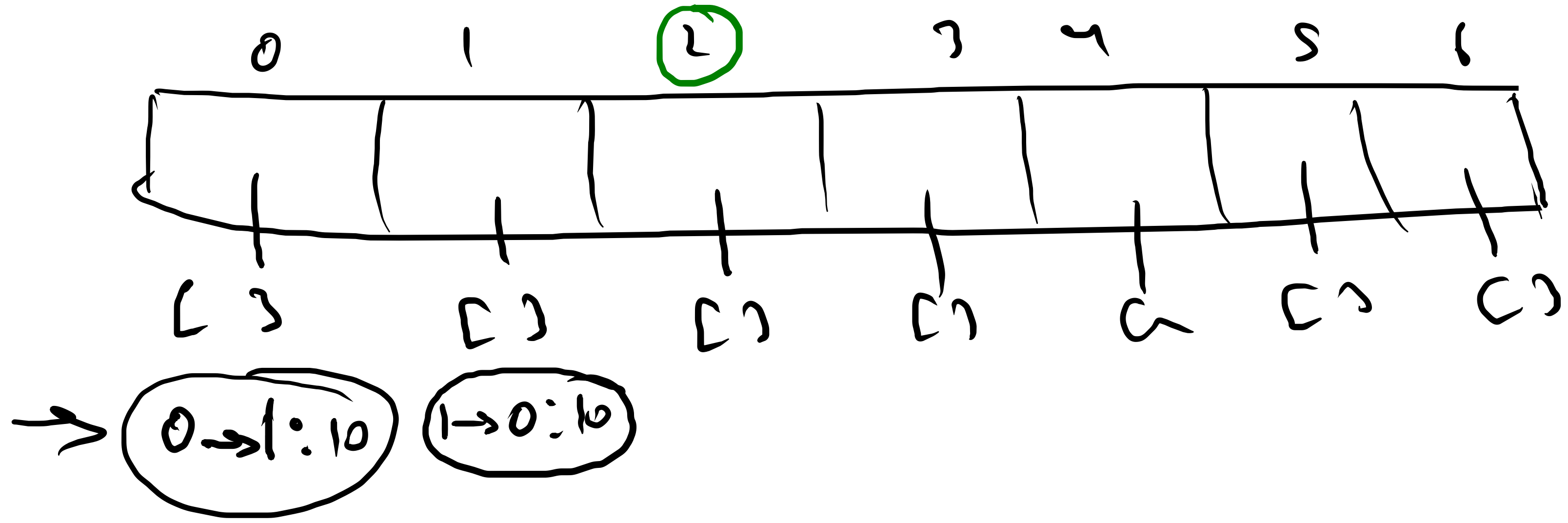
src 0

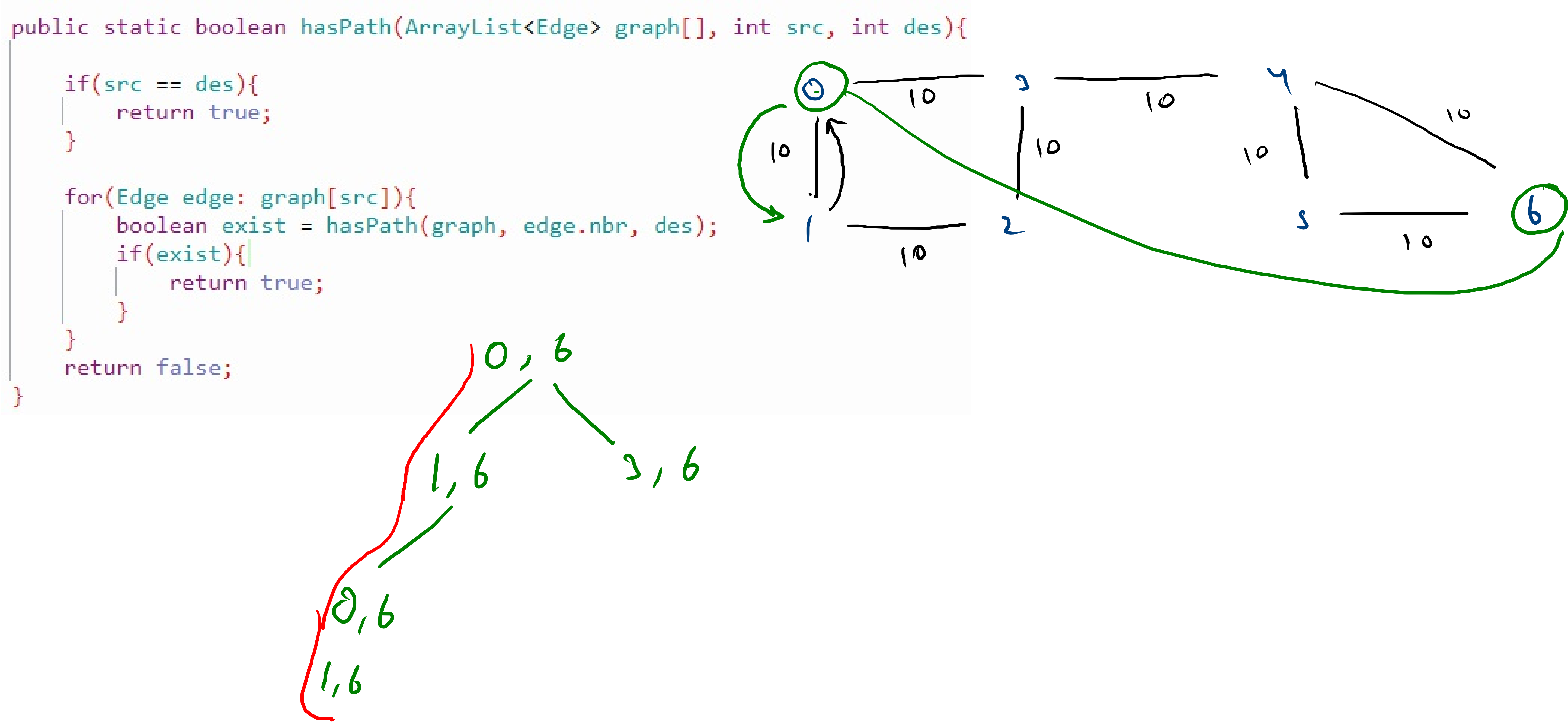
nbr 1

wt 10

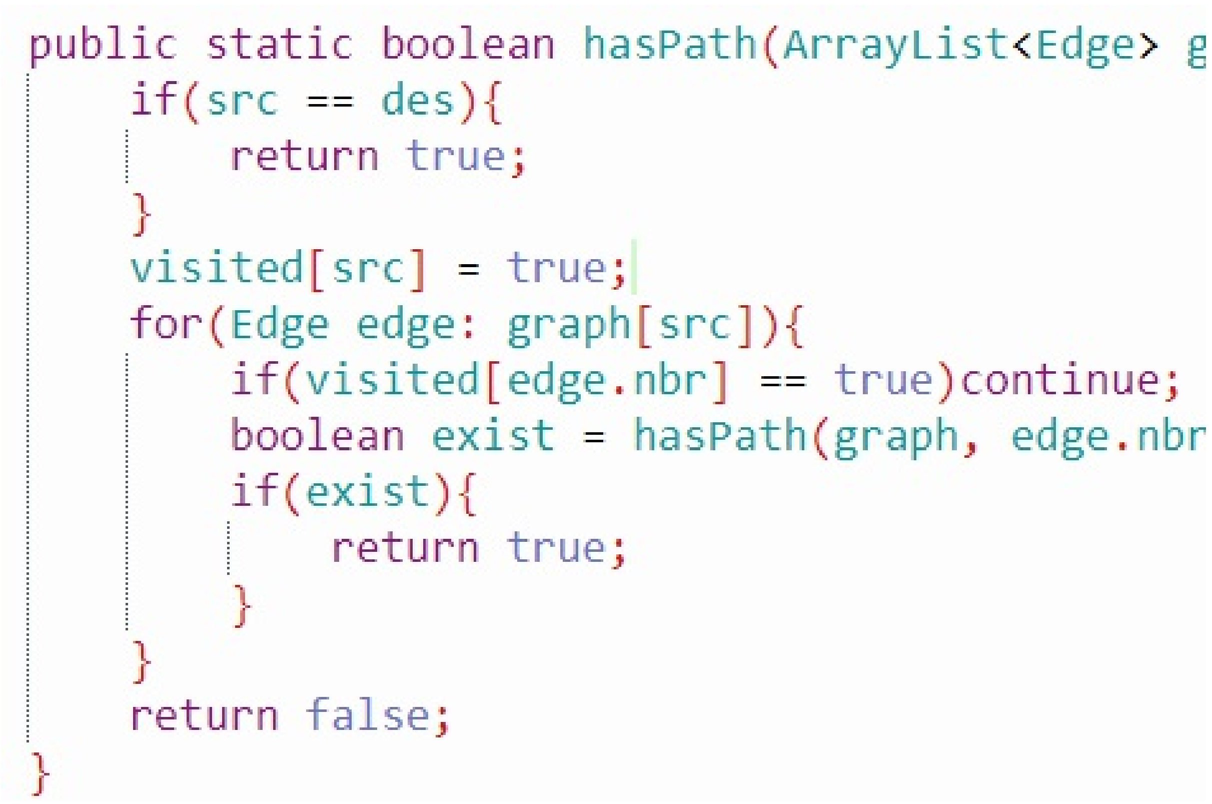
src→ nbr

nbr → src

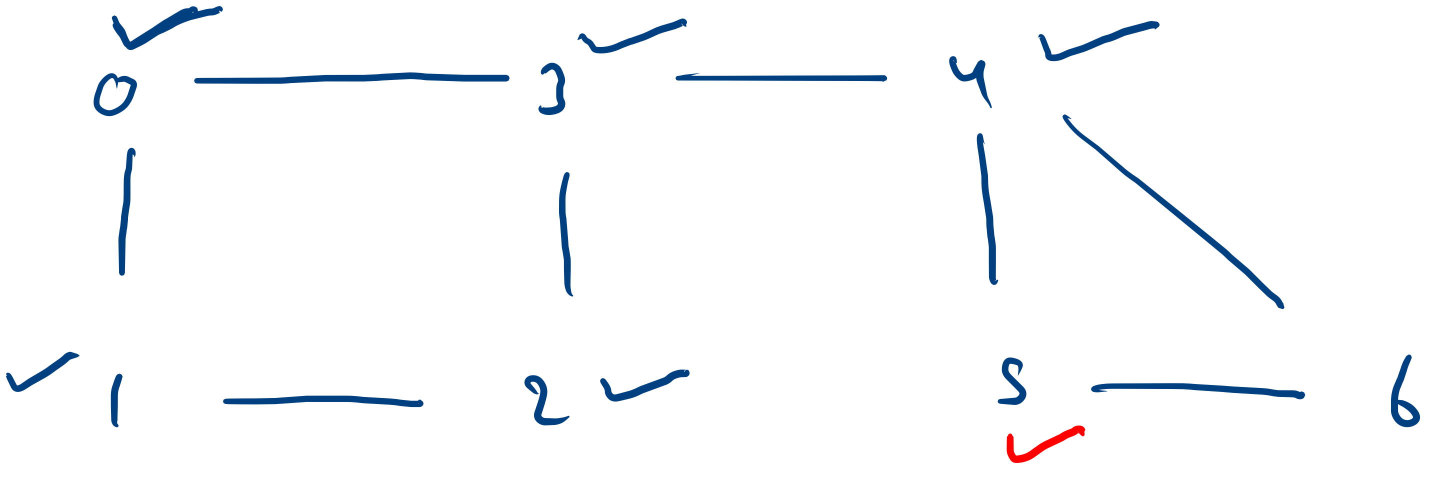


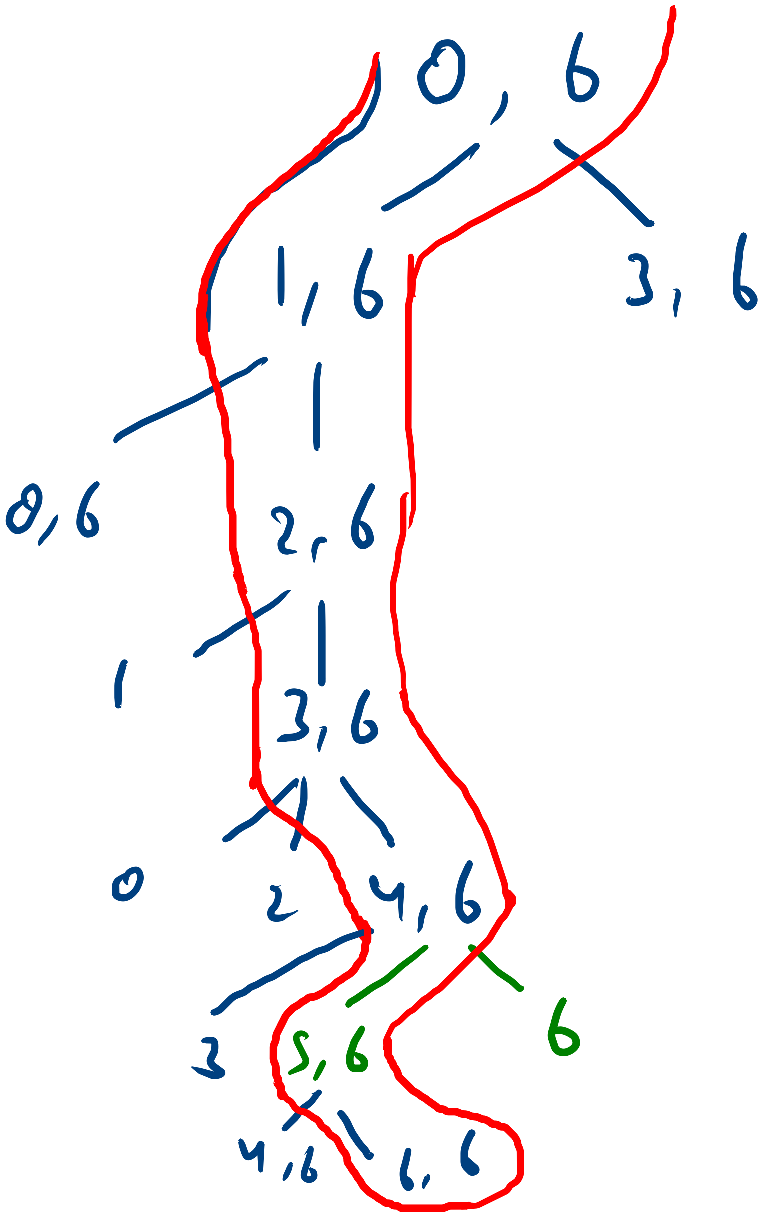


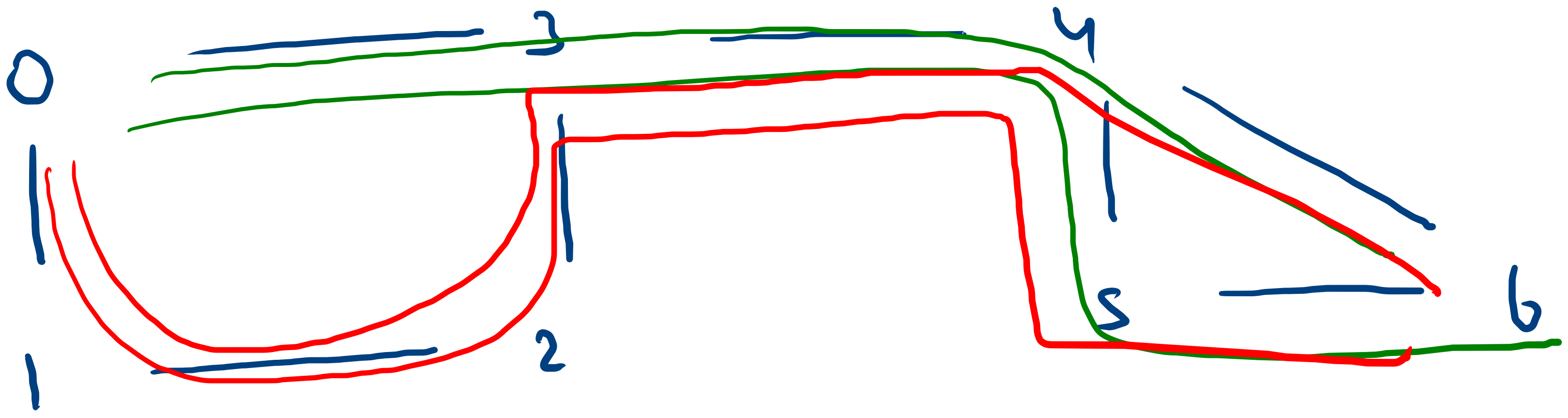
```
public static boolean hasPath(ArrayList<Edge> graph[], int src, int des){

    if(src == des){
        return true;
    }

    for(Edge edge: graph[src]){
        boolean exist = hasPath(graph, edge.nbr, des);
        if(exist){
            return true;
        }
    }
    return false;
}
```



```
public static boolean hasPath(ArrayList<Edge> g
    if(src == des){
        return true;
    }
    visited[src] = true;
    for(Edge edge: graph[src]){
        if(visited[edge.nbr] == true)continue;
        boolean exist = hasPath(graph, edge.nbr
        if(exist){
            return true;
        }
    }
    return false;
}
```







