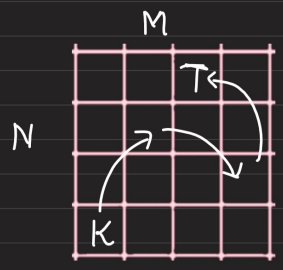
Shortest Path Problems

Q1



N X M

K - knight

T - target

$(N, M \leq 20)$

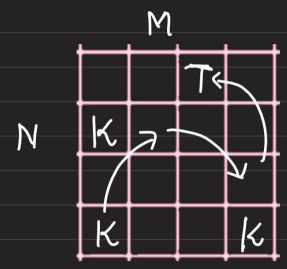
Find minimum no. of moves to move knight from K to T.

sol^n: Single source shortest path.

↓

BFS } ✓

Q2



N X M

K - knight

T - target

$(N, M \leq 20)$

Find minimum no. of moves to move knight from any knight to T.

sol^n: Multi – source shortest path.

↓

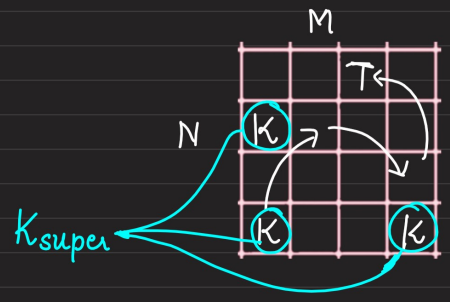
BFS } ✓

↓

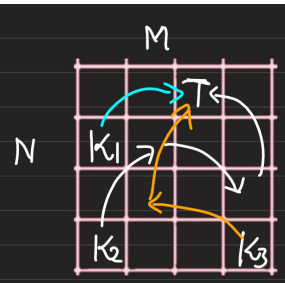
on K_{super}

dis[T] - 1

distance of T from closest K.



Q3.
**



N X M
K- knight
T- target

$(N, M \leq 20)$

Find sum of minimum no. of moves to reach T from each knight.

Soln:
$|K_1| = 1$
$|K_2| = 3$
$|K_3| = 2$
} Ans = 1+2+3 = 6

Naive approach:

do BFS for each knight.

TC: $O(|K| * NM) \equiv O(N^2 M^2)$

Best approach:

Do BFS from target.

↳ Ans = $\sum dis(K_i)$

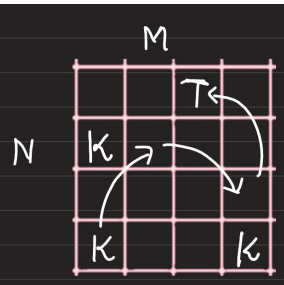
Since, going from $A_i \rightarrow B$, go from $B \rightarrow A_i$

Reversal Approach.

Note: the difference b/w MSSP & reversal approach.

Single source multiple targets → BFS
Single target multiple source → Reversal BFS

Q4



N X M

K - knight

T - target

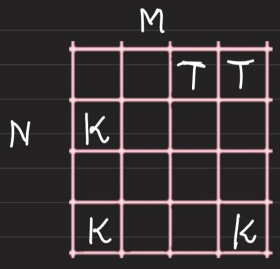
$(N, M \leq 20)$

Find min(any K, T).

Soln: App 1: Ksuper ↙ Easier

App 2: BFS(T) & take min across all the Ks.

Q5



N X M

K - knight

T - target

$(N, M \leq 20)$

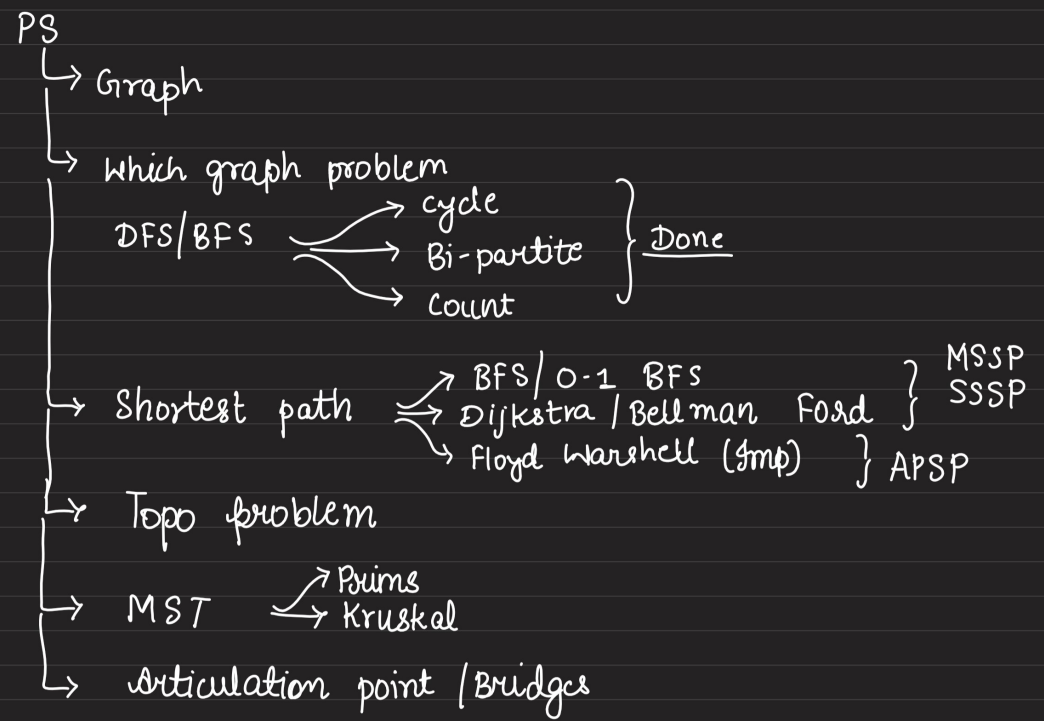
Find minimum no. of moves to move knight from any K to any T.

Soln: Ksuper then take minimum across all the Ts.

↳ most optimal.

Simply push all the K's into Queue.

Formulation :-



- PS (Graph) → Algorithm (Black box) → output

- Modelling Graph ⇒ Modelling state.

