

TY CSE AY-2022-23 Sem-I

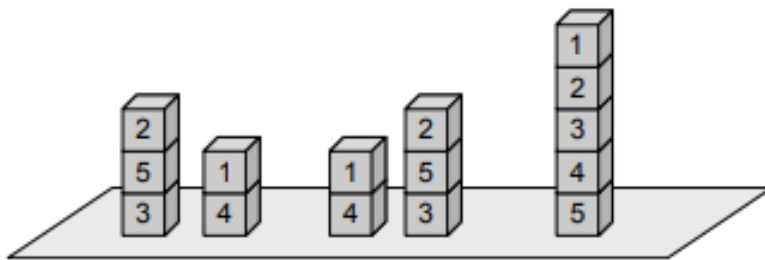
Artificial Intelligence and Machine Learning Lab

Assignment No 1

Due date- 26/08/2022

1. The block's world is a well-known toy domain used in AI for planning problems related to robots. It consists of a set of blocks of various size, shape, and color, possibly identified by a letter or by a number, and placed on a surface (e.g., on a table); the blocks can be stacked into one or more piles, possibly with some constraints (depending, e.g., on their size); the goal is to form a target set of piles starting from a given initial configuration, by moving one block at a time, either to the table or atop another pile of blocks, and only if it is not under another block. Consider a simple version of this problem, in which there are five blocks with identical shape and size, numbered from 1 to 5.

The figure below shows two possible configurations of such blocks; note that the relative position of the piles does not matter, thus the configuration on the left is equivalent to the one in the middle. Every block can be either on the table (there are no constraints on the number of piles) or atop any another block. The goal is to stack the blocks in a single pile, in the order shown in the rightmost configuration in the figure, starting from any given set of piles, by moving the smallest possible number of blocks. Only blocks at the top of the current piles can be moved, and can be placed only on the table or a top another pile.



Formulate the above version of the block's world as a search problem, by precisely defining the state space, the initial state, the goal test, the set of possible actions and the path cost (based on cost of moving blocks).

Answer:-

The state space will be made up of set of distinct arrangements of the five blocks. The current state can be represented using the specific arrangement of the blocks in each pile. The possible or legal actions would be to move a single block from the top of the pile or ground and put it either on the top of the another pile or on the ground.

Here the goal state would be defined or said to be reached when in minimum number of moves we reach the target configuration and a path cost would be associated with it and the path cost will be given by the number of moves made to reach the goal state. Thus for each move the cost would be equal to 1.

2. You start with the sequence ABABAECCCEC, or in general any sequence made from A, B, C, and E. You can transform this sequence using the following equalities: $AC = E$, $AB = BC$, $BB = E$, and $E * x = x$ for any x . For example, ABBC can be transformed into AEC, and then AC, and then E. Your goal is to produce the sequence E.

Q2. You start with the sequence ABABAECCCEC or in general any sequence made from A, B, C and E. You can transform this sequence made from A, B, C and E. You can transform this sequence using the following equalities: $AC = E$, $AB = BC$, $BB = E$ and $E * x = x$ for any x . For example ABBC can be transformed into AEC, and then AC and then E. Your goal is to produce the sequence E.

