**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN I**



**BÀI TẬP ĐIỀU KIỆN**

**Bộ Môn : Toán Rời Rạc 2**

- **HỌ TÊN** : MẠC VĂN THÀNH **THẦY GIÁO :** VŨ VĂN THỎA

- **MÃ SINH VIÊN** : B21DCCN677

- **SỐ ĐIỆN THOẠI:** 0982316213

- **LỚP** : D21CQCN05-B

1. **Viết hàm có tên là DFS(int u) trên C/C++ mô tả thuật toán duyệt theo chiều sâu các đỉnh của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề a[ ] [ ].**

#define MAX 100

int a[MAX][MAX]; *// Ma trận kề*

int n; *// Số đỉnh của đồ thị*

bool visited[MAX]; *// Mảng đánh dấu các đỉnh đã được duyệt*

*// Hàm DFS duyệt đồ thị theo chiều sâu*

void DFS(int u)

{

    visited[u] = true; *// Đánh dấu đỉnh u đã được duyệt*

    cout << u << " "; *// In ra đỉnh u*

    for (int v = 1; v <= n; v++)

    {

*// Nếu có cạnh nối từ u tới v và v chưa được duyệt*

        if (a[u][v] == 1 && visited[v] == false)

        {

DFS(v); *// Duyệt đỉnh v*

        }

    }

}

1. **Viết hàm có tên là BFS( int u) trên C/C++ mô tả thuật toán duyệt theo chiều rộng các đỉnh của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề a[ ] [ ].**

#define MAX 100

int n, a[MAX][MAX]; *// Số đỉnh và ma trận kề của đồ thị*

bool visited[MAX]; *// Mảng đánh dấu các đỉnh đã được duyệt*

void BFS(int u)

{

    queue<int> Q;

    visited[u] = true;

    Q.push(u);

    while (Q.size() > 0)

    {

        int u = Q.front(); *// Lấy đỉnh đầu tiên trong hàng đợi*

        cout << u << " "; *// In ra đỉnh vừa lấy*

        Q.pop(); *// Xóa đỉnh khỏi hàng đợi*

        for (int v = 1; v <= n; v++)

        {

            if (a[u][v] == 1 && visited[v] == false) *// Kiểm tra xem có cạnh nối từ đỉnh u đến đỉnh v hay không*

            {

                visited[v] = true; *// Đánh dấu đã duyệt đỉnh v*

                Q.push(v); *// Thêm đỉnh v vào hàng đợi*

            }

        }

    }

}

1. **Viết hàm có tên là int TPLT\_DFS(int a[ ] [ ]) trên C/C++ tìm số thành phần liên thông của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề a[ ] [ ] bằng cách sử dụng hàm DFS(int u) đã biết mô tả thuật toán duyệt theo chiều sâu các đỉnh của đồ thị G.**

#define MAX 100

int a[MAX][MAX]; *// ma trận kề*

bool visited[MAX]; *// mảng đánh dấu các đỉnh đã được thăm hay chưa*

int n; *// số đỉnh của đồ thị*

void DFS(int u)

{

    visited[u] = true; *// Đánh dấu đỉnh u đã được duyệt*

    for (int v = 1; v <= n; v++)

    {

        if (a[u][v] == 1 && visited[v] == false) *// Nếu đỉnh v chưa được duyệt và có cạnh nối với u*

        {

            DFS(v); *// Duyệt đỉnh v*

        }

    }

}

int TPLT\_DFS()

{

    int coun = 0; *// Khởi tạo số thành phần liên thông*

    for (int i = 1; i <= n; i++)

    {

        if (visited[i] == false)*// Nếu đỉnh i chưa được duyệt*

        {

            coun++; *// Tăng số thành phần liên thông*

            DFS(i); *// Duyệt các đỉnh liên thông với đỉnh i*

        }

    }

    return coun; *// Trả về số thành phần liên thông*

}

1. **Viết hàm có tên là int TPLT\_BFS(int a[ ] [ ]) trên C/C++ tìm số thành phần liên thông của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề a[ ] [ ] bằng cách sử dụng hàm BFS(int u) đã biết mô tả thuật toán duyệt theo chiều rộng các đỉnh của đồ thị G.**

