



Experiment No. 6

Title: Implementation of problem based on Graph Theory



Batch:A2

Roll No: 16010421063

Experiment No.:3

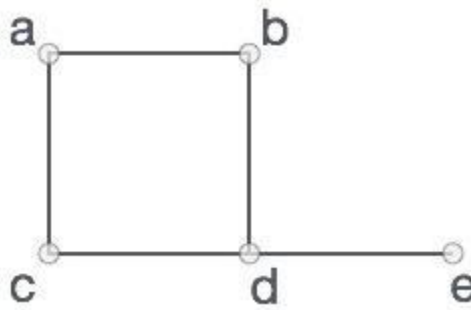
Aim: To study Graph Theory and graph traversal for implementation of problem statement that is based on BFS, DFS & topological sort and verify given test cases.

Resources needed: Text Editor, C/C++ IDE

Theory:

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.

Formally, a graph is a pair of sets (V, E) , where V is the set of vertices and E is the set of edges, connecting the pairs of vertices. Take a look at the following graph –



In the above graph,

$V = \{a, b, c, d, e\}$

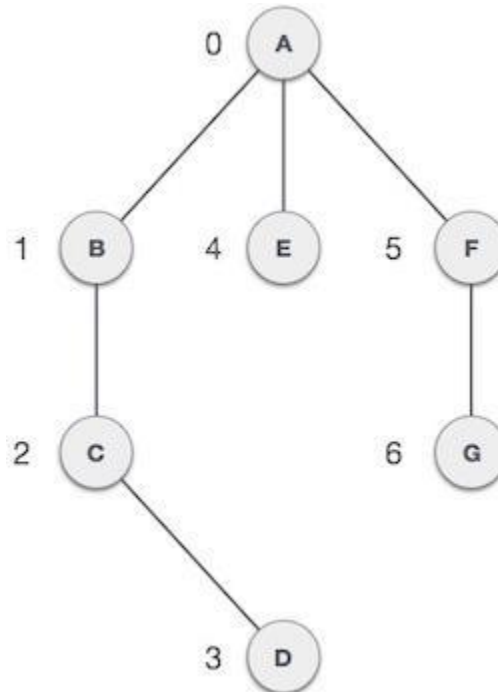
$E = \{ab, ac, bd, cd, de\}$

Graph Data Structure

Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms –

- **Vertex** – Each node of the graph is represented as a vertex. In the following example, the labeled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.
- **Edge** – Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.

- **Adjacency** – Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.
- **Path** – Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.

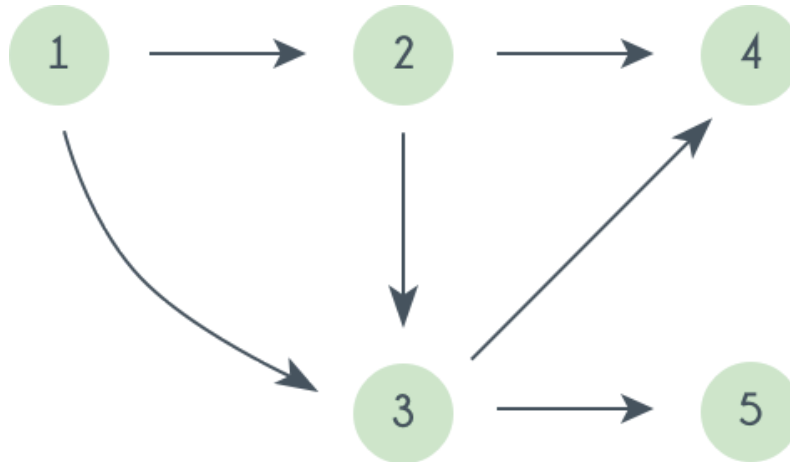


Basic Operations

Following are basic primary operations of a Graph –

- **Add Vertex** – Adds a vertex to the graph.
- **Add Edge** – Adds an edge between the two vertices of the graph.
- **Display Vertex** – Displays a vertex of the graph.

Topological sorting of vertices of a **Directed Acyclic Graph** is an ordering of the vertices v_1, v_2, \dots, v_n in such a way, that if there is an edge directed towards vertex v_j from vertex v_i , then v_i comes before v_j . For example consider the graph given below:



A topological sorting of this graph is: 1 2 3 4 5. There are multiple topological sorting possible for a graph. For the graph given above one another topological sorting is: 1 2 3 5 4

In order to have a topological sorting the graph must not contain any cycles. In order to prove it, let's assume there is a cycle made of the vertices $v_1, v_2, v_3 \dots v_n$. That means there is a directed edge between v_i and v_{i+1} ($1 \leq i < n$) and between v_n and v_1 . So now, if we do topological sorting then v_n must come before v_1 because of the directed edge from v_n to v_1 . Clearly, v_{i+1} will come after v_i , because of the directed from v_i to v_{i+1} , that means v_1 must come before v_n . Well, clearly we've reached a contradiction, here. So topological sorting can be achieved for only directed and acyclic graphs.

