ICPC National 2022

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

STL Help

Customize set<>

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

Hàm unique()

Hàm unique loại bỏ các phần tử giống nhau liền 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 & Lazy (Lưu ý có thể chặt nhị phân trên segment tree trong các bước tìm kiếm)

```
int ST[4*NUM];
int lazy[4*NUM];
void down(int rt) {
    lazy[2*rt] += lazy[rt];
    lazy[2*rt+1] += lazy[rt];
    st[2*rt] += lazy[rt];
    st[2*rt+1] += lazy[rt];
    lazy[rt] = 0;
}
void update(int rt, int l, int r, int u, int v, int val) {
    if (v < l || r < u) {return ;}
    if (u <= l && r <= v) {
       ST[rt] += val; lazy[rt] += val; return;  }
    down(rt);
    update(rt<<1, l, (l+r)>>1, idx, val);
    update(rt<<1 | 1, ((l+r)>>1) + 1, r, idx, val);
//merge(st[2*id].begin(),st[2*id].end(),st[2*id+1].begin(),st[2*
id+1].end(),back_inserter(st[id]));
    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];
    down(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;}
```

Persistent IT & BIT

```
//IT
struct Node {
    int val; Node* l; Node* r;
    Node()  {};
    Node(int val, Node* l, Node* r) : val(val) , l(l) , r(r) {};
};
void build(Node* root, int l,int r) {
    if (l == r) {
        root->val = a[l];
        return;
    }
    int mid = (l + r)>>1;
    root->l = new Node(0,NULL,NULL);
    root->r = new Node(0,NULL,NULL);
    build(root->l,l,mid);
    build(root->r,mid+1,r);
    root->val = root->l->val + root->r->val;
}
int query(Node* root,int l,int r,int u,int v) {
    if (v < l || u > r) return 0;
    if (u <= l && r <= v) return root->val;
    int mid=(l+r)>>1;
    return query(root->l,l,mid,u,v)+query(root->r,mid+1,r,u,v);
}
void upgrade(Node* prev, Node* cur, int l, int r, int u, int x)
{
    if (u < l || u > r || l > r) return;
    if (l == r) { cur->val = x; return;}
    int mid=(l+r)>>1;
    if (u <= mid) {
```

```
        cur->r = prev->r; cur->l = new Node(0,NULL,NULL);
        upgrade(prev->l,cur->l,l,mid,u,x);
    }
    else {
        cur->l = prev->l; cur->r = new Node(0,NULL,NULL);
        upgrade(prev->r,cur->r,mid+1,r,u,x);
    }
    cur->val = cur->l->val + cur->r->val;
}
//BIT
void update(int u, int val, int time) {
    while(u <= N) {
        if (BIT[u].empty()) BIT[u].push_back({time,val});
        else {
            int cur = BIT[u][BIT[u].size() - 1].second;
            BIT[u].push_back({time,cur+val});
        }u += (u & (-u));}}
int get(int u, int time) {
    int res = 0;
    while (u > 0) {
        if (BIT[u].size()) {
            if (BIT[u][BIT[u].size() - 1].first <= time) {
                res += BIT[u][BIT[u].size() - 1].second;
            }
            else {
                int pos =
upper_bound(BIT[u].begin(),BIT[u].end(),make_pair(time,maxN)) -
BIT[u].begin() - 1;
                if (pos >= 0) res += BIT[u][pos].second;
            }
        }u -= (u & (-u));}return res;}
```

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)

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 = 1e5 + 1;
vector <int> chainHead(maxN,0);
vector <int> chainInd(maxN,0);
vector <int> posInBase(maxN,0);
vector <int> g[maxN];
vector <int> par(maxN,-1);
vector <int> visited(maxN,0);
vector <int> nChild(maxN,0);

void dfs(int u) {
    visited[u] = true;
    nChild[u] += g[u].size();
    for (auto v : g[u]) {
        if (visited[v]) continue;
        par[v] = u;
        dfs(v);
        nChild[u] += nChild[v];
    }
```

