

**HaUI. Boss Long**

1. **Nguyễn Bá Thuận**
2. **Nguyễn Công Long**
3. **Nguyễn Thế Anh**

1.Template

1.1 Debug

#define bug(...) \_\_f (#\_\_VA\_ARGS\_\_, \_\_VA\_ARGS\_\_)

template <typename Arg1>void \_\_f (const char\* name, Arg1 && arg1) {cout << name << " : " << arg1 << endl; }

template <typename Arg1, typename... Args>

void \_\_f (const char\* name, Arg1 && arg1, Args && ... args)

{

const char\* comma = strchr (name + 1, ',');

cout.write (name, comma - name) << " : " << arg1 << " | "; \_\_f (comma + 1, args ...);

}

1.2. Sinh test

#include <bits/stdc++.h>

using namespace std;

// Tên chương trình

const string NAME = "TASK";

// Số test kiểm tra

const int NTEST = 100;

mt19937 rd(chrono::steady\_clock::now().time\_since\_epoch().count());

#define rand rd

// Viết lại hàm random để sử dụng cho thuận tiện. Hàm random này sinh ngẫu nhiên số trong phạm vi long long, số sinh ra >= l và <= h.

long long Rand(long long l, long long h) {

assert(l <= h);

return l + abs(rd() \* 1LL \* rd()) % (h - l + 1);

}

int main()

{

srand(time(NULL));

for (int iTest = 1; iTest <= NTEST; iTest++)

{

ofstream inp((NAME + ".inp").c\_str());

//code o day

inp.close();

// Nếu dùng Linux thì "./" + Tên chương trình

system((NAME + ".exe").c\_str());

system((NAME + "\_trau.exe").c\_str());

// Nếu dùng linux thì thay fc bằng diff

if (system(("fc " + NAME + ".out " + NAME + ".anss").c\_str()) != 0)

{

cout << "Test " << iTest << ": WRONG!\n";

return 0;

}

cout << "Test " << iTest << ": CORRECT!\n";

}

return 0;

}

#include <bits/stdc++.h>

using namespace std;

// Tên chương trình

const string NAME = "template";

// Số test kiểm tra

const int NTEST = 100;

mt19937\_64 rd(chrono::steady\_clock::now().time\_since\_epoch().count());

#define rand rd

// Viết lại hàm random để sử dụng cho thuận tiện.

// Hàm random này sinh ngẫu nhiên số trong phạm vi long long

// Số sinh ra nằm trong [L;R].

long long Rand(long long L, long long R) {

assert(L <= R);

return L + rd() % (R - L + 1);

}

int main()

{

srand(time(NULL));

for (int iTest = 1; iTest <= NTEST; iTest++)

{

ofstream inp((NAME + ".inp").c\_str());

// Code phần sinh test ở đây

inp.close();

// Nếu dùng Linux thì "./" + Tên chương trình

system(("./" + NAME).c\_str());

system(("./" + NAME + "\_trau").c\_str());

// Nếu dùng linux thì thay fc bằng diff

if (system(("diff " + NAME + ".out " + NAME + ".ans").c\_str()) != 0)

{

cout << "Test " << iTest << ": WRONG!\n";

return 0;

}

cout << "Test " << iTest << ": CORRECT!\n";

}

return 0;

}

2. Cấu trúc dữ liệu

2.1 BIT

BIT

BIT cập nhật trên đoạn, lấy giá trị điểm

Cho dãy số a[1] a[2] ... a[n]. Cần thực hiện hai thao tác sau:

update l r v: với mỗi vị trí i thuộc [l..r], ta gán a[i] += v

get i: trả về giá trị hiện tại của a[i]

int bit[MAX], n; // mảng bit độ dài n

void update(int x, int v) {

for (; x <= n; x += x & -x) bit[x] += v;

}

void updateRange(int l, int r, int v) {

update(l, v);

update(r + 1, -v);

}

int get(int x) {

int res = 0;

for (; x >= 1; x &= x - 1) res += bit[x];

return res;