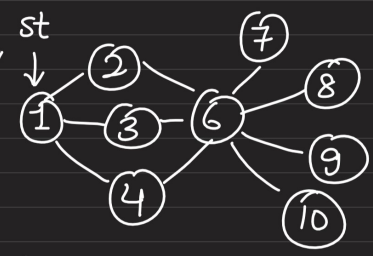
Shortest Path

↳ unweighted (BFS : O(V+E)) we use visiting queue.

BFS

```
queue q; q.push(st);
vis[st] = 1;
while (!q.empty())
{
    curr = q.front(); q.pop();
    for (v: g[curr])
    {
        if (!vis[v]){
            q.push(v);
        } → vis[v] = 1; }
    }
}
```

one problem with this code

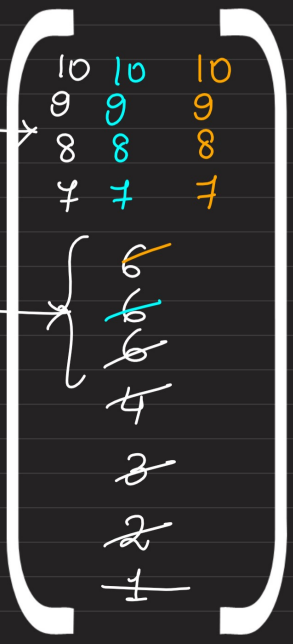


Dry run

visited:

1 ✓ 2 ✓ 3 ✓ 4 ✓ 6 ✓ 7 ✓ 8 ✓ 9 ✓ 10 ✓

for each 6



6 will be added 3 times

Hence, this code is wrong.

↓ correction

- mark visited after pushing it.
  q.push(v); vis[v] = 1;

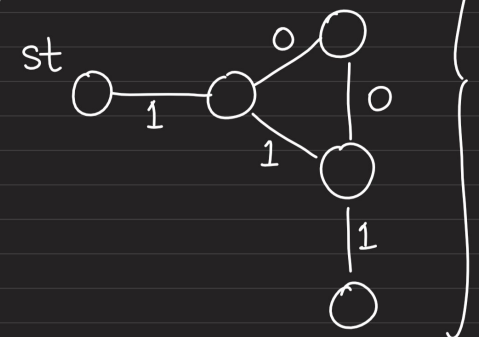
SIMPLE BFS (SSSP)

```
vector<int> dist;   dist.assign(n+1, 1e9);
void BFS(){
   queue q; q.push(st);
   vis[st] = 1;
   while(!q.empty())
   {
      curr = q.front(); q.pop();
      for(v: g[curr])
      {
         if(dist[v] > dist[curr]+1) // relaxing.
         {
            dist[v] = dist[curr] +1; q.push(v);
         }
      }
   }
```

Common Problem: } V. Imp
↳ Knight's problem (already done)

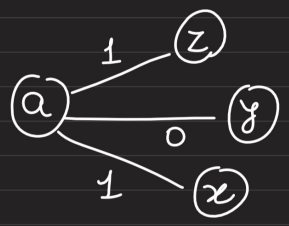
0-1 BFS

weight as 0/1.

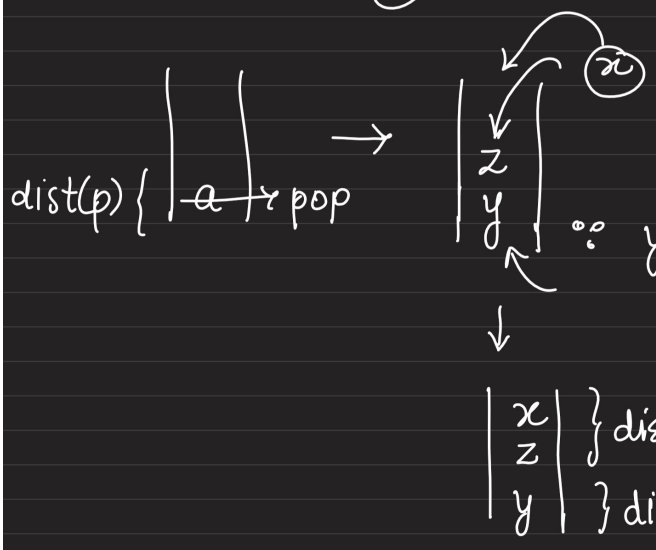


Instead of queue, we will use dequeue.

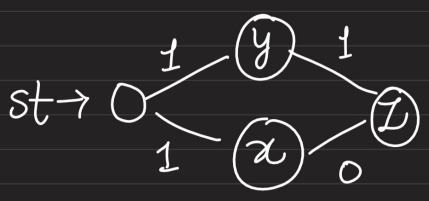
BFS explores layer wise :

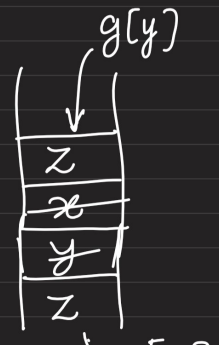


'y' distance from st same as 'a' distance.



```
deque dq;
dist[st] = 0
while (!dq.empty())
{
    x = dq.front(); dq.pop_front();
    for (auto v : g[x])
    {
        if (dist[v] > dist[x] + c)
        {
            dist[v] = dist[x] + c;
            if (c==0)
            {
                dq.push_front(v);
            }
            else
            {
                dq.push_back(v);
            }
        }
    }
}
```



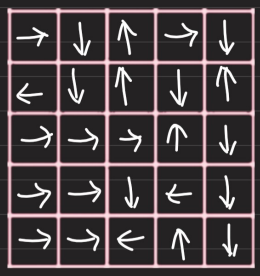


dist	st	x	y	z
	0	1	1	2 ↓ 1

z added twice

Any element can go into the deque atmax 2 time,
so, TC: $O(V+E)$ $t_{max} = 2(V+E)$

eg 1:



start (x_s, y_s)
end (x_d, y_d)

(UBER)

P1) If you follow the directions, can you go outside the board.

P2) What is the minimum cost of go from start to end.

P1) → use DFS to cheak rechability of all the other nodes from st, then check if there is an arrow towards outside the board.

