L22 : Graph1

1-Tut : Code : BFS Traversal

Send Feedback

Given an undirected graph G(V, E), print its BFS traversal.

Here you need to consider that you need to print BFS path starting from vertex 0 only.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

Note :

1. Take graph input in the adjacency matrix.
2. Handle for Disconnected Graphs as well

Input Format :

Line 1: Two Integers V and E (separated by space)
Next 'E' lines, each have two space-separated integers, 'a' and 'b', denoting that there exists an edge between Vertex 'a' and Vertex 'b'.

Output Format :

BFS Traversal (separated by space)

Constraints :

2 <= V <= 1000

1 <= E <= 1000

Sample Input 1:

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

Sample Output 1:

```
0 1 3 2
```

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

void printBFS(int **edges, int n, int sv, bool *visited)
{

    queue<int> pendingVertices;
    pendingVertices.push(sv);
    visited[sv] = true;

    while (!pendingVertices.empty())
    {

        int currentVertex = pendingVertices.front();
        pendingVertices.pop();
        cout << currentVertex << " ";
        for (int i = 0; i < n; i++)
        {

            if (i == currentVertex)
            {
                continue;
            }

            if (edges[currentVertex][i] == 1 && !visited[i])
            {
                pendingVertices.push(i);
                visited[i] = true;
            }
        }
    }
}

void BFS(int **edges, int n)
{
    bool *visited = new bool[n];
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
    }

    for (int i = 0; i < n; i++)
    {
        if (!visited[i])
        {
            printBFS(edges, n, i, visited);
        }
    }
    delete[] visited;
}

int main()
{

    int n;
    int e;

    cin >> n >> e;

    int **edges = new int *[n];
    for (int i = 0; i < n; i++)
    {
        edges[i] = new int[n];
        for (int j = 0; j < n; j++)
        {
            edges[i][j] = 0;
        }
    }

    for (int i = 0; i < e; i++)
    {
        int f, s;
        cin >> f >> s;
        edges[f][s] = 1;
        edges[s][f] = 1;
    }

    BFS(edges, n);

    for (int i = 0; i < n; i++)
    {
        delete[] edges[i];
    }

    delete[] edges;

    return 0;
}
```

2-Tut : Code : Has Path

Send Feedback

Given an undirected graph G(V, E) and two vertices v1 and v2(as integers), check if there exists any path between them or not. Print true or false.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

Input Format :

First line will contain T(number of test cases), each test case as follow.
Line 1: Two Integers V and E (separated by space)

Next E lines : Two integers a and b, denoting that there exists an edge between vertex a and vertex b (separated by space)
Line (E+2) : Two integers v1 and v2 (separated by space)

Output Format :

true or false for each test case in a newline.

Constraints :

1 <= T <= 10
2 <= V <= 1000
1 <= E <= 1000
0 <= v1, v2 <= V-1

Sample Input 1 :

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

Sample Output 1 :

```
true
```

Sample Input 2 :

```
1
6 3
5 3
0 1
3 4
0 3
```

Sample Output 2 :

```
false
```

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

bool printBFS(int **edges, int n, int sv, bool *visited, int v2)
{

    queue<int> pendingVertices;
    pendingVertices.push(sv);
    visited[sv] = true;

    while (!pendingVertices.empty())
    {

        int currentVertex = pendingVertices.front();
        pendingVertices.pop();
        if (currentVertex == v2)
        {
            return true;
        }
        for (int i = 0; i < n; i++)
        {

            if (i == currentVertex)
            {
                continue;
            }

            if (edges[currentVertex][i] == 1 && !visited[i])
            {
                pendingVertices.push(i);
                visited[i] = true;
            }
        }
    }

    return false;
}

void BFS(int **edges, int n, int v1, int v2)
{
    bool *visited = new bool[n];
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
    }

    int ans = printBFS(edges, n, v1, visited, v2);
    if (ans == 1)
    {
        cout << "true" << endl;
    }
    else
    {
        cout << "false" << endl;
    }

    delete[] visited;
}