}

Bài toán: Cho dãy số a[1] a[2] ... a[n], ta cần thực hiện 2 thao tác:

update l r v: với mỗi vị trí i thuộc [l..r], tăng thêm v đơn vị

getsum l r: tính tổng đoạn [l..r]

getsum(l, r) = get(r) - get(l - 1). Do đó, ta vẫn đưa truy vấn tính tổng về tính tổng đoạn [1..r]

int bitAdd[MAX], bitMul[MAX], n;

void updateBIT(int bit[], int x, int v) {

for (; x <= n; x += x & -x) bit[x] += v;

}

int getBIT(int bit[], int x) {

int res = 0;

for (; x >= 1; x &= x - 1) res += bit[x];

return res;

}

void updateRange(int l, int r, int v) {

updateBIT(bitMul, l, v);  
 updateBIT(bitMul, r + 1, -v);  
 updateBIT(bitAdd, l, -v \* (l - 1));  
 updateBIT(bitAdd, r + 1, v \* r);  
}

int getPrefixSum(int x) {

return getBIT(bitMul, x) \* x + getBIT(bitAdd, x);

}

int getSum(int l, int r) {

return getPrefixSum(r) - getPrefixSum(l - 1);

}

Thao tác trên BIT:



2.2 InOrderSet

#include<ext/pb\_ds/assoc\_container.hpp>

#include<ext/pb\_ds/tree\_policy.hpp>

using namespace \_\_gnu\_pbds;

typedef tree<int, null\_type, less\_equal<int>, rb\_tree\_tag, tree\_order\_statistics\_node\_update> A;

A a;

a.find\_by\_order(k) : trả về phần từ ở vị trí thứ k. tính từ 0

a.order\_of\_key(x): đếm số phần từ < x

Căn bậc 2 modulo

Các hàm thao tác Bit

2.3 MO

**struct** **node**{

**int** l, r, index;

}query[MAXN];

**bool** **cmp**(**const** node a, **const** node b){

**int** pa **=** a.r **/** block, pb **=** b.r **/** block;

**if** (pa **!=** pb) **return** pa **<** pb;

**return** ((pa **%** 2 **==** 0) **?** (a.l **<** b.l) **:** (a.l **>** b.l));

}

vector**<int>** v;

fu (i, 0, n) v.pb(values[i]);

sort(v.begin(), v.end());

v.resize(unique(v.begin(), v.end()) **-** v.begin());

fu (i, 0, n) {

values[i] **=** lower\_bound(v.begin(), v.end(), values[i]) **-** v.begin();

}

**int** L **=** 0, R **=** -1;

fu (i, 1, q){

**int** l **=** query[i].l-1, r **=** query[i].r;

**while** (L **>** l) add(**--**L);

**while** (R **<** r) add(**++**R);

**while** (L **<** l) del(L**++**);

**while** (R **>** r) del(R**--**);

ans[query[i].index] **=** cur;

}

3. String

3.1 Hash

typedef long long ll;

const int base = 31;

const ll MOD = 1000000003;

const ll maxn = 1000111;

using namespace std;

ll POW[maxn], hashT[maxn];

ll getHashT(int i,int j) {

return (hashT[j] - hashT[i - 1] \* POW[j - i + 1] + MOD \* MOD) % MOD;

}

int main() {

// Input

string T, P;

cin >> T >> P;

// Initialize

int lenT = T.size(), lenP = P.size();

T = " " + T;

P = " " + P;

POW[0] = 1;

// Precalculate base^i

for (int i = 1; i <= lenT; i++)

POW[i] = (POW[i - 1] \* base) % MOD;

// Calculate hash value of T[1..i]

for (int i = 1; i <= lenT; i++)

hashT[i] = (hashT[i - 1] \* base + T[i] - 'a' + 1) % MOD;

// Calculate hash value of P

ll hashP=0;

for (int i = 1; i <= lenP; i++)

hashP = (hashP \* base + P[i] - 'a' + 1) % MOD;