P2) → If in the direction, then cost = 0
else cost = 1.

```cpp
using lli = long long int;
using ii = pair<int,int>;
#define F first
#define S second
const lli mod = 1000000007;

int n,m;
vector<vector<char>> grid;
vector<vector<int>> dist;

unordered_map<char,int> dir = {{'D',0},{'U',1},{'R',2},{'L',3}};
// D, U, R, L
int dx[] = {1,-1,0,0};
int dy[] = {0,0,1,-1};

bool check_inside(int x, int y)
{
    if(x>=0 && y>=0 && x<n && y<m) return 1;
    else return 0;
}
```

```cpp
void solve()
{
    cin>>n>>m;
    grid.assign(n,vector<char>(m,'.'));
    dist.assign(n,vector<int>(m,mod));

    for(int i=0;i<n;i++){
        for(int j=0;j<m;j++){
            cin>>grid[i][j]; //R, L,
        }
    }
    ii st,end;
    cin>>st.F>>st.S;
    cin>>end.F>>end.S;

    bfs01(st);
    cout<<dist[end.F][end.S]<<'\n';
}

int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);cout.tie(0);

    solve();
}
```

```cpp
void bfs01(ii st)
{
    deque<ii> q;
    q.push_back(st);
    dist[st.F][st.S] = 0;

    while(!q.empty())
    {
        ii xx = q.front();
        int direction = dir[grid[xx.F][xx.S]];

        q.pop_front();
        for(int k=0;k<4;k++){ //exploring neighbours

            ii neigh = {xx.F + dx[k], xx.S + dy[k]};
            int weight = !(k==direction);

            if(check_inside(xx.F + dx[k], xx.S + dy[k]))
            {
                if(dist[neigh.F][neigh.S]>dist[xx.F][xx.S]+weight)
                {
                    dist[neigh.F][neigh.S]=dist[xx.F][xx.S]+weight;
                    if(weight==1){
                        q.push_back(neigh);
                    }else{
                        q.push_front(neigh);
                    }
                }
            }
        }
    }
}
```

eg 2:

1	1					2	2	
	1					2	2	2
#		1				2	2	
	#		1			2	2	3
#		#				3	3	3
	#		#			3		
							3	

1, 2, 3 components

Find minimum no. of bridges to connect all 3 components.

Soln: Here, 2 & 3 are already connected.

Ans = 2 Bridges

Observation :-

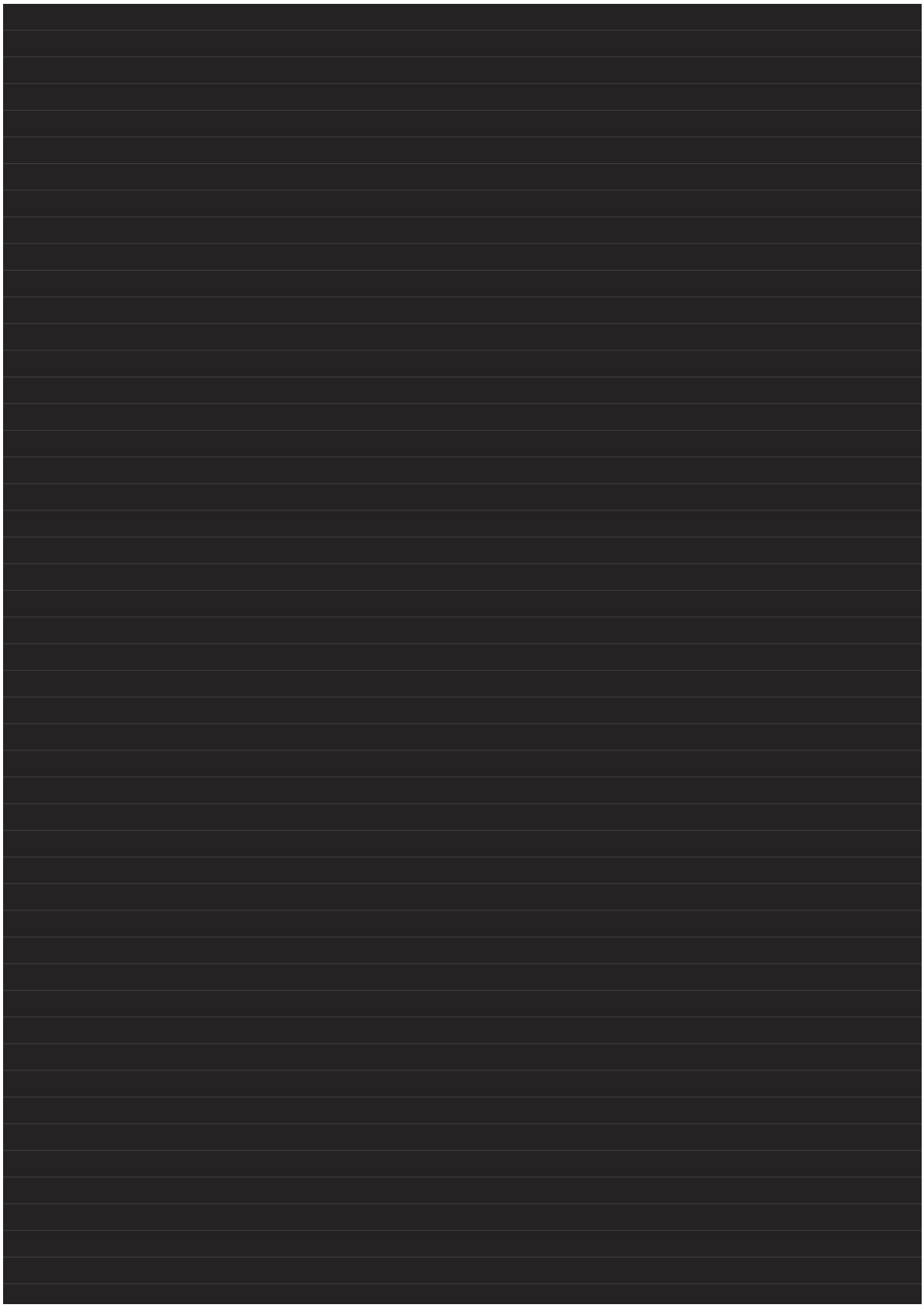
There is always a (X,Y) such that all the 3 components are closest to (X,Y)

1	1					2	2	
	1					2	2	2
#		1				2	2	
	#		1	B	B	2	2	3
#		#				3	3	3
	#		#			3		
							3	

We will build bridges from (X,Y) to comp 1, 2, 3.

eg:

1	1	B	B	B	2	2	2	← (X,Y) somewhere in the comp of 2.
1	1					2		
						B		
						B		
					3	3		
						3		



Dijkstra

Weight [0, ∞)