#define MAX 100

int n, a[MAX][MAX]; *// số đỉnh của đồ thị và ma trận kề*

bool visited[MAX]; *// mảng đánh dấu các đỉnh đã được thăm hay chưa*

void BFS(int u)

{

    queue<int> Q; *// khởi tạo hàng đợi*

    Q.push(u); visited[u] = true; *// đẩy đỉnh u vào hàng đợi và đánh dấu đã duyệt*

    while (Q.size() > 0)

    {

        int u = Q.front(); Q.pop(); *// lấy ra đỉnh hàng đợi*

        for(int v = 1; v <= n; v++){

          if(a[u][v] == 1 && visited[v] == false){

            Q.push(v); visited[v] = true; *// đẩy đỉnh vào hàng đợi và đánh dấu đã duyệt*

          }

        }

    }

}

int TPLT\_BFS()

{

    int coun = 0; *// Khởi tạo số thành phần liên thông*

    for (int i = 1; i <= n; i++)

    {

        if (visited[i] == false) *// Nếu đỉnh i chưa được duyệt*

        {

            coun++; *// Tăng số thành phần liên thông lên 1*

            BFS(i); *// Duyệt các đỉnh liên thông với đỉnh i*

        }

    } return coun; *// Trả về số thành phần liên thông*

}

1. **Viết hàm có tên là T\_DFS(int a[ ] [ ]) trên C/C++ tìm cây khung T[ ] của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề a[ ] [ ] bằng cách sử dụng hàm DFS(int u) đã biết mô tả thuật toán duyệt theo chiều sâu các đỉnh của đồ thị G.**

#define MAX 100

bool visited[MAX];

int n, a[MAX][MAX], parent[MAX]; *// ma trận kề và mảng lưu đỉnh cha của đỉnh hiện tại*

void DFS(int u)

{

    visited[u] = true; *// đánh dấu đã duyệt đỉnh u*

    for (int v = 1; v <= n; v++) *// duyệt ma trận kề của đỉnh u*

    {

        if (a[u][v] == 1 && visited[v] == false) *// nếu v chưa thăm và có đường đi từ u -> v thì duyệt đỉnh v*

        {

            parent[v] = u; *// cập nhật đỉnh cha cho đỉnh v*

            DFS(v);

        }

    }

}

void T\_DFS()

{

    for (int i = 1; i <= n; i++) *// duyệt tất cả các đỉnh*

    {

*// nếu đỉnh nào chưa được thăm thì DFS đến đỉnh đó để tìm cây khung*

        if (visited[i] == false)

        {

            DFS(i);

        }

    }

    cout << "Edges of MST:" << "\n";

*// In ra cây khung*

    for (int i = 1; i <= n; i++)

    {

        cout << parent[i] << " <- " << i << "\n";

    }

}

1. **Viết hàm có tên là T\_BFS(int a[ ] [ ]) trên C/C++ tìm cây khung T[ ] của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề a[ ] [ ] bằng cách sử dụng hàm BFS(int u) đã biết mô tả thuật toán duyệt theo chiều rộng các đỉnh của đồ thị G.**

#define MAX 100

int n, a[MAX][MAX], parent[MAX]; *// Số đỉnh của đồ thị , ma trận kề và mảng lưu đỉnh cha của đỉnh hiện tại*

bool visited[MAX]; *// Mảng đánh dấu các đỉnh đã được duyệt*

void BFS(int u)

{

    queue<int> Q; *// Khởi tạo hàng đợi*

    visited[u] = true; *// Đánh dấu đỉnh u đã duyệt*

    Q.push(u); *// Đẩy đỉnh u vào hàng đợi*

    while (Q.size() > 0)