```

```
int main()
{

    int t;
    cin >> t;
    while (t--)
    {
        int n;
        int e;

        cin >> n >> e;

        int **edges = new int *[n];
        for (int i = 0; i < n; i++)
        {
            edges[i] = new int[n];
            for (int j = 0; j < n; j++)
            {
                edges[i][j] = 0;
            }
        }

        for (int i = 0; i < e; i++)
        {
            int f, s;
            cin >> f >> s;
            edges[f][s] = 1;
            edges[s][f] = 1;
        }

        int v1, v2;

        cin >> v1 >> v2;

        if (edges[v1][v2] == 1)
        {
            cout << "true" << endl;
        }
        else
        {
            BFS(edges, n, v1, v2);
        }

        for (int i = 0; i < n; i++)
        {
            delete[] edges[i];
        }

        delete[] edges;
    }

    return 0;
}
```

3-Tut : Code : Get Path - DFS

Send Feedback

Given an undirected graph G(V, E) and two vertices v1 and v2(as integers), find and print the path from v1 to v2 (if exists). Print nothing if there is no path between v1 and v2.

Find the path using DFS and print the first path that you encountered.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

Print the path in reverse order. That is, print v2 first, then intermediate vertices and v1 at last.

Note : Save the input graph in Adjacency Matrix.

Input Format :

First line will contain T(number of test case), each test follow as.
Line 1: Two Integers V and E (separated by space)
Next E lines : Two integers a and b, denoting that there exists an edge between vertex a and vertex b (separated by space)
Line (E+2) : Two integers v1 and v2 (separated by space)

Output Format :

Path from v1 to v2 in reverse order (separated by space) for each test case in newline.

Constraints :

1 <= T <= 10
2 <= V <= 1000
1 <= E <= 1000
0 <= v1, v2 <= V-1

Sample Input 1 :

```
1
4 4
```

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

Sample Output 1 :

```
3 0 1
```

Sample Input 2 :

```
1
6 3
5 3
0 1
3 4
0 3
```

Sample Output 2 :

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

vector<int> get_path(int **arr, int v, bool *visited, int current_vertex, int v2)
{
    if (current_vertex == v2)
    {
        vector<int> ans;
        ans.push_back(current_vertex);
        return ans;
    }
    for (int i = 0; i < v; i++)
    {
        if (!visited[i] && i != current_vertex && arr[current_vertex][i] == 1)
        {
            vector<int> ans;
            visited[i] = true;
            ans = get_path(arr, v, visited, i, v2);
            if (ans.size() != 0)
            {
                ans.push_back(current_vertex);
                return ans;
            }
        }
    }
    vector<int> ans;
    return ans;
}
int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int v, e;
        cin >> v >> e;
        int **arr = new int *[v];
        for (int i = 0; i < v; i++)
        {
            arr[i] = new int[v];
            for (int j = 0; j < v; j++)
            {
                arr[i][j] = 0;
            }
        }
        bool *visited = new bool[v];
        for (int i = 0; i < v; i++)
        {
            visited[i] = false;
        }
        while (e--)
        {
            int a, b;
            cin >> a >> b;
            arr[a][b] = 1;
            arr[b][a] = 1;
        }
        int v1, v2;
        cin >> v1 >> v2;

        visited[v1] = true;
        vector<int> ans = get_path(arr, v, visited, v1, v2);
        if (ans.size() != 0)
        {
            for (int i = 0; i < ans.size(); i++)
            {
                cout << ans[i] << " ";
            }
        }
        else
        {
        }
        cout << endl;
    }

    return 0;
}
```

4-Tut : Code : Get Path - BFS

Send Feedback

Given an undirected graph G(V, E) and two vertices v1 and v2(as integers), find and print the path from v1 to v2 (if exists). Print nothing if there is no path between v1 and v2.

Find the path using BFS and print the shortest path available.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

Print the path in reverse order. That is, print v2 first, then intermediate vertices and v1 at last.

Note : Save the input graph in Adjacency Matrix.

Input Format :

First line of input will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next E lines : Two integers a and b, denoting that there exists an edge between vertex a and vertex b (separated by space)
Line (E+2) : Two integers v1 and v2 (separated by space)

Output Format :

Path from v1 to v2 in reverse order (separated by space) for each test case in new line.

Constraints :

1 <= T <= 10
2 <= V <= 1000
1 <= E <= 1000
0 <= v1, v2 <= V-1

Sample Input 1 :

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

Sample Output 1 :

