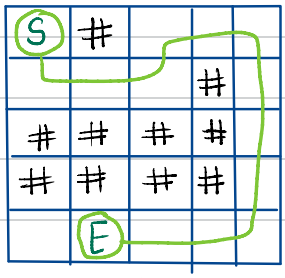
September 13, 2025

GRAPH FORMULATION 1

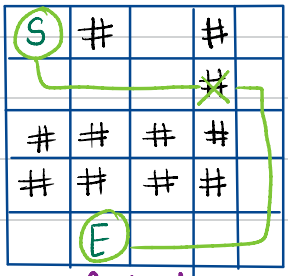
Ques 1. Given a grid with a starting point and the ending point. Find:

(a) Find the minimum no. of walls you need to break to go from point S to E.

Example:



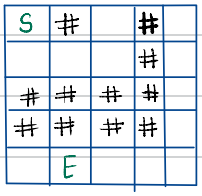
→ Ans = 0
No walls to break



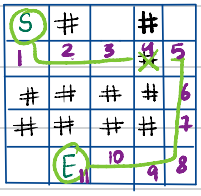
→ Ans = 1
Had to break 1 wall

(b) If you can break <= K walls, Find minimum moved needed

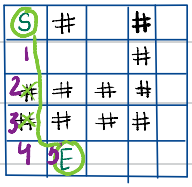
Example:



→ K = 0
Ans = 0
No way to go from S to E without breaking any wall.



→ K = 1
(How many moves needed to go from S → E if you are allowed to break just 1 wall) Ans = 11



→ K = 2
(Now you can break <= 2)
Ans = 5 moves

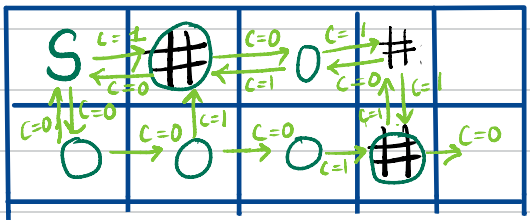
(c) Find all answer for (b) for K in the range: $1 \le K \le N.M$

I Constraints: $N, M \le 100$

SOLUTION a.

While traversing the cell to go from S → E.

1. If the cell is empty, then lets take the incurred cost = 0.
2. If the cell has a wall, then lets take the incurred cost to cross that cell = 1.



So, this is how we can build our graph and then simply use Shortest Path Algorithm.

Keep track of broken walls as when tracing these walls again will make the cost = 0 this time.

```
void bfs(state st){
    vis = vector<vector<int>>(n, vector<int>(m,0));
    dist = vector<vector<int>>(n, vector<int>(m,INF));

    deque<state> dq;
    // Add source
    dq.push_back(st);
    dist[st.F][st.S]=0;
    // Start BFS
    while(!q.empty()){
        state cur = dq.front(); dq.pop_front();
        if(vis[cur.F][cur.S])continue;

        // Explore
        vis[cur.F][cur.S]=1;
```

dq

Changes in BFS code

```
        for(state v:valid_neighbours(cur)){
            int w = 0;
            if(arr[v.F][v.S]=='#')w=1;

            if(!vis[v.F][v.S] && dist[v.F][v.S] > dist[cur.F][cur.S]+w){
                dist[v.F][v.S] = dist[cur.F][cur.S]+w;
                if(w==0){
                    dq.push_front(v);
                }else{
                    dq.push_back(v);
                }
            }
        }
    }
```

Input

```
5 5
S#.#.
..##.
####.
####.
.E...
```

Output

```
YES
1
```

→ For printing the path from S to E

```
void bfs(state st){
    vis = vector<vector<int>>(n, vector<int>(m,0));
    dist = vector<vector<int>>(n, vector<int>(m,INF));
    par = vector<vector<state>>(n, vector<state>(m,{-1,-1}));

    deque<state> dq;
    // Add source
    dq.push_back(st);
    dist[st.F][st.S]=0;
    // Start BFS
    while(!dq.empty()){
        state cur = dq.front(); dq.pop_front();
        if(vis[cur.F][cur.S])continue;
        // Explore
        vis[cur.F][cur.S]=1;
        for(state v:valid_neighbours(cur)){
            int w = 0;
            if(arr[v.F][v.S]=='#')w=1;

            if(!vis[v.F][v.S] && dist[v.F][v.S] > dist[cur.F][cur.S]+w){
                dist[v.F][v.S] = dist[cur.F][cur.S]+w;
                par[v.F][v.S] = cur;
                if(w==0){
                    dq.push_front(v);
                }else{
                    dq.push_back(v);
                }
            }
        }
    }
}
```