```
}
void hld(int u) {
    //chuoi dau
    if (chainHead[nChain] == 0) chainHead[nChain] = u;
    chainInd[u] = nChain; //danh dau, u thuoc nChain
    posInBase[u] = ++nBase;
    int mVtx = -1;
    for (auto v : g[u]) {
        if (v != par[u]) {
            if (mVtx == -1 || nChild[v] > nChild[mVtx]) {
                mVtx = v;
            }
        }
    }
    if (mVtx > -1) hld(mVtx);
    for (auto v : g[u]) {
        if (v != par[u] && v != mVtx) {
            nChain++; hld(v);}}}
 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) {
        if (uchain == achain) {
            updateIntervalTree(..., posInBase[a], posInBase[u],
...);
            break;
        }
        updateIntervalTree(..., posInBase[chainHead[uchain]],
posInBase[u], ...);
        u = parent[chainHead[uchain]];
       uchain = chainInd[u];}}
```

Z-algorithm

Đề: Cho chuỗi S. Tạo mảng Z sao cho Zi là k sao cho Si…Si+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]);
}
```

Matching (không trọng số)

```
int count_ = 0;
bool bfs() {
    queue <int> Q;
    for (int u = 1; u <= M; ++u) {
        if (!mx[u]) {
            dist[u] = 0;
            Q.push(u);
        } else dist[u] = oo;
    }

    bool found = false;
    while (!Q.empty()) {
        int u = Q.front();
        Q.pop();
        for (auto v : g[u]) {
            if (!my[v]) {
                found = true;
                break;
            } else if (dist[my[v]]==oo) {
                dist[my[v]] = dist[u] + 1;
                Q.push(my[v]);
            }
        }
    }
    return found;
}

bool dfs(int u) {
    if (dist[u] == oo) return false;
    for (auto v : g[u]) {
```

```
        if (!my[v] || (dist[my[v]] == dist[u] + 1 &&
dfs(my[v]))) {
            mx[u] = v;
            my[v] = u;
            return true;
        }
    }
    return false;
}

void match() {
    while (bfs()) {
        for (int u = 1; u <= M; ++u) {
            if (!mx[u]) count_ += dfs(u);
        }
    }
}//Định lý KONIG
void konig() {
    queue <int> que;
    for (int i = 1; i <= M; ++i) if (!mx[i]) que.push(i);
    for (int i = 1; i <= N; ++i) if (!my[numV - i])
que.push(numV - i);
    while (que.size()) {
        int u = que.front();
        que.pop();
        for (auto v : g[u]) {
            if (!choose[v]) {
                choose[v] = true;
                if (!mx[v]) que.push(mx[v]);
                if (!my[v]) que.push(my[v]);
            }
        }
```

```
    }
    for (int i = 1; i <= M; ++i)
        if (mx[i] && !choose[mx[i]] && !choose[i]) choose[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;
}
```

Giải thích: Nếu gcd(a,m) = 1 thì a^phi(m) ≡ 1 (mod). Suy ra a^(phi(m)-1) ≡ a^-1

Tính tất cả nghịch đảo modulo <= m nguyên tố

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

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 vế được ax ≡ 1 (mod 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;
    }
}
```

Nhân ma trận

```
using type = int; // Kiểu dữ liệu các phần tử của ma trận

//matrix square power K (Size N): O(N^3log(K))
//property: Cij = k[1:n]sigma(Aik * Bkj) <=> new problem: Cij = k[1:n]min(Aik + Bkj)

struct Matrix {
    vector <vector <type> > data;
    // Số lượng hàng của ma trận
    int row() const { return data.size(); }
    // Số lượng hàng của ma trận
    int col() const { return data[0].size(); }
    auto & operator [] (int i) { return data[i]; }
    const auto & operator[] (int i) const { return data[i]; }
    Matrix() = default;
    Matrix(int r, int c): data(r, vector <type> (c)) { }
    // In ra ma trận.
    friend ostream & operator << (ostream &out, const Matrix &d) {
        for (auto x : d.data) {
            for (auto y : x) out << y << ' ';
            out << '\n';
        }
        return out;
    }

