Câu 1:

* Đồ thị : 0-1 0-2 0-3 1-3 1-4 2-5 2-9 3-6 4-7 4-8 5-8 5-9 6-7 6-9 7-8

Ta thấy đồ thị trên có đỉnh 0 bậc 3 là bậc lẻ => không có chu trình Euler

Ta tìm được một chu trình Hamilton sau:

0-2-9-5-8-7-6-3-0-1-4

* Đồ thị: 0-1 0-2 0-3 1-3 0-3 2-5 5-6 3-6 4-7 4-8 5-8 5-9 6-7 6-9 8-8

Ta thấy đồ thị trên có đỉnh 0 bậc 3 là bậc lẻ => không có chu trình Euler

Ta tìm được một chu trình Hamilton sau:

3-1-0-2-5-9-6-7-4-8

* Đồ thị: 0-1 1-2 1-3 0-3 0-4 2-5 2-9 3-6 4-7 4-8 5-8 5-9 6-7 6-9 7-8

Ta thấy đồ thị trên có đỉnh 0 bậc 3 là bậc lẻ => không có chu trình Euler

Ta tìm được một chu trình Hamilton sau:

0-1-3-6-5-2-5-8-7-4

* Đồ thị: 4-1 7-9 6-2 7-3 5-0 0-2 0-8 1-6 3-9 6-3 2-8 1-5 9-8 4-5 4-7

Ta thấy đồ thị trên có đỉnh 4 bậc 3 là bậc lẻ => không có chu trình Euler

Ta tìm được một chu trình Hamilton sau:

1-4-7-9-3-6-2-8-5-0

Câu 2:

#include <bits/stdc++.h>

using namespace std;

long long Cnk(long long n, long long k)

{

if (k == 0 || k == n)

{

return 1;

}

return Cnk(n - 1, k - 1) + Cnk(n - 1, k);

}

long long graphNum(long long E, long long V)

{

return 2 \* E \* Cnk((V \* (V - 1) / 2), E);

}

int main()

{

long long v, e;

cin >> v >> e;

cout << graphNum(v, e) << endl;

}

Câu 3:

#include <iostream>

#include <vector>

#include <unordered\_set>

using namespace std;

int direction(int p, int q, int r)

{

int val = (q - p) \* (r - q);

if (val == 0)

{

return 0;

}

else if (val > 0)

{

return 1;

}

else

{

return 2;

}

}

bool inALine(int p, int q, int r)

{

if (q >= min(p, r) && q <= max(p, r))

{

return true;

}

return false;

}

bool doEdgesIntersect(int u1, int v1, int u2, int v2)

{

int direction1 = direction(u1, v1, u2);

int direction2 = direction(u1, v1, v2);

int direction3 = direction(u2, v2, u1);

int direction4 = direction(u2, v2, v1);

if (direction1 != direction2 && direction3 != direction4)

{

return true;

}

if (direction1 == 0 && inALine(u1, u2, v1))

{

return true;

}

if (direction2 == 0 && inALine(u1, v2, v1))

{

return true;

}

if (direction3 == 0 && inALine(u2, u1, v2))

{

return true;

}

if (direction4 == 0 && inALine(u2, v1, v2))

{

return true;

}

return false;

}

int Parallel(const vector<vector<int>> &graph)

{

int count = 0;

for (int u = 0; u < graph.size(); u++)

{

for (int v : graph[u])

{

for (int x = u + 1; x < graph.size(); x++)

{

for (int y : graph[x])

{

if (u != x && u != y && v != x && v != y)

{

if (doEdgesIntersect(u, v, x, y))

{

count++;

}

}

}

}

}

}

return count;

}

int main()

{

vector<vector<int>> graph;

int n;

cout << "Nhập số cạnh của đồ thị";

cin >> n;

cout << "Nhập tạo độ các cạnh";

while (n != 0)

{

int a, b;

cin >> a >> b;

graph.push\_back(vector<int>{a, b});

n--;

}

int parallelEdgesCount = Parallel(graph);

cout << "Number of parallel edges: " << parallelEdgesCount << endl;

return 0;

}

Câu 4:

Để chứng minh rằng một đồ thị là đồ thị hai màu (bipartite) khi và chỉ khi nó không chứa chu trình độ dài lẻ, ta sẽ sử dụng phương pháp phản chứng.

* Giả sử đồ thị G là một đồ thị hai màu và chứa một chu trình độ dài lẻ. Ta chứng minh dẫn đến mâu thuẫn.

Đồ thị hai màu, được chia thành hai tập đỉnh phân biệt, ký hiệu là U và V. Giả sử đồ thị G có một chu trình độ dài lẻ, nghĩa là một chu trình mà số lượng cạnh là số lẻ.

