Q19)

#include <iostream>

using namespace std;

class Graph

{

int n;

int \*\*arr;

public:

Graph(int a)

{

n = a;

arr = new int \*[n];

for (int i = 0; i < n; i++)

arr[i] = new int[n];

}

void inputGraph()

{

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

cin >> arr[i][j];

}

}

bool isEulerPath(int path[], int pathSize)

{

int visited[n][n];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

visited[i][j] = 0;

}

int u, v;

for (int i = 0; i < pathSize - 1; i++)

{

u = path[i];

v = path[i + 1];

if (!arr[u][v] || !arr[v][u])

return false;

else if (visited[u][v])

return false;

// visit/mark the undirected edge b/w u and v

visited[u][v] = 1;

visited[v][u] = 1;

}

// check for any unvisited edges which were present in graph but were not visited

for (int i = 0; i < n; i++)

{

for (int j = i + 1; j < n; j++)

{

bool isEdge = arr[i][j] && arr[j][i];

bool isVisited = visited[i][j] && visited[j][i];

if (isEdge && !isVisited)

return false;

}

}

return true;

}

bool isEulerCircuit(int path[], int pathSize)

{

return isEulerPath(path, pathSize) && (path[0] == path[pathSize - 1]);

}

};

int main()

{

Graph G(4);

G.inputGraph();

int path[] = {1, 0, 2, 3, 1,2};

int s = sizeof(path) / sizeof(path[0]);

cout << G.isEulerPath(path, s) << endl;

return 0;

}

/\*

input-

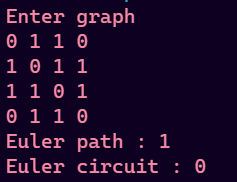
0 1 1 0

1 0 1 1

1 1 0 1

0 1 1 0

\*/



Q20)

#include <iostream>

using namespace std;

// l = (m − 1)i + 1

class M\_AryTree

{

int i, l, m;

public:

M\_AryTree(int i, int m)

{

this->i = i;

this->m = m;

l = (m - 1) \* i + 1;

}

int getLeaves()

{

return l;

}

};

int main()

{

int i, m;

cout << "Enter m (of m-ary tree) and number of internal nodes " << endl;

cin >> m >> i;

M\_AryTree tree(i, m);

cout << "Number of leaves : " << tree.getLeaves();

return 0;

}