src 0

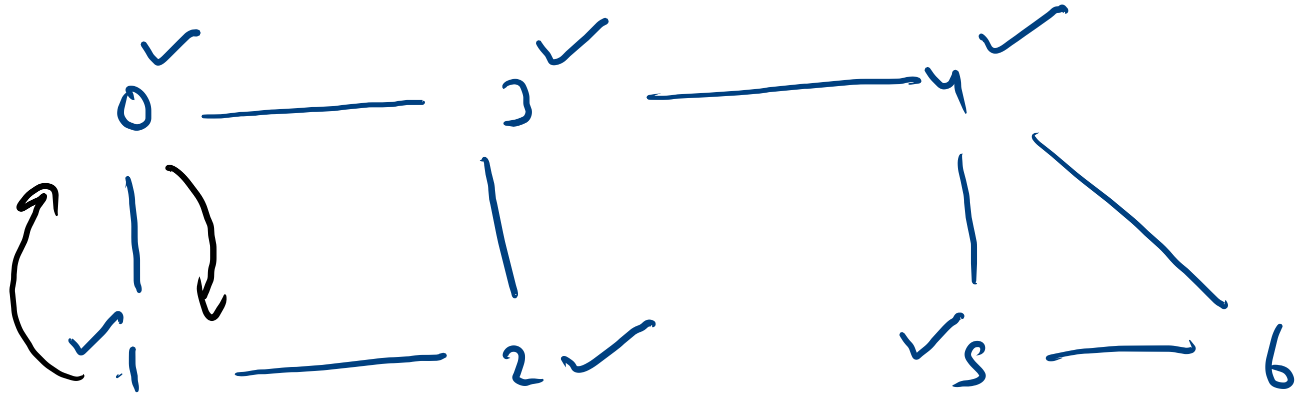
des 6

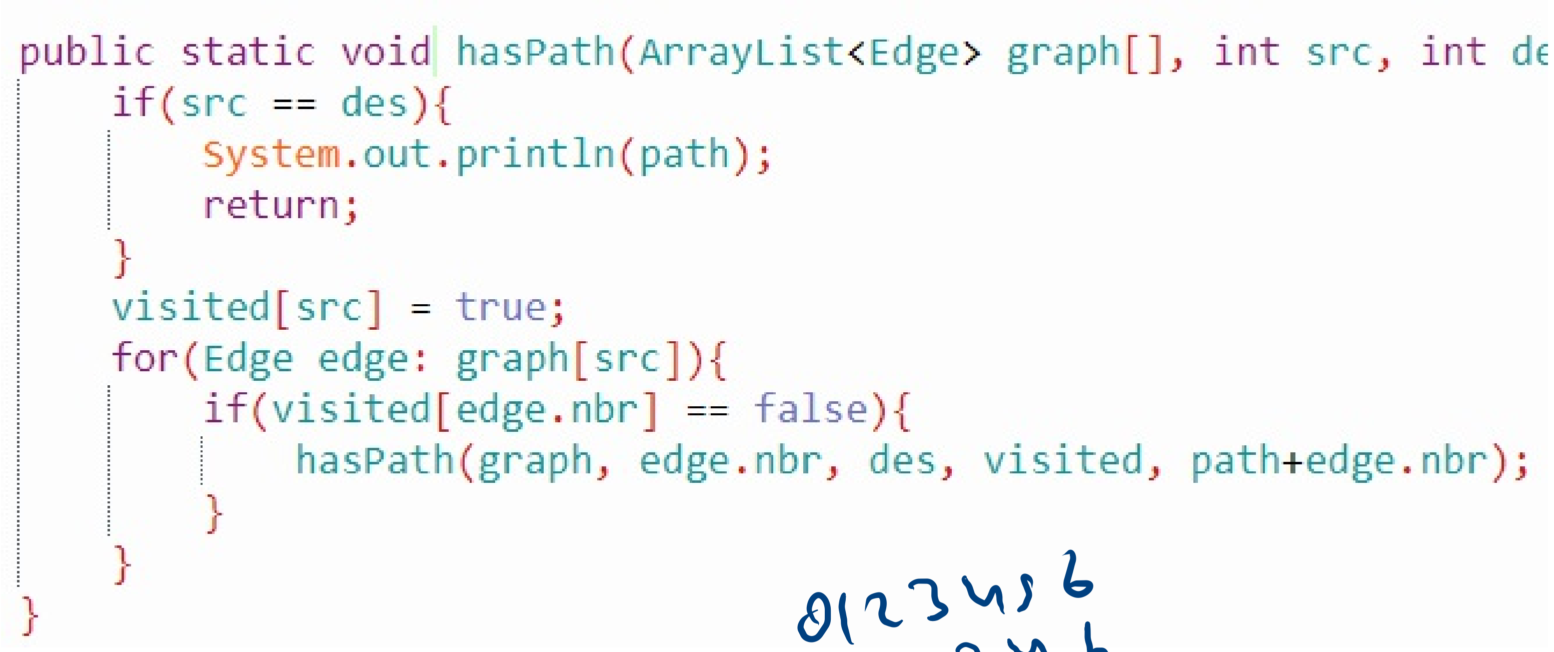
0 3 4 6

0 3 4 5 6

0 1 2 3 4 6

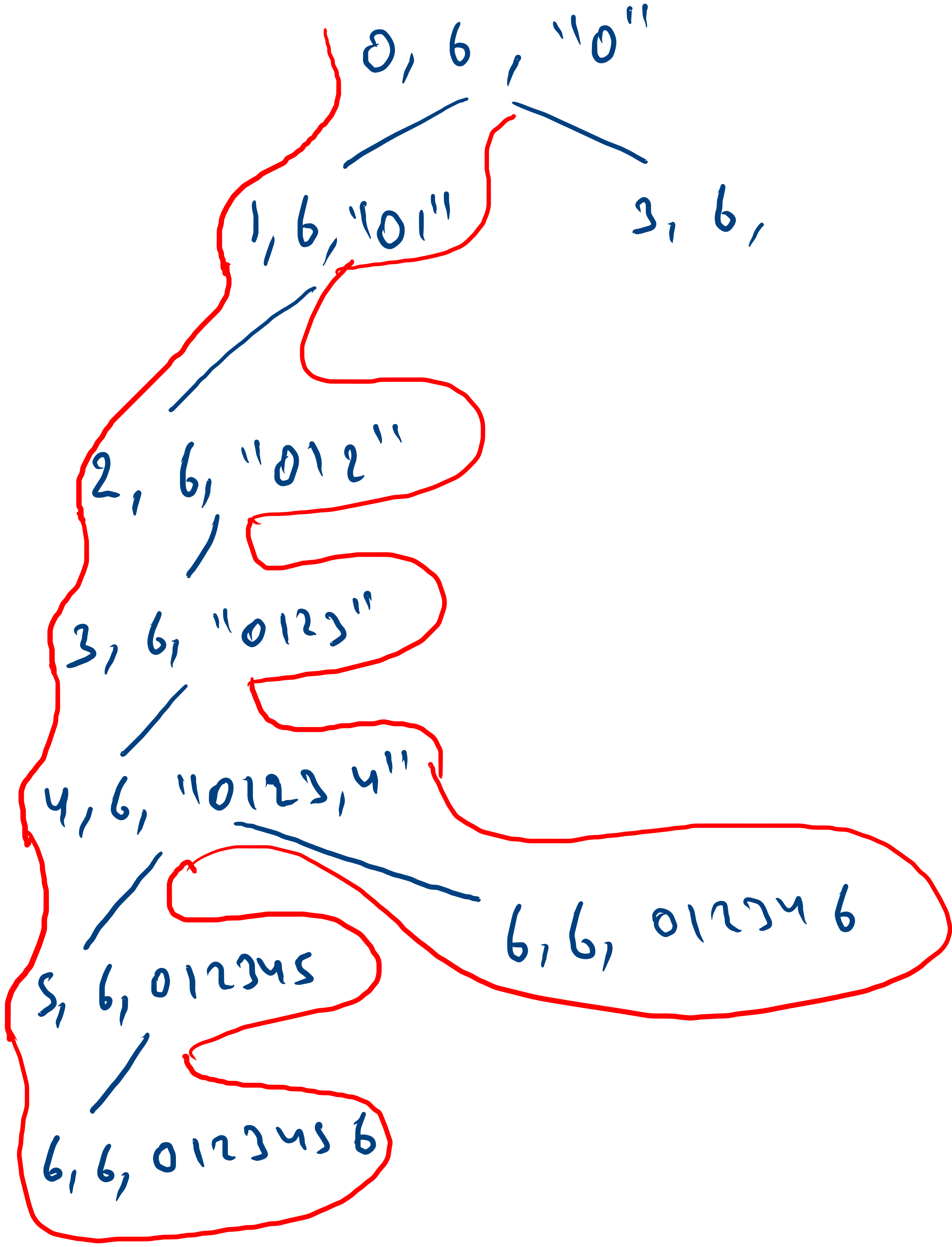
0 1 2 3 4 5 6