→ Answer:-

Approach 1:-

ABABAECCCEC

Step 1: Converting the AB to BC we get

ABBCAECCCEC

Step 2: Converting the BB to E

AEC AECCCEC

Step 3: Converting AE, EC, EC to corresponding

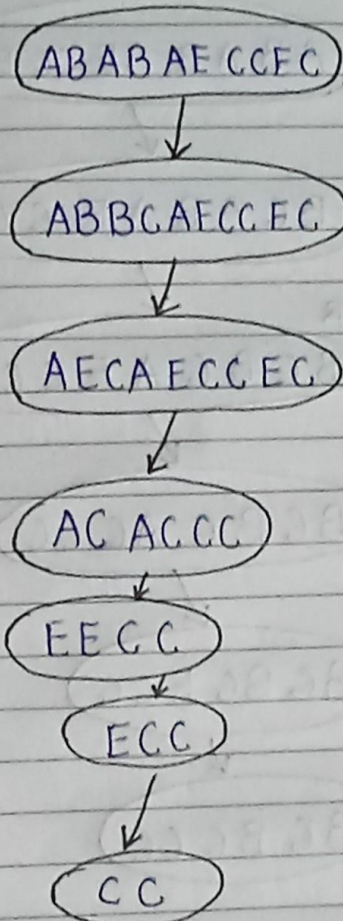
A, E, E

~~A~~ ~~A~~ ~~E~~ ~~C~~ ~~C~~ ~~A~~ ~~C~~ ~~A~~ ~~C~~ ~~C~~ ~~C~~

Step 4: Converting AC to E

~~A~~ ~~E~~ ~~E~~ ~~E~~ ~~E~~ ~~C~~ ~~C~~

Approach I:-



Finally we get CC as the reduced which is not our goal state hence the approach is discarded.

Step 5:- Converting EC to C
ECC

Step 6:- Converting EC to C
CC

So we can conclude as this is the least possible reduced by this method so we can understand that this method doesn't give solution so we need to try an alternative method.

