BFS → shortest path

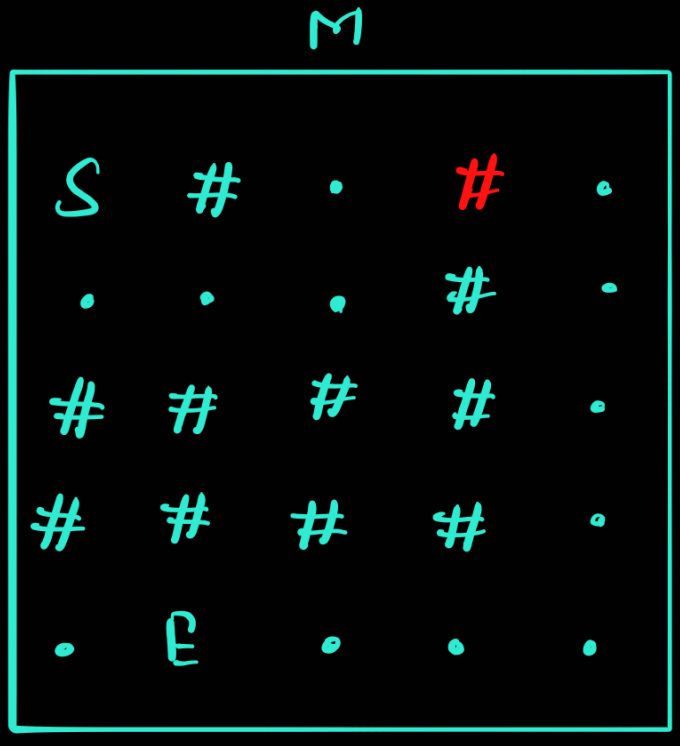
Many more Algo

SP → Algorithm (fixed, easy)

SP → Modelling

Given a problem statement, where is the graph? How to form a graph with least no. of nodes and edges efficiently?