    {

        int u = Q.front(); *// Lấy đỉnh đầu tiên trong hàng đợi*

        cout << u << " "; *// In ra đỉnh vừa lấy*

        Q.pop(); *// Xóa đỉnh khỏi hàng đợi*

        for (int v = 1; v <= n; v++)

        {

            if (a[u][v] == 1 && visited[v] == false) *// Kiểm tra xem có cạnh   nối từ đỉnh s đến đỉnh i hay không*

            {

                visited[v] = true; *// Đánh dấu đã duyệt đỉnh i*

                Q.push(v); *// Thêm đỉnh i vào hàng đợi*

                parent[v] = u; *// cập nhật đỉnh cha cho đỉnh v*

            }

        }

    }

}

void T\_BFS()

{

    for (int i = 1; i <= n; i++) *// duyệt tất cả các đỉnh*

    {

*// nếu đỉnh nào chưa được thăm thì DFS đến đỉnh đó để tìm cây khung*

        if (visited[i] == false)

        {

            BFS(i);

        }

    }

    cout << "Edges of MST:" << "\n";

*// In ra cây khung*

    for (int i = 1; i <= n; i++)

    {

        cout << parent[i] << " <- " << i << "\n";

    }

}

1. **Viết hàm có tên là EULER(int a[ ] [ ]) trên C/C++ tìm chu trình/đường đi Euler CE[ ] của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận kề a[ ] [ ], biết rằng G là đồ thị Euler/nửa Euler.**

#define MAX 100

int n, a[MAX][MAX]; *// Số đỉnh của đồ thị và ma trận kề*

bool visited[MAX];

int isEulerian() {

    int odd = 0; *// Biến odd để đếm số đỉnh bậc lẻ*

    for (int i = 1; i <= n; i++) {

        int degree = 0; *// đếm bậc của từng đỉnh i*

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

            degree += a[i][j];

        }

        if (degree % 2 != 0) {

            odd++; *// nếu bậc của các đỉnh i là lẻ thì odd tăng thêm 1*

        }

    }

    if (odd == 0) {

        return 1; *// => đồ thị là Euler*

    }

    if (odd == 2) {

        return 2; *// => đồ thị là Nửa Euler*

    }

    return 0; *// => Không có chu trình*

}

void Fleury(int u, vector<int>& CE) {

    for (int v = 1; v <= n; v++) {

        if (a[u][v] == 1) {

            a[u][v] = a[v][u] = 0; *// đánh dấu cạnh đã duyệt*

            Fleury(v, CE); *// đệ quy đến cạnh kề với đỉnh u để duyệt tiếp*

        }

    }

*// Khi mà đỉnh u cô lập thì sẽ thêm vào chu trình CE*

    CE.push\_back(u);

}

void EULER() {

*// 1.Nếu không có chu trình euler*

    if (isEulerian() == 0) {

        cout << "Khong ton tai chu trinh / duong di Euler!" << "\n";

        return;

    }

*// 2.Chu trình / đường đi của đồ thị nửa Euler*

    else if(isEulerian() == 2){

        int start = 0; *// tìm đỉnh đầu tiên có bậc lẻ để duyệt chu trình và kết thúc ở đỉnh bậc lẻ còn lại.*

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

            int degree = 0; *// đếm bậc của từng đỉnh*

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

                degree += a[i][j];

            }

*// Nếu đỉnh i nào đó có bậc lẻ thì đánh dấu lại*

            if (degree % 2 != 0) {

                start = i;

                break;

            }

        }

        vector<int> CE; *// Khởi tạo chu trình CE*

        Fleury(start, CE); *// Thuật toán bắt đầu từ đỉnh vừa tìm được (start)*

        reverse(CE.begin(), CE.end()); *// Lật ngược chu trình lại*

*// In ra chu trình*

        cout << "Chu trinh/duong di Euler la: ";

        for (int i = 0; i < CE.size(); i++) {

            cout << CE[i] << " ";

        }

        cout << "\n";