    // Ma trận đơn  vị
    static Matrix identity(long long n) {
        Matrix a = Matrix(n, n);
        while (n--) a[n][n] = 1;
        return a;
    }
```

```
    // Nhân ma trận
    Matrix operator * (const Matrix &b) {
        Matrix a = *this;

        // Kiểm tra điều kiện nhân ma trận
        assert(a.col() == b.row());

        Matrix c(a.row(), b.col());
        for (int i = 0; i < a.row(); ++i)
            for (int j = 0; j < b.col(); ++j)
                for (int k = 0; k < a.col(); ++k)
                    c[i][j] += a[i][k] * b[k][j];
        return c;
    }

    // Lũy thừa ma trận
    Matrix pow(long long exp) {

        // Kiểm tra điều kiện lũy thừa ma trận (là ma trận vuông)
        assert(row() == col());

        Matrix base = *this, ans = identity(row());
        for (; exp > 0; exp >>= 1, base = base * base)
            if (exp & 1) ans = ans * base;
        return ans;
    }
};
```

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;}
```

Flow (edmondskarp)

1. Tìm đường tăng luồng và Tăng luồng
2. Nếu không tăng được nữa thì dừng lại.
3. Thông thường BFS (sẽ tốt hơn khi tìm đường tăng luồng)

```
//O(log(|f|)*E)
bool bfs(int start, int target) {
    queue <int> que;
    memset(trace,0,sizeof(trace));
    trace[start] = -1;
    que.push(start);
    while (que.size()) {
        int u = que.front();
        que.pop();
        if (u == target) return true;
        for (auto v : g[u]) {
            if (trace[v] == 0 && f[u][v] < c[u][v]) {
                trace[v] = u;
                que.push(u);
            }
        }
    }
    return false;
}

int min_cut(bool tracing = false) {
    int r = 0;

    for (int u = 1; u <= n; ++u) {
        for (auto v : g[u]) {
            if (trace[u] && !trace[v]) {
                r += c[u][v];
            }
```

```
            if (tracing) cout << u << " " << v << endl;
        }
    }

    return r;
}

void enlarge() {
    int delta = maxN;
    for (int i = target; i != start; i = trace[i])
        delta = min(delta,c[trace[i]][i] - f[trace[i]][i]);
    for (int i = target; i != start; i = trace[i]) {
        f[trace[i]][i] += delta;
        f[i][trace[i]] -= delta;
    }
}
```

Một số biến thể:

- **Định mức nút:** tách nút u thành u_in = 2 * u; u_out = 2* u - 1
- **Multi source, multi target:** tạo 2 đỉnh start và target giả kết nối với các source và target.
- **Định mức cạnh:**

  $c'((s', v)) = \sum_{u \in V} d((u, v))$ for each edge (s', v).

  $c'((v, t')) = \sum_{w \in V} d((v, w))$ for each edge (v, t').

  $c'((u, v)) = c((u, v)) - d((u, v))$ for each edge (u, v) in the old network.

  $c'((t, s)) = \infty$

- **Min-cost max-flow:**
  Dùng spfa để tìm đường đi ngắn nhất (lưu ý chu trình âm)
  Với đường đi ngắn nhất vừa tìm được tăng luồng theo đường đi ngắn nhất đó.

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(nlogn).

```
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];
}
```

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

Bellman Ford

```
bool spfa(ll s) {
    queue <ll> q;
    d[s] = 0;
    in_queue[s] = true;
    q.push(s);
    while (q.size()) {
        ll u = q.front();
        q.pop();
        in_queue[u] = false;
        for (auto edge : g[u]) {
            ll v = edge.v;
            ll w = edge.w;
            if (d[u] != INF && d[u] + w < d[v]) {
                d[v] = d[u] + w;
                trace[v] = u;
                if (!in_queue[v]) {
                    q.push(v);
                    in_queue[v] = true;
                    cnt[v]++;
                    if (cnt[v] > N) {
                        check = v;
                        return false;
                    }
                }}}}
    return true;
}

//tìm chu trình âm
int u = check;
```

```
for (int i = 0; i < N; ++i) u = trace[u];
vector <int> negCycle;
negCycle.push_back(u);
for (int v = trace[u]; v != u; v = trace[v]) {
     negCycle.push_back(v);
}
reverse(negCycle.begin(),negCycle.end());
for (auto ele : negCycle) cout << ele << " ";
////////////////////////////////////
vector <int> trace_path(int S,int u) {
    if (u != S && trace[u] == -1) return vector <int> (0);
    vector <int> path;
    while (u != -1) {
        path.push_back(u);
        u = trace[u];
    }
    reverse(path.begin(),path.end());
    return path;
}
```