Let's see how we can find a topological sorting in a graph. So basically we want to find a permutation of the vertices in which for every vertex v_i , all the vertices v_j having edges coming out and directed towards v_i comes before v_i . We'll maintain an array T that will denote our topological sorting. So, let's say for a graph having N vertices, we have an array in `degree[]` of size N whose i th element tells the number of vertices which are not already inserted in T and there is an edge from them incident on vertex numbered i . We'll append vertices v_i to the array T , and when we do that we'll decrease the value of `in degree[vj]` by 1 for every edge from v_i to v_j . Doing this will mean that we have inserted one vertex having edge directed towards v_j . So at any point we can insert only those vertices for which the value of `in degree[]` is 0.

Activity:

Consider the following problem statement and other information provided along with it:

Problem Statement: Lonely Island

There are many islands that are connected by one-way bridges, that is, if a bridge connects islands a and b , then you can only use the bridge to go from a to b but you cannot travel back by using the same. If you are on island a , then you select (uniformly and randomly) one of the islands that are directly reachable from a through the one-way bridge and move to that island. You are stuck on an island if you cannot move any further. It is guaranteed that after leaving any island it is not possible to come back to that island.

Find the island that you are most likely to get stuck on. Two islands are considered equally likely if the absolute difference of the probabilities of ending up on them is $\leq 10^{-9}$.

Input format

First line: Three integers n (the number of islands), m (the number of one-way bridges), and r (the index of the island you are initially on)

Next m lines: Two integers u_i and v_i representing a one-way bridge from island u_i to v_i .

Output format

Print the index of the island that you are most likely to get stuck on. If there are multiple islands, then print them in the increasing order of indices (space separated values in a single line).

Input Constraints

$$1 \leq n \leq 200000$$

$$1 \leq m \leq 500000$$

$$1 \leq u_i, v_i, r \leq n$$

Sample Input	Sample Output
5 7 1 1 2 1 3 1 4 1 5 2 4 2 5 3 4	4

Program:

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
vector<int> vis;
vector<int> topo_sorted;

int dfs(int node, vector<int> adj[])
{
    vis[node] = 1;
    for (auto &nbr : adj[node])
```

```

{
    if (!vis[nbr])
    {
        dfs(nbr, adj);
    }
}
topo_sorted.push_back(node);
return 0;
}

int32_t main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
#ifdef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
#endif
    int n, m, r;
    cin >> n >> m >> r;
    vis.resize(n + 1);
    vector<int> adj[n + 1];
    while (m--)
    {
        int a, b;
        cin >> a >> b;
        adj[a].push_back(b);
    }
    dfs(r, adj);
    reverse(topo_sorted.begin(), topo_sorted.end());
    map<int, double> prob, leaf_prob;
    prob[r] = 1;

    for (auto &node : topo_sorted)
    {
        int bridges = adj[node].size();
        if (bridges == 0)
        {
            leaf_prob[node] += prob[node];
        }
    }
}

```

```
    for (auto &nbr : adj[node])
    {
        prob[nbr] += prob[node] / bridges;
    }a
}
double mx = 0;
for (auto &i : leaf_prob)
{
    mx = max(mx, i.second);
}
set<int> ans;
for (auto &i : leaf_prob)
{
    if ((abs(i.second - mx)) <= 1e-9)
    {
        ans.insert(i.first);
    }
}
for (auto &i : ans)
{
    cout << i << " ";
}
}
```












Output:

Submission ID: 81579679 / 12 seconds ago

RESULT:  Accepted[Refer Judge e](#)

Score	Time (sec)	Memory (KiB)	Language
0	1.57344	13056	C++17

Input	Result	Time (sec)	Memory (KiB)	Score	Your Output	Correct Output	Diff
Input #1	 Accepted	0.073995	10192	4			
Input #2	 Accepted	0.075146	9828	4			
Input #3	 Accepted	0.058483	7848	4			
Input #4	 Accepted	0.081883	9224	4			
Input #5	 Accepted	0.089963	10844	4			
Input #6	 Accepted	0.067011	8996	4			
Input #7	 Accepted	0.06607	8776	4			
Input #8	 Accepted	0.057435	7964	4			
Input #9	 Accepted	0.082829	10960	4			
Input #10	 Accepted	0.081775	11128	4			
Input #11	 Accepted	0.074196	11520	4			
Input #12	 Accepted	0.090907	13056	4			
Input #13	 Accepted	0.098079	12680	4			

Outcomes: CO3 Understand the Graphs,related algorithms, efficient implementation of

those algorithms and applications

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

Successfully solved the hackerrank question using DFS and topological sorting algorithm

References:

1. <https://www.hackerearth.com/practice/algorithms/graphs/topological-sort/practice-problems/algorithm/lonelyisland-49054110/>
2. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, "Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
3. Antti Laaksonen, "Guide to Competitive Programming", Springer, 2018
4. Gayle Laakmann McDowell, "Cracking the Coding Interview", CareerCup LLC, 2015
5. Steven S. Skiena Miguel A. Revilla, "Programming challenges, The Programming Contest Training Manual", Springer, 2006
6. Antti Laaksonen, "Competitive Programmer's Handbook", Hand book, 2018
7. Steven Halim and Felix Halim, "Competitive Programming 3: The Lower Bounds of Programming Contests", Handbook for ACM ICPC

