

# ICPC National 2022

Team: King's Landing  
Member: Nguyen Vo, Bach Ly, Phu Nguyen



|  |           |
|--|-----------|
| <b>STL Help</b>                                  | <b>3</b>  |
| 1. Customize set<>                               | 3         |
| 2. Hàm unique()                                  | 3         |
| <b>Segment tree</b>                              | <b>5</b>  |
| <b>Segment tree lazy</b>                         | <b>6</b>  |
| <b>Disjoint set</b>                              | <b>8</b>  |
| <b>Cầu, khớp</b>                                 | <b>9</b>  |
| <b>Heavy-Light Decomposition (HLD)</b>           | <b>10</b> |
| <b>Z-algorithm</b>                               | <b>11</b> |
| <b>Convex Hull (Bao lồi)</b>                     | <b>12</b> |
| <b>Hàm phi euler</b>                             | <b>15</b> |
| <b>Nghịch đảo modulo</b>                         | <b>16</b> |
| Extended Euclid                                  | 16        |
| Dùng hàm phi Euler                               | 17        |
| Tính tất cả nghịch đảo modulo $\leq m$ nguyên tố | 17        |
| <b>Định lý Wilson - factorial</b>                | <b>18</b> |
| <b>Trie</b>                                      | <b>19</b> |
| <b>Hình học tọa độ</b>                           | <b>21</b> |
| Tích vô hướng                                    | 21        |
| Tích có hướng                                    | 21        |
| <b>Suffix Array</b>                              | <b>22</b> |
| <b>Các bổ đề vector</b>                          | <b>25</b> |

|   |           |
|---|-----------|
| <b>Heavy-light Decomposition (HLD)</b>    | <b>26</b> |
| <b>Đường đi ngắn nhất</b>                 | <b>30</b> |
| Dijkstra                                  | 30        |
| Bellman Ford                              | 32        |
| Chu trình âm trong Bellman Ford           | 33        |
| Floyd Warshal                             | 35        |
| <b>Hash</b>                               | <b>37</b> |
| <b>Thuật toán Kruskal - Spanning tree</b> | <b>39</b> |
| <b>Tarjan</b>                             | <b>42</b> |

## STL Help

### 1. Customize set<>

---

```
class Comp {
public:
    bool operator() (const int &a, const int &b) const {
        return a<b;
    }
};
set<int, Comp> s;
```

---

### 2. Hàm unique()

Hàm unique loại bỏ các phần tử giống nhau liên kề. Cần sort nếu dùng cho mục đích extract các phần tử phân biệt.

---

```
int myints[] = {10,20,20,20,30,30,20,20,10};
vector<int> myvector (myints,myints+9);
// 10 20 20 20 30 30 20 20 10

vector<int>::iterator it;
it = unique(myvector.begin(), myvector.end());
// 10 20 30 20 10 ? ? ? ?
//                ^

myvector.resize( distance(myvector.begin(),it) );
// 10 20 30 20 10

// Có thể thêm hàm comp, trả về khi 2 phần tử = nhau
// it = unique(myvector.begin(), myvector.end(), comp);
```

---

**Segment tree**

---

```
int ST[4*NUM];

void update(int rt, int l, int r, int idx, int val) {
    if (idx < l || idx > r) {return ;}
    if (l==r) { ST[rt] = val; return; }

    update(rt<<1, l, (l+r)>>1, idx, val);
    update(rt<<1 | 1, ((l+r)>>1) + 1, r, idx, val);

    ST[rt] = ST[rt*2] + ST[rt*2+1];
}

int get(int rt, int l, int r, int u, int v) {
    if (r < u || l > v ) return 0;
    if (u<=l && r<=v) return ST[rt];

    int lbr = get(rt<<1, l, (l+r)>>1, u, v);
    int rbr = get(rt<<1|1, ((l+r)>>1)+1, r, u, v);
    return lbr+rbr;
}
```