Q

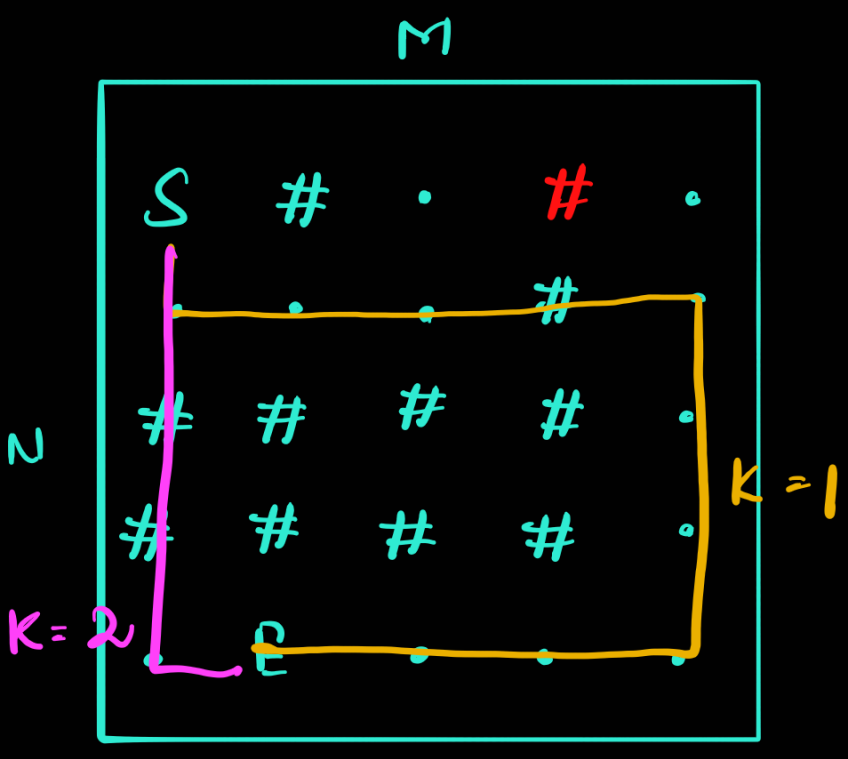


2D grid

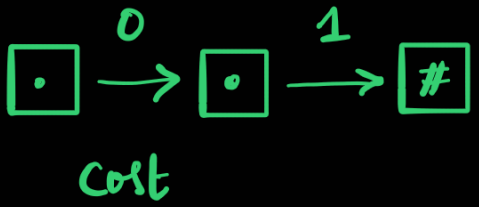
① Min walls to break to go from S → E

② If You can break atmost k walls find the SP

③ Solve ② for $1 \le k \le N \cdot M$



① Any path with Min walls broken



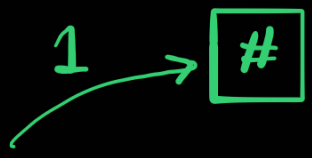
AlgoZenith

Cost = 2

path with cost 0 or 1

shortest path will automatically find min cost.

Min (# of walls)



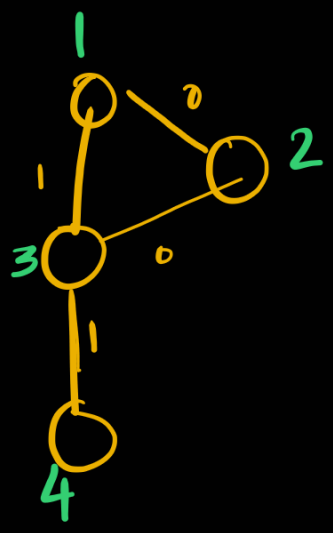
whatever we want to minimize, that determines edge cost