```
3 0 1
```

Sample Input 2 :

```
1
6 3
5 3
0 1
3 4
0 3
```

Sample Output 2 :

```
#include <bits/stdc++.h>
using namespace std;
void printbfs(int **arr, int v, int current_element, int v2, bool *visited)
{
    queue<int> q;
    visited[current_element] = true;
    q.push(current_element);
    map<int, int> m;
    while (!q.empty())
    {
        int current = q.front();
        if (current == v2)
        {
            break;
        }
        for (int i = 0; i < v; i++)
        {
            if (!visited[i] && arr[current][i] == 1 && i != current)
            {
                q.push(i);
                visited[i] = true;
                m[i] = current;
            }
        }
        q.pop();
        if (q.empty())
        {
            return;
        }
    }
    int i = v2;
    cout << v2 << " ";
    while (i != current_element)
    {
        cout << m[i] << " ";
        i = m[i];
    }
}
int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int v, e;
        cin >> v >> e;
        int **arr = new int *[v];
        for (int i = 0; i < v; i++)
        {
            arr[i] = new int[v];
            for (int j = 0; j < v; j++)
            {
                arr[i][j] = 0;
            }
        }
        while (e--)
        {
            int a, b;
            cin >> a >> b;
            arr[a][b] = 1;
            arr[b][a] = 1;
        }
        int v1, v2;
        cin >> v1 >> v2;
        bool *visited = new bool[v];
        for (int i = 0; i < v; i++)
        {
            visited[i] = false;
        }
        printbfs(arr, v, v1, v2, visited);
        cout << endl;
    }
}
```

5-Tut : Code : Is Connected ?

Send Feedback

Given an undirected graph G(V,E), check if the graph G is connected graph or not.

V is the number of vertices present in graph G and vertices are numbered from 0 to V-1.

E is the number of edges present in graph G.

Input Format :
First line will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next 'E' lines, each have two space-separated integers, 'a' and 'b', denoting that there exists an edge between Vertex 'a' and Vertex 'b'.
Output Format :
Print "true" or "false" for each test case in new line
Constraints :
1 <= T <= 10
2 <= V <= 1000
1 <= E <= 1000
Sample Input 1:
```
1
4 4
0 1
0 3
1 2
2 3
```
Sample Output 1:
```
true
```
Sample Input 2:
```
1
4 3
0 1
1 3
0 3
```
Sample Output 2:
```
false
```
Sample Output 2 Explanation

The graph is not connected, even though vertices 0,1 and 3 are connected to each other but there isn’t any path from vertices 0,1,3 to vertex 2.

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

void printBFS(int **edges, int n, int sv, bool *visited)
{

    queue<int> pendingVertices;
    pendingVertices.push(sv);
    visited[sv] = true;

    while (!pendingVertices.empty())
    {

        int currentVertex = pendingVertices.front();
        pendingVertices.pop();
        for (int i = 0; i < n; i++)
        {

            if (i == currentVertex)
            {
                continue;
            }

            if (edges[currentVertex][i] == 1 && !visited[i])
            {
                pendingVertices.push(i);
                visited[i] = true;
            }
        }
    }
}

void BFS(int **edges, int n)
{
    bool *visited = new bool[n];
    for (int i = 0; i < n; i++)
    {
        visited[i] = false;
    }

    printBFS(edges, n, 0, visited);

    int flag = 0;
    for (int i = 0; i < n; i++)
    {
        if (!visited[i])
        {
            flag = 1;
            break;
        }
    }

    if (flag == 1)
    {
        cout << "false" << endl;
    }
    else
    {
        cout << "true" << endl;
    }
    delete[] visited;
}

int main()
{

    int t;
    cin >> t;
    while (t--)
    {
        int n;
        int e;

        cin >> n >> e;

        int **edges = new int *[n];
        for (int i = 0; i < n; i++)
        {
            edges[i] = new int[n];
            for (int j = 0; j < n; j++)
            {
                edges[i][j] = 0;
            }
        }

        for (int i = 0; i < e; i++)
        {
            int f, s;
            cin >> f >> s;
            edges[f][s] = 1;
            edges[s][f] = 1;
        }

        BFS(edges, n);

        for (int i = 0; i < n; i++)
        {
            delete[] edges[i];
        }
```

```

            delete[] edges;
        }

        return 0;
    }
```

6-Tut : Code : All connected components

Send Feedback

Given an undirected graph G(V,E), find and print all the connected components of the given graph G.