    }

*// 3.Chu trình / đường đi của đồ thị Euler*

    else if(isEulerian() == 1){

        vector<int> CE; *// Khởi tạo chu trình CE*

        Fleury(1, CE); *// bắt đầu duyệt từ đỉnh 1*

        reverse(CE.begin(), CE.end()); *// Lật ngược chu trình CE*

*// In ra chu trình*

        cout << "Chu trinh/duong di Euler la: ";

        for (int i = 0; i < CE.size(); i++) {

            cout << CE[i] << " ";

        }

        cout << "\n";

    }

}

1. **Viết hàm có tên là DIJKSTRA(int u) trên C/C++ tìm đường đi ngắn nhất d[v] xuất phát từ đỉnh u đến các đỉnh v của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận trọng số a[ ] [ ].**

#define MAX 100

int n, a[MAX][MAX]; *// ma trận trọng số*

int d[MAX]; *// Mảng lưu khoảng cách từ đỉnh u -> mọi đỉnh*

void DIJKTRA(int u)

{

    for (int i = 1; i <= n; i++) {

        d[i] = 100000000; *// khởi tạo khoảng cách từ đỉnh u -> các đỉnh còn lại là vô cùng*

    }

    d[u] = 0; *// Đỉnh u có khoảng cách bắt đầu = 0*

*// Hàng đợi ưu tiên mà ở đỉnh luôn lưu khoảng cách nhỏ nhất lưu cặp {distance, đỉnh}*

    priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> Q;

    Q.push({0, u}); *// Đẩy đỉnh u và khoảng cách ban đầu vào trong hàng đợi ưu tiên để bắt đầu duyệt*

    while (Q.size() > 0)

    {

*// Lấy ra khoảng cách nhỏ nhất và đỉnh hiện tại để xét*

        int distance = Q.top().**first**; *// Khoảng cách hiện tại mà ta đang xét*

        int u = Q.top().**second**; *// lấy ra đỉnh hiện tại*

        Q.pop();

        if (distance > d[u]) continue; *// - Điều kiện if này có nghĩa là trong hàng đợi nó đã có phiên bản tốt hơn (tức là đường đi tốt hơn) rồi nên continue*

  for (int v = 1; v <= n; ++v)

        {

            int W = a[u][v]; *// lấy ra trọng số của đỉnh u,v*

            if (d[v] > d[u] + W) *// Điều kiện để cập nhật lại khoảng cách từ đỉnh u -> v*

            {

                d[v] = d[u] + W;

                Q.push({d[v], v}); *// Đẩy lại vào trong hàng đợi để cập nhật cho đến hết.*

            }

        }

    }

*// In ra khoảng cách từ đỉnh u -> mọi đỉnh*

    for(int i = 1; i <= n; i++){

        cout << d[i] << "\n";

    }

}

1. **Viết hàm có tên là FLOYD(int a[ ] [ ]) trên C/C++ tìm đường đi ngắn nhất d[ ] [ ] giữa các cặp đỉnh của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận trọng số a[ ] [ ].**

#define MAX 100

int n, a[MAX][MAX], d[MAX][MAX];

void FLOYD()

{

*// Khởi tạo ma trận d[][] ban đầu bằng ma trận trọng số a[][].*

    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= n; j++)

        {

            if (i == j) d[i][j] = 0;

            else d[i][j] = a[i][j];

        }

    }

*// Duyệt lần lượt qua tất cả các đỉnh từ 1 đến n, và cập nhật lại ma trận d[][].*

    for (int k = 1; k <= n; k++)

    {

        for (int i = 1; i <= n; i++)

        {

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

            {

                if (d[i][j] > d[i][k] + d[k][j]) *// Nếu đường đi từ đỉnh i -> j lớn hơn đường đi từ i -> k + đường đi từ k -> j thì sẽ cập nhật lại d[i][j].*

                {

                    d[i][j] = d[i][k] + d[k][j];