Approach 2:-

ABABAECCEC

Step 1:- Converting AB to BC
BCABAECC

Step 2:- Converting AB to BC
BCBCAECC

Step 3:- Converting EC to C
BCBC ACCC

~~We cannot~~

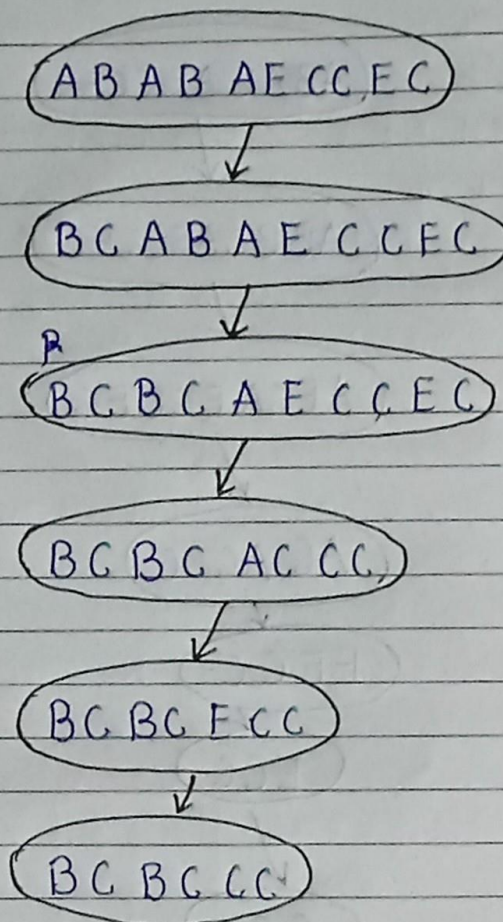
Step 4:- Converting AC to E
BCBC ECC

Step 5:- Converting EC to C

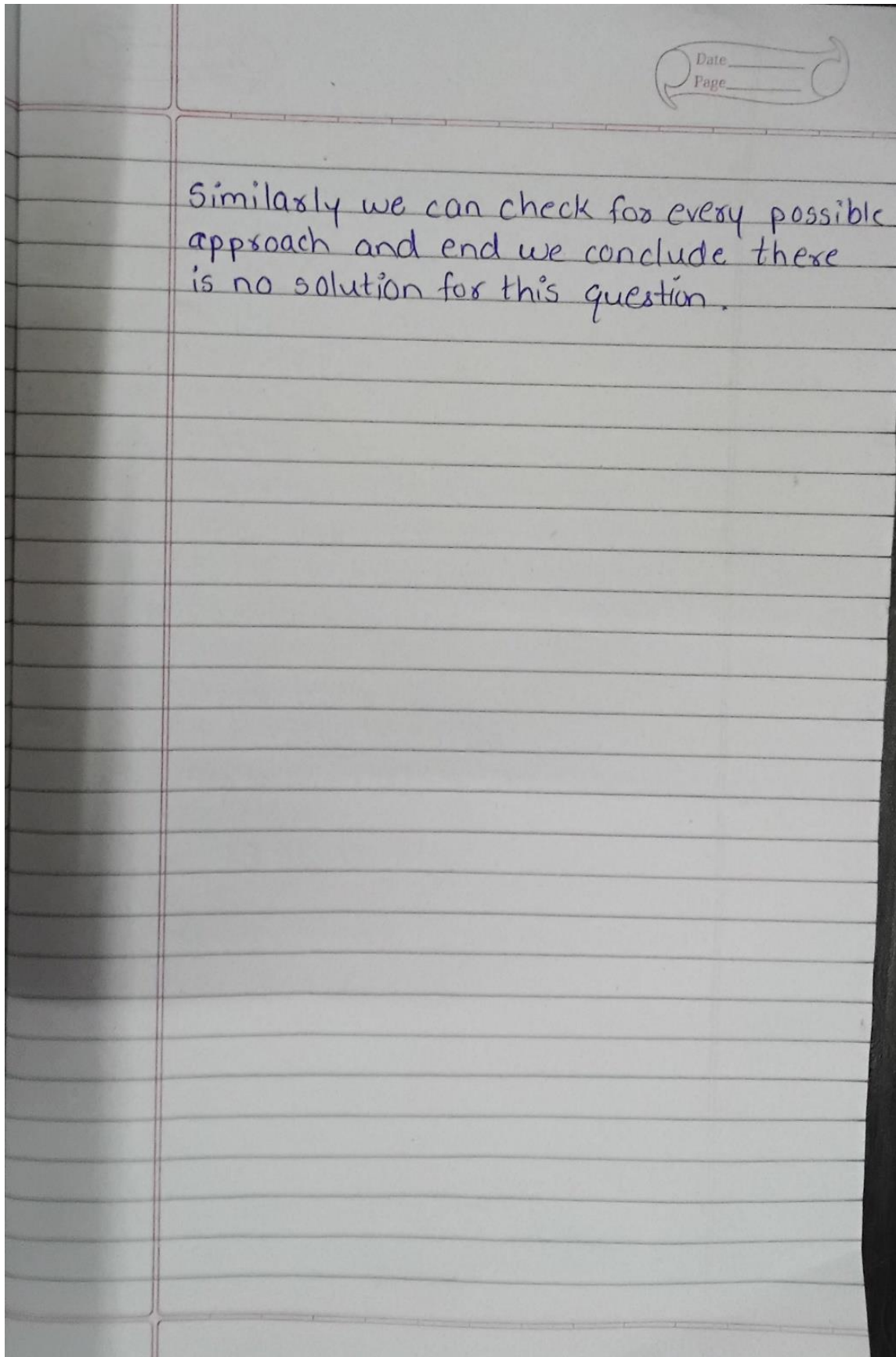
BCBCCC

We cannot further reduce the string

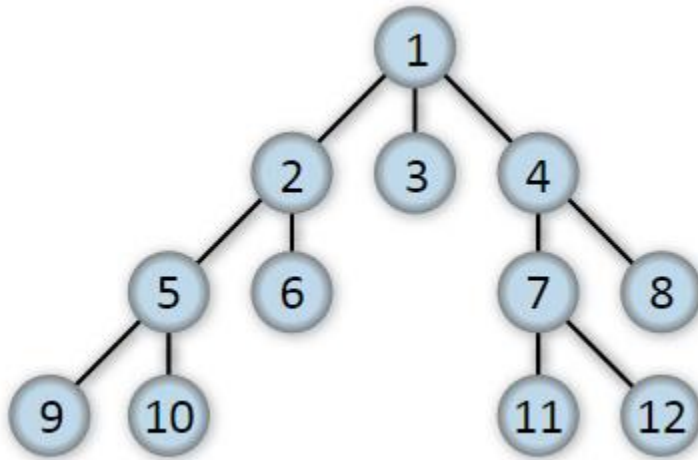
Approach 2:-



Finally we get BCBGCC as the reduced string and we cannot further reduce it hence this approach is also discarded as it is not equal to E which is our goal state.



3. For the tree given below:



If starting state is 1 and goal state is 7, implement BFS and DFS search strategies for above tree. Which searching strategy will be best for this problem? Justify your answer with proper explanation.

