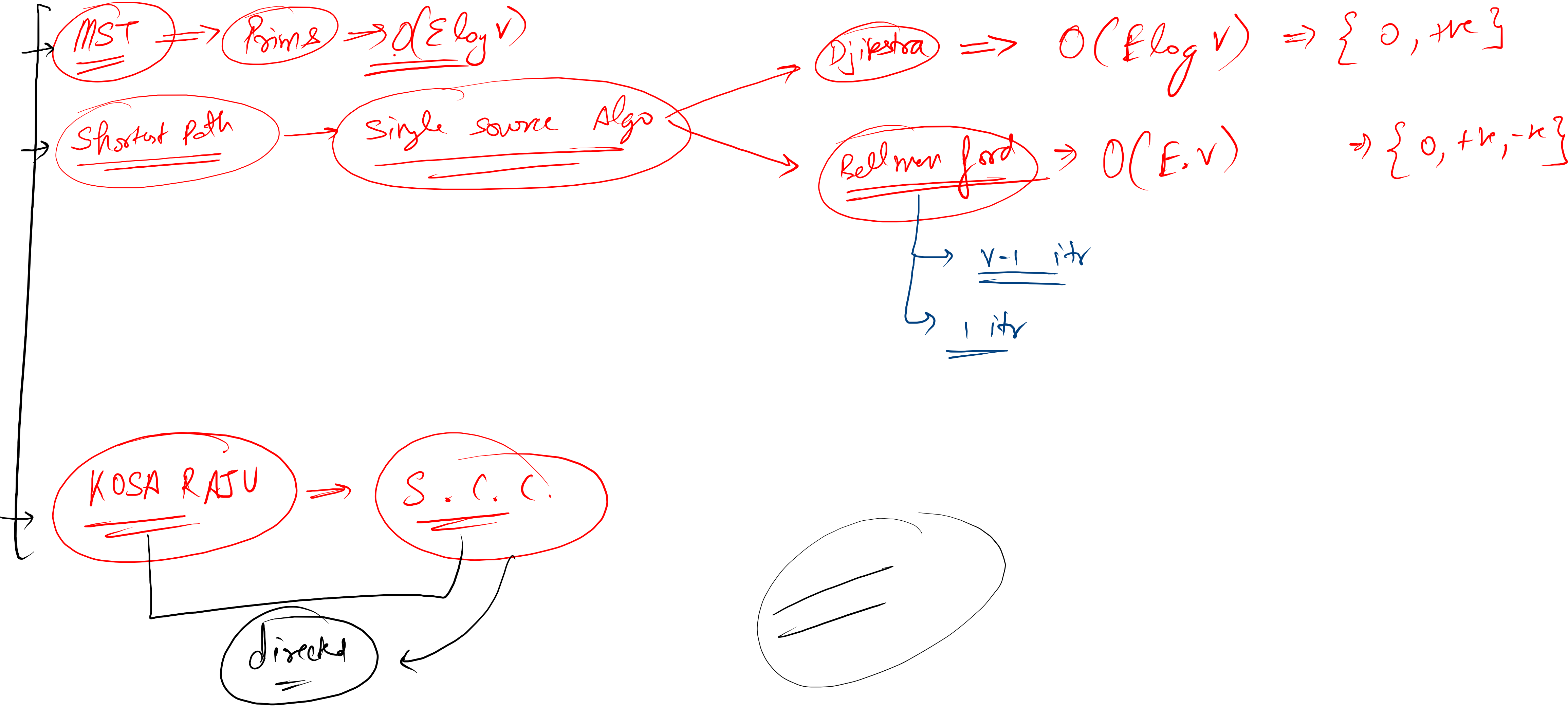


MST → Prims → $O(E \log V)$

Shortest Path → Single source Algo

Djikstra ⇒ $O(E \log V)$ ⇒ $\{0, +ve\}$

Bellman ford → $O(E.V)$ ⇒ $\{0, +ve, -ve\}$

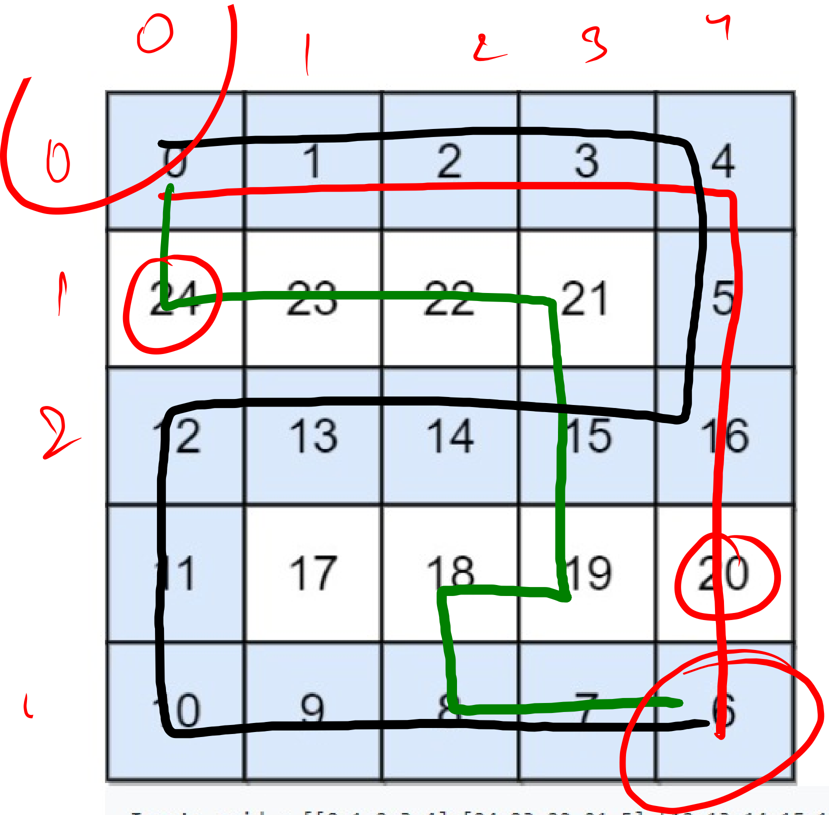
V-1 itr

1 itr

KOSA RAJU → S.C.C.

directed

{0,0} → {4,4}



| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 3 | 4 |
| 1 | 24 | 23 | 22 | 21 | 5 |
| 2 | 12 | 13 | 14 | 15 | 16 |
| | 11 | 17 | 18 | 19 | 20 |
| | 10 | 9 | 8 | 7 | 6 |

Input: grid = [[0,1,2,3,4],[24,23,22,21,5],[12,13,14,15,16],[11,17,18,19,20],[10,9,8,7,6]]
Output: 16
Explanation: The final route is shown.
We need to wait until time 16 so that (0, 0) and (4, 4) are connected.