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());
    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() {
    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';
}
//prim
int prim(int s) {
    int count = 0;
    dist.resize(N + 1);
    fill(dist.begin(),dist.end(),oo);
    priority_queue <pii,vector<pii>,greater<pii>> pq;
    dist[s] = 0;
    pq.push({dist[s],s});
    while (!pq.empty()) {
        int u = pq.top().second;
        int w = pq.top().first;
        pq.pop();
        if (w != dist[u]) {
            continue;
        }
        count += w;
        dist[u] = -oo; //mark xet roi
        for (auto vc : g[u]) {
            int v = vc.first;
            int c = vc.second;
            if (dist[v] > c) {
                dist[v] = c;
                pq.push({dist[v],v});
            }
        }
    }
    return count;
```

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;
}
```

Basic Geometry

```
bool inter_straight(double a, double b, double c, double d) {
   if (a > b) swap(a,b);
   if (c > d) swap(c,d);
   return max(a,c) <= max(b,d);
}

bool check_intersection(const point& a, const point& b, const point& c, const
point& d) {
   if (abs(c.cross(a,d)) <= EXP && abs(c.cross(b,d)) <= EXP) return
inter_straight(a.x,b.x,c.x,d.x) && inter_straight(a.y,b.y,c.y,d.y);
   return sign(a.cross(b,d)) != sign(a.cross(b,c)) &&
      sign(c.cross(a,d)) != sign(c.cross(b,d));
}

ftype abs(point2d a) {
   return sqrt(norm(a));
}
ftype proj(point2d a, point2d b) {
   return dot(a,b) / abs(b);
}
ftype angle(point2d a, point2d b) {
   return acos(dot(a,b)/(abs(a)*abs(b)));
}
double cross(point2d a, point2d b) {
   return a.x * b.y - a.y * b.x;
}
//get sign_triangle
double get_sign_triangle(point2d p1, point2d p2, point2d p3) {
   return cross(p2 - p1,p3 - p2);
}
```

```
double get_triangle(point2d p1, point2d p2, point2d p3) {
    double res = get_sign_triangle(p1,p2,p3) * 1.0/2;
    if (abs(res) <= ESP) return 0;
    return res;
}
double clockwise(point2d p1, point2d p2, point2d p3) {
    return get_sign_triangle(p1,p2,p3) < 0;
}
double counter_clockwise(point2d p1, point2d p2, point2d p3) {
    return get_sign_triangle(p1,p2,p3) > 0;
}
```