AlgoZenith

Graph → (V, E)
↳ determined by what we want to minimize

. → 0
→ 1
} 0/1 BFS Algo

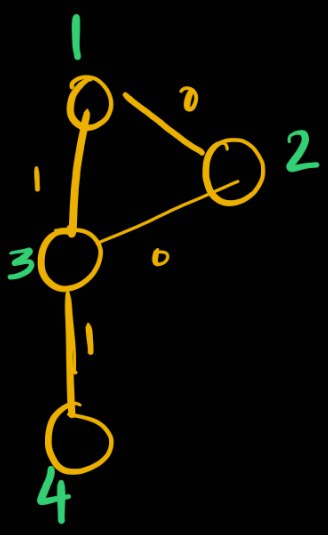
if you find SP, we get min walls broken

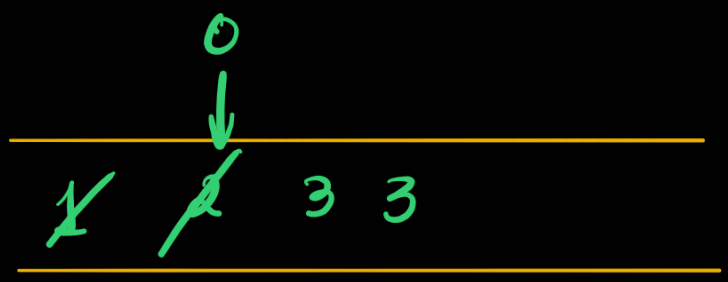


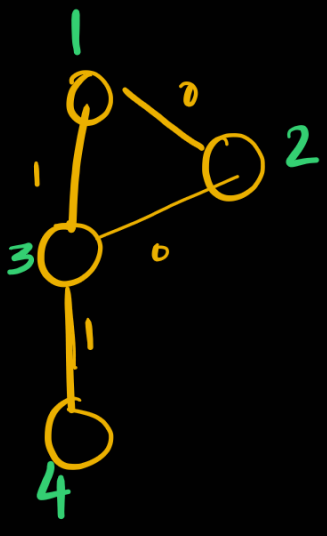
✗

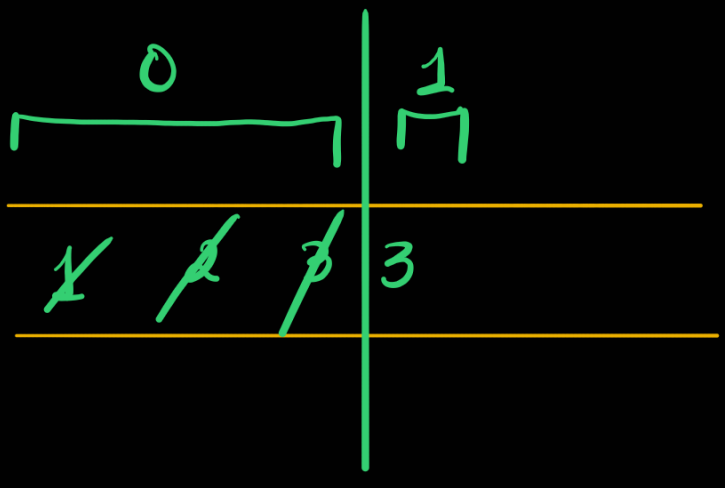
---

which nbr to process first?









```cpp
vector<pair<state,int>> neighbour(state s){
    vector<pair<state,int>> ans;
    for(int k=0;k<4;k++){
        state temp = make_pair(s.F+dx[k],s.S+dy[k]);
        if(inside(temp.F,temp.S)){
            if(arr[temp.F][temp.S]=='#')
                ans.push_back({temp,1});
            else
                ans.push_back({temp,0})
        }
    }
    return ans;
}
```

```
while(!q.empty()){
    state cur = q.front();
    q.pop();
        front
    if(vis[cur.F][cur.S])continue;
    vis[cur.F][cur.S]=1;
    // process the node
    for(auto [v,c]:neighbour(cur)){
        // relaxing edge.
        if(dist[v.F][v.S] > dist[cur.F][cur.S]+c){
            dist[v.F][v.S] = dist[cur.F][cur.S]+c;
            if(c==1)
                q.push_back(v);
            else
                q.push_front(v);

        }
    }
}
```

deque< state > q;

(2)

G(V, E)

↑

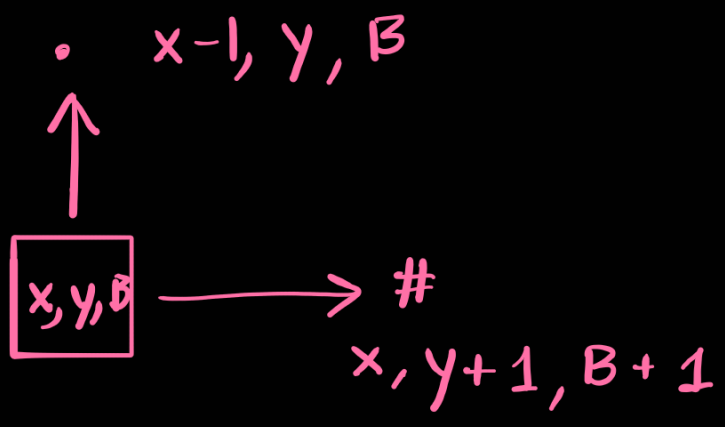
Restriction (# walls_broken ≤ K)