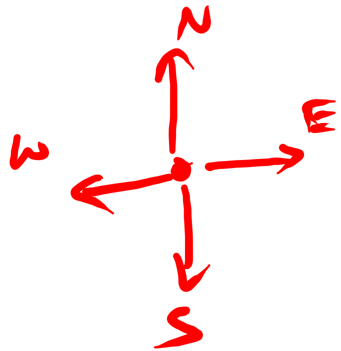
val → Output → min Cost of path

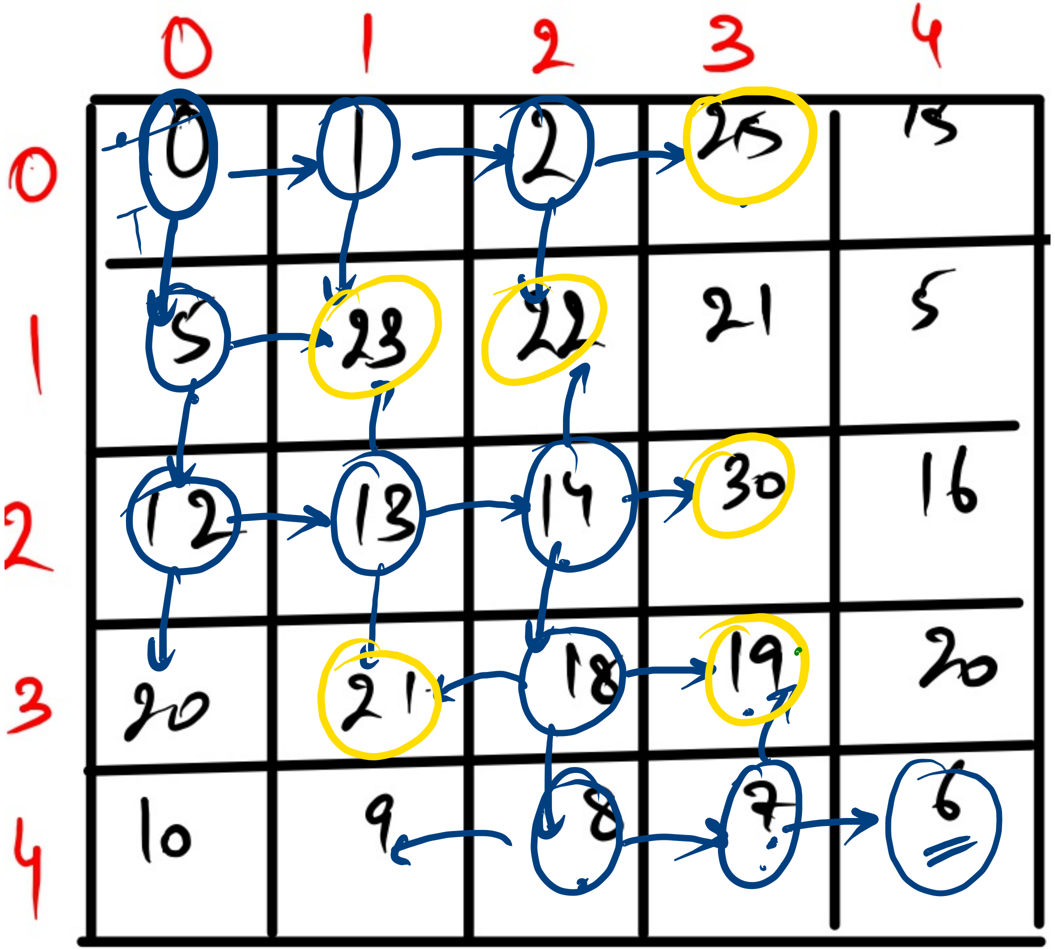
↳ Maximum value encountered along path.

16 → Output

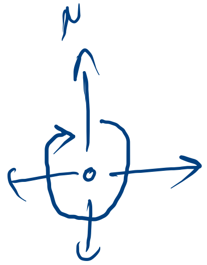
20

24





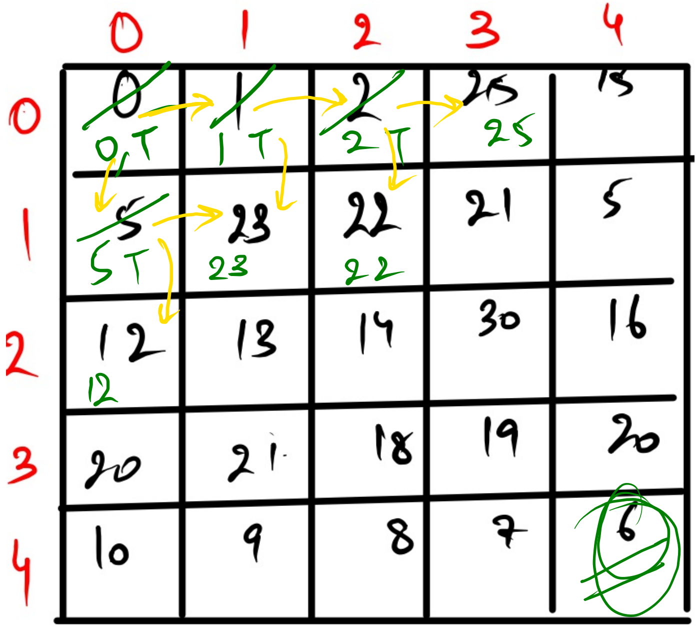
|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 2 | 25 | 15 |
| 1 | 5 | 23 | 22 | 21 | 5 |
| 2 | 12 | 13 | 14 | 30 | 16 |
| 3 | 20 | 21 | 18 | 19 | 20 |
| 4 | 10 | 9 | 8 | 7 | 6 |



