



Breadth First Search (BFS)

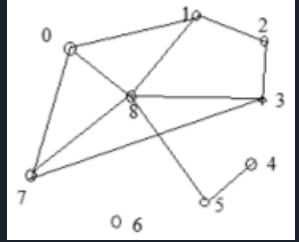
Jaxon Ballentine, Aiden Xu, Jack Lowry

What is BFS?

BFS is an algorithm that explores every possible path from one point to every other.

The algorithm is as follows:

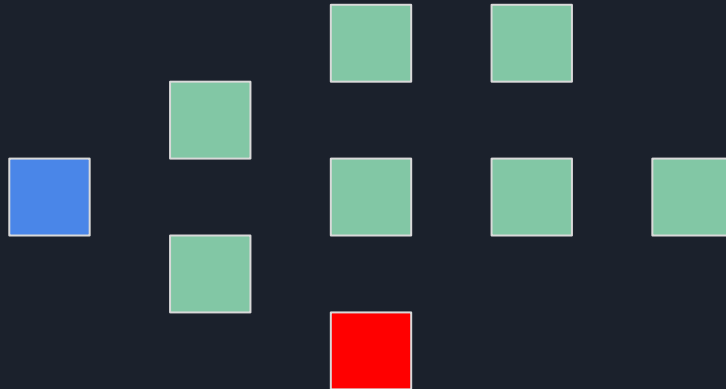
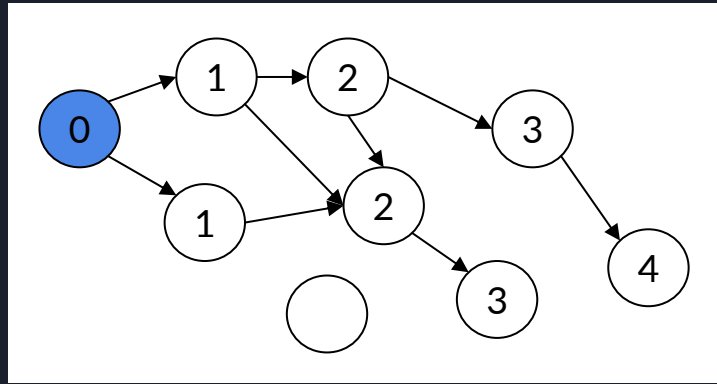
1. Create a distance vector (houses all distances from the source node to all other nodes) and initialize all as -1
2. Create a queue, and add source node to queue*
3. While the queue is not empty:
 - a. Remove first element and store in a temporary variable
 - b. For each adjacent node that hasn't been visited yet (or $\neq -1$):
 - i. Set the adjacent node to the source node's value + 1
 - ii. Add the adjacent node to the queue
4. Distance vector should now contain all shortest distances from source node, and any still at -1 are not reachable from the source



*Queue:

Similar to a vector, but only the first value is accessible in code. Values can only be added to the back.

BFS Animation





Application: Modulo Solitaire

- We use BFS to find the minimum number of moves needed to find 0.
- Each move we can make is an edge in our graph.
- Each number from 0 to $m - 1$ is a vertex in our graph.

[< Hide](#)

Modulo Solitaire

Modulo Solitaire is a game that can be played when you are bored. You can even play it without a phone, just on paper. First, you pick a number m . Then you pick two sequences of numbers a_i and b_i . Finally, you pick a starting number s_0 . Now, your goal is to go from s_0 to 0 in as few moves as possible. In each move, you choose an i , then multiply your current number by a_i , add b_i to it, and reduce the result modulo m . That is, $s_j = (s_{j-1} \cdot a_{i_j} + b_{i_j}) \bmod m$.

Input

The first line of input contains three integers $1 \leq m \leq 10^6$, $0 \leq n \leq 10$, and $1 \leq s_0 < m$. The next n lines each contain two integers, a pair $0 \leq a_i \leq 10^9$ and $0 \leq b_i \leq 10^9$.

Output

Output a single integer, the minimum number of moves needed to reach 0 starting from s_0 . If it is not possible to reach 0 in any number of moves, output -1 .