---

**Segment tree lazy**

---

```
struct Node {
    int lazy; // giá trị T trong phân tích trên
    int val; // giá trị lớn nhất.
} nodes[MAXN * 4];

void down(int id) {
    int t = nodes[id].lazy;
    nodes[id*2].lazy += t;
    nodes[id*2].val += t;

    nodes[id*2+1].lazy += t;
    nodes[id*2+1].val += t;

    nodes[id].lazy = 0;
}

void update(int id, int l, int r, int u, int v, int val) {
    if (v < l || r < u) {
        return ;
    }
    if (u <= l && r <= v) {
        // đảm bảo val được cập nhật ĐỒNG THỜI với giá trị lazy
        nodes[id].val += val;
        nodes[id].lazy += val;
        return ;
    }
    int mid = (l + r) / 2;

    down(id); // đẩy giá trị lazy propagation xuống các con
    update(id*2, l, mid, u, v, val);
```

```

        update(id*2+1, mid+1, r, u, v, val);
        nodes[id].val = max(nodes[id*2].val, nodes[id*2+1].val);
    }

    int get(int id, int l, int r, int u, int v) {
        if (v < l || r < u) {
            return -INFINITY;
        }
        if (u <= l && r <= v) {
            return nodes[id].val;
        }
        int mid = (l + r) / 2;
        down(id); // đẩy giá trị lazy propagation xuống các con

        return max(get(id*2, l, mid, u, v),
                    get(id*2+1, mid+1, r, u, v));
        // Trong các bài toán tổng quát, giá trị ở nút id có thể bị
        // thay đổi (do ta đẩy lazy propagation
        // xuống các con). Khi đó, ta cần cập nhật lại thông tin của
        // nút id dựa trên thông tin của các con.
    }

```

### Disjoint set

Đề: Có  $n$  cái hộp và  $n$  viên sỏi, 2 loại truy vấn:

1. bỏ hết sỏi của hộp  $u$  và  $v$  vào cùng 1 hộp
2. check xem viên sỏi  $i$  và viên  $j$  có cùng hộp ko

---

```

// par[i] = x nếu sỏi i và sỏi x cùng hộp. Nếu par[i] < 0 thì
// sỏi i trong hộp i, và -par[i] là số sỏi trong hộp đó.
// Ban đầu, khởi tạo par[i] = -1 với mọi i.

```

```

int root(int v) { // Cho 1 số v, tìm hộp chứa viên sỏi v
    return par[v] < 0 ? v : (par[v] = root(par[v]));
}

```

```

/* Sỏi v cùng hộp với viên sỏi chứa par[v]. Chú ý, gán lại
par[v] = root(par[v]), kĩ thuật này là Path Compression, giúp
giảm độ phức tạp mỗi thao tác xuống  $\log(n)$  */
}

```

```

void merge(int x, int y) {
    // Gộp 2 hộp chứa viên sỏi x và y vào cùng 1 hộp
    if ((x = root(x)) == (y = root(y))) { return ; }
    // x và y cùng 1 hộp, pass
    if (par[y] < par[x]) { swap(x, y); }
}

```

```

/* Gộp vào hộp chứa nhiều sỏi hơn (Union by Rank), giảm độ phức
tạp mỗi thao tác xuống  $\log(n)$ . Kết hợp Union-by-rank và Path
Compression thì mỗi thao tác là ackerman(n), rất nhỏ so với  $n$  */

```

```

    par[x] += par[y];
    par[y] = x;
}

```

**Cầu, khớp**

Đề: Cho đồ thị vô hướng, tìm cầu, khớp.

---

```
void dfs(int u, int pre) {
    low[u]=num[u]=++timeDfs;

    int child=0;
    for (int v:g[u]) {
        if (v==pre) continue;
        if (!num[v]) {
            dfs(v, u);
            child++;

            low[u]=min(low[u], low[v]);

            if (low[v]==num[v]) cau++; // đếm cầu

            // xét khớp
            if (pre==u) {
                if (child>1) khop[u]=true;
            }
            else if (low[v]>=num[u]) khop[u]=true;
        }
        else low[u]=min(low[u], num[v]);
    }

    tail[u] = timeDfs;
}
```

---

**Heavy-Light Decomposition (HLD)**

## Z-algorithm

Đề: Cho chuỗi S. Tạo mảng Z sao cho  $Z_i$  là k sao cho  $S_i \dots S_{i+k}$  là một tiền tố của S.