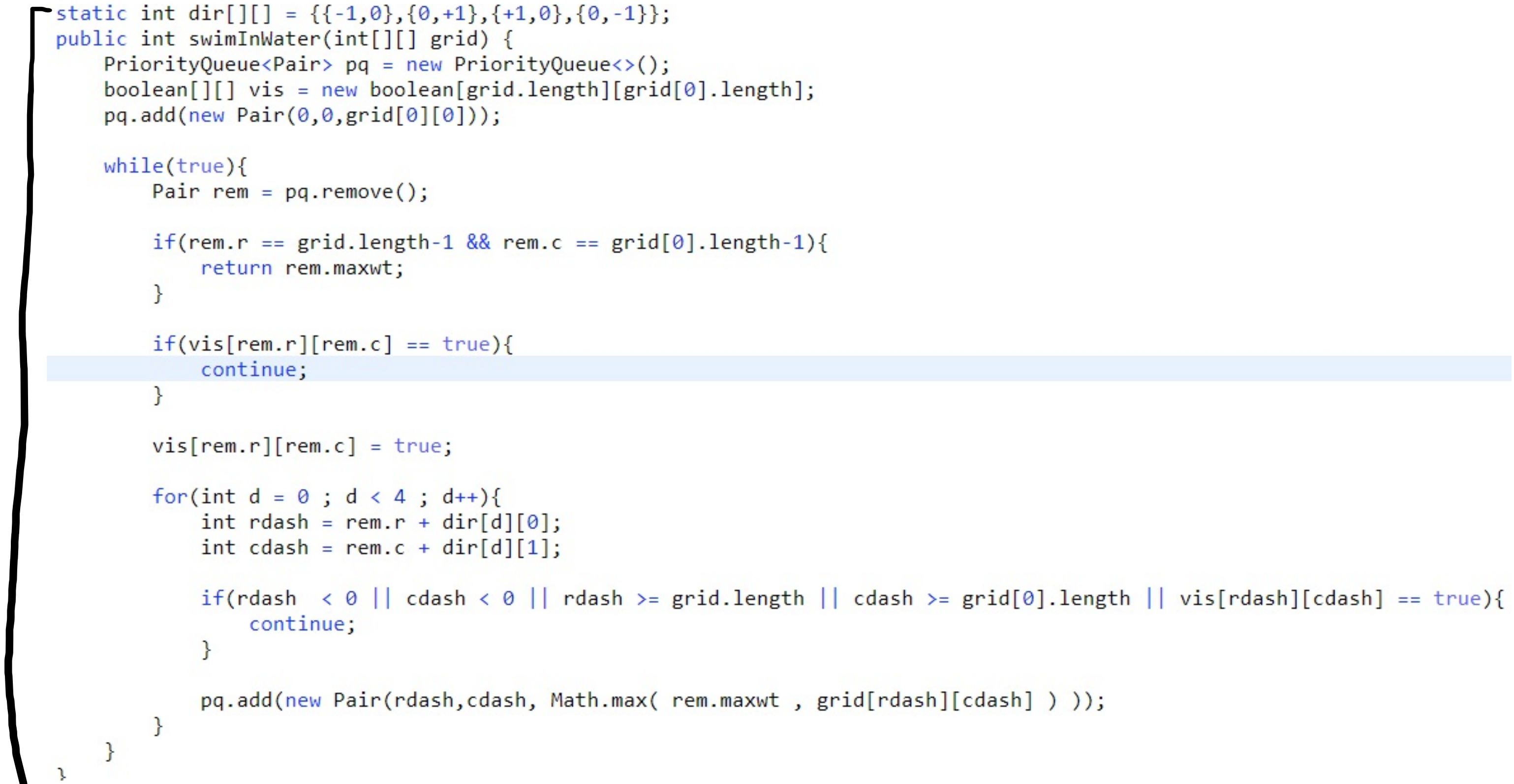
{r, c, mwt}

{1,1,23}
{0,3,25}
{1,1,23}
{3,0,20}
{1,1,23}
{1,2,22}
~~{4,3,18}~~
{3,3,19}

{3,1,21}
{1,2,22}
{2,3,30}
{3,3,19}
{3,1,21}
~~{4,2,18}~~
{4,4,18}
~~{4,4,18}~~



|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | ~~0~~<br>0,T | ~~1~~<br>1 T | ~~2~~<br>2 T | 25<br>25 | 15 |
| 1 | ~~5~~<br>5 T | 23<br>23 | 22<br>22 | 21 | 5 |
| 2 | 12<br>12 | 13 | 14 | 30 | 16 |
| 3 | 20 | 21 | 18 | 19 | 20 |
| 4 | 10 | 9 | 8 | 7 | ~~6~~ |