                }

            }

        }

    }

}

1. **Viết hàm có tên là PRIM(int a[ ] [ ], int u) trên C/C++ tìm cây khung T[ ] nhỏ nhất bắt đầu tại đỉnh u của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận trọng số a[ ] [ ] bằng cách sử dụng thuật toán Prim.**

#define MAX 100

int n, m, a[MAX][MAX];

bool visited[MAX];

void PRIM(int u)

{

    long long d = 0; *// Cây khung có tổng trọng số nhỏ nhất*

    priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> Q; *// min\_heap : lưu min ở đỉnh hàng đợi*

    Q.push({0, u}); *// Queue lưu cặp {trọng số, đỉnh} cần xét*

    visited[u] = true; *// đánh dấu đỉnh u đã duyệt*

    while (Q.size() > 0)

    {

        pair<int, int> top = Q.top(); Q.pop();

        int W = top.**first**; *// W : trọng số hiện tại*

        int u = top.**second**; *// u : đỉnh hiện tại đang xét của đồ thị*

        if (visited[u] == true) *// Nếu đỉnh u đã duyệt*

            continue;

        d += W; *// cộng trọng số vào cây khung cực tiểu*

        visited[u] = true; *// Đánh dấu đỉnh đã duyệt*

        for(int v = 1; v <= n; v++){

            if(visited[v] == false){

                Q.push({a[u][v], v}); *// Đẩy vào trong hàng đợi cặp {trọng số, đỉnh kề} tiếp theo*

            }

        }

    }

*// In ra cây khung cực tiểu*

    cout << d << "\n";

}

1. **Viết hàm có tên là KRUSKAL(int a[ ] [ ]) trên C/C++ tìm cây khung T[ ] nhỏ nhất của đồ thị G = <V, E> được biểu diễn dưới dạng ma trận trọng số a[ ] [ ] bằng cách sử dụng thuật toán Kruskal.**

#define MAX 100

int n, m, a[MAX][MAX];;

int parent[MAX], sz[MAX];

*// Khởi tạo 1 struct lưu đỉnh đầu , đỉnh cuối và trọng số*

struct edge{

    int **u**, **v**, **w**;

};

vector<edge> adj; *// sử dụng vector kiểu struct để lưu*

*// Khởi tạo*

void init(){

    cin >> n >> m;

    for(int u = 1; u <= n; u++){

        for(int v = 1; v <= m; v++){

            if(a[u][v] != 0){

                adj.push\_back({u, v, a[u][v]});

*// u: đỉnh đầu* *, v: đỉnh cuối*, *a[u][v]: trọng số*

            }

        }

    }

*// Khởi tạo 2 mảng để sử dụng cho Disjoint Set Union*

    for(int i = 1; i <= n; i++){

        parent[i] = i;

        sz[i] = 1;

    }

}

*// DSU : Disjoint Set Union dùng để nối 2 đỉnh xem có tạo thành chu trình hay không.*

int Find(int u){

    if(u == parent[u])

        return u;

    return parent[u] = Find(parent[u]);

}

bool Union(int x, int y){

    x = Find(x); *// // Tìm đỉnh cha của đỉnh x*

    y = Find(y); *// Tìm đỉnh cha của đỉnh y*

    if(x == y)

        return false; *// Nếu thấy 2 đỉnh cần nối có cùng cha , thì không thể nối*

    if(sz[x] <= sz[y]){

        parent[x] = y;

        sz[x] += sz[y];

    }

    else{

        sz[y] += sz[x];

        parent[y] = x;

    }

    return true; *// Ngược lại ta có thể nối 2 đỉnh đó với nhau*

}

*// Sắp xếp theo trọng số tăng* *dần của các cạnh tương ứng*

bool cmp(edge a, edge b){

    return a.**w** < b.**w**;