-ve cycle → wrong answer

-ve edge → TLE

Now, we will use priority queue to maintain layers of nodes.

priority_queue <pair> q; } min, priority queue

q.push(0, st); dist[st] = 0;

<int, vector<int>, greater<int>>

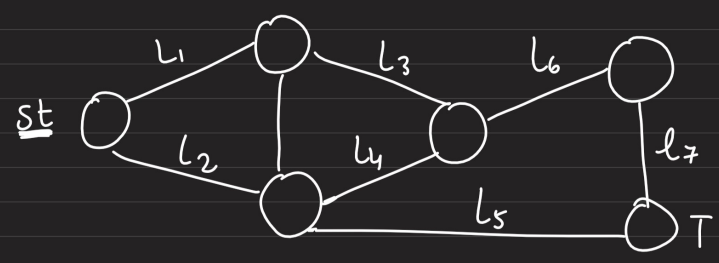
min priority_queue

```
while (!q.empty())
{
    x = q.top();
    q.pop();
    for (v : g[x])
    {
        if (vis[v] == 1) continue;
        vis[v] = 1;
        if (dist[v] > dist[x] + c)
        {
            dist[v] = dist[x] + c;
            q.push(d[v], v);
            vis[v] = 1;   // wrong
        }
    }
}
```

Problem 1.

V nodes, E edges

initial total fuel = 0



{C = capacity of fuel tank}

- fuel used to travel b/w them = length of path b/w them.
- Each city has fuel tank, where you will have to pay P[i] rupees to recharge of fuel.

Find minimum amount of money to reach to T from st.

$|V| \leq 1000$ $\quad C \leq 100$

$|E| \leq 1000$

} constraints

Soln: Dijkstra ?

Formulations

We will create a new graph

node fuel left

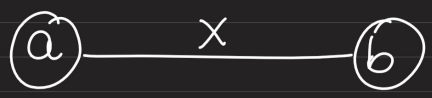
State

Initial state

Initially 0 fuel

P[st]

Transitions :



\# price used

$(a, fuel) \xrightarrow{0} (b, fuel - x)$

↳ possible only if $fuel \geq x$

$P[s]$ } new graph with cV vertices and cE edges.

$(s, fuel + 1)$

if $(fuel + 1 \leq c)$

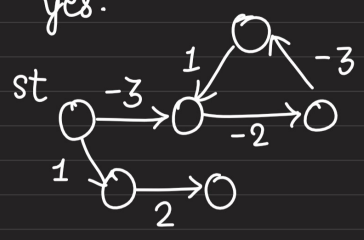
Time complexity :

$O(c * (E + V))$

Bellman Ford

Q) How to find if graph has -ve cycle or not?

soln: a) Run Dijkstra, if it runs in infinite loop, then yes.



Algorithm : G(V, E)

smallest distance to A

for (|V| - 1) times
relax all edges
→ all nodes with fixed shortest distance will be marked.

relax one more time.

↳ if distance of any node is still decreasing, then graph contains : -ve cycle.

↳ all nodes whose value will change are part of negative cycle.

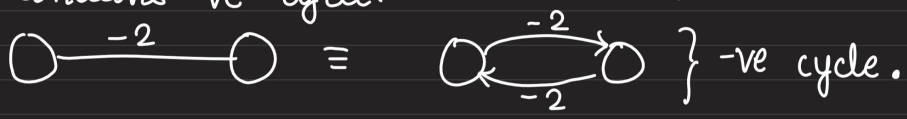
Time complexity : O(V·E)

Bellman Ford Algorithm

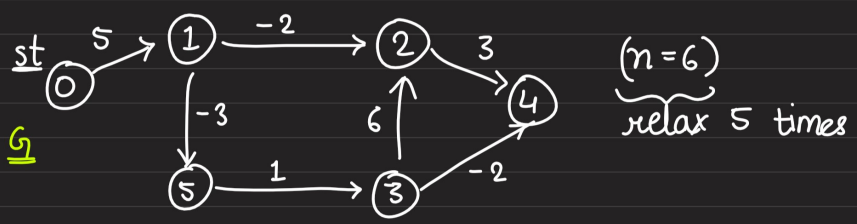
If there is a -ve cycle, then we cannot find the shortest path as nodes of -ve cycle will have chepest path as $-\infty$.

Bellman Ford helps us to find -ve cycle. If -ve cycle is found, then finding ASSP is irrelevant.

- If any undirected graph contains any -ve edge, it means it contains -ve cycle.



- Using Bell-man ford for directed - graph makes more sense.



i) Relax all the edges exactly N-1 times.

ii) Relaxation

```
if (dist[u] + wt < dist[v])    (v being neighbour of u)
    dist[v] = dist[u] + wt;
```

Initial

dist[]:

0	1	2	3	4	5		
0	∞	∞	∞	∞	∞		

Initial

	0	1	2	3	4	5
dist[]:	0	∞	∞	∞	∞	∞

(u, v)	wt
(3, 2)	6
(5, 3)	1
(0, 1)	5
(1, 5)	-3
(1, 2)	-2
(3, 4)	-2
(2, 4)	3

edge list

1st time

relax these edges 5 times ←

- dist[3] + 6 < dist[2] (∞) (∞) ×
- dist[5] + 1 < dist[3] (∞) (∞) ×
- dist[0] + 5 < dist[1] ✓ (0) (5) (∞)

∴ dist[1] = 5.

- dist[1] + (-3) < dist[5] ✓ (5) (∞)

dist[5] = 2

- dist[1] + (-2) < dist[2] ✓ (5) (∞)

dist[2] = 3

- dist[3] + (-2) < dist[4] × (∞) (∞)
- dist[2] + 3 < dist[4] ✓ (3) (∞)

dist[4] = 6

After 1st it:

	0	1	2	3	4	5
dist[]:	0	5	3	∞	6	2

Repeat this process 4 more times.

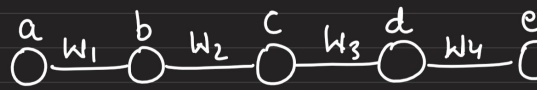
Final after 5 iterations :

	0	1	2	3	4	5
dist[] :	0	5	3	3	1	2

} Shortest distance psbl.

If we relax one more time and dist[] array reduces, then graph has -ve cycle.

Why N-1 loops ?