CHECK MỘT ĐIỂM CÓ TRONG POLYGON

```
struct pt {
    ll x,y;
    pt() {};
    pt(ll x, ll y) : x(x), y(y) {};
    pt operator+(const pt &p) const {return pt(x + p.x,y + p.y);}
    pt operator-(const pt &p) const {return pt(x - p.x,y - p.y);}
    ll cross(const pt& p) const {return x * p.y - y * p.x;}
    ll dot(const pt& p) const {return x * p.x + y * p.y;}
    ll cross(const pt& a, const pt& b) const {return (a - *this).cross(b - *this);
    }
    ll dot(const pt& a, const pt& b) const {return (a - *this).dot(b - *this);}
    ll sqrLen() const {return this->dot(*this);}
};

bool lexComp(const pt& l, const pt& r) {
    return l.x < r.x || (l.x == r.x && l.y < r.y);
```

```
}

int sign(ll val) {
    return val > 0 ? 1 : (val == 0 ? 0 : -1);
}

vector<pt> seq;
vector<pt> points;
pt translation;
int n;

bool pointInTriangle(pt a, pt b, pt c, pt point) {
    ll s1 = abs(a.cross(b,c));
    ll s2 = abs(point.cross(a,b)) + abs(point.cross(b,c)) + abs(point.cross(c,a));
    return s1 == s2;
}

void prepare() {
    int pos = 0;
    for (int i = 1; i < points.size(); ++i) {
        if (lexComp(points[i],points[pos])) {
            pos = i;
        }
    }
    rotate(points.begin(),points.begin()+pos,points.end());
    for (int i = 0; i < points.size() - 1; ++i) {
        seq[i] = points[i + 1] - points[0];
    }
    translation = points[0];
}

bool pointInConvexPolygon(pt point) {
```

```
    point = point - translation;
    int n = seq.size();
    if (seq[0].cross(point) != 0 && sign(seq[0].cross(point)) !=
sign(seq[0].cross(seq[n-1])))
        return false;
    if (seq[n-1].cross(point) != 0 && sign(seq[n-1].cross(point)) !=
sign(seq[n-1].cross(seq[0])))
        return false;
    if (seq[0].cross(point) == 0) {
        return seq[0].sqrLen() >= point.sqrLen();
    }
    int l = 0;
    int r = n - 1;
    while (r - l > 1) {
        int mid = (r+l)>>1;
        if (seq[mid].cross(point)>=0) l = mid;
        else r = mid;
    }
    int pos = l;
    return pointInTriangle(points[pos],points[pos+1],points[0],point+translation);
}

//area - Shoelace formula (O(N))
double get_area(vector<pair<double,double>>points) {
    double res = 0.0;
    for (int i = 0; i < points.size(); ++i) {
        pair<double,double> q = i ? points[i - 1] : points.back();
        res += (points[i].x - q.x) * (points[i].y + q.y);
    }
    return abs(res) / 2;
}
```

MINKOWSKI-SUM (tìm khoảng cách giữa 2 polygon)

```
bool complex_y(struct pt& a, struct pt& b) {
   return a.y < b.y || (a.y == b.y && a.x < b.x);
}

void reorder(vector <pt>& points) {
   int pos = 0;
   for (int i = 1; i < points.size(); ++i) {
      if (complex_y(points[i],points[pos])) {
         pos = i;
      }
   }
   rotate(points.begin(),points.begin()+pos,points.end());
}
vector <pt> get_Minkowski() {
   reorder(P);
   reorder(Q);
   //ensure cycle
   P.push_back(P[0]);
   P.push_back(P[1]);
   Q.push_back(Q[0]);
   Q.push_back(Q[1]);
   int i = 0;
   int j = 0;
   vector <pt> res;
   while (i < P.size() - 2 && j < Q.size() - 2) {
      res.push_back(P[i] + Q[j]);
      int comp = (P[i+1] - P[i]).cross(Q[i+1] - Q[i]);
      if (comp >= 0) ++i;
      if (comp <= 0) ++j;}    return res;}
```

LCA (Binary-Lifting)

```
int up[maxN][17];
void preprocess_k() {
    for (int u = 1; u <= n; ++u) up[u][0] = par[u];
    for (int j = 1; j < 17; j++) {
        for (int u = 1; u <= n; ++u) {
            up[u][j] = up[up[u][j - 1]][j - 1];
        }
    }
}
int ancestor_u_v(int u,int v) {
    //h[ancestor_i[u]] = h[v]
    int k = h[u] - h[v];
    for (int j = 0; (1 << j) <= k; ++j)
        if (k >> j & 1) u = up[u][j];
    return u;
}

int LCA(int u,int v) {
    if (u == v) {
        return u;
    }
    if (h[u] < h[v]) swap(u,v);
    //find ancestor u so: h[ancestor_i[u]] = h[v]
    u = ancestor_u_v(u,v);
    int k = __lg(h[u]);
    for (int j = k; j >= 0; j--) {
        if (up[u][j] != up[v][j]) u = up[u][j], v = up[v][j];
    }
    return up[u][0];
}
```