DFS Output:-

```
#include<bits/stdc++.h>
using namespace std;

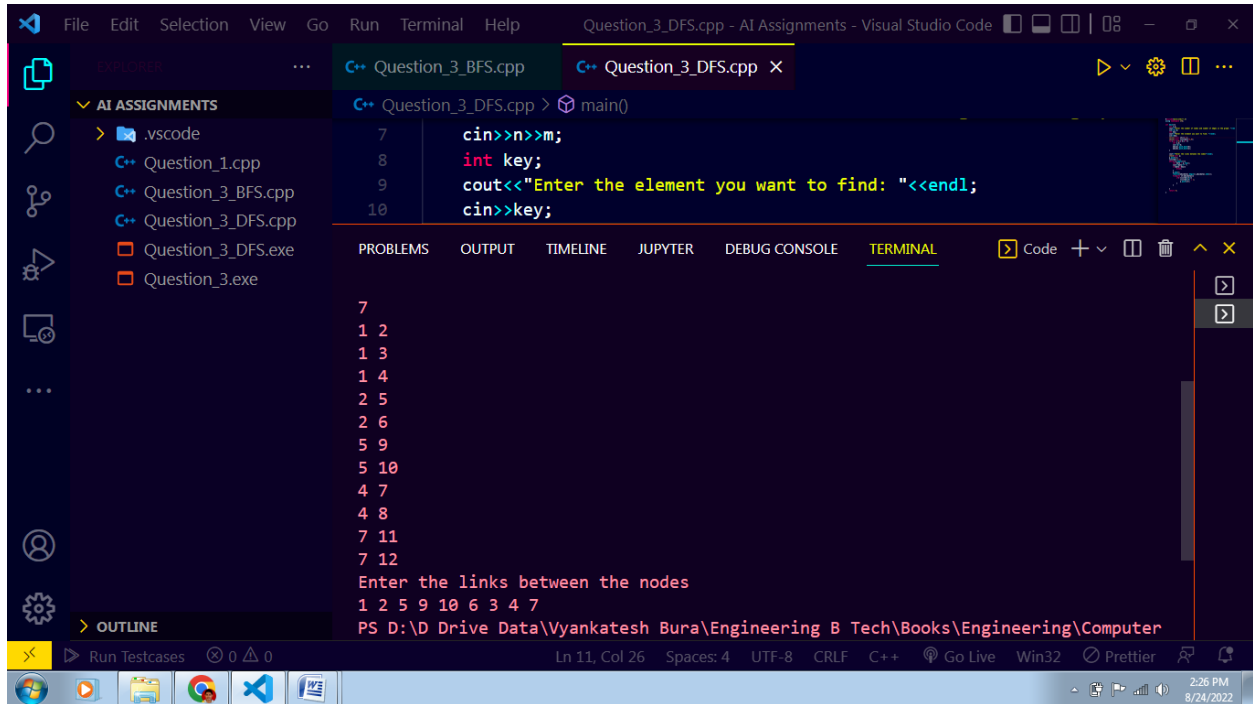
int main(){
    int n,m;
    cout<<"Enter the number of nodes and number of edges in the graph: "<<endl;
    cin>>n>>m;
    int key;
    cout<<"Enter the element you want to find: "<<endl;
    cin>>key;
    vector<int> adj[n+1];
    vector<int> visited(n+1,0);
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    cout<<"Enter the links between the nodes"<<endl;
    stack<int> q;
    q.push(1);
    visited[1]=1;
    while(!q.empty()){
        int data = q.top();
        cout<<data<<" ";
        if(data==key){
            break;
        }
        q.pop();
        reverse(adj[data].begin(),adj[data].end());
        for(auto it:adj[data]){
            if(!visited[it]){
                visited[it] = 1;
            }
        }
    }
}
```



```
        q.push(it);
    }
}

return 0;
}
```



The screenshot shows the Visual Studio Code interface with two C++ files open: Question_3_BFS.cpp and Question_3_DFS.cpp. The Explorer sidebar on the left shows a project named 'AI ASSIGNMENTS' containing several files. The main editor displays the code for Question_3_DFS.cpp, which includes a main function that takes input for the number of nodes (n) and edges (m), a key to find, and a list of edges. The output window at the bottom shows the execution results, including the input values and the edges entered.

```
Question_3_DFS.cpp - AI Assignments - Visual Studio Code
C++ Question_3_BFS.cpp C++ Question_3_DFS.cpp X
C++ Question_3_DFS.cpp > main()
7      cin>>n>>m;
8      int key;
9      cout<<"Enter the element you want to find: "<<endl;
10     cin>>key;

PROBLEMS OUTPUT TIMELINE JUPYTER DEBUG CONSOLE TERMINAL
7
1 2
1 3
1 4
2 5
2 6
5 9
5 10
4 7
4 8
7 11
7 12
Enter the links between the nodes
1 2 5 9 10 6 3 4 7
PS D:\D Drive Data\Vyankatesh Bura\Engineering B Tech\Books\Engineering\Computer

Ln 11, Col 26 Spaces: 4 UTF-8 CRLF C++ Go Live Win32 Prettier 2:26 PM 8/24/2022
```