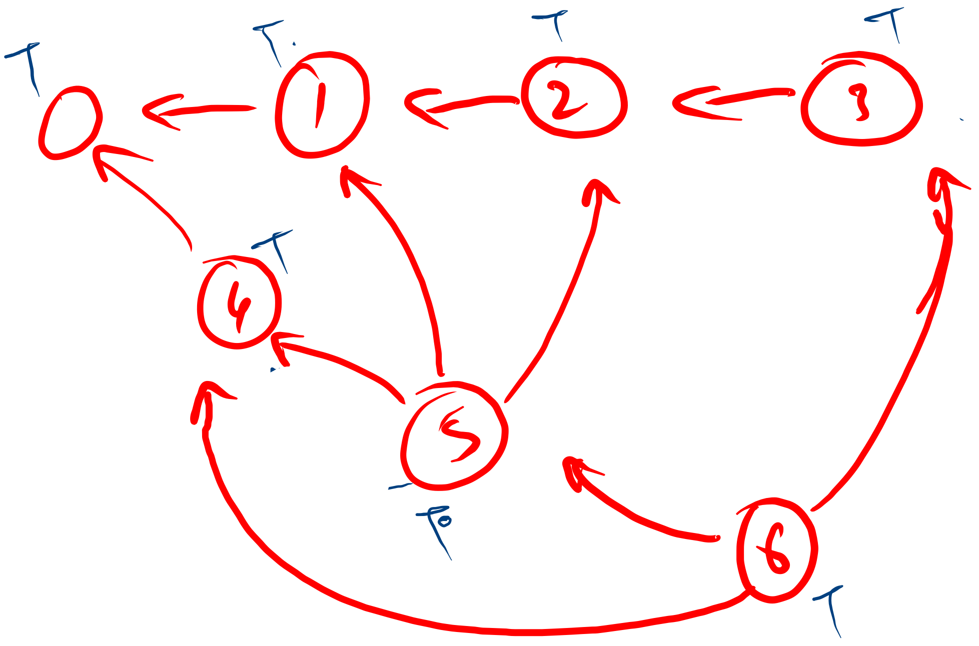


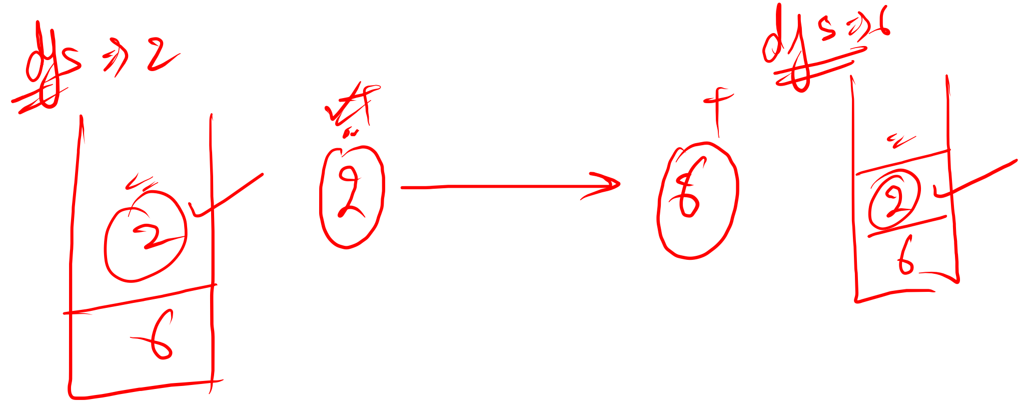
```java
static int dir[][] = {{-1,0},{0,+1},{+1,0},{0,-1}};
public int swimInWater(int[][] grid) {
    PriorityQueue<Pair> pq = new PriorityQueue<>();
    boolean[][] vis = new boolean[grid.length][grid[0].length];
    pq.add(new Pair(0,0,grid[0][0]));

    while(true){
        Pair rem = pq.remove();

        if(rem.r == grid.length-1 && rem.c == grid[0].length-1){
            return rem.maxwt;
        }

        if(vis[rem.r][rem.c] == true){
            continue;
        }

        vis[rem.r][rem.c] = true;

        for(int d = 0 ; d < 4 ; d++){
            int rdash = rem.r + dir[d][0];
            int cdash = rem.c + dir[d][1];

            if(rdash  < 0 || cdash < 0 || rdash >= grid.length || cdash >= grid[0].length || vis[rdash][cdash] == true){
                continue;
            }

            pq.add(new Pair(rdash,cdash, Math.max( rem.maxwt , grid[rdash][cdash] ) ));
        }
    }
}
```

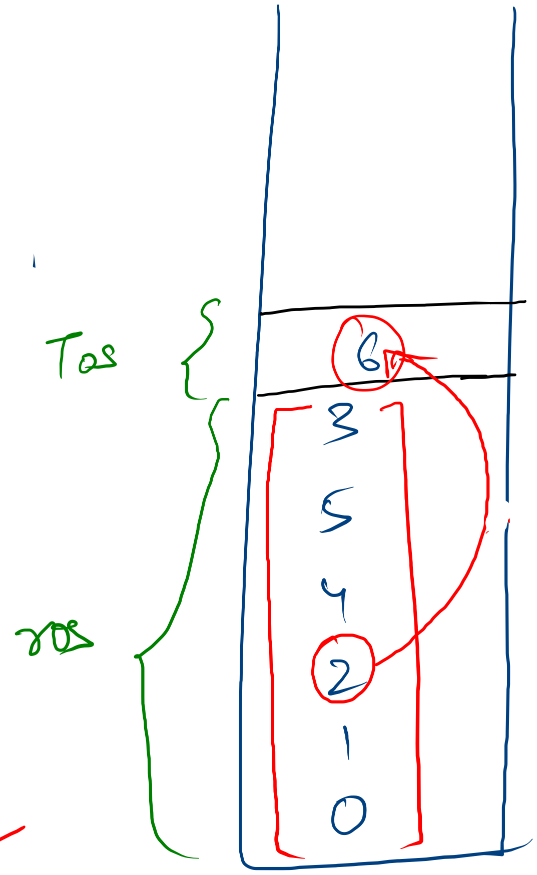
NOTES

| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 5 | 21 | 22 | 4 |
| 6 | 7 | 6 | 5 |
| 8 | 23 | 24 | 25 |
| 7 | 5 | 3 | 1 |

✓ 1



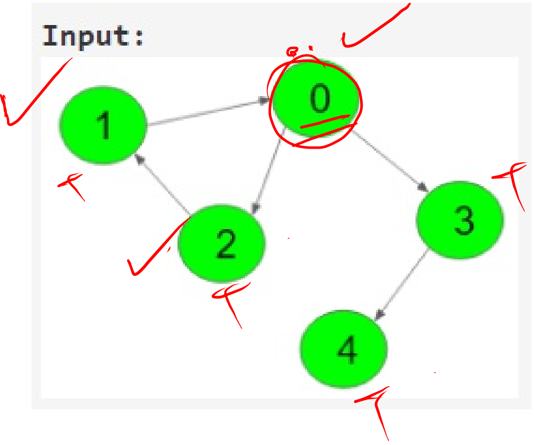


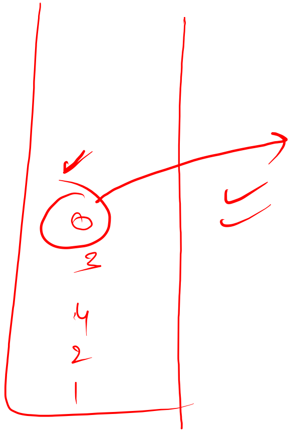


✓

| | Tos | Ros |
|---|---|---|
| C. | ✓ | ✓ |
| C. | X | X |
| C. | ✓ | X |
| | X | ✓ |