}

void KRUSKAL(){

    long long ans = 0; *// Cây khung có tổng trọng số nhỏ nhất*

    vector<edge> MST; *// Khởi tạo cây khung MST*

    sort(begin(adj), end(adj), cmp); *// Sắp xếp theo trọng số tăng dần*

    for(int i = 0; i < m; i++){

*// Nếu MST.size() == n - 1 thì break , tức là đã xây dựng được cây khung kết nối với tất cả các đỉnh của đồ thị, có n - 1 cạnh*

        if(MST.size() == n - 1)

            break;

        int x = adj[i].**u**, y = adj[i].**v**, z = adj[i].**w**;

*// Ta sẽ nối 2 đỉnh x với y*

*// + Nếu nối được ta sẽ cộng trọng số vào ans và thêm nó vào cây khung*

*// + Nếu không ta sẽ nhảy sang cạnh tiếp theo*

        if(Union(x, y) == true){

            ans += z;

            MST.push\_back(adj[i]);

        }

    }

*// Nếu MST.size() != n - 1 thì đồ thị KHÔNG liên thông*

*// In ra cây khung nhỏ nhất*

    cout << ans << "\n";

    for(auto it : MST){

        cout << it.**u** << " " << it.**v** << " " << it.**w** << "\n";

    }

}

1. **Viết chương trình hoàn chỉnh tìm luồng cực đại f[][] trên mạng G = <V, E> được biểu diễn dưới dạng ma trận trọng số c[ ] [ ] với đỉnh phát s và đỉnh thu t bằng cách sử dụng thuật toán đường tăng luồng dựa trên Ford-Fulkerson :**

#include<bits/stdc++.h>

using namespace std;

#define MAX 1000

int n, c[MAX][MAX], f[MAX][MAX];

int trace[MAX];

bool visited[MAX];

bool BFS(int s, int t)

{

    queue<int> Q;

    Q.push(s);

    visited[s] = true;

    while (Q.size())

    {

        int u = Q.front(); Q.pop();

        for (int v = 1; v <= n; v++)

        {

            if (visited[v] == false && c[u][v] > f[u][v])

            {

                Q.push(v); visited[v] = true;

                trace[v] = u;

                if (v == t) return true;

            }

        }

    }

    return false;

}

int Ford\_Fulkerson(int s, int t)

{

    int max\_flow = 0;

    while (BFS(s, t) == true)

    {

        int delta = INT\_MAX;

        for (int v = t; v != s; v = trace[v])

        {

            int u = trace[v];

            delta = min(delta, c[u][v] - f[u][v]);

        }

        for (int v = t; v != s; v = trace[v])

        {

            int u = trace[v];

            f[u][v] += delta;

            f[v][u] -= delta;

        }

        max\_flow += delta;

        for (int i = 1; i <= n; i++) visited[i] = false;

    }

    return max\_flow;

}

int main()

{

    #ifndef ONLINE\_JUDGE

    freopen("DT.INP.txt", "r", stdin);

*// nhap du lieu*

*/\**

*6*

*0 5 5 0 0 0*

*0 0 0 6 3 0*

*0 0 0 3 1 0*

*0 0 0 0 0 6*

*0 0 0 0 0 6*

*0 0 0 0 0 0*

*\*/*

    freopen("DT.OUT.txt", "w", stdout);

*/\**

*- Kết quả:*

*9*

*0 5 4 0 0 0*

*0 0 0 3 2 0*

*0 0 0 3 1 0*

*0 0 0 0 0 6*

*0 0 0 0 0 3*

*0 0 0 0 0 0*

*\*/*

    #endif

    cin >> n;

    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= n; j++)

        {

            cin >> c[i][j];

        }

    }

    int max\_flow = Ford\_Fulkerson(1, n);

*// In ra giá trị luồng cực đại*

    cout << "val : " << max\_flow << "\n\n";

*// In ra đồ thị f sau khi thực hiện xong thuật toán*

    cout << "Ma trận f : \n";

    for (int i = 1; i <= n; i++)

    {

        for (int j = 1; j <= n; j++)

        {

            if(i >= j) f[i][j] = 0;

            cout << f[i][j] << " ";

        }

        cout << "\n";

    }

    return 0;

}