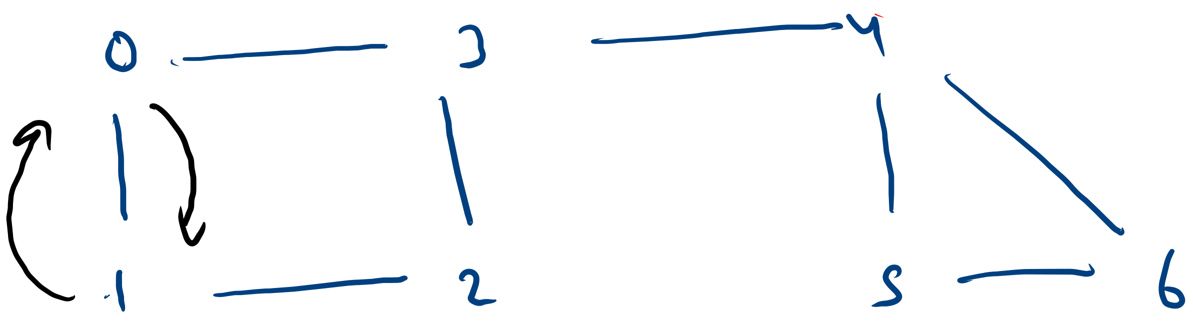


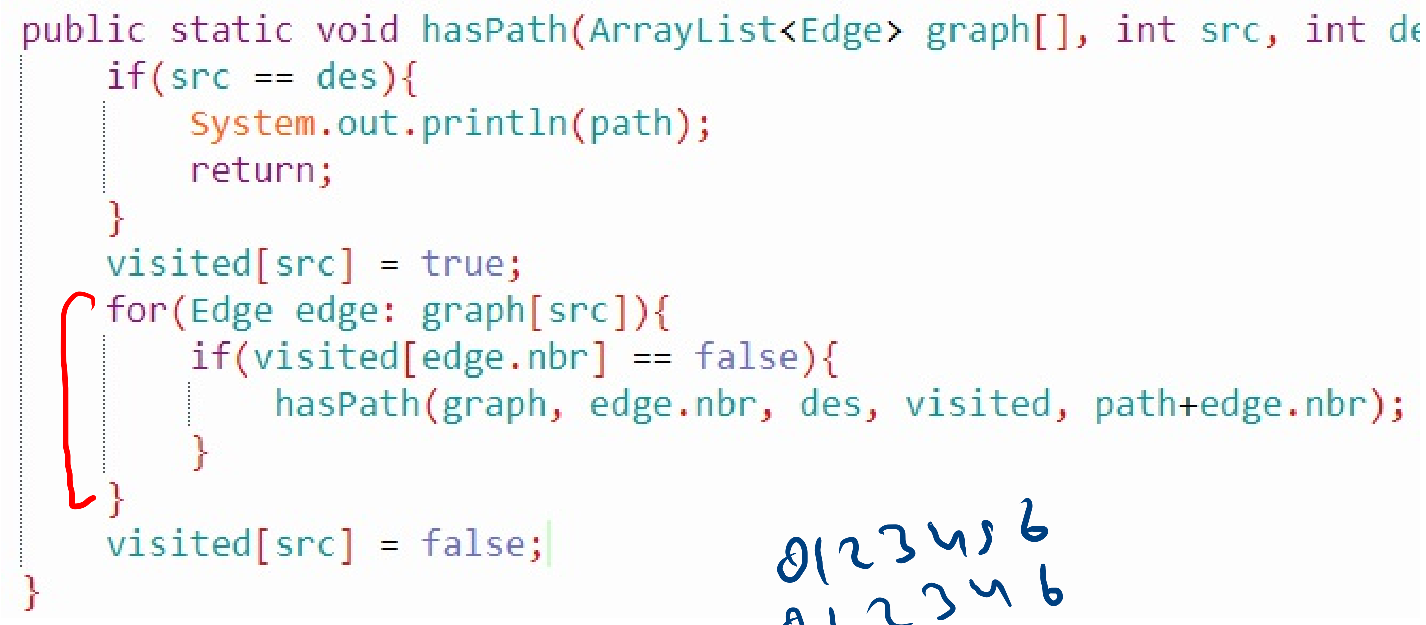


```
public static void hasPath(ArrayList<Edge> graph[], int src, int de
    if(src == des){
        System.out.println(path);
        return;
    }
    visited[src] = true;
    for(Edge edge: graph[src]){
        if(visited[edge.nbr] == false){
            hasPath(graph, edge.nbr, des, visited, path+edge.nbr);
        }
    }
}
```