BFS Output:-

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    int n,m;
    cout<<"Enter the number of nodes and number of edges in the graph: "<<endl;
    cin>>n>>m;
    int key;
    cout<<"Enter the element you want to find: "<<endl;
    cin>>key;
    vector<int> adj[n+1];
    vector<int> visited(n+1,0);

    cout<<"Enter the links between the nodes"<<endl;
    for(int i=0;i<m;i++){
        int a,b;
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

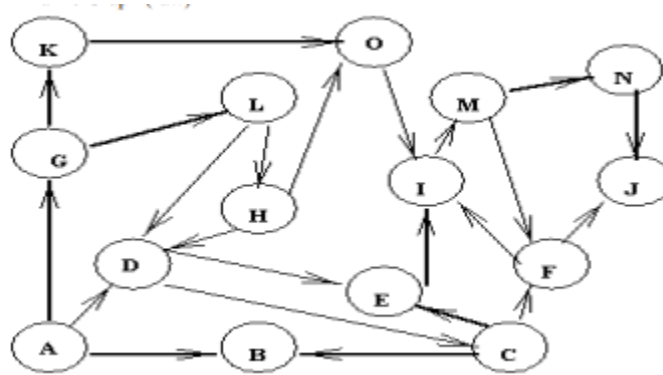
    queue<int> q;
    q.push(1);
    visited[1]=1;
    while(!q.empty()){
```

```
int data = q.front();  
cout<<data<<" ";  
if(data==key){  
    break;  
}  
q.pop();  
for(auto it:adj[data]){  
    if(!visited[it]){  
        visited[it] = 1;  
        q.push(it);  
    }  
}  
}  
return 0;  
}
```

The screenshot shows the Visual Studio Code interface with the file `Question_3.cpp` open. The code is a C++ program implementing a Breadth-First Search (BFS) algorithm on an adjacency list. The terminal output shows the sequence of nodes visited in a BFS traversal starting from node 1.

```
1 4  
2 5  
2 6  
5 9  
5 10  
4 7  
4 8  
7 11  
7 12  
1 2 3 4 5 6 7 8 9 10 11 12  
PS D:\D Drive Data\Vyankatesh B  
ce and Engineering\Third Year\A
```

4. Consider the traffic map of certain city is given below



Identify different paths using

a. BFS with start state A and goal state N

CPP Implementation:-

```
#include <bits/stdc++.h>
using namespace std;

void printpath(vector<int>& path)
{
    char ch='A';
    map<int,char>mp;

    for(int i=0;i<15;i++)
    {
        mp[i]=ch;
        ch++;
    }

    int size = path.size();
    for (int i = 0; i < size; i++)
        cout << mp[path[i]] << " ";
    cout << endl;
}

int isNotVisited(int x, vector<int>& path)
{
    int size = path.size();
    for (int i = 0; i < size; i++)
        if (path[i] == x)
            return 0;
    return 1;
}

void findpaths(vector<vector<int> >&g, int src,int dst, int v)
{
    queue<vector<int> > q;

    vector<int> path;
```

```
path.push_back(src);
q.push(path);
while (!q.empty()) {
    path = q.front();
    q.pop();
    int last = path[path.size() - 1];

    if (last == dst)
        printpath(path);

    for (int i = 0; i < g[last].size(); i++) {
        if (isNotVisited(g[last][i], path)) {
            vector<int> newpath(path);
            newpath.push_back(g[last][i]);
            q.push(newpath);
        }
    }
}

int main()
{
    int n=16;
    vector<vector<int> > g(n+1);
    // number of vertices

    int m=23;

    for(int i=0;i<m;i++)
    {
        char c1,c2;
        cin>>c1>>c2;

        int u=c1-'A';
        int v=c2-'A';

        // cout<<u<<" "<<v<<endl;

        g[u].push_back(v);
    }

    int src = 0, dst = 13;
    char sr = 'A'+0;
    char ds = 'A'+13;
    cout << "path from src " << sr
        << " to dst " << ds << " are \n";

    // function for finding the paths
    findpaths(g, src, dst, n);

    return 0;
}
```




The screenshot shows a Visual Studio Code editor with a C++ file named 'Question_no_4.cpp'. The code implements a Depth-First Search (DFS) algorithm to find all possible paths from a source state 'A' to a goal state 'N'. The graph is represented as an adjacency list. The output of the program is displayed in the console, showing multiple paths from 'A' to 'N'.

