Find the shortest path

Write a C program to determine if a path exists between a given source node and a destination node in an undirected graph. If a path exists, also find the shortest distance between them. If no path exists, print "False" and do not print a distance. In this graph: Each node is connected to its adjacent nodes by edges, and the distance between any two adjacent nodes is 1 unit.

Input Format

Number of nodes(n)

Number of edges(c)

Source node

Destination node

u1 v1 (For each edge, provide two integers u and v, where u and v are the nodes connected by that edge.)

u2 v2

.

.

uc vc

Constraints

Values of the nodes are from 0 to n-1

Graph is undirected.

Output Format

True or False(True if path exists between source and destination else False)

The shortest distance between them(if path exists else don't print anything)

Sample Input 0

```
3
3
0
2
1 0
2 0
1 2
```

Sample Output 0

```
True
1
```

Explanation 0

There is a path from node 0 to node 2 with a shortest distance of 1.

Contest ends in an hour

Submissions: 19
Max Score: 10
Difficulty: Easy

Rate This Challenge:

More

C

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
  int vertex;
  struct Node *next;
} Node;

typedef struct Graph {
  int numVertices;
  Node **adjLists;
  bool *visited;
} Graph;

int queue[100];
int f_queue = 0;  // front of the queue
int r_queue = -1; // rear of the queue

Graph *createGraph(int vertices);
void addEdge(Graph *graph, int src, int dest);
int path(Graph *graph, int src, int dest);

int main() {
  int n, m, s, d;
  scanf("%d %d %d %d", &n, &m, &s, &d);

  Graph *graph = createGraph(n);

  for (int i = 0; i < m; i++) {
    int u, v;
    scanf("%d %d", &u, &v);
    addEdge(graph, u, v);
  }

  int ans;
  if ((ans = path(graph, s, d)) >= 0) {
    printf("True\n%d", ans);
  } else {
    printf("False\n");
  }

  return 0;
}

// Initialize a new graph with a specified number of vertices
Graph *createGraph(int vertices) {
  Graph *g = (Graph *)malloc(sizeof(Graph));
  g->numVertices = vertices;
  g->adjLists = (Node **)malloc(sizeof(Node *) * vertices);
  g->visited = (bool *)malloc(sizeof(bool) * vertices);
  for (int i = 0; i < vertices; i++) {
    g->adjLists[i] = NULL;
    g->visited[i] = false;
  }
  return g;
}
```

```c

// Add an edge to the graph (undirected)
void addEdge(Graph *graph, int src, int dest) {
  Node *s = (Node *)malloc(sizeof(Node));
  s->vertex = dest;
  s->next = graph->adjLists[src];
  graph->adjLists[src] = s;

  Node *d = (Node *)malloc(sizeof(Node));
  d->vertex = src;
  d->next = graph->adjLists[dest];
  graph->adjLists[dest] = d;
}

// BFS to find the shortest path from src to dest
int path(Graph *graph, int src, int dest) {
  // Reset the visited array
  for (int i = 0; i < graph->numVertices; i++) {
    graph->visited[i] = false;
  }

  // Initialize the queue
  queue[++r_queue] = src;
  graph->visited[src] = true;
  int depth = 0;

  while (f_queue <= r_queue) {
    int size = r_queue - f_queue + 1;
    for (int i = 0; i < size; i++) {
      int current = queue[f_queue++];
      if (current == dest) {
        return depth; // Found the destination
      }

      // Traverse neighbors
      Node *traverse = graph->adjLists[current];
      while (traverse) {
        if (!graph->visited[traverse->vertex]) {
          graph->visited[traverse->vertex] = true;
          queue[++r_queue] = traverse->vertex;
        }
        traverse = traverse->next;
      }
    }
    depth++;
  }
  return -1; // No path found
}

```

Line: 106 Col: 1

Upload Code as File Test against custom input Run Code Submit Code

Testcase 0 ✔

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
3
3
0
2
1 0
2 0
1 2
```

Your Output (stdout)

```
True
1
```

Expected Output

```
True
1
```