V is the number of vertices present in graph G and vertices are numbered from 1 to V.

E is the number of edges present in graph G.

You need to take input in main and create a function which should return all the connected components.

And then print them in the main, not inside function.

Print different components in new line. And each component should be printed in increasing order (separated by space). Order of different components doesn't matter.

Input Format :

First line of input will contain T(number of test case), each test case follows as.
Line 1: Two Integers V and E (separated by space)
Next 'E' lines, each have two space-separated integers, 'a' and 'b', denoting that there exists an edge between Vertex 'a' and Vertex 'b'.

Output Format :

For each test case and each connected components print the connected components in sorted order in new line.
Order of connected components doesn't matter (print as you wish).

Constraints :

2 <= V <= 10000
1 <= E <= 10000

Sample Input 1:

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

Sample Output 1:

```
1 2

4 3
```

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

#define ll int

int arr[10001][10001];
```

```

struct query
{
    int aa;
    int bb;
};

struct node
{
    vector<int> nb;
};

vector<ll> vector_creator(ll n, ll starting_vertex, bool *visited, int max, node *nodes)
{
    vector<ll> temp;
    queue<ll> q;

    q.push(starting_vertex);

    visited[starting_vertex] = true;

    while (!q.empty())
    {
        ll current_element = q.front();
        temp.push_back(current_element);
        q.pop();
        vector<int> t1;
        t1 = nodes[current_element].nb;

        for (int i = 0; i < t1.size(); i++)
        {
            if (t1[i] == current_element)
            {
                continue;
            }
            if (!visited[t1[i]])
            {
                q.push(t1[i]);
                visited[t1[i]] = true;
            }
        }
    }
    return temp;
}
```

```

vector<vector<ll>> vector_return(ll n, int max, node *nodes)
{

    bool *visited = new bool[n + 1];
    for (ll i = 0; i <= n; i++)
    {
        visited[i] = false;
    }

    vector<vector<ll>> super;
    vector<ll> tempo;
    for (ll i = 1; i <= max; i++)
    {
        if (!visited[i])
        {
            tempo = vector_creator(n, i, visited, max, nodes);
            if (tempo.size() != 0)
            {
                super.push_back(tempo);
            }
        }
    }
    for (int i = max + 1; i <= n; i++)
    {
        vector<int> t1;
        t1.push_back(i);
        super.push_back(t1);
    }

    return super;
}

int main()
{
    ll t;
    cin >> t;
    while (t--)
    {
        ll v, e;
        cin >> v >> e;
        int max = 0;

        node *nodes = new node[v + 1];
```

```cpp
    for (int i = 0; i <= v; i++)
    {
        nodes[i].nb.push_back(i);
    }

      query q[e];
      for (int i = 0; i < e; i++)
      {

          ll a, b;
          cin >> a >> b;

          arr[a][b] = 1;
          arr[b][a] = 1;

          q[i].aa = a;
          q[i].bb = b;

          nodes[a].nb.push_back(b);
          nodes[b].nb.push_back(a);

          if (max < a)
          {
              max = a;
          }
          if (max < b)
          {
              max = b;
          }
      }

      vector<vector<ll>> super;

      super = vector_return(v, max, nodes);

      for (ll i = 0; i < super.size(); i++)
      {
          sort(super[i].begin(), super[i].end());
          for (ll j = 0; j < super[i].size(); j++)
          {
              cout << super[i][j] << " ";
          }
          cout << endl;
      }
```