STRONG BICONNECTED

Nén các thành phần liên thông mạnh lại thành một đỉnh

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

void dfs(int u) {
   visited[u] = true;
   par[u] = u;
   stk.push(u);

   for (auto v : g[u]) {
      if (visited[v]) {
         v = find(child_first[v]);
         while (stk.size() && stk.top() != v) {
            par[find(stk.top())] = v;
            stk.pop();
         }
      }
   }

   for (auto v : g[u]) {
      if (!visited[v]) {
         child_first[u] = v;
         dfs(v);
      }
   }

   if (stk.top() == u) stk.pop();}
```

HUNGARY (MATCHING WITH WEIGHT)

```
long long c[MN][MN];
long long fx[MN], fy[MN];
int mx[MN], my[MN];
int trace[MN], qu[MN], arg[MN];
long long d[MN];
int front, rear, start, finish;
void init() {
   FOR(i,1,N) {
      fy[i] = mx[i] = my[i] = 0;
      FOR(j,1,N) c[i][j] = inf;
   }
}
void addEdge(int i, int j, long long cost) {
   c[i][j] = min(c[i][j], cost);
}
inline long long getC(int i, int j) {
   return c[i][j] - fx[i] - fy[j];
}
void initBFS() {
   front = rear = 1;
   qu[1] = start;
   memset(trace, 0, sizeof trace);
   FOR(j,1,N) {
      d[j] = getC(start, j);
      arg[j] = start;
   }
   finish = 0;
}
void findAugPath() {
   while (front <= rear) {
```

```
        int i = qu[front++];
        FOR(j,1,N) if (!trace[j]) {
            long long w = getC(i, j);
            if (!w) {
                trace[j] = i;
                if (!my[j]) {
                    finish = j;
                    return ;
                }
                qu[++rear] = my[j];
            }
            if (d[j] > w) {
                d[j] = w;
                arg[j] = i;
            }
        }
    }
}
void subx_addy() {
    long long delta = inf;
    FOR(j,1,N)
        if (trace[j] == 0 && d[j] < delta) delta = d[j];
    // xoay
    fx[start] += delta;
    FOR(j,1,N)
        if (trace[j]) {
            int i = my[j];
            fy[j] -= delta;
            fx[i] += delta;
        }
        else d[j] -= delta;
    FOR(j,1,N)
```

```
        if (!trace[j] && !d[j]) {
            trace[j] = arg[j];
            if (!my[j]) { finish = j; return ; }
            qu[++rear] = my[j];
        }
    }
    void enlarge() {
        do {
            int i = trace[finish];
            int next = mx[i];
            mx[i] = finish;
            my[finish] = i;
            finish = next;
        } while (finish);
    }
    long long mincost() {
        FOR(i,1,N) fx[i] = *min_element(c[i]+1, c[i]+N+1);
        FOR(j,1,N) {
            fy[j] = c[1][j] - fx[1];
            FOR(i,1,N) {
                fy[j] = min(fy[j], c[i][j] - fx[i]);
            }
        }
        FOR(i,1,N) {
            start = i;
            initBFS();
            while (finish == 0) {
                findAugPath();
                if (!finish) subx_addy();
            }
            enlarge();
        }
```

```
    long long res = 0;
    FOR(i,1,N) res += c[i][mx[i]];
    return res;
}
```

MO ALGORITHM

Độ phức tạp O(Nsqrt(N) + Qsqrt(N))

```
struct Query
{ int L, R; };
bool compare(Query x, Query y) {
    if (x.L/block != y.L/block)
        return x.L/block < y.L/block;
    return x.R/block < y.R/block;
}
void queryRes() {
    block = (int)sqrt(n);
    sort(q,q+m,compare);
    int currL = 0;    int currR = 0;    int currSum = 0;
    for (int i=0; i<m; i++)
    {
        int L = q[i].L, R = q[i].R;
        while (currL < L) {
            currSum -= a[currL];currL++; }
        while (currL > L) {
            currSum += a[currL-1];currL--; }
        while (currR <= R){
            currSum += a[currR];currR++;}
        while (currR > R+1){
            currSum -= a[currR-1]; currR--; }
    }}
```