```
g L
L H
K O
I N
Expected Output: Copy
Received Output: Copy
path from src A to dst
N are
A D E I M N
A G K O I M N
A D C F I M N
A D C E I M N
A G L H O I M N
A G L D E I M N
A G L H D E I M N
A G L D C F I M N
A G L D C E I M N
A G L H D C F I M N
A G L H D C E I M N
+ New Testcase
Run All + New
Stop Help Delete
Ln 91, Col 6 Tab Size: 4 UTF-8 CRLF C++ Go Live Win32 Prettier 2:12 PM 8/26/2022
```

b. DFS with start state A and goal state N

Cpp Implementation:-

```
#include <bits/stdc++.h>
using namespace std;

void printpath(vector<int>& path)
{
    char ch='A';
    map<int,char>mp;

    for(int i=0;i<15;i++)
    {
        mp[i]=ch;
        ch++;
    }

    int size = path.size();
    for (int i = 0; i < size; i++)
        cout << mp[path[i]] << " ";
    cout << endl;
}

int isNotVisited(int x, vector<int>& path)
{
    int size = path.size();
    for (int i = 0; i < size; i++)
        if (path[i] == x)
            return 0;
    return 1;
}
```

```
void findpaths(vector<vector<int> >&g, int src,int dst, int v)
{
    stack<vector<int> > q;

    vector<int> path;
    path.push_back(src);
    q.push(path);
    while (!q.empty()) {
        path = q.top();
        q.pop();
        int last = path[path.size() - 1];

        if (last == dst)
            printpath(path);

        for (int i = 0; i < g[last].size(); i++) {
            if (isNotVisited(g[last][i], path)) {
                vector<int> newpath(path);
                newpath.push_back(g[last][i]);
                q.push(newpath);
            }
        }
    }
}

int main()
{
    int n=16;
    vector<vector<int> > g(n+1);
    // number of vertices

    int m=23;

    for(int i=0;i<m;i++)
    {
        char c1,c2;
        cin>>c1>>c2;

        int u=c1-'A';
        int v=c2-'A';

        // cout<<u<<" "<<v<<endl;

        g[u].push_back(v);
    }

    int src = 0, dst = 13;
    char sr = 'A'+0;
    char ds = 'A'+13;
    cout << "path from src " << sr
```

```
<< " to dst " << ds << " are \n";

// function for finding the paths
findpaths(g, src, dst, n);

return 0;
}
```



The screenshot shows a Visual Studio Code window with a C++ file named `Question_no_4_DFS.cpp`. The code defines a `printpath` function that takes a vector of integers representing a path and prints the corresponding nodes. The main function calls `printpath` with a path vector. The output window shows the expected output and the received output, which lists multiple paths from node A to node N.

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 void printpath(vector<int>& path)
5 {
6     char ch='A';
7     map<int,char>mp;
8
9     for(int i=0;i<path.size();i++)
10     {
11         mp[path[i]]=ch;
12         ch++;
13     }
14
15     int size = path.size();
16     for (int i = 0; i < size; i++)
17         cout << mp[path[i]] << " ";
18     cout << endl;
19 }
20
21 int isNotVisited(int x, vector<int>& path)
22 {
23     int size = path.size();
24     for (int i = 0; i < size; i++)
25         if (path[i] == x)
26             return false;
27     return true;
28 }
```

