HackerRank | Prepare Certify Compete Apply Search 2

All Contests > CSE-g-Week-14 > Cycle Hunt

Cycle Hunt

Problem Submissions Leaderboard

Given an undirected graph represented as an adjacency list, determine if there exists a cycle in the graph. A cycle exists if a path starts and ends at the same vertex without reusing any edge in between. Print "True" if there exists a cycle; otherwise, print "False".

Input Format

1. The first line contains two space-separated integers, n (the number of nodes) and m (the number of edges).
2. Each of the next m lines contains two space-separated integers, u and v, representing an undirected edge between nodes u and v.

Constraints

1 ≤ n ≤ 10^5
0 ≤ m ≤ 10^5
The graph may be disconnected.

Output Format

A single line containing either "True" or "False".

Sample Input 0

```
4 4
0 1
1 2
2 3
3 0
```

Sample Output 0

```
True
```

Contest ends in an hour

Submissions: 21
Max Score: 10
Difficulty: Medium

Rate This Challenge:
☆☆☆☆☆

More

C

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

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

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

// Function to create the graph
Graph* createGraph(int vertices);

// Function to add an edge to the graph
void addEdge(Graph* graph, int src, int dest);

// DFS utility function to check for cycles
bool dfs(Graph* graph, int vertex, bool* visited, int parent);

// Function to check if the graph contains a cycle
bool containsCycle(Graph* graph);

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

    Graph* graph = createGraph(n);

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

    if (containsCycle(graph)) {
        printf("True\n");
    } else {
        printf("False\n");
    }

    return 0;
}

// Function to create a graph with the specified number of vertices
Graph* createGraph(int vertices) {
    Graph *g = (Graph *) malloc(sizeof(Graph));
    g->numVertices = vertices;
    g->adjLists = (Node **)malloc(sizeof(Node *) * vertices);
    for (int i = 0; i < vertices; i++) {
        g->adjLists[i] = NULL;
    }
    return g;
}

// Function to add an edge between src and dest
void addEdge(Graph* graph, int src, int dest) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->vertex = dest;
    newNode->next = graph->adjLists[src];
    graph->adjLists[src] = newNode;

    // Since it's an undirected graph, add an edge in the opposite direction as well
    newNode = (Node*)malloc(sizeof(Node));
    newNode->vertex = src;
    newNode->next = graph->adjLists[dest];
    graph->adjLists[dest] = newNode;
}

// DFS function to detect a cycle in the graph
bool dfs(Graph* graph, int vertex, bool* visited, int parent) {
    visited[vertex] = true;

    Node* current = graph->adjLists[vertex];
    while (current != NULL) {
        int neighbor = current->vertex;

        // If the neighbor is not visited, recurse on it
        if (!visited[neighbor]) {
            if (dfs(graph, neighbor, visited, vertex)) {
                return true;
            }
        }
        // If the neighbor is visited and not the parent, a cycle is found
        else if (neighbor != parent) {
            return true;
        }

        current = current->next;
    }

    return false;
}

// Function to check if the graph contains a cycle
bool containsCycle(Graph* graph) {
    bool* visited = (bool*)malloc(sizeof(bool) * graph->numVertices);
    for (int i = 0; i < graph->numVertices; i++) {
        visited[i] = false;
    }

    // Check for cycles in each component
    for (int i = 0; i < graph->numVertices; i++) {
        if (!visited[i]) {
            if (dfs(graph, i, visited, -1)) {
                free(visited);
                return true;
            }
        }
    }

    free(visited);
    return false;
}
```

Line: 117 Col: 2

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)

```
4 4
0 1
1 2
2 3
3 0
```

Your Output (stdout)

```
True
```

Expected Output

```
True
```