```
        for (int i = 0; i < e; i++)
        {

            arr[q[i].aa][q[i].bb] = 0;
            arr[q[i].bb][q[i].aa] = 0;
        }
    }
}
```

7-Ass : Islands

Send Feedback

An island is a small piece of land surrounded by water . A group of islands is said to be connected if we can reach from any given island to any other island in the same group . Given N islands (numbered from 0 to N - 1) and M pair of integers (u and v) denoting island, u is connected to island v and vice versa. Can you count the number of connected groups of islands?

Input Format:

The first line of input will contain T(number of test cases), each test case follows as.
Line 1: Two Integers N and M (separated by space)
Next 'M' lines, each have two space-separated integers, 'u' and 'v', denoting that there exists an edge between Vertex 'u' and Vertex 'v'.

Output Format:

Print number of Islands for each test case in new line.

Constraints:

1 <= T <= 10
1 <= N <= 1000
1 <= M <= min((N*(N-1))/2, 1000)
0 <= u[i] ,v[i] < N

Output Return Format :

The count the number of connected groups of islands

Sample Input :

```
1
2 1
0 1
```

Sample Output :

```
1
```

```
#include <bits/stdc++.h>
using namespace std;
void visited_filler(int **arr, int n, int starting_vertex, bool *visited)
{
    queue<int> q;
    q.push(starting_vertex);
    visited[starting_vertex] = true;
    while (!q.empty())
    {
        int current_element = q.front();
        for (int i = 0; i < n; i++)
        {
            if (i == current_element)
            {
                continue;
            }
            if (visited[i])
            {
                continue;
            }
            if (arr[current_element][i] == 1)
            {
                q.push(i);
                visited[i] = true;
            }
        }
        q.pop();
    }
}

int main()
{

    int t;
    cin >> t;
    while (t--)
    {

        int N, M;
        cin >> N >> M;

        int **arr = new int *[N + 1];
        for (int i = 0; i < N + 1; i++)
        {
            arr[i] = new int[N + 1];
            for (int j = 0; j < N + 1; j++)
            {
                arr[i][j] = 0;
            }
        }
    
        for (int i = 0; i < M; i++)
        {
            int u, v;
            cin >> u >> v;
            arr[u][v] = 1;
            arr[v][u] = 1;
        }

        bool *visited = new bool[N + 1];
        for (int i = 0; i < N + 1; i++)
        {
            visited[i] = false;
        }

        int count = 0;

        for (int i = 0; i < N; i++)
        {
            if (!visited[i])
            {
                visited_filler(arr, N + 1, i, visited);
                count += 1;
            }
        }
        cout << count << endl;
    }

    return 0;
}
```

8-Ass : Coding Ninjas

Send Feedback

Given a NxM matrix containing Uppercase English Alphabets only. Your task is to tell if there is a path in the given matrix which makes the sentence “CODINGNINJA” .

There is a path from any cell to all its neighbouring cells. A neighbour may share an edge or a corner.

Input Format :

First line will contain T(number of test case), each test case follows as.
Line 1 : Two space separated integers N and M, where N is number of rows and M is number of columns in the matrix.
Next N lines : N rows of the matrix. First line of these N line will contain 0th row of matrix, second line will contain 1st row and so on

Assume input to be 0-indexed based

Output Format :
Print 1 if there is a path which makes the sentence “CODINGNINJA” else print 0, for each test case in a new line
Constraints :
1 <= T <= 10
1 <= N <= 1000
1 <= M <= 1000
Sample Input :
1
2 11
CXDXNXNXNXA
XOXIXGXIXJX
Sample Output :

1

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

#define MAXN 500

int validPoint(int x, int y, int n, int m)
{

    return (x >= 0 && x < n && y >= 0 && y < m);
}

bool dfs(char **board, vector<vector<bool>> &used, string &word, int x, int y, int
wordIndex, int n, int m)
{

    if (wordIndex == 11)
    {
        return true;
    }

    used[x][y] = true;
    bool found = false;

    int dxdy[8][2] = {{-1, -1},
                      {-1, 0},
                      {-1, -1},
                      {0, -1},
                      {0, 1},
                      {1, -1},
                     {1, 0},
                     {1, 1}};

    for (int i = 0; i < 8; ++i)
    {

        int newX = x + dxdy[i][0];
        int newY = y + dxdy[i][1];

        if (validPoint(newX, newY, n, m) && board[newX][newY] == word[wordIndex] && !used[newX][newY])
        {

            found = found | dfs(board, used, word, newX, newY, wordIndex + 1, n, m);
        }
    }

    used[x][y] = false;

    return found;
}

bool hasPath(char **board, int n, int m)
{

    bool foundPath = false;
    string word = "CODINGNINJA";
    vector<vector<bool>> used(n, vector<bool>(m, false));

    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < m; j++)
        {

            if (board[i][j] == word[0])
            {
                foundPath = dfs(board, used, word, i, j, 1, n, m);
                if (foundPath)
                    break;
            }
        }

        if (foundPath)
            break;
```

```cpp
    }

    return foundPath;
}

int main()
{

    int t;
    cin >> t;
    while (t--)
    {
        int N, M;
        cin >> N >> M;

        char **arr = new char *[N];
        for (int i = 0; i < N; i++)
        {
            arr[i] = new char[M];
            for (int j = 0; j < M; j++)
            {
                arr[i][j] = ' ';
            }
        }

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

        {
            for (int j = 0; j < M; j++)

            {
                cin >> arr[i][j];
            }
        }

        bool ans = hasPath(arr, N, M);

        if (ans)
        {
            cout << 1 << endl;
        }
        else
        {
            cout << 0 << endl;
        }
    }

    return 0;
}
```