doesnot Exist

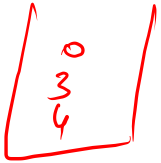


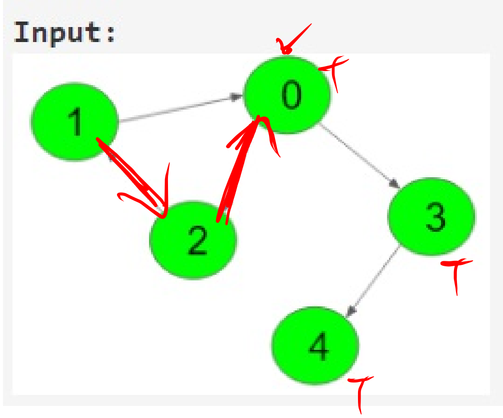


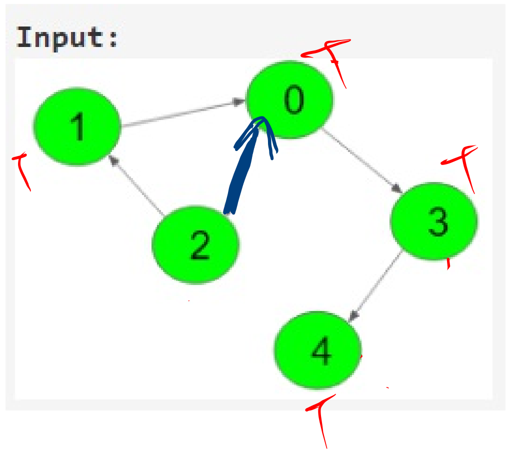
0 1 2 3 4

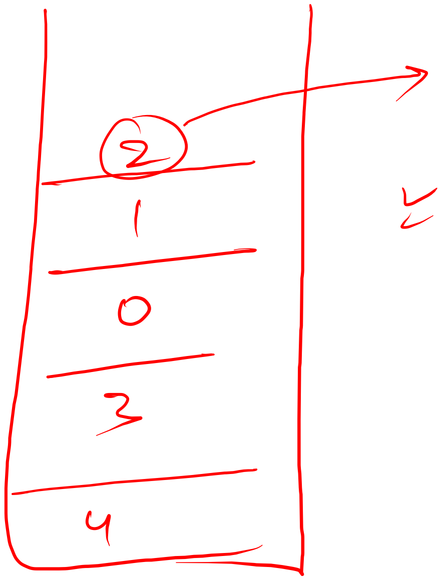
T T T

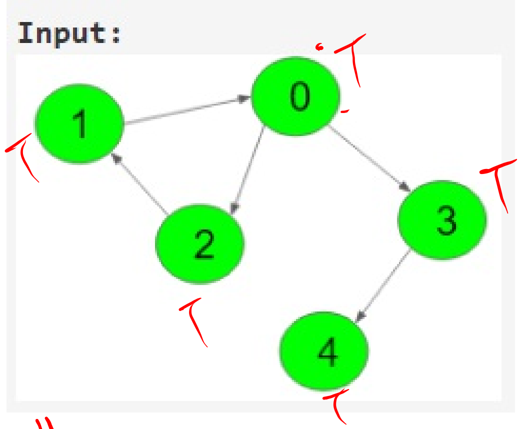
5-7 mins





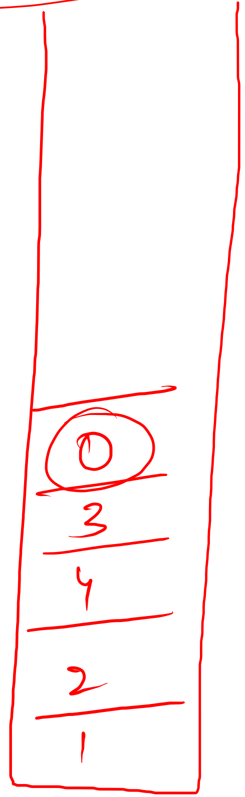


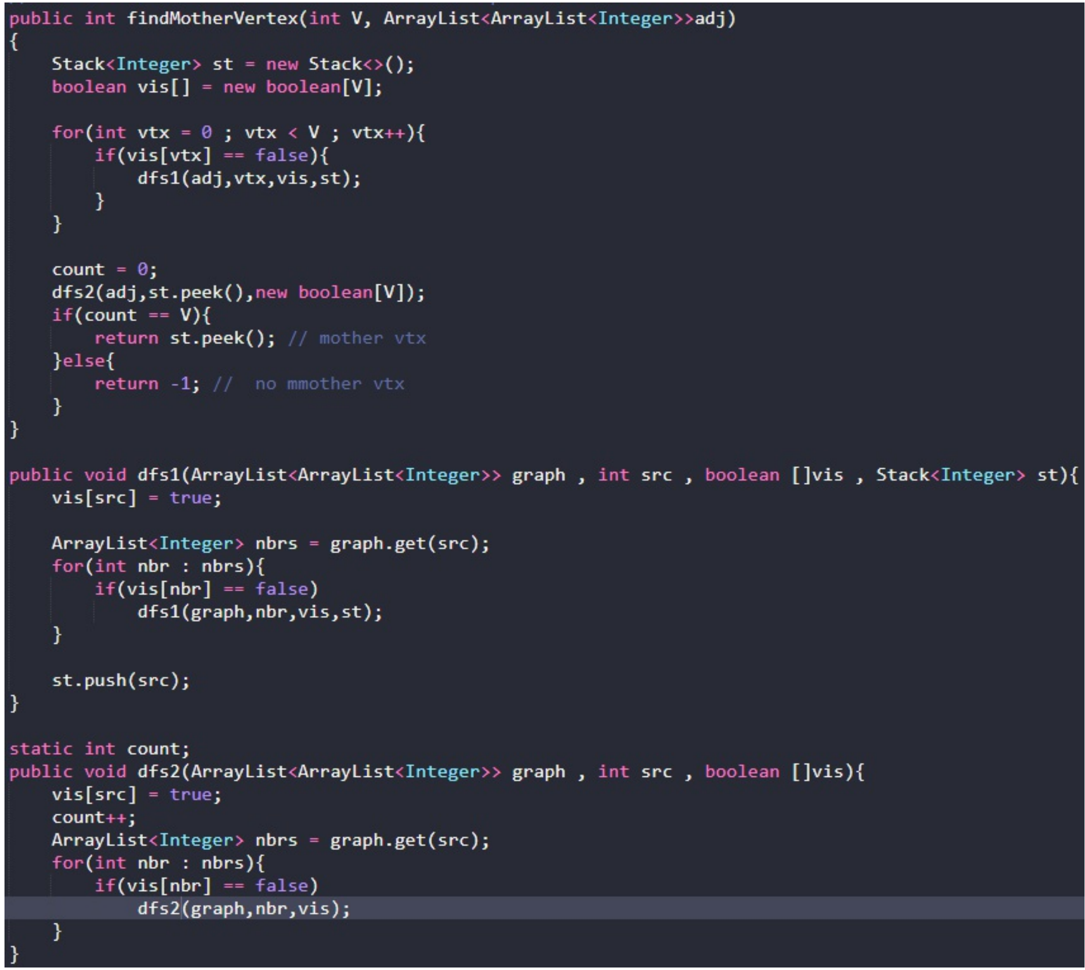




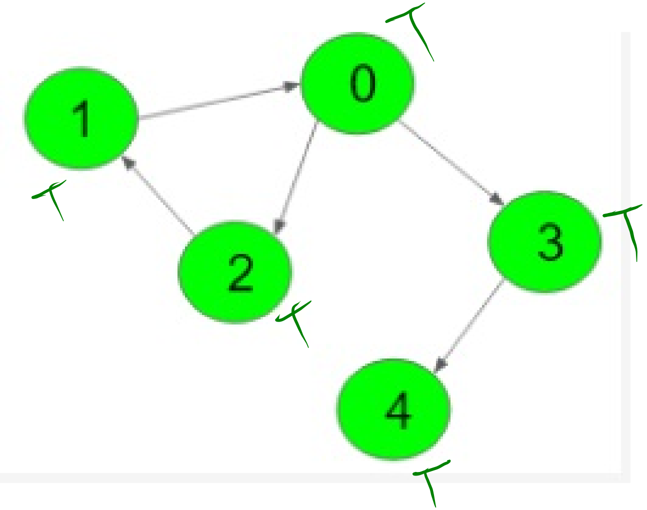
Count = ~~1~~ ~~2~~ ~~3~~ ~~4~~ 5

| 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| f | f | f | f | f |





```
public int findMotherVertex(int V, ArrayList<ArrayList<Integer>>adj)
{
    Stack<Integer> st = new Stack<>();
    boolean vis[] = new boolean[V];

    for(int vtx = 0 ; vtx < V ; vtx++){
        if(vis[vtx] == false){
            dfs1(adj,vtx,vis,st);
        }
    }

    count = 0;
    dfs2(adj,st.peek(),new boolean[V]);
    if(count == V){
        return st.peek(); // mother vtx
    }else{
        return -1; //  no mmother vtx
    }
}

public void dfs1(ArrayList<ArrayList<Integer>> graph , int src , boolean []vis , Stack<Integer> st){
    vis[src] = true;

    ArrayList<Integer> nbrs = graph.get(src);
    for(int nbr : nbrs){
        if(vis[nbr] == false)
            dfs1(graph,nbr,vis,st);
    }

    st.push(src);
}

static int count;
public void dfs2(ArrayList<ArrayList<Integer>> graph , int src , boolean []vis){
    vis[src] = true;
    count++;
    ArrayList<Integer> nbrs = graph.get(src);
    for(int nbr : nbrs){
        if(vis[nbr] == false)
            dfs2(graph,nbr,vis);
    }
}
```



2

1

0

|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 4 | 3 | 2 | 3 | 2 |
| 1 | 3 | 3 | 3 | 3 | 3 |
| 2 | 3 | 3 | 2 | 3 | 1 |
| 3 | 3 | 3 | 3 | 3 | 4 |
| 4 | 4 | 3 | 1 | 3 | 2 |

(1,1)