// Finding substrings of T equal to string P

for (int i = 1; i <= lenT - lenP + 1; i++)

if (hashP == getHashT(i, i + lenP - 1))

printf("%d ", i);

}

3.2 Trie

struct Trie{

struct Node{

Node\* child[26];

int exist, cnt;

Node() {

for (int i = 0; i < 26; i++) child[i] = NULL;

exist = cnt = 0;

}

};

int cur;

Node\* root;

Trie() : cur(0) {

root = new Node();

};

void add\_string(string s) {

Node\* p = root;

for (auto f : s) {

int c = f - 'a';

if (p->child[c] == NULL) p->child[c] = new Node();

p = p->child[c];

p->cnt++;

}

p->exist++;

}

bool delete\_string\_recursive(Node\* p, string& s, int i) {

if (i != (int)s.size()) {

int c = s[i] - 'a';

bool isChildDeleted = delete\_string\_recursive(p->child[c], s, i + 1);

if (isChildDeleted) p->child[c] = NULL;

}

else p->exist--;

if (p != root) {

p->cnt--;

if (p->cnt == 0) {

delete(p); // Khác với cài đặt bằng mảng,

// ta có thể thực sự xóa đỉnh này đi

return true;

}

}

return false;

}

void delete\_string(string s) {

if (find\_string(s) == false) return;

delete\_string\_recursive(root, s, 0);

}

bool find\_string(string s) {

Node\* p = root;

for (auto f : s) {

int c = f - 'a';

if (p->child[c] == NULL) return false;

p = p->child[c];

}

return (p->exist != 0);

}

};

4. Đồ thị

4.1 Khớp cầu

#include <bits/stdc++.h>

using namespace std;

const int maxN = 10010;

int n, m;

bool joint[maxN];

int timeDfs = 0, bridge = 0;

int low[maxN], num[maxN];

vector <int> g[maxN];

void dfs(int u, int pre) {

int child = 0; // Số lượng con trực tiếp của đỉnh u trong cây DFS

num[u] = low[u] = ++timeDfs;

for (int v : g[u]) {

if (v == pre) continue;

if (!num[v]) {

dfs(v, u);

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

if (low[v] == num[v]) bridge++;

child++;

if (u == pre) { // Nếu u là đỉnh gốc của cây DFS

if (child > 1) joint[u] = true;

}

else if (low[v] >= num[u]) joint[u] = true;

}

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

}

}

int main() {

cin >> n >> m;

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

int u, v;

cin >> u >> v;

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

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

}

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

if (!num[i]) dfs(i, i);

int cntJoint = 0;

for (int i = 1; i <= n; i++) cntJoint += joint[i];

cout << cntJoint << ' ' << bridge;

}

4.2 Đường đi Euler

// N - số đỉnh nhiều nhất

// M - số cạnh nhiều nhất

struct Edge {

int target, id;

Edge(int \_target, int \_id): target(\_target), id(\_id) {}

};

vector<Edge> adj[N]; // Danh sách kề lưu cạnh và chỉ số

bool used\_edge[M]; // Mảng đánh dấu cạnh đã thăm

list<int> euler\_walk(int u) {

// Sử dụng cấu trúc danh sách liên kết để lưu kết quả

list<int> ans;

// Xuất phát từ đỉnh u

ans.push\_back(u);

while (!adj[u].empty()) {

// Chọn một cạnh bất kì chưa thăm

int v = adj[u].back().target;

int eid = adj[u].back().id;

// Xoá cạnh vừa đi qua khỏi đồ thị

// Lưu ý việc xoá cạnh có thể \*\*ảnh hưởng\*\* tới các

// thao tác trên đồ thị về sau do việc xoá cạnh sẽ

// \*\*phá huỷ\*\* hoàn toàn danh sách cạnh

// Nên sao lưu danh sách cạnh ra biến khác nếu cần dùng lại

adj[u].pop\_back();

// Bỏ qua nếu cạnh đã thăm

if (used\_edge[eid]) continue;