0123456
012346

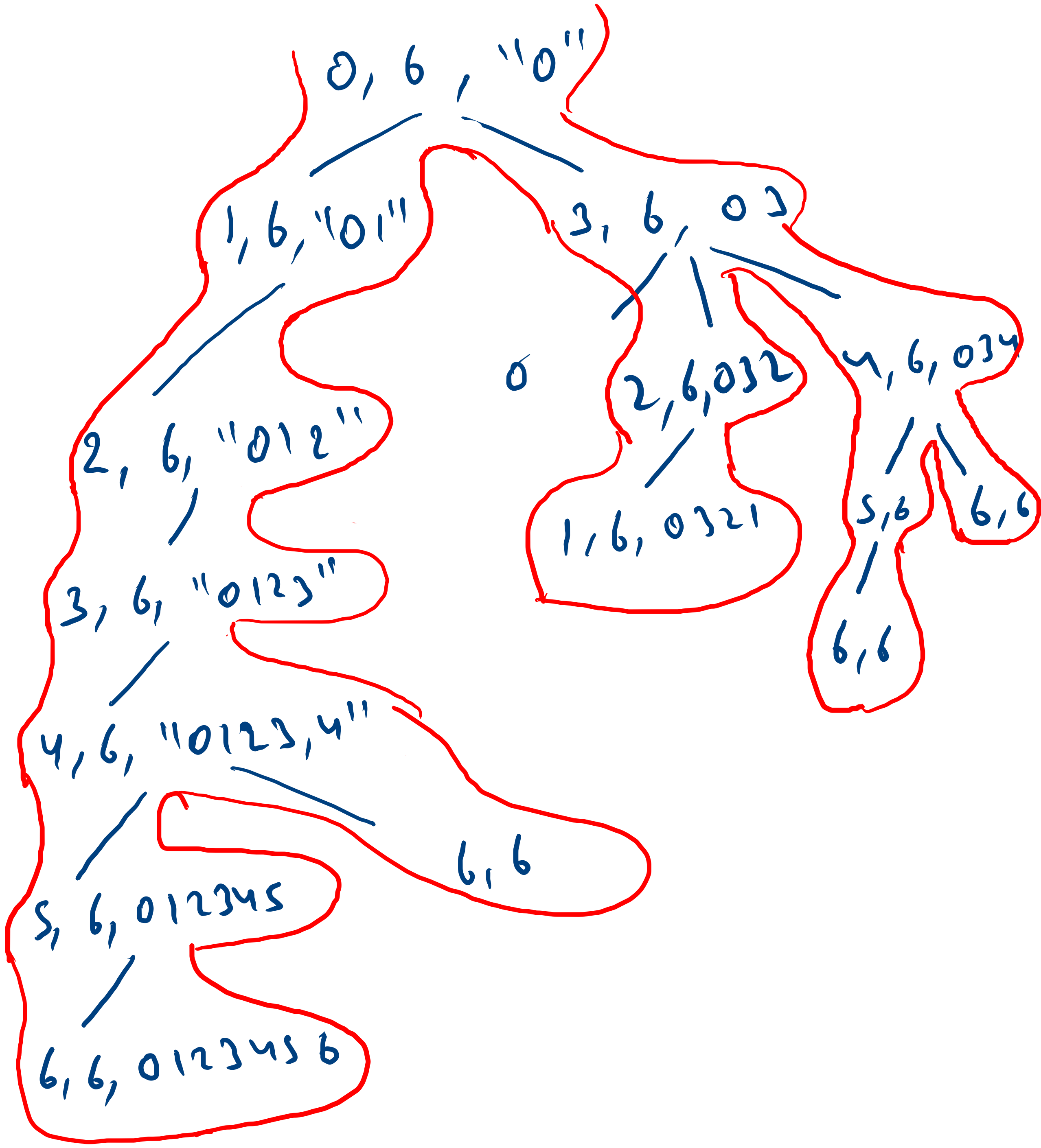


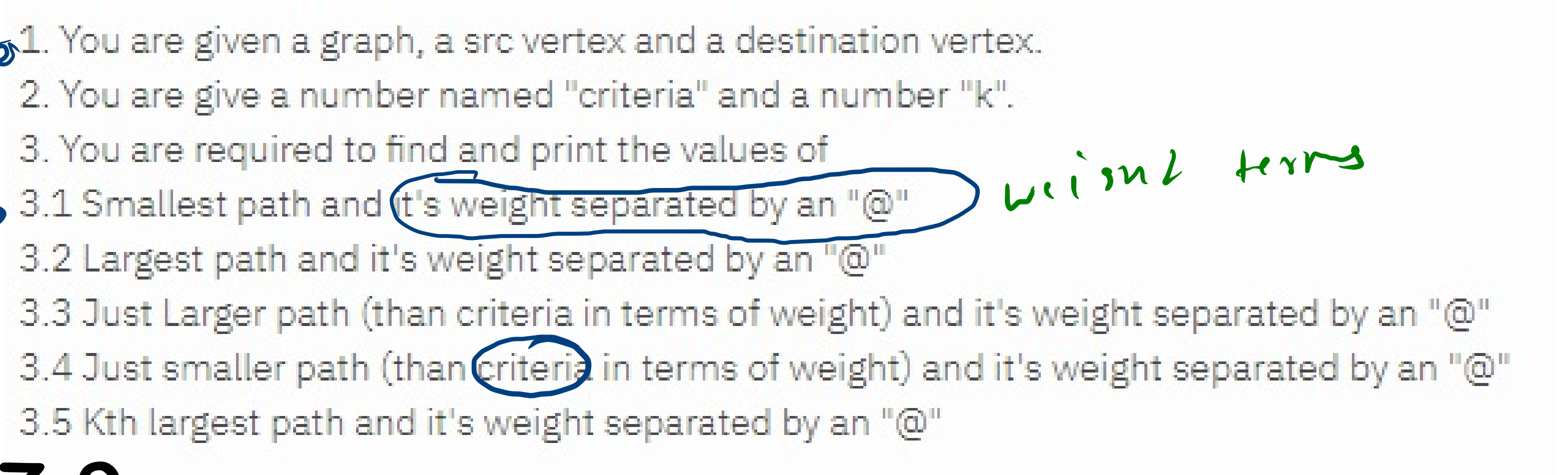




```
public static void hasPath(ArrayList<Edge> graph[], int src, int de
    if(src == des){
        System.out.println(path);
        return;
    }
    visited[src] = true;
    for(Edge edge: graph[src]){
        if(visited[edge.nbr] == false){
            hasPath(graph, edge.nbr, des, visited, path+edge.nbr);
        }
    }
    visited[src] = false;
}
```

0123456
012346
03456
0346





1. You are given a graph, a src vertex and a destination vertex.
2. You are give a number named "criteria" and a number "k".
3. You are required to find and print the values of

3.1 Smallest path and it's weight separated by an "@" weight terms