---

```
int L = 0, R = 0;
Z[0] = n;
for (int i = 1; i < n; i++) {
    if (i > R)
    {
        L = R = i;
        while (R < n && S[R] == S[R - L]) R++;
        Z[i] = R - L; R--;
    }
    else
    {
        int k = i - L;
        if (Z[k] < R - i + 1) Z[i] = Z[k];
        else
        {
            L = i;
            while (R < n && S[R] == S[R - L]) R++;
            Z[i] = R - L; R--;
        }
    }
}
```

---

## Convex Hull (Bao lồi)

Dùng thuật toán **monotone chain (chuỗi đơn điệu)**: lấy 2 điểm ngoài cùng bên trái, bên phải, xong xây dựng bao lồi cho phần trên và phần dưới.

Lưu ý các trường hợp suy biến: các điểm trùng nhau, 3 điểm thẳng hàng. Trong thuật này:

- Các điểm thuộc bao lồi lưu trong a trả về, theo thứ tự **clockwise**
- Code này đã bỏ qua các điểm thẳng hàng (chỉ lấy 2 điểm rìa)
- Code chưa xét đến trường hợp 2 điểm trùng nhau lặp lại trong input

---

```
class pt { // Kiểu điểm
public:
    double x, y;
    pt(double x, double y) {this->x=x; this->y=y;}
};

bool cmp (pt a, pt b) { // so x trước, xong so y
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

bool cw (pt a, pt b, pt c) { // true if a -> b -> c clockwise
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) < 0;
}

bool ccw (pt a, pt b, pt c) { // a -> b -> c counter-clockwise
    return a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y) > 0;
}

void convex_hull (vector<pt> & a) {
    if (a.size() == 1) { // chỉ có 1 điểm
        return;
    }
}
```

```
}

sort (a.begin(), a.end(), &cmp);

pt p1 = a[0], p2 = a.back();

vector<pt> up, down; // chuỗi trên và chuỗi dưới
up.push_back (p1);
down.push_back (p1);

for (size_t i=1; i<a.size(); ++i) { // xét lần lượt các điểm
    // Thêm vào chuỗi up
    if (i==a.size()-1 || cw (p1, a[i], p2)) {
        while (up.size()>=2 && !cw (up[up.size()-2],
up[up.size()-1], a[i]))
            {up.pop_back();}
        up.push_back (a[i]);
    }

    // Thêm vào chuỗi down
    if (i==a.size()-1 || ccw (p1, a[i], p2)) {
        while (down.size()>=2 && !ccw (down[down.size()-2],
down[down.size()-1], a[i]))
            {down.pop_back();}
        down.push_back (a[i]);
    }
}

// Gộp 2 chuỗi up và down để lấy bao lồi
a.clear();
for (size_t i=0; i<up.size(); ++i)
    a.push_back (up[i]);
```

```
for (size_t i=down.size()-2; i>0; --i)
    a.push_back (down[i]);
}
```

---

## Hàm phi euler

---

```
int phi(int n) {
    int res = n;
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            while (n % i == 0) {
                n /= i;
            }
            res -= res / i;
        }
    }
    if (n != 1) {
        res -= res / n;
    }
    return res;
}
```

---

## Nghịch đảo modulo

Đề: tính  $a^{-1}$

### Extended Euclid

Giải thích: nếu  $\gcd(a,m) = 1$  thì luôn tìm được  $x, y$  nguyên sao cho:  $ax+my=1$ .  
Lấy mode 2 về được  $ax \equiv 1 \pmod{m}$ .

---

```
int x, y;

int extendedEuclid(int A, int B) {
    if (B == 0) {
        x = 1;
        y = 0;
        return A;
    }

    int d = extendedEuclid(B, A%B);
    int temp = x; x = y; y = temp - (A/B)*y;

    return d;
}

int main() {
    int g = extended_euclidean(a, m);
    if (g != 1) cout << "No solution!";
    else {
        result = (x % m + m) % m;
    }
}
```

---



**Dùng hàm phi Euler**

Giải thích: Nếu  $\gcd(a, m) = 1$  thì  $a^{\phi(m)} \equiv 1 \pmod{m}$ . Suy ra  $a^{\phi(m)-1} \equiv a^{-1} \pmod{m}$

**Tính tất cả nghịch đảo modulo  $\leq m$  nguyên tố**

---

```
r[1] = 1;
for(int i = 2; i < m; ++i)
    r[i] = (m - (m/i) * r[m%i] % m) % m;
```