(N-5)

longest path b/w any 2 nodes is atmax N-1.

It 1

de
cd
bc
ab
} edge list

∞ ∞
dist[d] + w4 < dist[e]
↳ no updation

Similarly for cd, bc → no updation.

dist[a] + w1 ≤ dist[b]
∵ (dist[b] = w1) ✓

It 2
↳ In this, bc will also be relaxed.

It 3
↳ In this, cd will also be relaxed

It 4
↳ de, cd, bc, ab all relaxed.

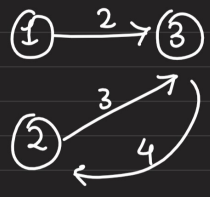
That's why (N-1) times relaxation is important.

All Pair Shortest Path

If no negative cycle. We can run Dijktra for each node.

TC: $(V \cdot (E+V) \log E) \approx O(V^3 \log V)$ for dense graph $E = O(V^2)$

Floyd Warshell:



	1	2	3	
1	0	∞	2	(1 → 3)
2	∞	0	3	(2 → 3)
3	∞	4	0	(3 → 2)

V × V matrix

Algorithm:

loop) for $k \in [1, |V|]$
  for i from 1 to |V|
    for j from 1 to |V|

dist[i][j] = min(dis[i][j], dis[i][k] + dist[k][j])

dist[i][j]: distance from i to j

4 line code. TC $(O(V^3))$

logic

relaxing i to j using i to k then k to j

→ outer loop (looping over intermediate vertex)

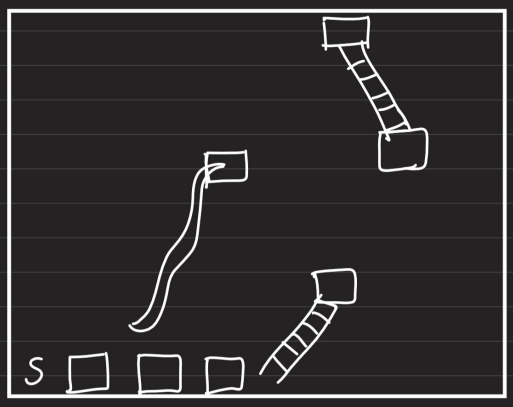
Code :

```
int dist[404][404];

void solve()
{
    lli n,m;cin>>n>>m;
    for(int i=0;i<=n;i++)
    {
        for(int j=0;j<=n;j++)
        {
            if(i!=j)dist[i][j]=1e9;
            else dist[i][j]=0;
        }
    }

    for(int i=0;i<m;i++)
    {
        int a,b,c;
        cin>>a>>b>>c;
        dist[a][b] = min(dist[a][b],c);
    }

    //Floyd Warshell - Note the order k,i,j
    for(int k=0;k<n;k++)
    {
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                dist[i][j] = min(dist[i][j],dist[i][k]+dist[k][j]);
            }
        }
    }
    int a,b; cin>>a>>b;
    cout<<dist[a][b]<<'\n';
```

Reversal Problem (Nodes deletion)

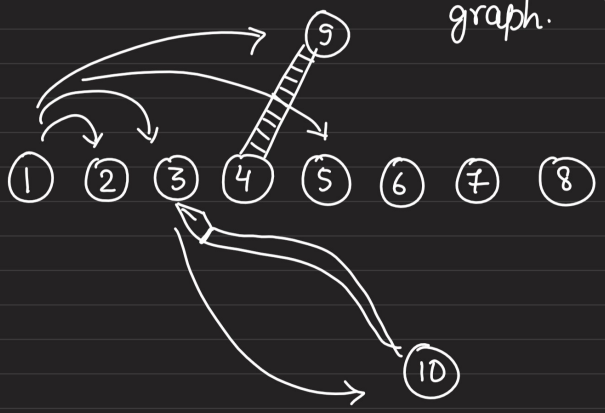
Graphs Modelling and Formulation

Q1. Find minimum #moves to complete snake and ladder game.



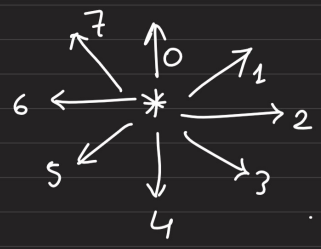
eg: N = 10

Can be formulated using graph.



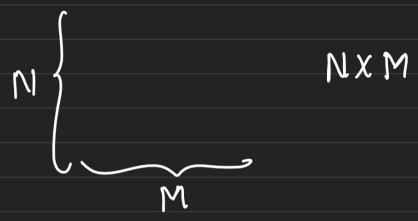
Q2 River - Boat Problem :-

Directions



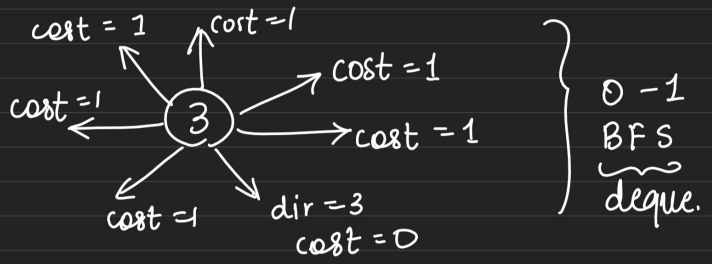
cost of movement along * direction is 0, & 1 otherwise.

I/p : Grid :- N M

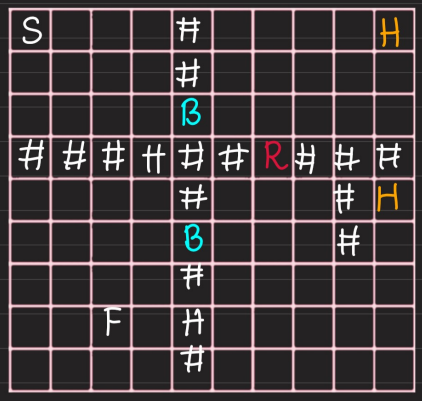


Modelling

find min cost to move from $(r_s, c_s) \rightarrow (r_d, c_d)$



Q3



R → Red door
B → Blue door

H → switch

at one time you can either open all red door or all blue doors.