Khi xét các cạnh trong chu trình này, ta thấy rằng các đỉnh xen kẽ nằm trong tập U và tập V => số lượng cạnh nối từ đỉnh từ tập U sang V bằng số lượng cạnh nối từ đỉnh từ tập V sang U. Nên số lượng cạnh nối sẽ luôn là chẵn nếu xuất hiện 1 cạnh nữa tạo chu trình thì phải nối cùng tập => cùng màu.

Do đó, đồ thị không thể cùng lúc là đồ thị hai màu và chứa chu trình độ dài lẻ. Điều này chứng minh rằng một đồ thị là đồ thị hai màu khi và chỉ khi nó không chứa chu trình độ dài lẻ.

Câu 5:

Chứng minh rằng một đồ thị không có điểm articulation sẽ là một đồ thị biconnected có thể được thực hiện như sau:

Giả sử đồ thị G không có điểm articulation. Chọn hai đỉnh bất kỳ s và t trong đồ thị và xác định một đường đi nối chúng, gọi đường đi này là đường đi P từ s đến t.

Vì không có điểm articulation trong G, nên việc xóa bất kỳ đỉnh nào trên đường đi P cũng không làm đồ thị mất tính liên thông. Điều này có nghĩa là mọi đỉnh trên đường đi P có thể được bắt đầu từ s hoặc kết thúc tại t mà không làm đồ thị G mất tính liên thông.

Bây giờ, để chứng minh rằng đồ thị G là biconnected, chúng ta cần xây dựng hai đường đi không giao nhau từ s đến t.

Xét đường đi P từ s đến t. Vì không có điểm articulation nào trên đường đi này, chúng ta có thể chia đường đi P thành hai đường đi không giao nhau, gọi chúng là đường đi Q và R. Điểm chung giữa Q và R sẽ là t, và không có đỉnh nào khác trên Q và R ngoài s và t.

Do đó, chúng ta đã xây dựng được hai đường đi không giao nhau từ s đến t trên đồ thị G, không làm mất tính liên thông của nó. Vì vậy, đồ thị không có điểm articulation sẽ là đồ thị biconnected.

Câu 6:

#include "bits/stdc++.h"

using namespace std;

int n,m;

vector<int> a[1005];

bool visited[1005];

vector<pair<int, int>> edge;

void DFS(int u, int s, int t) {

visited[u] = true;

for (auto v : a[u]) {

//Neu xet u,v == s,t thi khong xet canh nay

if (u == s && v == t || u == t && v == s) {

continue;

}

if (!visited[v])

DFS(v,s,t);

}

}

int tplt(int s, int t) {

int count = 0;

memset(visited,false,sizeof(visited));

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

if (!visited[i]) {

count++;

DFS(i,s,t);

}

}

return count;

}

void canh\_cau() {

int cc = tplt(0,0);//la goc

for (pair<int,int> e : edge) {

int s = e.first, t = e.second;

if (cc < tplt(s,t)) {

//cout << s << " " << t << "\n";

cout << "Đồ thị không có tính chất liên thông cạnh";

return;

}

}

cout << "Đồ thị có tính chất liên thông cạnh";

}

int main() {

cin >> n >> m;

for (int i = 1; i <= m; i++) {

int u,v; cin >> u >> v;

a[u].push\_back(v);

a[v].push\_back(u);

}

}

Câu 7:

\*/

#include <iostream>

#include <vector>

using namespace std;

void floodFillDFS(vector<vector<int>>& matrix, int x, int y, int targetColor, int replacementColor) {

int n = matrix.size();

if (x < 0 || x >= n || y < 0 || y >= n) {

return;

}

if (matrix[y][x] != targetColor) {

return;

}

matrix[y][x] = replacementColor;

floodFillDFS(matrix, x + 1, y, targetColor, replacementColor);

floodFillDFS(matrix, x - 1, y, targetColor, replacementColor);

floodFillDFS(matrix, x, y + 1, targetColor, replacementColor);

floodFillDFS(matrix, x, y - 1, targetColor, replacementColor);

}

int main() {

int n = 5; // Kích thước ma trận vuông n x n

vector<vector<int>> matrix = {

{1, 1, 0, 0, 0},

{0, 1, 1, 1, 0},

{0, 0, 1, 0, 0},

{0, 0, 1, 1, 1},

{0, 0, 0, 0, 1}

};

// Hiển thị ma trận trước khi thực hiện flood fill

cout << "Ma tran truoc khi thuc hien flood fill:\n";

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

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

cout << matrix[i][j] << " ";

}

cout << endl;

}

int x = 2, y = 2; // Thay đổi x, y tùy thuộc vào điểm bạn muốn bắt đầu flood fill

int targetColor = matrix[y][x]; // Màu của điểm (x, y)

int replacementColor = 2; // Thay đổi màu tại đây

floodFillDFS(matrix, x, y, targetColor, replacementColor);

// Hiển thị ma trận sau khi thực hiện flood fill

cout << "Ma tran sau khi thuc hien flood fill:\n";

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

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

cout << matrix[i][j] << " ";

}

cout << endl;

}

return 0;

}