---

**Định lý Wilson - factorial**

Định lý:  $n > 1$  là số nguyên tố khi và chỉ khi  $(n-1)! \equiv -1 \pmod{n}$

**Trie**

---

```
struct TrieNode {
    TrieNode* child[26];
    int cnt;
    TrieNode() {
        for (int i=0; i<26; ++i)
            child[i]=NULL;
        cnt=0;
    }
};

void TrieInsert(const string &s)
{
    int n=s.length();
    TrieNode* p=root;
    for (int i=0; i<n; ++i) {
        int nxt=s[i]-'a';
        if (p->child[nxt]==NULL)
            p->child[nxt]=new TrieNode();
        p=p->child[nxt];
    }
}

bool TrieFind(const string &s)
{
    int n=s.length();
    TrieNode* p=root;
    for (int i=0; i<n; ++i) {
        int nxt=s[i]-'a';
        if (p->child[nxt]==NULL)
            return false;
        p=p->child[nxt];
    }
}
```

```
    }
    return p->cnt>0;
}
```

---

### Hình học tọa độ

Vector  $u = (x1, y1)$ , vector  $v = (x2, y2)$ .

$Pi = 3.14159265359$

### Tích vô hướng

$$u.v = x1*x2 + y1*y2$$

$$u.v = ||u|| * ||v|| * \cos(u,v)$$

### Tích có hướng

Lưu ý tích có hướng âm khi góc  $(u,v) < 0$ :

$$u \times v = x1*y2 - x2*y1$$

$$u \times v = ||u|| * ||v|| * \sin(u,v)$$

### Suffix Array

Xây dựng bằng đệ quy từ các dãy có độ dài  $2^k$ . Xây dựng xong sắp xếp từng đoạn. Độ phức tạp  $O(n \log n)$ .

---

```
void count_sort(vector<int>& p, vector<int>& c){
    int n = p.size();
    vector<int> cnt(n);
    for (auto x : c){
        cnt[x]++;
    }
    vector<int> p_new(n);
    vector<int> pos(n);
    pos[0] = 0;
    for (int i = 1; i < n; ++i){
        pos[i] = pos[i - 1] + cnt[i - 1];
    }
    for (auto x : p){
        int i = c[x];
        p_new[pos[i]] = x;
        pos[i]++;
    }
    p = p_new;
}
```

```
int main(){
    string s;
    cin >> s;
    s += "$";
    int n = s.size();
    vector<int> p(n), c(n);
    {
```

```

vector <pair<char,int>> a(n);
for (int i = 0; i < n; ++i) a[i] = {s[i],i};
sort(a.begin(), a.end());
for (int i = 0; i < n; ++i) p[i] = a[i].second;
c[p[0]] = 0;

for (int i = 1; i < n; ++i){
    if (a[i].first == a[i - 1].first)
        c[p[i]] = c[p[i - 1]];
    else c[p[i]] = c[p[i - 1]] + 1;
}

}

int k = 0;
while ((1 << k) < n){ //k -> k + 1
    for (int i = 0; i < n; ++i){
        p[i] = (p[i] - (1 << k) + n) % n;
    }

    count_sort(p,c);
    vector <int> c_new(n);
    c_new[p[0]] = 0;
    for (int i = 1; i < n; ++i){
        pair<int,int> now = {c[p[i]],c[(p[i] + (1 << k)) %
n]};
        pair<int,int> prev = {c[p[i - 1]],c[(p[i - 1] + (1
<< k)) % n]};
        if (now == prev) c_new[p[i]] = c_new[p[i - 1]];
        else c_new[p[i]] = c_new[p[i - 1]] + 1;
    }
    c = c_new;
    k++;
}

```

```

}
vector <int> lcp(n);
k = 0;
for (int i = 0; i < n - 1; ++i){
    int pi = c[i];
    int j = p[pi - 1];
    // lcp[i] = lcp(s[i..],s[j..])
    while (s[i + k] == s[j + k]) k++;
    lcp[pi] = k;
    k = max(k - 1,0);
}
}