Sample Input 1

```
5 2 1
2 1
3 1
```



Sample Output 1

```
2
```



Modulo Solitaire code

- Get the modulo, the number of moves, and the starting point.
- Read in the moves.
- Define the distance vector and active queue.
- Loop until we reach zero or the queue is empty.
- Loop through each possible move.
- Add the new number to the queue and update its distance.

```
1 // JAKOB Ballentine
2 // 6/23/2025
3 // SI@UCF Introduction to Competitive Programming via C++ Final Project
4 // https://open.kattis.com/problems/modulosolitaire
5
6 #include<bits/stdc++.h>
7 using namespace std;
8
9 int main() {
10     // Input.
11     int m, n, s;
12     cin >> m >> n >> s;
13
14     // Get the possible moves.
15     vector<pair<long long, long long>> moves(n);
16
17     for (int i = 0; i < n; i++) {
18         cin >> moves[i].first >> moves[i].second;
19     }
20
21     // Define the distance array and activeValues queue.
22     vector<int> distance(m, -1);
23     distance[s] = 0;
24
25     queue<int> activeValues;
26     activeValues.push(s);
27
28     // Loop through each active value until we find 0 or we run out of values.
29     while (activeValues.size() > 0 && distance[0] == -1) {
30
31         // Get an active value.
32         long long cur = activeValues.front();
33         activeValues.pop();
34
35         // Loop through each possible move.
36         for (int i = 0; i < n; i++) {
37             // Find next value.
38             int next = (cur * moves[i].first + moves[i].second) % m;
39
40             // Add the next value if we haven't found it yet.
41             if (distance[next] == -1) {
42                 distance[next] = distance[cur] + 1;
43                 activeValues.push(next);
44             }
45         }
46     }
47
48     // Output.
49     cout << distance[0];
50
51     return 0;
52 }
```

Application: Fire 2

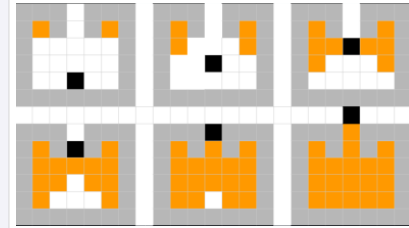
- Fire moves to all adjacent every second
- Can move to any adjacent every second
- Create distance vector for the fire
- Fire vector acts an advisory (e.g. after 4 seconds the bottom left will be engulfed)
- Run through each path for the person and stop if fire is there before or at the same time.

Sample Input 1

```
5
4 3
####
#*@.
####
7 6
###.###
#*#.###
#....#
#....#
#..@..#
#####
7 4
###.###
#....*#
#@....#
.#####
5 5
.....
.***.
.*@*.
.***.
.....
3 3
###
#@#
###
```

Sample Output 1

```
2
5
IMPOSSIBLE
IMPOSSIBLE
IMPOSSIBLE
```



	1	2
1	2	3
2	3	4

Fire Code:

```
bool inBounds(int x, int y, int r, int c) {
    return x >= 0 && x < r && y >= 0 && y < c;
}

int solve(vector<vector<char>> maze, pair<int, int> start, vector<vector<int>> fires) {
    vector<vector<int>> dist(h, vector<int>(w, -1));
    queue<pair<int, int>> q;
    q.push(start);
    int res = -1;
    dist[start.first][start.second] = 0;
    while (q.size() > 0) {
        pair<int, int> cur = q.front(); q.pop();

        //Its out
        if (cur.first == 0 || cur.first == h - 1 || cur.second == 0 || cur.second == w - 1) {
            res = dist[cur.first][cur.second];
            break;
        }

        for (int i = 0; i < DX.size(); i++) {
            int nextX = cur.first + DX[i];
            int nextY = cur.second + DY[i];

            if (!inBounds(nextX, nextY, h, w)) continue;

            if (dist[cur.first][cur.second] + 1 >= fires[nextX][nextY] && fires[nextX][nextY] >= 0) continue;

            if (maze[nextX][nextY] == '#') continue;
            if (dist[nextX][nextY] != -1) continue;

            dist[nextX][nextY] = dist[cur.first][cur.second] + 1;
            q.push(pair<int, int>(nextX, nextY));
        }
    }
    return res;
}
```

```
const vector<int> DX = {-1,0,0,1};
const vector<int> DY = {0,-1,1,0};
```

```
vector<vector<int>> fireTimes(vector<vector<char>> maze) {
    vector<vector<int>> dist(h, vector<int>(w, -1));
    queue<pair<int, int>> q;
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
            if (maze[i][j] == '*') {
                q.push(pair<int, int>(i, j));
                dist[i][j] = 0;
            }
        }
    }
    while (q.size() > 0) {
        pair<int, int> cur = q.front(); q.pop();
        for (int i = 0; i < DX.size(); i++) {
            int nextX = cur.first + DX[i];
            int nextY = cur.second + DY[i];
            if (!inBounds(nextX, nextY, h, w)) continue;
            if (maze[nextX][nextY] == '#') continue;
            if (dist[nextX][nextY] != -1) continue;
            dist[nextX][nextY] = dist[cur.first][cur.second] + 1;
            q.push(pair<int, int>(nextX, nextY));
        }
    }
    return dist;
}
```

```
int main() {
    int nC;
    cin >> nC;
    pair<int, int> start;
    for (int i = 0; i < nC; i++) {
        cin >> w >> h;
        vector<vector<char>> maze(h);
        for (int k = 0; k < h; k++) {
            for (int j = 0; j < w; j++) {
                char x;
                cin >> x;
                maze[k].push_back(x);
                if (x == '@') start = pair<int, int>(k, j);
            }
        }
        vector<vector<int>> fire = fireTimes(maze);
        int res = solve(maze, start, fire);
        if (res != -1) {
            cout << res + 1 << endl;
        }
        else cout << "IMPOSSIBLE" << endl;
    }
    return 0;
}
```