9-Ass : Connecting Dots

Send Feedback

Gary has a board of size NxM. Each cell in the board is a coloured dot. There exist only 26 colours denoted by uppercase Latin characters (i.e. A,B,...,Z). Now Gary is getting bore and wants to play a game. The key of this game is to find a cycle that contain dots of same colour. Formally, we call a sequence of dots d1, d2, ..., dk a cycle if and only if it meets the following condition:

1. These k dots are different: if i ≠ j then di is different from dj.
2. k is at least 4.
3. All dots belong to the same colour.
4. For all 1 ≤ i ≤ k - 1: di and di + 1 are adjacent. Also, dk and d1 should also be adjacent. Cells x and y are called adjacent if they share an edge.

Since Gary is colour blind, he wants your help. Your task is to determine if there exists a cycle on the board.

Assume input to be 0-indexed based.

Input Format :

Line 1 : Two integers N and M, the number of rows and columns of the board

Next N lines : a string consisting of M characters, expressing colors of dots in each line. Each character is an uppercase Latin letter.

Output Format :

Return 1 if there is a cycle else return 0

Constraints :

2 ≤ N, M ≤ 400

Sample Input :

```
3 4
AAAA
ABCA
AAAA
```

Sample Output :

```
1
```

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, char> pi;

bool vis[51][51];
char graph[51][51];
```

```
int dx[] = {1, -1, 0, 0}; // only left, right , up ,down are allowed here
int dy[] = {0, 0, 1, -1};
int m, n;
bool ok = false;

void dfs(int i, int j, int frmi, int frmj, char co)
{
    if (i < 1 || j < 1 || i > m || j > n)
        return;

    if (graph[i][j] != co)
        return;

    if (vis[i][j])
    {
        ok = true;
        return;
    }

    vis[i][j] = true;

    for (int y = 0; y < 4; y++)
    {
        int nxti = i + dx[y];
        int nxtj = j + dy[y];

        if (nxti == frmi && nxtj == frmj)
            continue; // it doesn't go back from where it comes from

        dfs(nxti, nxtj, i, j, co);
    }
}

int main()

{

    memset(vis, false, sizeof(vis));
    int x, y, u, v;
    char c;
    cin >> m >> n;
    for (int i = 1; i <= m; i++)