```

mảng LCP để tính longest common prefix của 2 suffix  $i$  và  $i - 1$ , tính lcp của 2 prefix bất kì, có thể dùng sparse table

```

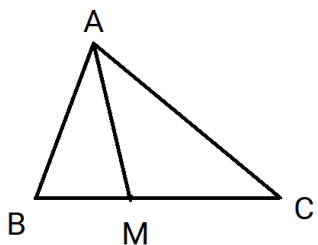
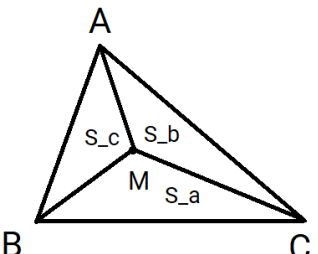
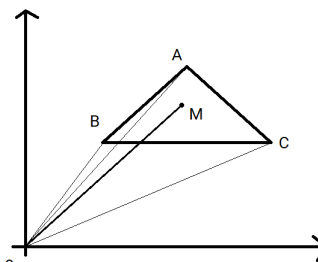
void process2(int M[MAXN][LOGMAXN], int A[MAXN], int N){
    int i, j;

    // Khởi tạo M với các khoảng độ dài 1
    for (i = 0; i < N; i++)
        M[i][0] = i;

    // Tính M với các khoảng dài  $2^j$ 
    for (j = 1; 1 << j <= N; j++)
        for (i = 0; i + (1 << j) - 1 < N; i++)
            if (A[M[i][j - 1]] < A[M[i + (1 << (j - 1))][j - 1]])
                M[i][j] = M[i][j - 1];
            else
                M[i][j] = M[i + (1 << (j - 1))][j - 1];
}

```

## Các bổ đề vector

|  |  |
|--|--|
|   | $\overrightarrow{AM} = \frac{MC}{BC} \cdot \overrightarrow{AB} + \frac{MB}{BC} \cdot \overrightarrow{AC}$  |
|   | $S_a \cdot \overrightarrow{MA} + S_b \cdot \overrightarrow{MB} + S_c \cdot \overrightarrow{MC} = \vec{0}$  |
|  | <p>Cho tam giác ABC. Có một bộ (a,b,c) thỏa :</p> $a \cdot \overrightarrow{OA} + b \cdot \overrightarrow{OB} + c \cdot \overrightarrow{OC} = \overrightarrow{OM}.$ <p>Cả 3 số a,b,c đều dương <math>\Leftrightarrow</math> M nằm trong tam giác ABC.</p> |

## Heavy-light Decomposition (HLD)

Đề: Update và Query từ nút u tới nút v trên một cây.

Giải thích:

- HLD giúp cắt cây thành những nhánh dài, sau đó có thể dùng Segment tree để update, query
- Với nChain là số chuỗi, các chuỗi là: 0, 1, ... nChain.
- Với mỗi chuỗi, biết đỉnh của nó với chainHead,
- Với một đỉnh, biết chuỗi mà nó nằm trong với chainInd
- Mảng posInBase[] lưu lại vị trí của các đỉnh sau khi chúng ta "trải" các chuỗi trên lên một đường thẳng => giúp cài đặt Segment gọn gàng hơn.

---

```
const int maxN=1e4;
```

```
int n;
vector<int> adj[maxN+1];
int nChain=0, nBase=0, chainHead[maxN+1], chainInd[maxN+1],
posInBase[maxN+1];
int parent[maxN+1], nChild[maxN+1];
```

```
// nChain chuỗi hiện tại. Sau khi kết thúc việc phân tách thì
// đây sẽ là tổng số chuỗi.
```

```
// chainHead[c] đỉnh đầu của chuỗi c
```

```
// chainInd[u] chuỗi mà đỉnh u nằm trong.
```

```
void hld(int u) {
    // Nếu chuỗi hiện tại chưa có đỉnh đầu (đỉnh gần gốc nhất)
    // thì đặt u làm đỉnh đầu của nó.
    if (chainHead[nChain] == 0) chainHead[nChain] = u;
```

```

// Gán chuỗi hiện tại cho u
chainInd[u] = nChain;

// Giải thích bên dưới
posInBase[u] = ++nBase;

// Biến lưu đỉnh con đặc biệt của u
int mxVtx = -1;

// Tìm đỉnh con đặc biệt trong số những đỉnh con của u
for (int i = 0; i < adj[u].size(); i++) {
    int v = adj[u][i];
    if (v != parent[u]) {
        if (mxVtx == -1 || nChild[v] > nChild[mxVtx]) {
            mxVtx = v;
        }
    }
}

// Nếu tìm ra đỉnh con đặc biệt (u không phải là đỉnh lá)
thì di chuyển đến đỉnh đó
if (mxVtx > -1) hld(mxVtx);

// Sau khi đi hết một chuỗi thì tăng nChain lên và bắt đầu
một chuỗi mới
for (int i = 0; i < adj[u].size(); i++) {
    int v = adj[u][i];
    if (v != parent[u] && v != mxVtx) {
        nChain++;
        hld(v);
    }
}