Input

```
5 5
S#...
..##.
####.
####.
.E...
```

Output

```
YES
1
RRRRDDDDLLL
```

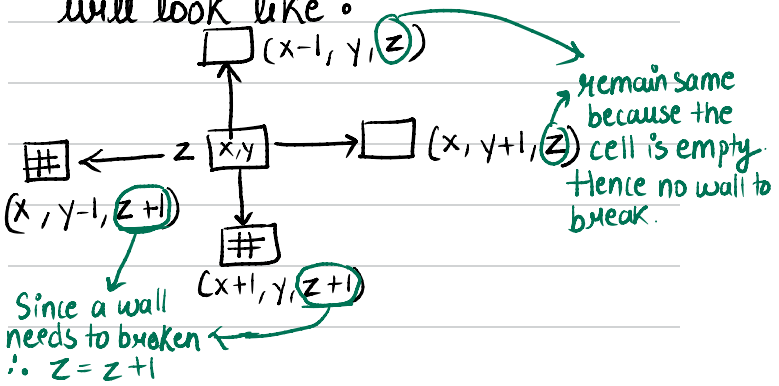
SOLUTION b. $O(N*M*K)$

Now every step cost either 1 or 0. So, our goal becomes to minimise the number of moves.

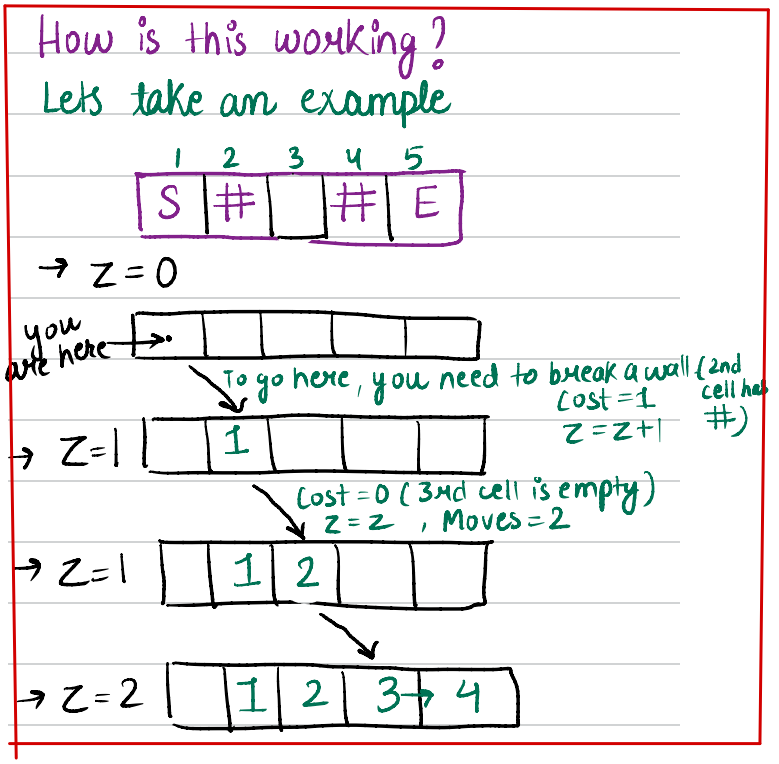
No. of walls broken $<= K$ → Node (V)

Node is now (x, y, z). It will store → min cost to (x, y) with z wall broken till that point.

(x, y, z) now means that you are at point (x, y) with z broken walls. The configuration will look like :



How is this working?
Lets take an example



Starting configuration in the Graph :

[(St.F)(St.S)(0)]

Code link :

https://azpremiumb11.slack.com/archives/C09026YB5ME/p1757768879733089

AlgoZenith

* Budget Travelling

You want to visit the country of Wonderland. There are N cities in the country. Not all cities are connected by roads, but you know which cities are connected. You landed in city A, and you want to visit city B. You already booked your car, but it doesn't have any petrol. The capacity of the tank of the car is C. You know the **per liter cost** of petrol in each city, and you also have the map of the country (i.e., you know the length of the road between two cities).

To travel one unit of distance, you need one liter of petrol.

Your task is to find the **minimum cost** to travel from city A to city B.

Petrol is a restriction here.

(z, petrol-d) ← 0 (city, petrol) ← configuration we will use

↓ p(city)

(city, petrol+1)

Use Dijkstra here.

SOLUTION c.

Max. value of K = N*M

Is N*M truly possible?

No, As for even the farthest point, we can cover with (N+M)

Use K constraint till N+M.

```
void solve(){
    cin>>n>>m;
    k=n+m;
    arr.resize(n);
    for(int i=0;i<n;i++){
        cin >> arr[i];
    }
    pair<int,int> st,en;
    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            if(arr[i][j]=='S')st={i,j};
            else if(arr[i][j]=='E')en={i,j};
        }
    }
    bfs({st,0});
    int ans = INF;
    for(int i=0;i<=n+m;i++){
        // cout<<i<<" "<<dist[en.F][en.S][i]<<en
        ans=min(ans,dist[en.F][en.S][i]);
        cout<<ans<<endl;
    }
    for(int i=n+m+1;i<=n*m;i++){
        cout<<ans<<endl;
    }
    // cout<<ans<<endl;
```

Full Code link :

https://azpremiumb11.slack.com/archives/C09026YB5ME/p1757771676672799