(N, M ≤ 1000)

Find minimum time to reach F from S. Initially all the blue doors are open.

soln: Here, we go from S → B → H (open Red door)

F ← B ← H ← R

(open blue)

- Graph with state as

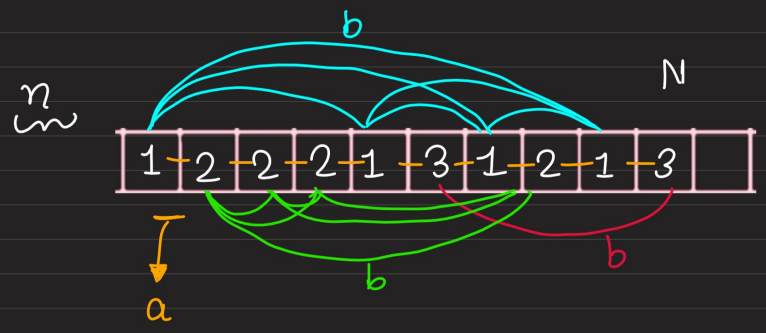
(x, y, open_door)

Implementation :

```
lli n,m;
vector<string> arr; //grid
using state = pair<int,ii>;

lli dist[2][1001][1001];

int dx[] = {0,0,1,-1};
int dy[] = {1,-1,0,0};

vector<state> get_neigh(state x)
{
    vector<state> res;
    for(int k=0;k<4;k++)
    {
        state y = x;
        y.S.F += dx[k];
        y.S.S += dy[k];
        res.push_back(y);
    }
    return res;
}

bool check(state a)
{
    int x = a.S.F; int y = a.S.S;
    int col = a.F;
    if(x<0 || y<0 || x>=n || y>=m || arr[x][y]=='#') return false;
    if(col==0 && arr[x][y]=='R') return false;
    if(col==1 && arr[x][y]=='B') return false;
    return true;
}
```

```
void bfs(state init)
{
    for(int i=0;i<2;i++){
        for(int j=0;j<=n;j++){
            for(int k=0;k<=m;k++){
                dist[i][j][k] = INF;
            }
        }
    }
    dist[init.F][init.S.F][init.S.S]=0;
    queue<state> q;
    q.push(init);
    while(!q.empty())
    {
        state cur = q.front();
        q.pop();
        for(auto neigh : get_neigh(cur))
        {
            if(check(neigh) && dist[neigh.F][neigh.S.F][neigh.S.S] == INF){
                dist[cur.F][cur.S.F][cur.S.S] = dist[neigh.F][neigh.S.F][neigh.S.S] + 1;
                q.push(neigh);
            }
        }
        if(arr[cur.S.F][cur.S.S]=='H'){ //if it is a switch
            state neigh = cur; neigh.F ^= 1;
            if(check(neigh) && dist[neigh.F][neigh.S.F][neigh.S.S] == INF)
            {
                dist[cur.F][cur.S.F][cur.S.S] = dist[neigh.F][neigh.S.F][neigh.S.S] + 1;
                q.push(neigh);
            }
        }
    }
}
```

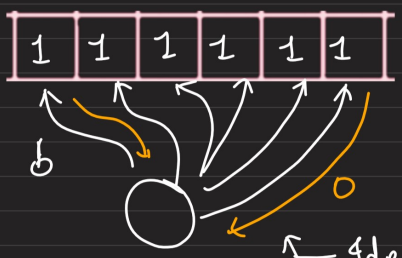
Q4 Given an array of integers, you can go to any other element of same value with cost b.
You can also go to the neighbouring elements with cost = a.



soln: Make graph connecting all the required nodes
& then run dijkstra

TC : $O(n^2 \log n) \rightarrow O((|V|+|E|)\log E)$
as $|E| \cong O(V^2)$

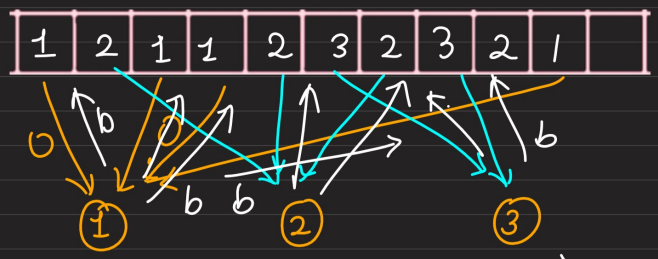
Can we do better?



$|E| = {}^nC_2$

← Idea for optimizing.

Now, there will be $O(N)$ edges.



this is how we will optimize

→ super node (teleportation cost)

for each 1 → 2 edges
for each 2 → 2 edges
for each 3 → 2 edges

(1) ⇄ super 1 (0 / b)

Edges = $\underbrace{2n}_{\text{connect. with super edges}} + \underbrace{2(n-1)}_{\text{b/w 2 neighbouring elements}} = \underline{4n-2}$

Nodes = (N + D)
↳ # distincts – elements

Coding :-

1) Creating Graph.
2) Run Dijkstra from st.

Q5. String (start) "01100...010" → 20 digits (fixed)

↓

final "11001 00 ...110"

You can flip any digit.

a) Minimum no. of flips?

Soln: We will simply count no. of unequal positions.

b) Some strings are banned [ ]

Now, what are the minimum no. of moves.

Soln: 101 ⟶ 000 for size = 3

Banned [100, 001]

101 ⟶ 111 ⟶ 011 ⟶ 010 ⟶ 000

Ans → (4)

for size = 20

each step might give TLE

str → 1 → str(i) flipping 1st digit.

str → ..... → 20 strings − # banned

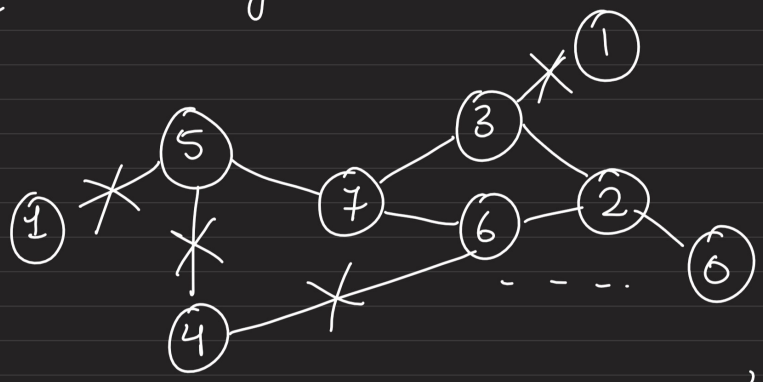
Approach :-

(2^{20} nodes)

\# 101 ⟶ 000 for size = 3

Banned $\begin{bmatrix} 100 \\ 001 \end{bmatrix}$

same as going to node 0 from node 5 with 4 & 1 being blocked.



