Page No
Date

0-1 BFS

In normal BFS of a graph, all edges have equal weight but in 0-1 BFS some edges may have 0 weight and some may have 1 weight. In this, we will not use a bool array to mark visited nodes but at each step, we will check for the optimal distance condition. We use a double-ended queue to store the node. While performing BFS if an edge having weight = 0 is found node is pushed at front of the double-ended queue and if an edge having weight = 1 is found, it is pushed to the back of the double-ended queue.

Q. CHEF AND REVERSING (CODECHEF)

Sometimes mysteries happen. Chef found a directed graph with N vertices and M edges in his kitchen!

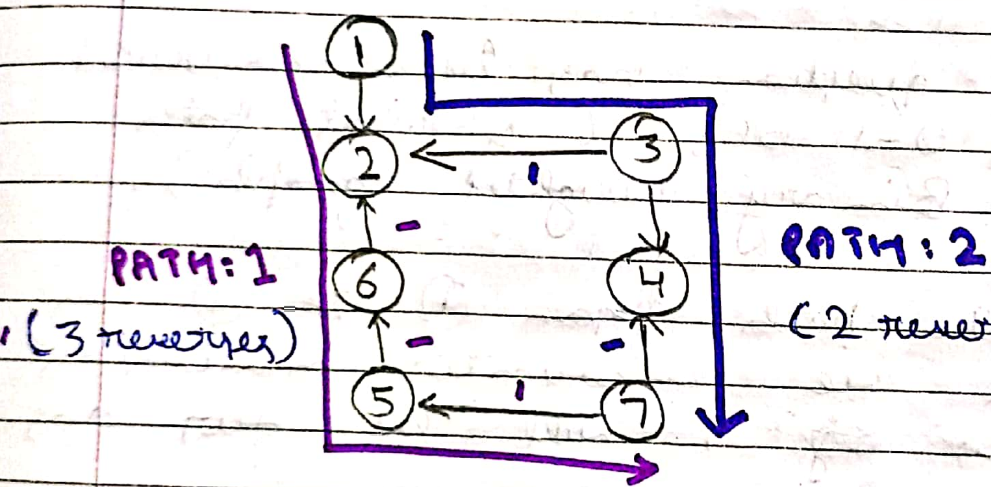
The evening was boring and chef has nothing else to do, so to entertain himself, Chef thought about a question "What is the minimum number of edges he needs to reverse in order to have at least one path from vertex 1 to vertex N, where the vertices are numbered from 1 to N.

INPUT :

~~99~~
7 7 → vertices and edges resp.
1 2
3 2
3 4
7 4
6 2
5 6
7 5

EDGES

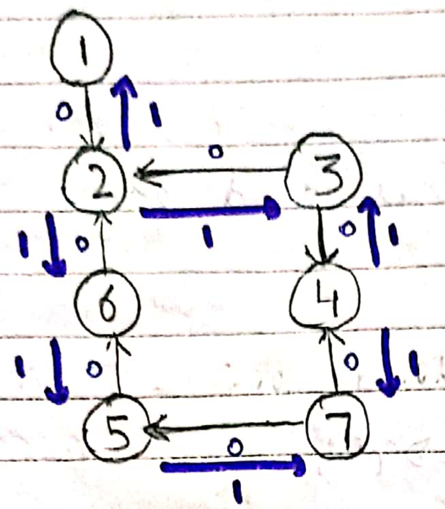
The graph will look something like this:



OUTPUT :

2

APPROACH: The question doesn't look like a BFS question or shortest path question. But it is a ~~01~~ 0-1 BFS (Shortest path in a Binary graph question).



→ Mark given direction of edges = 0 and make opposite direction of edges for each given edges and mark them 1.

∴ The question has been converted into 0-1 BFS (Shortest path in a Binary weighted graph).

Now the path from ① to ⑦ in which we encounter minimum number of 1 will be our ans.

CODE

```
CONST INT N = 1e5+10;
CONST INT INF = 1e9+10;

VECTOR <PAIR <INT, INT>> G[N]; // vector of pairs to store pair of node and its weight.
```

→ Level vector initialized with INF

```
VECTOR <INT> LEV(N, INF);
```

Scanned with CamScanner

Page No.
Date

→ Vertices and Edges

```
INT n, m;

INT BFS( )
{
    DEQUE <INT> q;
    q.PB(1); → Given starting point
    LEV[1] = 0;

    WHILE (!q.EMPTY())
    {
        INT CUR_V = q.FRONT();
        q.POP_FRONT();

        FOR (AUTO CHILD : G[CUR_V])
        {
            INT CHILD_V = CHILD.F;
            INT WT = CHILD.S;
```

→ Here, in 0-1 BFS, we don't need visited array, because each node can be visited twice first with 1 weight and then with 0 weight.
→ So, here we use level as kind of visited array.

```
            IF (LEV[CUR_V] + WT < LEV[CHILD_V])
            {
                LEV[CHILD_V] = LEV[CUR_V] + WT;
                IF (WT == 1) q.PB(CHILD_V);
                ELSE q.PUSH_FRONT(CHILD_V);
            }
```

```
        }
    }
    RETURN LEV[n] == INF ? -1 : LEV[n];
```
-> Returning level of n^{TH} node which will also be its shortest path from 1.

```
}

INT MAIN()
{
    CIN >> n >> m;
    FOR (INT i = 0; i < m; i++)
    {
        INT x, y;
        CIN >> x >> y;
        IF (x == y) CONTINUE;
```
-> To prevent self loop

```
        g[x].PB({y, 0});
```
-> Given edges with 0 as weight

```
        g[y].PB({x, 1});
```
-> Opposite direction edges created by me with 1 as weight

```
    }
    COUT << BFS() << "\n";
    RETURN 0;
}
```