LUỒNG CỰC ĐẠI TRÊN MẠNG

I. Các khái niệm và bài toán

1.1 Mạng

Mạng (network) là một bộ năm $G = (V, E, c, s, t)$ trong đó:

- V và E lần lượt là tập đỉnh và tập cung của một đồ thị có hướng không có khuyên (cung từ mỗi đỉnh đi đến chính nó)
- s và t là hai đỉnh phân biệt thuộc V, s gọi là đỉnh phát (source) và t gọi là đỉnh thu (sink)
- c là một hàm xác định trên tập cung E:

$$c: E \to [0, +\infty)$$
$$e \to c(e)$$

gán cho mỗi cung $e \in E$ một số không âm gọi là sức chứa (capacity) $c(e) \geq 0$.
Bằng cách thêm vào mạng một số cung có sức chứa 0, ta có thể giả thiết rằng mỗi cung $e = (u, v)$ luôn tương ứng duy nhất một cung ngược chiều $-e = (v, u) \in E$ gọi là cung đối của cung e. Ta cũng coi e là cung đối của $-e$ (tức $-(-e) = e$).
Chú ý rằng mạng là *đa đồ thị* tức là giữa hai đỉnh có thể có nhiều cung.
Để thuận tiện cho việc trình bày ta qui ước các ký hiệu sau:
Với X, Y là hai tập con của V và $f: E \to \mathbb{R}$ là một hàm xác định trên tập cung E. Khi đó đặt:

$$\{X \to Y\} \equiv \{e = (u, v) \in E : u \in X, v \in Y\}$$
$$f(X, Y) = \sum_{e \in \{X \to Y\}} f(e)$$

1.2 Luồng

Luồng (flow) trên mạng G là một hàm:

$$f: E \to \mathbb{R}$$
$$e \mapsto f(e)$$

Gán cho mỗi cung e một số thực $f(e)$, gọi là luồng trên cung e, thỏa mãn ba tính chất dưới đây:

- Ràng buộc về sức chứa: $\forall e \in E : \ f(e) \leq c(e)$
- Ràng buộc về tính đối xứng lệch: $\forall e \in E : f(e) = -f(-e)$
- Ràng buộc về tính bảo tồn: $\forall v \in V - \{s, t\} : f(\{v\}, V) = 0$

Với ràng buộc về tính đối xứng lệch và tính bảo tồn ta suy ra được: $\forall v \in V - \{s, t\} : f(V, \{v\}) = 0$
Giá trị luồng trên mạng được định nghĩa bằng tổng luồng trên các cung đi ra khỏi đỉnh phát:

$$|f| = f(\{s\}, V)$$

Bài toán luồng cực đại trên mạng: Cho một mạng G với đỉnh phát s và đỉnh thu t, hàm sức chứa c, hãy tìm một luồng có giá trị lớn nhất trên G.

1.3 Mạng thặng dư:

Với f là một luồng trên mạng $G = (V, E, c, s, t)$ ta xét mạng G_f cũng là mạng G nhưng với hàm sức chứa mới cho bởi:

$$c_f : E \to [0, +\infty)$$
$$e \mapsto c_f(e) = c(e) - f(e)$$

Mạng G_f như vậy được gọi là mạng thặng dư của mạng G sinh ra bởi luồng f. Sức chứa của cung e trên G_f thực chất là lượng luồng tối đa có thể thêm vào luồng $f(e)$ mà không vượt quá sức chứa $c(e)$ Cung $e \in E$ được gọi là cung bão hòa nếu $c_f(e) = 0$ ngược lại nó được gọi là cung thặng dư. Đường đi chỉ qua các cung thặng dư goi là đường thặng dư.

Địn lý 1: Cho f là một luồng trên mạng $G = (V, E, c, s, t)$. Khi đó nếu f' là luồng trên mạng G_f thì hàm:

$$f + f': E \to \mathbb{R}$$
$$e \longmapsto (f + f')(e) = f(e) + f'(e)$$

Cũng là một luồng trên mạng G với giá trị luồng $|f + f'| = |f| + |f'|$

Định lý 2: Cho f và f' là hai luồng trên mạng $G = (V, E, c, s, t)$ khi đó hàm:

$$f - f': E \to \mathbb{R}$$
$$e \longmapsto (f - f')(e) = f(e) - f'(e)$$

là một luồng trên mạng thặng dư G_f với giá trị luồng $|f - f'| = |f| - |f'|$

1.4 Đường tăng luồng

Với f là một luồng trên mạng $G = (V, E, c, s, t)$. Gọi P là đường đi từ đỉnh s đến đỉnh t trên mạng thặng dư G_f. Giá trị thặng dư của đường P ký hiệu là Δ_P được định nghĩa bằng sức chứa nhỏ nhất trên các cung dọc theo đường đi P:

$$\Delta_P = \min\{c_f(e) : (e) \textit{ nằm trên } P\}$$

Vì các sức chứa $c_f(e) \geq 0$ nên $\Delta_P \geq 0$. Nếu $\Delta_P > 0$ thì đường đi P là một đường thặng dư, khi đó ta gọi P là một đường tăng luồng tương ứng với luồng f

Định lý 3: Cho f là một luồng trên mạng $G = (V, E, c, s, t)$, P là một đường tăng luồng trên G_f. Khi đó hàm $f_P: E \to \mathbb{R}$ định nghĩa như sau:

$$f_P(e) = \begin{cases} +\Delta_P & \text{nếu } e \in P \\ -\Delta_P & \text{nếu } -e \in P \\ 0, \text{trường hợp khác} \end{cases} \qquad (1)$$

là một luồng trên G_f với giá trị luông $|f_P| = \Delta_P > 0$

Định lý 1 và 3 cho hệ quả:

Hệ quả: Cho f là một luồng trên mạng $G = (V, E, c, s, t)$ và P là một đường tăng luồng trên G_f, gọi f_P là luồng trên G_f định nghĩa như trong công thức (1). Khi đó $f + f_P$ là một luồng mới trên G với giá trị là $|f + f_P| = |f| + \Delta_P$

Có chế cộng luồng f_P vào luồng f hiện có gọi là tăng luồng dọc theo đường tăng luồng P.

1.5 Phương pháp Ford - Fulkerson

```
f ←≪Một luồng bất kỳ≫
while ≪Tìm được đường tăng luồng P≫ do f=f+fP
output←f
```

II. Thuật toán Dinitz

2.1 Đồ thị tầng

Gọi dist(u) là độ dài đường đi thặng dư ngắn nhất (tính theo số cung) từ s đến u trên G_f. Giả sử f là một luồng trên mạng $G = (V, E, c, s, t)$ với mạng thặng dư G_f. Đồ thị tầng là hạn chế của G_f trên các cung:

$$E_L = \{e = (u, v) \in E_f : dist(v) = dist(u) + 1\}$$

2.3 Thuật toán Dinitz

1. Khởi tạo $f(e) = 0 \; \forall e \in E$
2. Tính $dist(u): \forall u \in V$ nếu $dist(t) = \infty$ thì dừng và đưa ra kết quả f
3. while ≪còn đường tăng luồng≫ do ≪ tăng luồng dọc theo đường tăng luồng≫
4. Quay lại bước 2

2.4 Cài đặt Dinitz bằng ma trận kề.

```
// Các biến chính
int n;                                      // số đỉnh
vector<int> g[maxn];                        // Mô tả tập cug E
int c[maxn][maxn];                          // Mô tả độ thông qua
int S,T;                                    // S-đỉnh phát, T-đỉnh thu
int f[maxn][maxn];                          // Mô tả luồng
int dist[maxn];                             // Khoảng cách từ S
int id, cl[maxn];                           // Biến đếm, mảng đánh dấu

// Hàm tìm luồng cực đại
int MaxFlow() {
    id=0;
    int mf=0;
    while (BFS()) {
        while (int delta=(++id, FindPath(s,oo)))
        mf += delta;
    }
    return mf;
}

// Lập mảng dist bằng BFS, trả về 0 nếu không đến được T
int q[maxn]; // Hàng đợi
int BFS() {
    for(int i=1;i<=n;++i) dist[i]=0;  // dist[u]=0 - đỉnh chưa xét
    int L=1, R=0;
    q[++R]=S, dist[S]=1;
    while (L<=R) {
        int u=q[L++];
        for(auto &v : g[u]) if (!dist[v] && f[u][v]<c[u][v]) {
            dist[v]=dist[u]+1;
            q[++R]=v;
        }
    }
    return (dist[T]!=0);
}

// Hàm tìm đường và tăng luồng
int FindPath(int u,int val) {
    if (u==T) return val;
    if (cl[u]==id) return 0;
    cl[u]=id;
    for(auto &v: g[u])
    if (cl[v]!=id && dist[v]==dist[u]+1)
```

```
    if (c[u][v]>f[u][v]) {
        if (int delta=FindPath(v,min(val,c[u][v]-f[u][v]))) {
            f[u][v] += delta;
            f[v][u] -= delta;
            return delta;
        }
    }
    return 0;
}

int main() {
    ....
    // Khởi tạo mạng:
    cin >> n >> m;
    cin >> S >> T;
    for(int i=1;i<=m;++i) {
        int u, v, w; cin >> u >> v >> w;
        g[u].push_back(v);
        g[v].push_back(u);
        c[u][v] += w;
    }
    id=0;
    cout << MaxFlow();
}
```