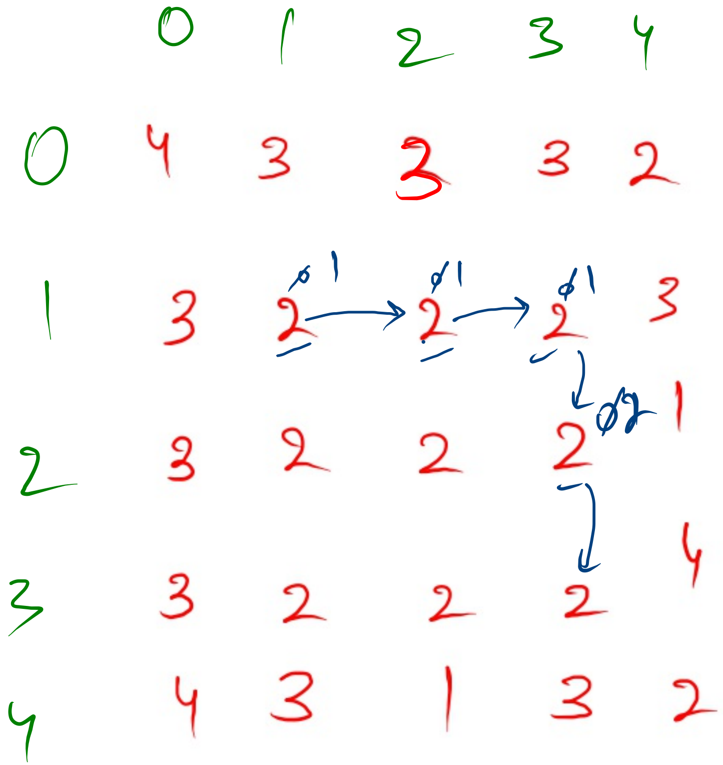
Color → 3

P. Color = 2

N

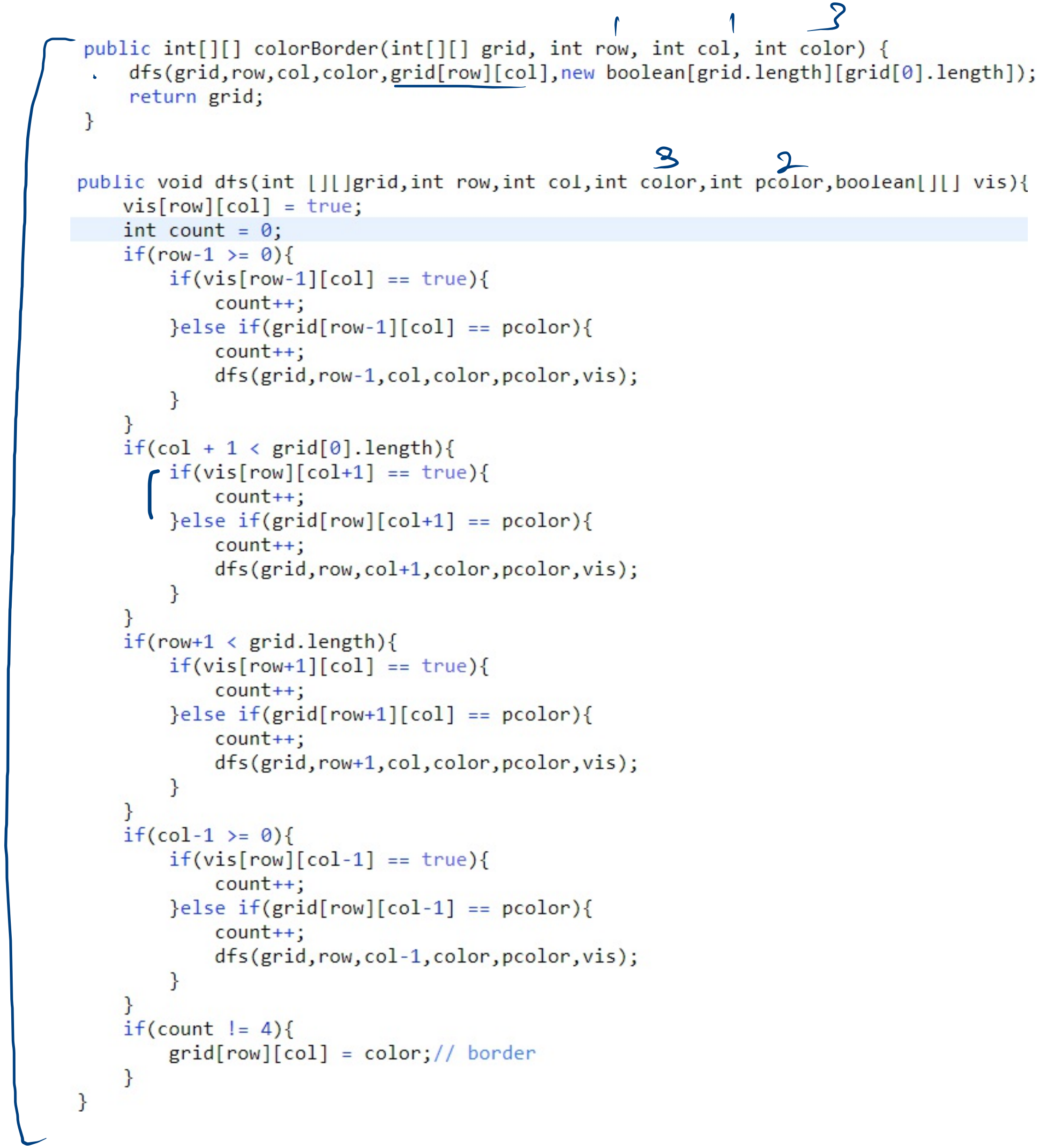
Count ++ → visited → col

Count != 4 → border



|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 4 | 3 | 3 | 3 | 2 |
| 1 | 3 | 2 | 2 | 2 | 3 |
| 2 | 3 | 2 | 2 | 2 | 1 |
| 3 | 3 | 2 | 2 | 2 | 4 |
| 4 | 4 | 3 | 1 | 3 | 2 |

(1,1) → 3



```java
public int[][] colorBorder(int[][] grid, int row, int col, int color) {
    dfs(grid,row,col,color,grid[row][col],new boolean[grid.length][grid[0].length]);
    return grid;
}

public void dfs(int [][]grid,int row,int col,int color,int pcolor,boolean[][] vis){
    vis[row][col] = true;
    int count = 0;
    if(row-1 >= 0){
        if(vis[row-1][col] == true){
            count++;
        }else if(grid[row-1][col] == pcolor){
            count++;
            dfs(grid,row-1,col,color,pcolor,vis);
        }
    }
    if(col + 1 < grid[0].length){
        if(vis[row][col+1] == true){
            count++;
        }else if(grid[row][col+1] == pcolor){
            count++;
            dfs(grid,row,col+1,color,pcolor,vis);
        }
    }
    if(row+1 < grid.length){
        if(vis[row+1][col] == true){
            count++;
        }else if(grid[row+1][col] == pcolor){
            count++;
            dfs(grid,row+1,col,color,pcolor,vis);
        }
    }
    if(col-1 >= 0){
        if(vis[row][col-1] == true){
            count++;
        }else if(grid[row][col-1] == pcolor){
            count++;
            dfs(grid,row,col-1,color,pcolor,vis);
        }
    }
    if(count != 4){
        grid[row][col] = color;// border
    }
}
```

1 BFS

H.W.