3.2 Largest path and it's weight separated by an "@"

ceil 3.3 Just Larger path (than criteria in terms of weight) and it's weight separated by an "@"

floor 3.4 Just smaller path (than criteria in terms of weight) and it's weight separated by an "@"

→ 3.5 Kth largest path and it's weight separated by an "@"

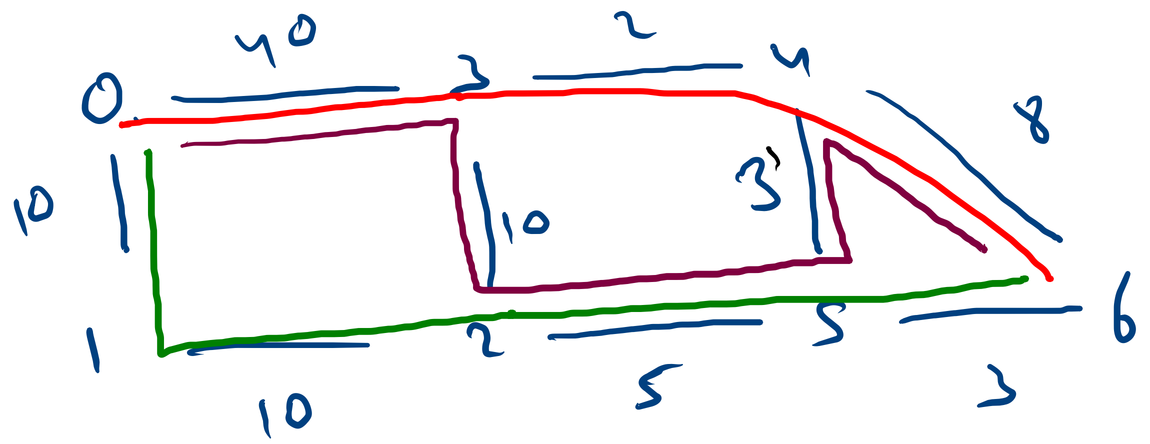
```
7 9
0 1 10
1 2 10
2 3 10
0 3 40
3 4 2
4 5 3
5 6 3
4 6 8
2 5 5
0 6 30 4
```

src → 0

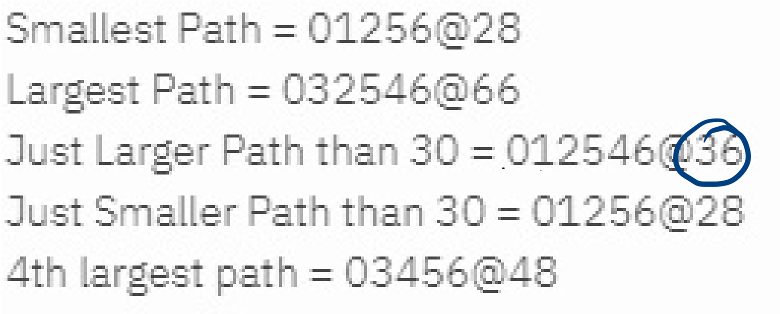
des → 6

criter → 30

K = 4



0 3 2 5 4 6 @ 66



Smallest Path = 01256@28
Largest Path = 032546@66
Just Larger Path than 30 = 012546@36
Just Smaller Path than 30 = 01256@28
4th largest path = 03456@48

K.

1 → 66

2 → 60

3 → 58

4 → 56

5 → 55