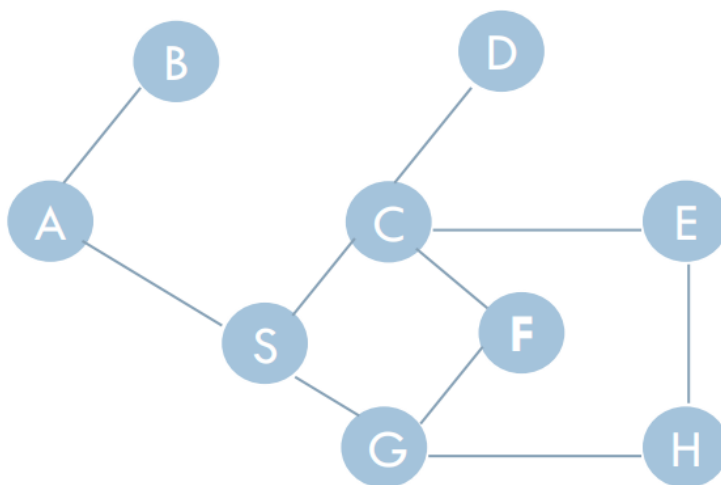
Expected Output: Copy

Received Output: Copy

path from src A to dst N are

A D E I M N
A D C E I M N
A D C F I M N
A G L D E I M N
A G L D C E I M N
A G L D C F I M N
A G L H O I M N
A G L H D E I M N
A G L H D C F I M N
A G K O I M N

5. For the graph given below



Implement BFS and DFS technique to identify a path from start node A to goal node E.

Answer:

BFS Traversal:

Implementation:

```
#include<bits/stdc++.h>
using namespace std;

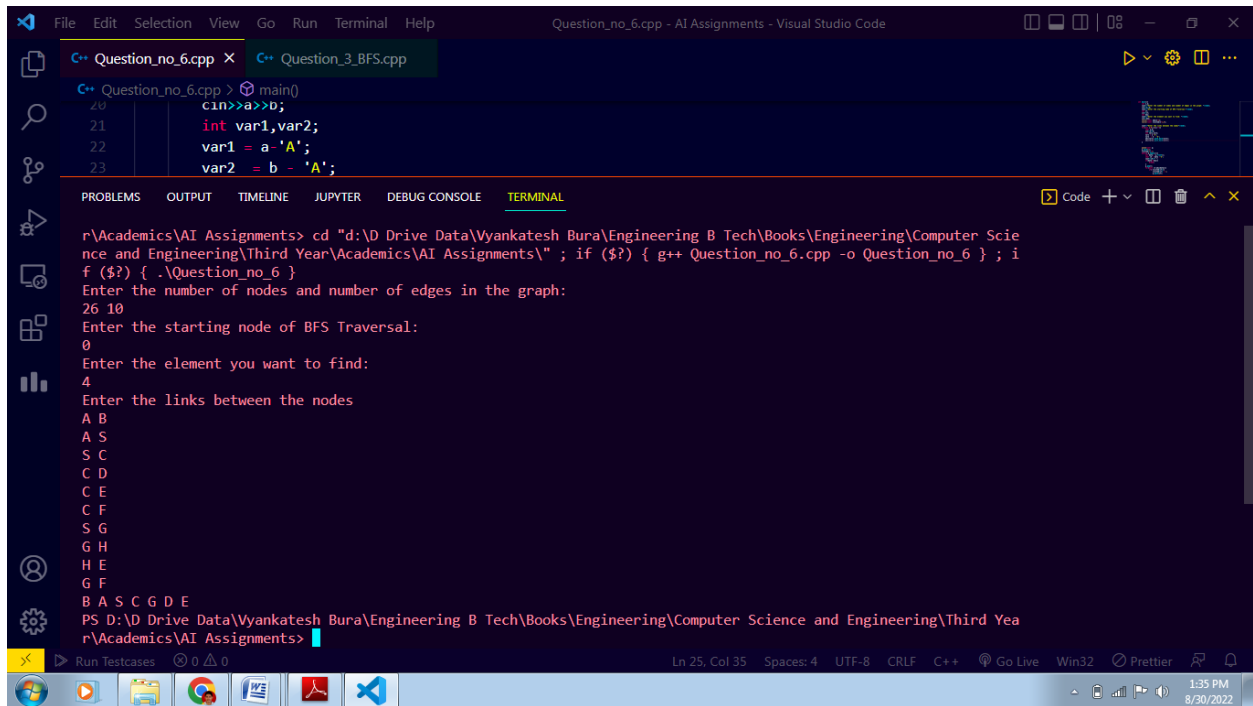
int main(){
    int n,m;
    cout<<"Enter the number of nodes and number of edges in the graph: "<<endl;
    cin>>n>>m;
    cout<<"Enter the starting node of BFS Traversal:"<<endl;
    int st;
    cin>>st;
    int key;
    cout<<"Enter the element you want to find: "<<endl;
    cin>>key;
    vector<int> adj[n+1];
    vector<int> visited(n+1,0);

    cout<<"Enter the links between the nodes"<<endl;
    for(int i=0;i<m;i++){
        char a,b;
        cin>>a>>b;
        int var1,var2;
        var1 = a-'A';
        var2 = b - 'A';
        adj[var1].push_back(var2);
        adj[var2].push_back(var1);
    }

    queue<int> q;
    q.push(1);
    visited[1]=1;
    while(!q.empty()){
        int data = q.front();
        char ch = data +'A';
        cout<<ch<<" ";
        if(data==key){
            break;
        }
        q.pop();
        for(auto it:adj[data]){
            if(!visited[it]){
                visited[it] = 1;
                q.push(it);
            }
        }
    }

    return 0;
}
```