NOW, BFS

- for each 2^{20} nodes, we can have 20 edges atmost :
- Simply go over no., flip bits & create the graph.

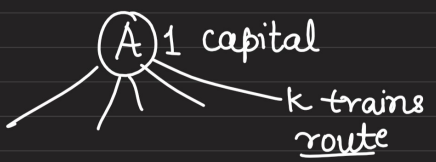
Adding decimal no.s is very efficient way to formulate the graph.

Jzzhu and Cities

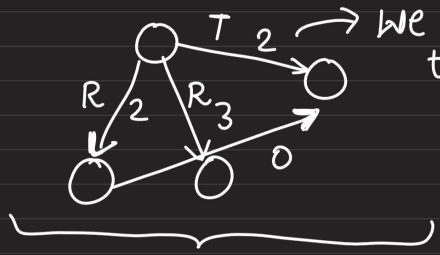
https://codeforces.com/problemset/problem/449/B

Steps to Solve :

i) We will have to decide in the dikstra that which nodes are very important.



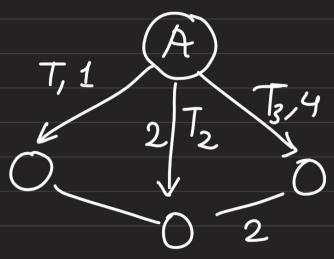
ii)



→ We can eliminate this T

If 2 paths, have same weight, then relax using road first and then only trains.

iii)



T_2 → eliminates T_3

At-coders Two Currencies

https://atcoder.jp/contests/abc164/tasks/abc164_e

N cities, M railroads

(1) 10^{100} gold coins, S silver coins

we can consider this as infinite.

state

• (node, silver_coins) → (n2, silver − A1) via time taken → B1

→ (n2, silver − A2) via B2

→ (n2, silver − Aj) via Bj

↓ Di

(nodes, silver_coins + Ci)

↳ this state is bounded to Amax(N−1)

SSSP (1, S)

E: Two Currencies

Let $A_{max} = max\{A_i \mid i = 1, 2, ..., M\}$. Consider the strategy when you have more than $A_{max}(N-1)$ silver coins. Obviously, it is optimal to go to the destination along the shortest path (during that, at most $A_{max}(N-1)$ coins are consumed, so you never run out the silver coins). Therefore, adding the constraints below does not change the answer of the problem.

Additional constraints

If the number of silver coins exceed $A_{max}(N-1)$ during the travel, discard them until it becomes $A_{max}(N-1)$

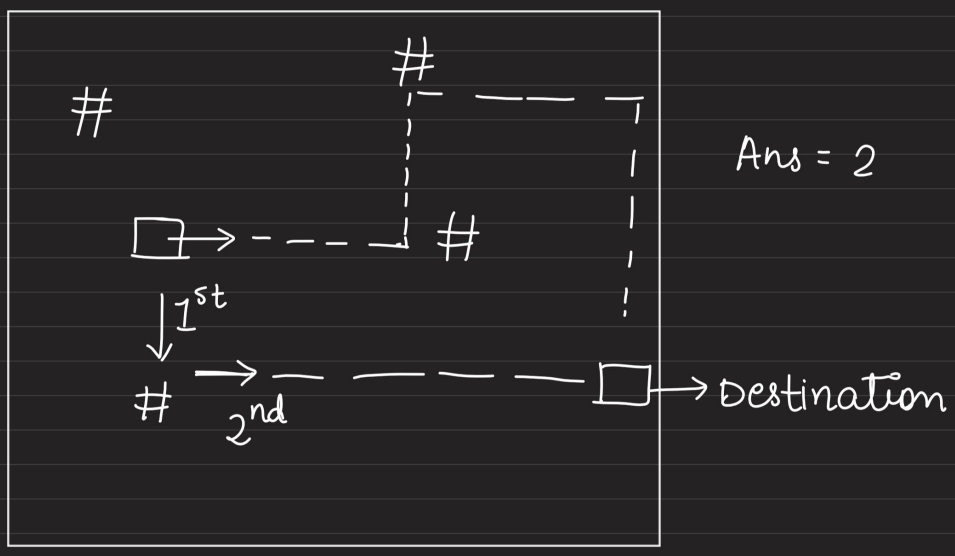
Let us consider the algorithm to the problem under the constraints above. By applying Dijkstra method while holding the state of (current vertex, the number of silver coins), the problem can be solved in a total of $O(A_{max}NMlog(A_{max}N))$ time. Since $A_{max} \leq 50, N \leq 50, M \leq 100$, it is fast enough.

All pair shortest path

Using BFS → run BFS on each node
$O(V+E) \cdot V$

Using Floyd warshell → $O(V^3)$ } we use this because it is easy to code.

Q6



Ans = 2

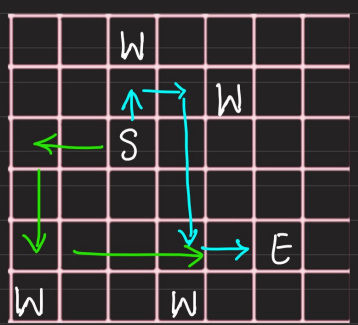
find minimum no. of pushes to get the block into D.

soln:

Approach 1 :

We can connect each atmax 4 nodes.

creating graph



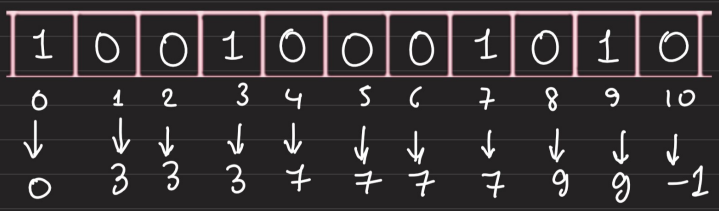
Ans = 3.

from every place, we can find the reachable nodes in R, L, U, D.

How to find nearest wall?

Q For every index find the nearest 1 to left

eg:



keep position of last seen 1. ← traverse in this array at each 1 update index.