// Đánh dấu cạnh đã đi qua

used\_edge[eid] = true;

// Di chuyển sang đỉnh mới

u = v;

// Thêm cạnh vào đường đi hiện tại

// Có nhiều cách lưu chu trình như lưu đỉnh, cạnh,

// chỉ số cạnh, ...

ans.push\_back(u);

}

// Tìm cạnh chưa thăm từ một đỉnh trên chu trình hiện tại

// Bắt đầu từ đỉnh thứ hai trong chu trình do ta biết

// rằng đỉnh đầu tiên trong chu trình (u) đã không còn

// cạnh ra

for (auto it = ++ans.begin(); it != ans.end(); ++it) {

// Gọi đệ quy tiếp tục tìm chu trình mới

auto t = euler\_walk(\*it);

// Nối chu trình tìm được vào chu trình hiện tại

t.pop\_back();

ans.splice(it, t);

}

return ans;

}

4.3 Bellman – Ford cùng chu trình âm

bool findNegativeCycle(int n, vector<Edge> &e, 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) {

trace[v] = u;

negStart = v; // đã tìm thấy -INF

break;

}

}

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[v]) {

negCycle.push\_back(v); // truy vết một vòng

}

reverse(negCycle.begin(), negCycle.end());

return true;

}

4.4 Euler tour trên cây

Đường đi Euler trên cây

vector<int> tour, st, en;

void dfs(int u, int par) {

tour.push\_back(u);

st[u] = tour.size();

for (int v : g[u]) {

if (v == par) continue;

dfs(v, u);

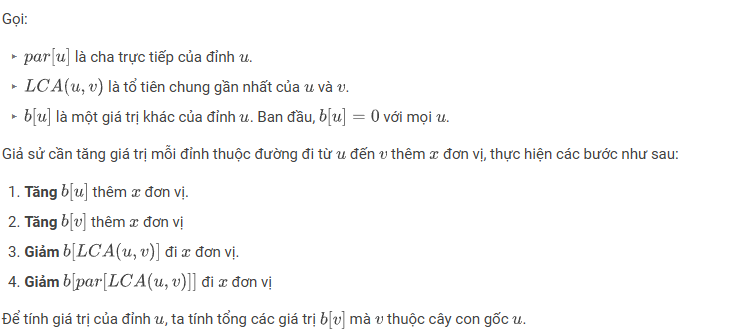
}

en[u] = tour.size();

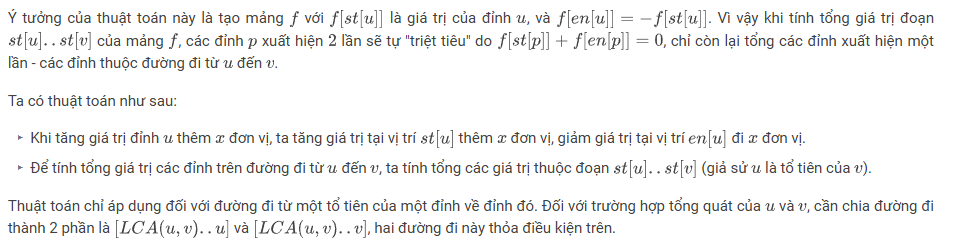
}

Đỉnh v thuộc cây con gốc u khi st[u] <= st[v] <= en[v] <= en[u]

Cập nhật đường đi truy vấn đỉnh

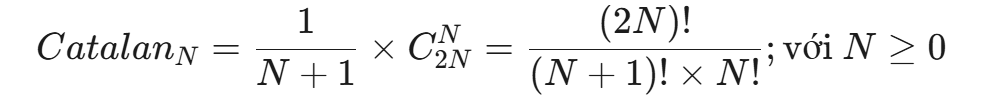


**Cập nhật đỉnh truy vấn đường đi**

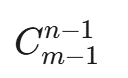


5. Toán học

5.1 Catalan



5.2 Chia kẹo

Ít nhất 1 chiếc kẹo: 

Có thể 0, 1 nhiều chiếc kẹo: 