```

```

}

void dfs(int u, int pre) {
    int height=1;
    for (int v:adj[u]) {
        if (v==pre) continue;
        parent[v]=u;
        dfs(v, u);
        height=max(nChild[v], height);
    }
    nChild[u]=height;
}

int main() {
    cin>>n;

    for (int i=1; i<n; ++i) {
        int u,v; cin>>u>>v;
        adj[u].push_back(v);
        adj[v].push_back(u);
    }

    parent[1]=-1;
    dfs(1, -1);

    // Lưu ý phải khởi tạo chainHead
    for (int i=0; i<n; ++i) chainHead[i]=0;
    hld(1);

    return 0;
}

```

Mã giả của hàm update:

---

```
void update(int u, int a) {
    // uchain chuỗi hiện tại của u
    // achain chuỗi của a
    int uchain = chainInd[u], achain = chainInd[a];

    while (1) {
        // Nếu u và a cùng nằm trên một chuỗi thì update đoạn từ
        // u đến a và kết thúc.
        if (uchain == achain) {
            updateIntervalTree(..., posInBase[a], posInBase[u],
...);
            break;
        }
        // Nếu u và a không nằm trên cùng một chuỗi thì update
        // đoạn từ u đến đỉnh đầu của chuỗi hiện tại.
        updateIntervalTree(..., posInBase[chainHead[uchain]],
posInBase[u], ...);

        // Nhảy lên đỉnh cha của đỉnh đầu hiện tại.
        u = parent[chainHead[uchain]];
        uchain = chainInd[u];
    }
}
```

---

## Đường đi ngắn nhất

### Dijkstra

---

```
const long long INF = 2000000000000000LL;
struct Edge{// kiểu dữ liệu tự tạo để lưu thông số của một cạnh.
    int v;
    long long w;
};
struct Node{// kiểu dữ liệu để lưu đỉnh u và độ dài của đường đi
    ngắn nhất từ s đến u.
    int u;
    long long Dist_u;
};
struct cmp{
    bool operator() (Node a, Node b) {
        return a.Dist_u > b.Dist_u;
    }
};
void dijkstraSparse(int n, int s, vector<vector<Edge>> &E,
vector<long long> &D, vector<int> &trace) {
    D.resize(n, INF);
    trace.resize(n, -1);
    vector<bool> P(n, 0);

    D[s] = 0;
    priority_queue<Node, vector<Node>, cmp> h; // hàng đợi ưu
    tiên, sắp xếp theo dist[u] nhỏ nhất trước
    h.push({s, D[s]});

    while(!h.empty()) {
        Node x = h.top();
```

```
h.pop();

int u = x.u;
if(P[u] == true) // Đỉnh u đã được chọn trước đó, bỏ qua
    continue;

P[u] = true; // Đánh dấu đỉnh u đã được chọn
for(auto e : E[u]) {
    int v = e.v;
    long long w = e.w;

    if(D[v] > D[u] + w) {
        D[v] = D[u] + w;
        h.push({v, D[v]});
        trace[v] = u;
    }
}
}
```

---

Để tracing trong thuật toán Dijkstra

```
vector<int> trace_path(vector<int> &trace, int S, int u) {
    if (u != S && trace[u] == -1) return vector<int>(0); //
không có đường đi

    vector<int> path;
    while (u != -1) { // truy vết ngược từ u về S
        path.push_back(u);
        u = trace[u];
    }
}
```

```
reverse(path.begin(), path.end()); // cần reverse vì đường
đi lúc này là từ u về S

return path;
}
```