Goes to vertex of graph

Build a new graph

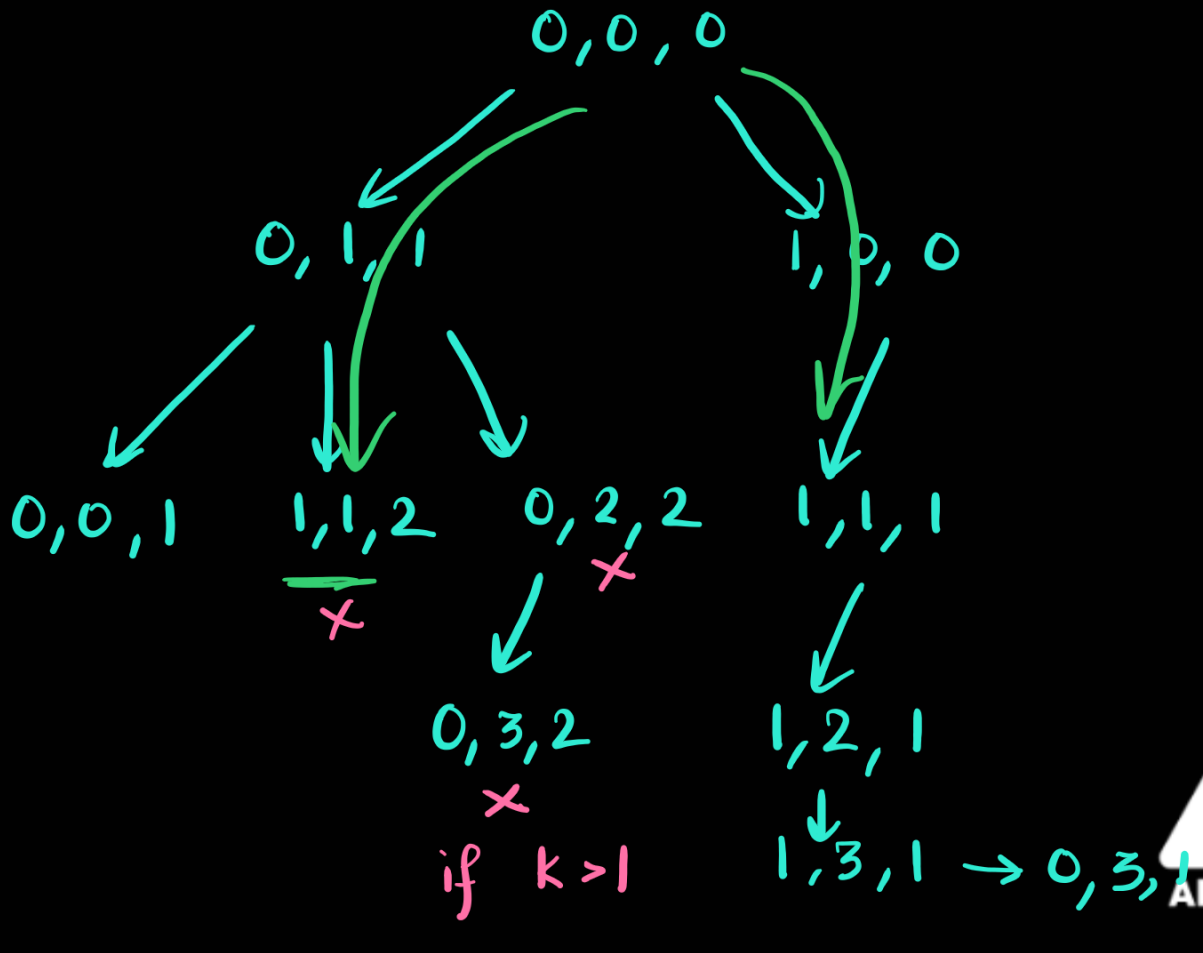
AlgoZenith

Node = (x, y, walls broken)