2.5 Cài đặt Dinitz bằng danh sách cung

```
// Mô tả cung và luồng
struct Edge{
    int dau,                        // Điểm đầu
        cuoi,                               // Điểm cuối
        capa,                               // Độ thông qua
        flow;                       // Luồng
};

int n,m;
Edge E[2*maxm];                             // Danh sách cung (m+1...2m là các cung giả)
vector<int> g[maxn];
int S, T;
int dist[maxn];

int q[maxn];
bool BFS() {
    for(int i=1;i<=n;++i) dist[i]=oo;
    int L=1, R=0;
    q[++R]=S; dist[S]=0;
    while (L<=R) {
        int u=q[L++];
        for(auto &i : g[u]) {
            int v=E[i].cuoi;
            if (dist[v]==oo && E[i].capa-E[i].flow>0) {
                q[++R]=v;
                dist[v]=dist[u]+1;
            }
```

```
        }
    }
    return (dist[T]<oo);
}

int cl[maxn], id;
int FindPath(int u,int val) {
    if (u==T) return val;
    if (cl[u]==id) return 0;
    cl[u]=id;
    for(auto &i : g[u]) {
        int v=E[i].cuoi;
        if (cl[v]!=id && E[i].capa>E[i].flow && dist[v]==dist[u]+1) {
            if (int delta=FindPath(v,min(val,E[i].capa-E[i].flow))) {
                E[i].flow += delta;
                E[2*m+1-i].flow -= delta;
                return delta;
            }
        }
    }
    return 0;
}

int MaxFlow() {
    int mf=0;
    while (BFS()) {
        while (int delta=(++id,FindPath(S,oo))) {
            mf += delta;
        }
    }
    return mf;
}

int main() {
    ...
    // Dựng mạng, thêm cung giả
    for(int i=1;i<=m;++i) {
        E[2*m+1-i].dau=E[i].cuoi, E[2*m+1-i].cuoi=E[i].dau;
        E[2*m+1-i].capa=0, E[2*m+1-i].flow=0;
    }
    for(int i=1;i<=2*m;++i) {
        int u=E[i].dau;
        g[u].push_back(i);
    }
    printf("%d",MaxFlow());

}
```

III. Các bài toán ứng dụng.

1) Tìm cặp ghép cực đại không trọng số

Thuật toán Hopkroft - Karp

```
int m, n, res = 0;
vector<int> g[MAXN];
int x[MAXN], y[MAXN], d[MAXN], q[MAXN];

bool FindPath() {
    int L = 1, R = 0;
    for (int u = 1; u <= n; u++)
        if (x[u] == 0) {
            d[u] = 0;
            q[++R] = u;
        } else
            d[u] = INF;
    d[0] = INF;
    while (L <= R) {
        int u = q[L++];
        for (auto v : g[u]) {
            if (d[y[v]] == INF) {
                d[y[v]] = d[u] + 1;
                if (y[v])
                    q[++R] = y[v];
            }
        }
    }
    return (d[0] != INF);
}

bool dfs (int u) {
    //if (u==0) return(true);
    for (auto v : g[u]) {

        if (y[v] == 0) {
            x[u] = v;
            y[v] = u;
            d[u] = INF;
            return true;
        }
        if (d[y[v]] == d[u] + 1)
            if (dfs (y[v])) {
                x[u] = v;
                y[v] = u;
                d[u] = INF;
                return true;
            }
    }
    d[u] = INF;
    return false;
}
```

```
void GhepMax() {
    while (FindPath()) {
        for (int u = 1; u <= n; u++)
            if (x[u] == 0)
                if (dfs (u))
                    res++;
    }
    printf ("%d", res);
}
```