---

### Bellman Ford

---

```
const long long INF = 2000000000000000LL;
struct Edge {
    int u, v;
    long long w; // cạnh từ u đến v, trọng số w
};

void bellmanFord(int n, int S, vector<Edge> &e, vector<long
long> &D, vector<int> &trace) {
    // e: danh sách cạnh
    // n: số đỉnh
    // S: đỉnh bắt đầu
    // D: độ dài đường đi ngắn nhất
    // trace: mảng truy vết đường đi
    // INF nếu không có đường đi
    // -INF nếu có đường đi âm vô tận
    D.resize(n, INF);
    trace.resize(n, -1);

    D[S] = 0;
    for(int T = 1; T < n; T++) {
        for (auto E : e) {
            int u = E.u;
            int v = E.v;
            long long w = E.w;
            if (D[u] != INF && D[v] > D[u] + w) {
```



```

        D[v] = D[u] + w;
        trace[v] = u;
    }
}
}

```

Truy vết trong Belman Ford:

```

vector<int> trace_path(vector<int> &trace, int S, int u) {
    if (u != S && trace[u] == -1) return vector<int>(0); //
    không có đường đi

    vector<int> path;
    while (u != -1) { // truy vết ngược từ u về S
        path.push_back(u);
        u = trace[u];
    }
    reverse(path.begin(), path.end()); // cần reverse vì đường
    đi lúc này là từ u về S

    return path;
}

```

### Chu trình âm trong Bellman Ford

```

// sau khi chạy xong N-1 vòng lặp Bellman-Ford
for(int T = 0; T < n; T++){
    for (auto E : e) {
        int u = E.u;
        int v = E.v;

```

```

        long long w = E.w;
        if (D[u] != INF && D[v] > D[u] + w) {
            // vẫn còn tối ưu được --> âm vô cực
            D[v] = -INF;
            trace[v] = u;
        }
    }
}

```

Tìm chu trình âm:

```

bool findNegativeCycle(int n, vector<long long> &D, vector<int>
&trace, vector<int> &negCycle) {
    // mảng D và trace đã được chạy qua thuật toán Bellman-Ford
    int negStart = -1; // đỉnh bắt đầu
    for (auto E : e) {
        int u = E.u;
        int v = E.v;
        long long w = E.w;
        if (D[u] != INF && D[v] > D[u] + w) {
            D[v] = -INF;
            trace[v] = u;
            negStart = v; // đã tìm thấy -INF
        }
    }

    if (negStart == -1) return false; // không có chu trình âm

    int u = negStart;
    for (int i = 0; i < n; i++) {
        u = trace[u]; // đưa u về chu trình âm

```

```
    }

    negCycle = vector<int>(1, u);
    for (int v = trace[u]; v != u; v = trace[u]) {
        negCycle.push_back(v); // truy vết một vòng
    }
    reverse(negCycle.begin(), negCycle.end());

    return true;
}
```

---

### Floyd Warshal

- $W[u][v]$  lưu trọng số cạnh, không có cạnh thì là dương vô cùng
- $D[u][v]$  lưu trọng số đường đi ngắn nhất, khởi tạo =  $W[u][v]$

---

```
void init_trace(vector<vector<int>> &trace) {
    int n = trace.size();
    for (int u = 0; u < n; u++) {
        for (int v = 0; v < n; v++) {
            trace[u][v] = u;
        }
    }
}

void floydWarshall(int n, vector<vector<long long>> &w,
vector<vector<long long>> &D, vector<vector<int>> &trace) {
    D = w;
    init_trace(trace); // nếu cần dò đường đi

    for (int k = 0; k < n; k++) {
```

---

```
        for (int u = 0; u < n; u++) {
            for (int v = 0; v < n; v++) {
                if (D[u][v] > D[u][k] + D[k][v]) {
                    D[u][v] = D[u][k] + D[k][v];
                    trace[u][v] = trace[k][v];
                }
            }
        }
    }
}
```

---

### Truy vết trong Floyd Warshal:

---

```
vector<int> trace_path(vector<vector<int>> &trace, int u, int v)
{
    vector<int> path;
    while (v != u) { // truy vết ngược từ v về u
        path.push_back(v);
        v = trace[u][v];
    }
    path.push_back(u);

    reverse(path.begin(), path.end()); // cần reverse vì đường
    đi từ v ngược về u

    return path;
}
```