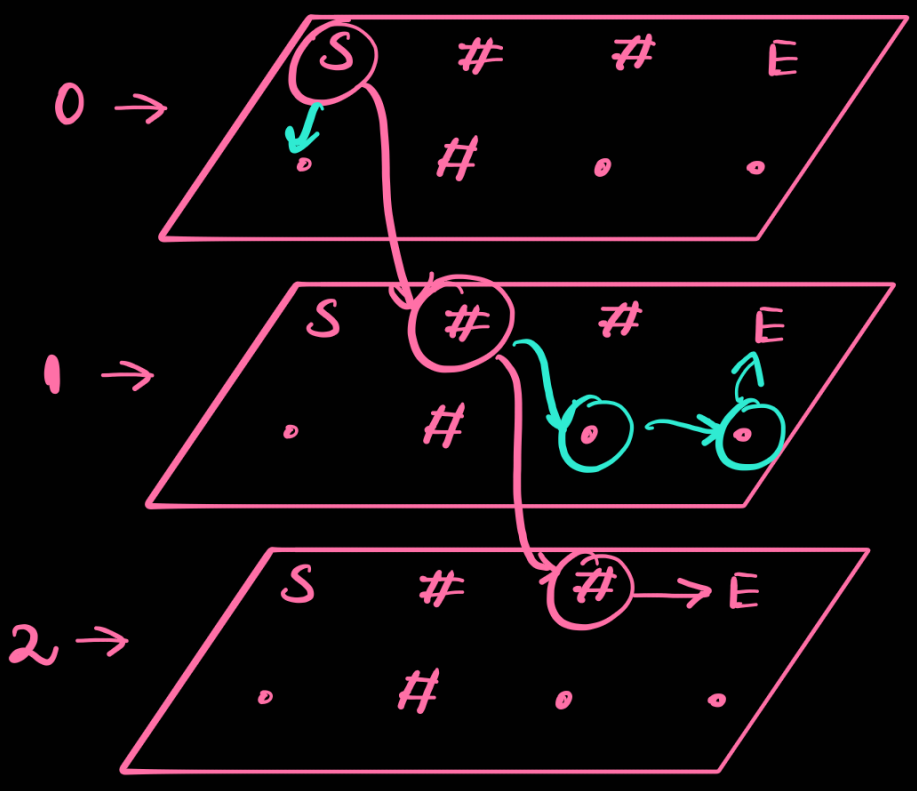


New graph to do shortest path on

S # # E
. # . .



find SP from (0,0) to (0,3)




```
using state = pair<pair<int,int>,int>;
```

```
vector<state> neighbour(state s){
    vector<state> ans;
    for(int dir=0;dir<4;dir++){
        pair<int,int> temp = make_pair(s.F.F+dx[dir],s.F.S+dy[dir]);
        if(inside(temp.F,temp.S)){
            if(arr[temp.F][temp.S]=='#' && s.S<k)
                ans.push_back({temp,s.S+1});
            else
                ans.push_back({temp,s.S});
        }
    }
    return ans;
}
```

AlgoZenith

```
vector<vector<vector<int>>> vis, dist;
```

```
vis = vector<vector<vector<int>>>(n,vector<vector<int>>(m,
vector<int>(k+1,0)));
dist = vector<vector<vector<int>>>(n,vector<vector<int>>(m,
vector<int>(k+1,INF)));
```

```
queue<state> q;


dist[st.F.F][st.F.S][st.S] = 0;
```

```
q.push(st);

while(!q.empty()){
    state cur = q.front();
    q.pop();

    if(vis[cur.F.F][cur.F.S][cur.S])continue;
    vis[cur.F.F][cur.F.S][cur.S]=1;
```

```
// process the node
for(auto v:neighbour(cur)){
    // relaxing edge.
    if(dist[v.F.F][v.F.S][v.S] > dist[cur.F.F][cur.F.S][cur.S]
    +c){
        dist[v.F.F][v.F.S][v.S] = dist[cur.F.F][cur.F.S][cur.S]
        +1;
        q.push(v);
    }
}
```

```
bfs({st,0});
int minpath = INF;
for(int i=0;i<=k;i++){
    minpath = min(minpath,dist[en.F][en.S][i]);
}
cout<<minpath<<endl;
```

S
#
#
E

Max walls from S → E is Manhattan distance from 5 onwards same answer

$k = N + M - 1$

Comp → $(N * M)(N + M)$