2 pl problems

Hint

Toy

Approach

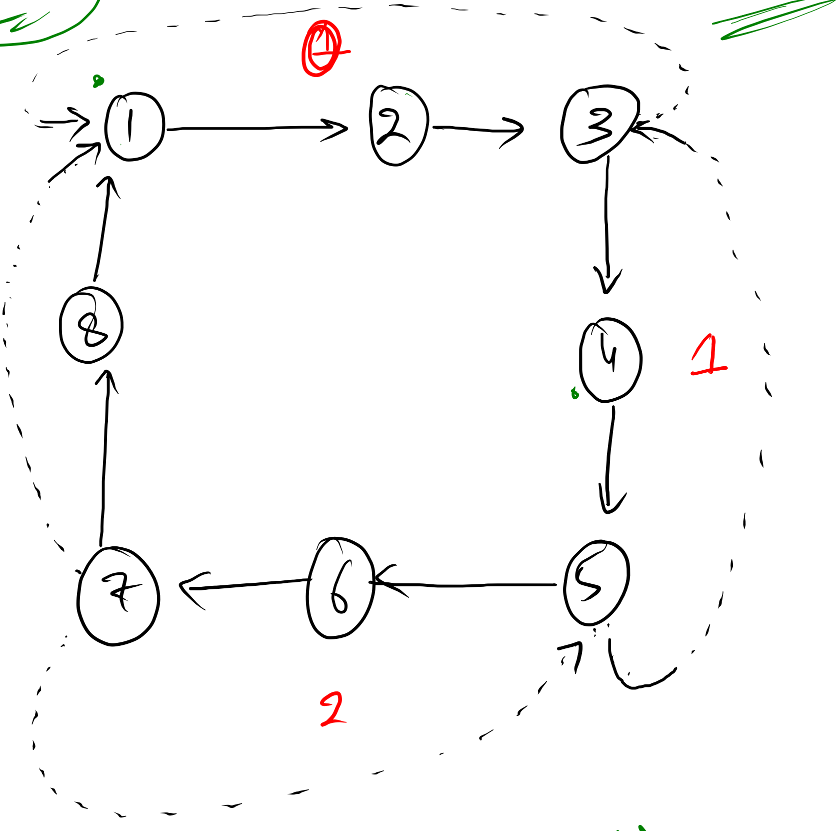
1 week

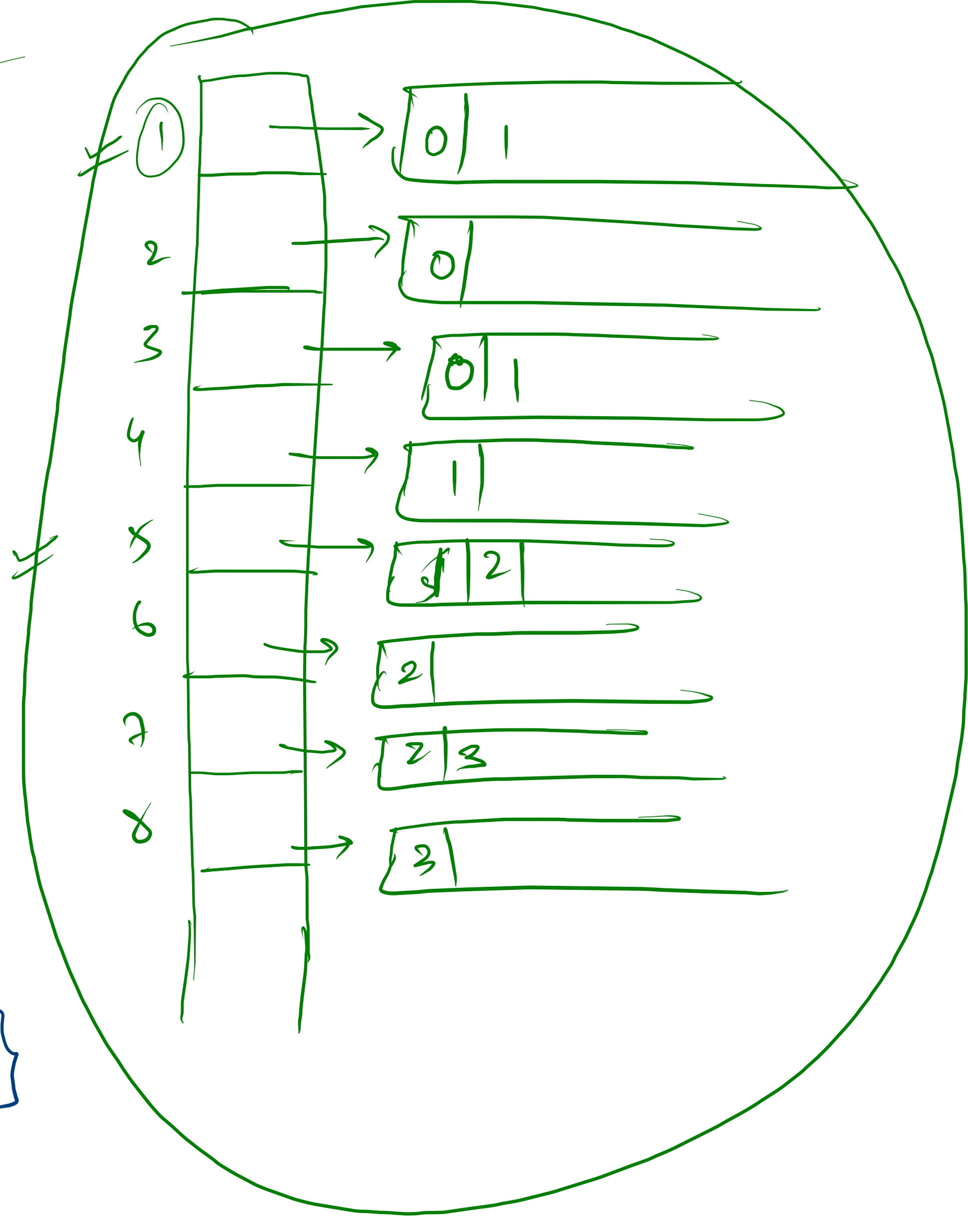
3

Bus stop

Bus no.

| | | | | |
|---|---|---|---|---|
| starting : | 1 | 2 | 1 | 1 |
| Target : | 4 | 1 | 5 | 10 |
| output : | 2 | 1 | 2 | -1 |





| | |
|---|---|
| 1 | 0, 1 |
| 2 | 0 |
| 3 | 0, 1 |
| 4 | 1 |
| 5 | 1, 2 |
| 6 | 2 |
| 7 | 2, 3 |
| 8 | 3 |

0 1 2 3

{ {1,2,3}, {3,4,5}, {5,6,7}, {7,8,1} }