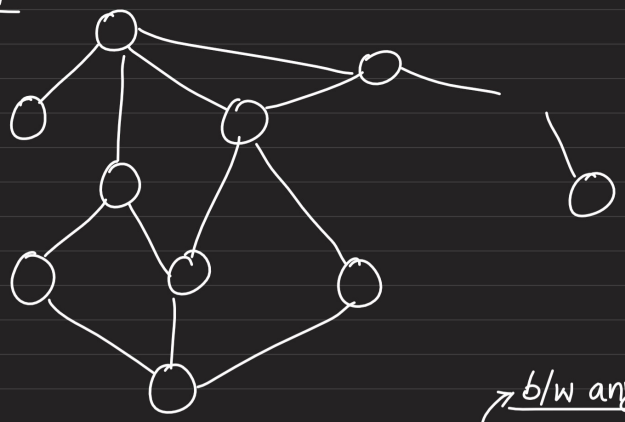
ans (i) → 0 → ans (i+1)
ans (i) → 1 → i

Now, we can create the graph and find the minimum pushes using BFS.

Approach 2 :

0-1 BFS (See code for transitions)

Q7



$N \leq 10^3$

$M \leq 10^4$

↗ b/w any 2 pairs

Find the longest shortest path, i.e, diameter of a graph.

Soln:

$$\underset{x,y}{\text{Max}} \; (dist(x,y))_{min}$$

Do BFS on each node to get all pair shortest path.

Q8.

0	1	2	3	4	5	6	7	8	9
3	5	8	2	1	.	.	.	.	.
9	8	7	6	.	.	.	.	2	1

→ H-Bomb inside

Initial (0,0,0)

(K) Banned numbers { 8 9 2 ; 5 3 1 ; 6 3 2 } → if these no. reaches, bomb will explode.

Soln: In one rotation we can go to 8 states.

total no. of nodes

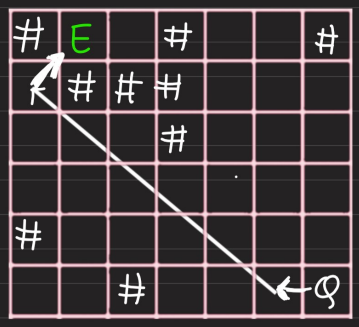
no. of nodes = $10 \times 10 \times 10 = 10^3$

from each node, atmax 8 edges.

1) Create Graph

2) Do BFS.

Q9.

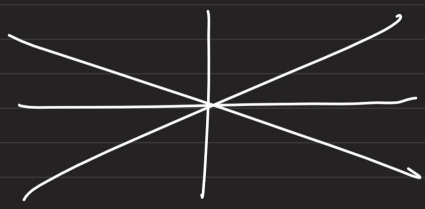


N × M chessboard

Find min moves Queen require to reach E.

Here, Ans = 3.

Soln: we can connect each cell with



O(N+M) no. of edges for each node

- we can create graph with $|E| = NM(N+M)$
- then do BFS.

↳ TC : $O(NM(N+M))$

Optimization :

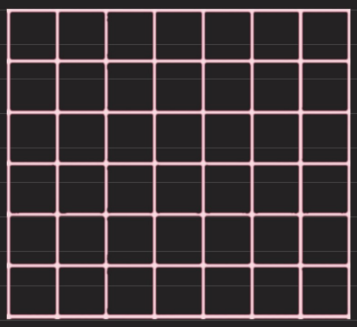
A 1) Whenever you change your direction, cost += 1

state (x, y, dir)

Now, $|V| = 8NM$, $|E| = 8(8NM)$

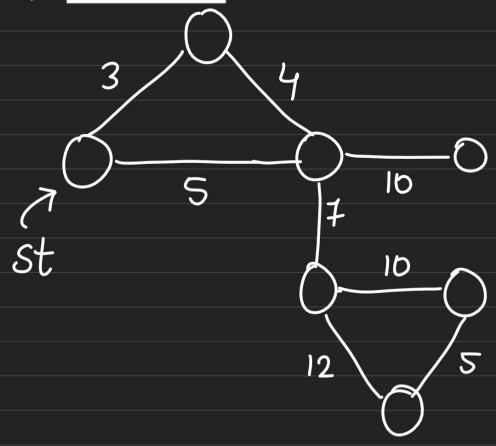
TC : $O(8^2 NM)$

A2) Right normal BFS (without creating graph) will give answer O(N*M)

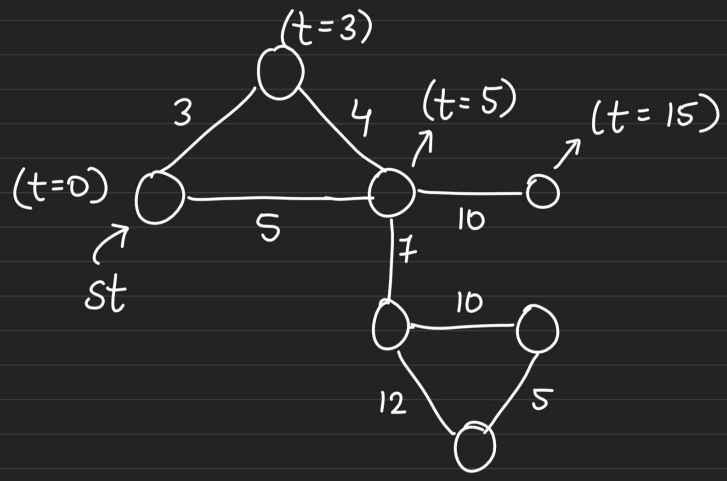


(Doubt)

Q10) Burn All :



Soln: t = burning time



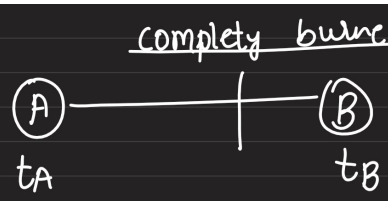
Shortest path from st junction to any junction gives us the time of its burning.

We can take max over all the answers.

burning from both sides

abs $(t_A - t_B) \geq x$ } time_taken to burn the edge

$\min(t_A - t_B) + x$.



say $t_A < t_B$

if $((t_B - t_A) < X)$

time to burn the edge $= (t - t_A) + (t - t_B) = X$

$$\Rightarrow \underbrace{t = \frac{X + t_A + t_B}{2}}$$

Take max over all the values of nodes & edges timing on burn