Output:



The screenshot shows the Visual Studio Code interface with a terminal window open. The terminal displays the execution of a C++ program for BFS Traversal. The user enters the number of nodes (26) and the number of edges (10). The starting node of BFS Traversal is 0, and the element to find is 4. The links between the nodes are entered as follows:

```
A B
A S
S C
C D
C E
C F
S G
G H
H E
G F
B A S C G D E
```

The terminal output shows the program's execution, including the input and output of the BFS Traversal algorithm.

DFS Traversal:

Implementation:

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    int n,m;
    // cout<<"Enter the number of nodes and number of edges in the graph: "<<endl;
    cin>>n>>m;
    // cout<<"Enter the starting node of BFS Traversal:"<<endl;
    int st;
    cin>>st;
    int key;
    // cout<<"Enter the element you want to find: "<<endl;
    cin>>key;
    vector<int> adj[n+1];
    vector<int> visited(n+1,0);

    // cout<<"Enter the links between the nodes"<<endl;
    for(int i=0;i<m;i++){
        char a,b;
        cin>>a>>b;
        int var1,var2;
        var1 = a-'A';
        var2 = b - 'A';
        adj[var1].push_back(var2);
        adj[var2].push_back(var1);
    }
}
```

```
stack<int> q;  
q.push(1);  
visited[1]=1;  
while(!q.empty()){  
    int data = q.top();  
    char ch = data + 'A';  
    cout<<ch<<" ";  
    if(data==key){  
        break;  
    }  
    q.pop();  
    for(auto it:adj[data]){  
        if(!visited[it]){  
            visited[it] = 1;  
            q.push(it);  
        }  
    }  
}  
return 0;  
}
```

Output:

The screenshot displays the Visual Studio Code interface. On the left, the 'Testcase' panel shows a single test case that has passed. The input consists of 26 nodes (A-Z) and 4 edges (A-B, A-S, S-C, S-G). The expected and received output is 'B A S G H E'. The main editor shows the C++ code for the BFS algorithm. The code takes the number of nodes (n) and edges (m) as input, followed by the starting node (st) and the element to find (key). It then takes the edges and performs a BFS traversal starting from 'st' until it finds 'key', printing the nodes in the order they are visited.

```
3  
4 int main(){  
5     int n,m;  
6     // cout<<"Enter the number of nodes and number of edges in the graph: "<<endl;  
7     cin>>n>>m;  
8     // cout<<"Enter the starting node of BFS Traversal:"<<endl;  
9     int st;  
10    cin>>st;  
11    int key;  
12    // cout<<"Enter the element you want to find: "<<endl;  
13    cin>>key;  
14    vector<int> adj[n+1];  
15    vector<int> visited(n+1,0);  
16  
17    // cout<<"Enter the links between the nodes"<<endl;  
18    for(int i=0;i<m;i++){  
19        char a,b;  
20        cin>>a>>b;  
21        int var1,var2;  
22        var1 = a - 'A';  
23        var2 = b - 'A';  
24        adj[var1].push_back(var2);  
25        adj[var2].push_back(var1);  
26    }  
27  
28  
29    stack<int> q;  
30
```
