RAJSHAHI UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Lab report: 05
Course No.: CSE 2202

Date of Experiment: 28.11.2018
Date of Submission: 14.01.2019

Submitted to:
Biprodip Pal
Assistant Professor,
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

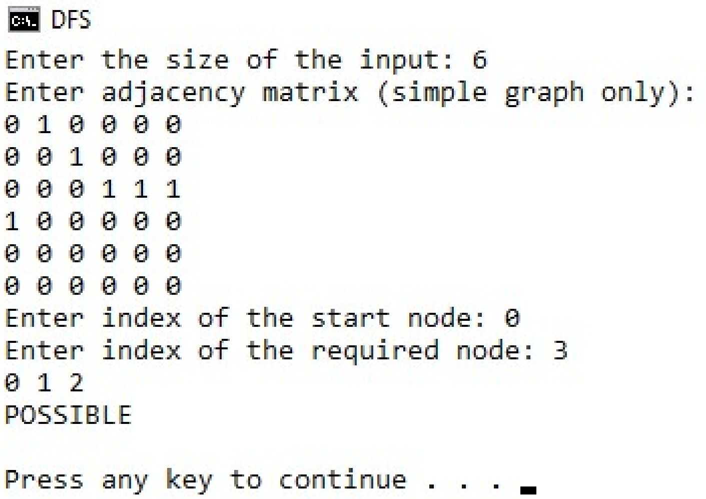
Submitted by:
Riyad Morshed Shoeb
Roll No: 1603013
Section: A
Department of Computer Science and Engineering
Rajshahi University of Engineering and Technology

Problem: Traverse a graph to find a node using Depth First Search (DFS).

Code:

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <stack>
#include <algorithm>
using namespace std;
int main(void)
{
    int INPUT_SIZE;
    int i, j;
    int START_NODE;
    int FIND_NODE;
    stack <int> NODE;
    int STACK_TOP;
    bool POSSIBLE = false;
    cout << "Enter the size of the input: ";
    cin >> INPUT_SIZE;
    bool PUSHED[INPUT_SIZE];
    bool ADJACENT[INPUT_SIZE][INPUT_SIZE];
    cout << "Enter adjacency matrix (simple graph only): " << endl;
    for(i=0; i<INPUT_SIZE; i++)
    {
        for(j=0; j<INPUT_SIZE; j++)
            cin >> ADJACENT[i][j];
    }
    for(i=0; i<INPUT_SIZE; i++)
        PUSHED[i] = false;
    cout << "Enter index of the start node: ";
    cin >> START_NODE;
    cout << "Enter index of the required node: ";
    cin >> FIND_NODE;
    NODE.push(START_NODE);
    PUSHED[START_NODE] = true;
    while(!NODE.empty())
    {
        STACK_TOP = NODE.top();
        cout << STACK_TOP << " ";
        NODE.pop();
        if(ADJACENT[STACK_TOP][FIND_NODE])
        {
            POSSIBLE = true;
            break;
        }
        for(i=0; i<INPUT_SIZE; i++)
        {
            if((ADJACENT[STACK_TOP][i]) && (!PUSHED[i]))
            {
                NODE.push(i);
                PUSHED[i] = true;
            }
        }
    }
    cout << endl;
    if(POSSIBLE)
        cout << "POSSIBLE" << endl;
    else
        cout << "NOT POSSIBLE" << endl;
}
```

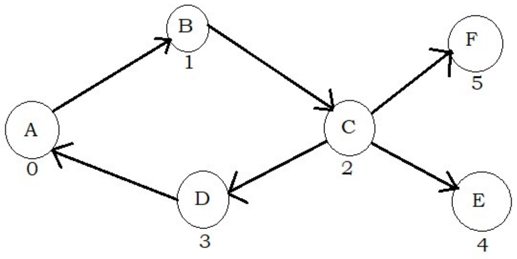
Output:



```
DFS
Enter the size of the input: 6
Enter adjacency matrix (simple graph only):
0 1 0 0 0 0
0 0 1 0 0 0
0 0 0 1 1 1
1 0 0 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0
Enter index of the start node: 0
Enter index of the required node: 3
0 1 2
POSSIBLE

Press any key to continue . . .
```

The graph used here is-



B
1
F
5
A
0
C
2
D
3
E
4