---

**Hash**

---

```
const int P = 1e6 + 3;

struct HashTable {
    vector< pair<int,int> > h[P];

public:
    void insert(int key, int value) {
        int hkey = getHash(key);
        for (auto p : h[hkey]) {
            if (p.first == key) {
                // key da ton tai trong Hash table, ta bo qua
                return;
            }
        }
        // Them (key, value) vao hash table
        h[hkey].emplace_back(key, value);
    }

    int find(int key) {
        int hkey = getHash(key);
        for(auto p : h[hkey]) {
            if (p.first == key) {
                // ton tai key trong Hash table, return value
                return p.value;
            }
        }
        // Khong tim thay
        return 0;
    }
}
```

```
private:
    int getHash(int key) {
        // Cho 1 key, tra lai Hash value la key % P
        return key % P;
    }
};
```

---

**Thuật toán Kruskal - Spanning tree**


---

```

/*input
4 4
1 2 1
2 3 2
3 4 3
4 1 4
*/
#include <bits/stdc++.h>
using namespace std;

// Cấu trúc để lưu các cạnh đồ thị
// u, v là 2 đỉnh, c là trọng số cạnh
struct Edge {
    int u, v, c;
    Edge(int _u, int _v, int _c): u(_u), v(_v), c(_c) {};
};

struct Dsu {
    vector<int> par;

    void init(int n) {
        par.resize(n + 5, 0);
        for (int i = 1; i <= n; i++) par[i] = i;
    }

    int find(int u) {
        if (par[u] == u) return u;
        return par[u] = find(par[u]);
    }
};

```

---

```

bool join(int u, int v) {
    u = find(u); v = find(v);
    if (u == v) return false;
    par[v] = u;
    return true;
}
} dsu;

// n và m là số đỉnh và số cạnh
// totalWeight là tổng trọng số các cạnh trong cây khung nhỏ nhất
int n, m, totalWeight = 0;
vector< Edge > edges;

int main() {
    // Fast IO
    ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    cin >> n >> m;

    for (int i = 1; i <= m; i++) {
        int u, v, c;
        cin >> u >> v >> c;
        edges.push_back({u, v, c});
    }

    dsu.init(n);

    // Sắp xếp lại các cạnh theo trọng số tăng dần
    sort(edges.begin(), edges.end(), [](Edge & x, Edge & y) {
        return x.c < y.c;
    });
};

```

```
// Duyệt qua các cạnh theo thứ tự đã sắp xếp
for (auto e : edges) {
    // Nếu không hợp nhất được 2 đỉnh u và v thì bỏ qua
    if (!dsu.join(e.u, e.v)) continue;

    // Nếu hợp nhất được u, v ta thêm trọng số cạnh vào kết
    quả
    totalWeight += e.c;
}

// Xuất ra kết quả
cout << totalWeight << '\n';
}
```

---

### Tarjan

- Hằng số maxN = 100010
- Biến timeDfs - Thứ tự DFS
- Biến scc - Số lượng thành phần liên thông mạnh
- Mảng low[], num[]
- Mảng deleted[] - Đánh dấu các đỉnh đã bị xóa
- Vector g[] - Danh sách cạnh kề của mỗi đỉnh
- Ngăn xếp st - Lưu lại các đỉnh trong thành phần liên thông mạnh

---

```
#include <bits/stdc++.h>

using namespace std;

const int maxN = 100010;

int n, m;
int timeDfs = 0, scc = 0;
int low[maxN], num[maxN];
bool deleted[maxN];
vector <int> g[maxN];
stack <int> st;

void dfs(int u) {
    num[u] = low[u] = ++timeDfs;
    st.push(u);
    for (int v : g[u]) {
        if (deleted[v]) continue;
        if (!num[v]){
            dfs(v);
            low[u] = min(low[u], low[v]);
        }
    }
}
```

```
        else low[u] = min(low[u], num[v]);
    }
    if (low[u] == num[u]) {
        scc++;
        int v;
        do {
            v = st.top();
            st.pop();
            deleted[v] = true;
        }
        while (v != u);
    }
}

int main() {
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v;
        cin >> u >> v;
        g[u].push_back(v);
    }
    for (int i = 1; i <= n; i++)
        if (!num[i]) dfs(i);

    cout << scc;
}
```

---