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

        {
            cin >> graph[i][j];
        }
    }
    for (int i = 1; i <= m; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            char z = graph[i][j];
            // cout<<z<<endl;
            if (!vis[i][j])
            {
                dfs(i, j, -1, -1, z);
                if (ok)
                {
                    cout << "1" << endl;
                    return 0;
                }
            }
        }
    }
    cout << "0" << endl;
}
```

10-Ass : Largest Piece

Send Feedback

Its Gary's birthday today and he has ordered his favourite square cake consisting of '0's and '1's . But Gary wants the biggest piece of '1's and no '0's . A piece of cake is defined as a part which consist of only '1's, and all '1's share an edge with eachother on the cake. Given the size of cake N and the cake , can you find the size of the biggest piece of '1's for Gary ?

Input Format :

First line will contain T(number of test cases), each test case follows as.
Line 1 : An integer N denoting the size of cake
Next N lines : N characters denoting the cake

Output Format :

Print the size of the biggest piece of '1's and no '0'sfor each test case in a newline.

Constraints:

1 <= T <= 10
1 <= N <= 1000

Sample Input :

```
1
2
11
01
```

Sample Output :

```
3
```

```
#include <bits/stdc++.h>
using namespace std;
#define NMAX 10005
char cake[NMAX][NMAX];

void dfs(char cake[NMAX][NMAX], int n, int &k, int i, int j)
{
    k++;
    cake[i][j] = '0';

    if (i + 1 < n && cake[i + 1][j] == '1')
        dfs(cake, n, k, i + 1, j);
    if (i - 1 >= 0 && cake[i - 1][j] == '1')
        dfs(cake, n, k, i - 1, j);
    if (j + 1 < n && cake[i][j + 1] == '1')
        dfs(cake, n, k, i, j + 1);
    if (j - 1 >= 0 && cake[i][j - 1] == '1')
        dfs(cake, n, k, i, j - 1);
}

int solve(int n, char cake[NMAX][NMAX])
{
    // Write your code here .
    int ans = 0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {

            if (cake[i][j] == '1')
            {
                int k1 = 0;
                dfs(cake, n, k1, i, j);
                ans = max(ans, k1);
            }
        }
    }
    return ans;
}

int main()
{

    // write your code here
    int t;
    cin >> t;
    while (t--)
    {
        int n;
        cin >> n;
        for (int i = 0; i < n; i++)
        {
            scanf("%s", cake[i]);
        }
        cout << solve(n, cake) << endl;
    }

    return 0;
}
```

11-Ass : 3 Cycle

Send Feedback

Given a graph with N vertices (numbered from 1 to N) and Two Lists (U,V) of size M where (U[i],V[i]) and (V[i],U[i]) are connected by an edge , then count the distinct 3-cycles in the graph. A 3-cycle PQR is a cycle in which (P,Q), (Q,R) and (R,P) are connected an edge.

Input Format :

Line 1 : Two integers N and M
Line 2 : List u of size of M
Line 3 : List v of size of M

Return Format :

The count the number of 3-cycles in the given Graph

Constraints :

1<=N<=100
1<=M<=(N*(N-1))/2
1<=u[i],v[i]<=N

Sample Input:

```
3 3
1 2 3
2 3 1
```

Sample Output:

```
1
```

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

int solve(int n, int m, vector<int> u, vector<int> v)
{
    // Write your code here .
    unordered_map<int, unordered_set<int>> adj;

    for (int i = 0; i < m; i++)
    {
        adj[u[i]].insert(v[i]);
        adj[v[i]].insert(u[i]);
    }

    int thcycle = 0;
    for (int i = 1; i <= n; i++)
    {
        int node = i;

        for (int p = 1; p < n; p++)
        {
            for (int q = p + 1; q <= n; q++)
            {

                if (adj[node].count(p) == 0)
                    break;
                if (adj[node].count(q) == 0)
                    continue;

                if (adj[node].count(p) && adj[node].count(q))
                {
                    if (adj[p].count(q))
                        thcycle++;
                }
            }
        }
    }
    return thcycle / 3;
}

int main()
```

```
{
    int n, m;
    vector<int> u, v;
    cin >> n >> m;
    for (int i = 0; i < m; i++)
    {
        int x;
        cin >> x;
        u.push_back(x);
    }
    for (int i = 0; i < m; i++)
    {
        int x;
        cin >> x;
        v.push_back(x);
    }
    cout << solve(n, m, u, v) << endl;
}
```