**Tarjan - Khớp cầu – Euler - Trie - KMP - Z-function - Catalan - Suffix Array - Manacher - MaxFlow - MinCost + MaxFlow - Matrix - Convex-hull - DP Convex-hull - LCA - GMC - Gauss**

**Tarjan:**

#include <iostream>

#include <vector>

#include <stack>

#include <algorithm>

using namespace std;

const int MAX = 1e5 + 10;

vector<int> graph[MAX];

vector<int> low, num;

vector<bool> found;

stack<int> st;

int n, m;

int counter;

void dfs(int u) {

counter++;

num[u] = low[u] = counter;

st.push(u);

for (int v : graph[u]) {

if (!found[v]){

if (num[v] > 0)

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

else {

dfs(v);

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

}

}

}

if (num[u] == low[u]){

cout << "Found a SCC:";

int v;

do {

v = st.top();

st.pop();

cout << " " << v;

found[v] = true;

}

while (v != u);

cout << "\n";

}

}

void tarjan() {

counter = 0;

low.assign(n + 1, 0);

num.assign(n + 1, 0);

found.assign(n + 1, false);

counter = 0;

st = stack<int>();

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

if (!num[i]){

dfs(i);

}

}

}

int main(){

ios::sync\_with\_stdio(false);

cin >> n >> m;

for (int u, v, i = 0; i < m; i++){

cin >> u >> v;

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

}

tarjan();

return 0;

}

**Khớp cầu**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

const int MAX = 1e5 + 10;

vector<int> graph[MAX];

int low[MAX];

int num[MAX];

bool isCut[MAX];

vector<pair<int, int>> bridges;

int n, m;

int counter;

void dfs(int u, int p) {

counter++;

num[u] = low[u] = counter;

int numChild = 0;

for (int v : graph[u]) {

if (v == p)

continue;

if (num[v] > 0)

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

else {

dfs(v, u);

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

numChild++;

if (low[v] > num[u])

bridges.push\_back(make\_pair(u, v));

if (low[v] >= num[u] && p != -1)

isCut[u] = true;

}

}

if (p == -1 && numChild > 1)

isCut[u] = true;

}

int main(){

ios::sync\_with\_stdio(false);

cin >> n >> m;

for (int u, v, i = 0; i < m; i++) {

cin >> u >> v;

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

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

}

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

if (!num[i])

dfs(i, -1);

cout << "Bridges:\n";

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

pair<int, int> b = bridges[i];

cout << b.first << " " << b.second << "\n";

}

cout << "Cut vertices:";

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

if (isCut[i])

cout << " " << i;

cout << "\n";

return 0;

}

**Euler**

#include <iostream>

#include <vector>

using namespace std;

const int MAX = 110;

int deg[MAX][MAX];

int n, m;

vector<int> result;

void findEulerPath(int u){

for (int v = 0; v < n; v++) {

if (deg[u][v] > 0) {

deg[u][v]--;

deg[v][u]--;

findEulerPath(v);

}

}

result.push\_back(u);

}

int main() {

cin >> n >> m;

for (int u, v, i = 0; i < m; i++) {

cin >> u >> v;

deg[u][v]++;

deg[v][u]++;

}

findEulerPath(0);

for (int v : result) cout << v << " ";

return 0;

}

**Trie**

const int MAX = 26;

struct Node {

Node\* child[MAX];

int count;

Node() {

count = 0;

fill(child, child + MAX, nullptr);

};

};

void add(Node\* root, string s) {

Node\* cur = root;

for (char ch : s) {

int c = ch - 'a';

if (cur->child[c] == nullptr)

cur->child[c] = new Node();

cur = cur->child[c];

}

cur->count++;

}

bool search(Node\* root, string s) {

Node\* cur = root;

for (char ch : s) {

int c = ch - 'a';

if (cur->child[c] == nullptr)

return false;

cur = cur->child[c];

}

return cur->count > 0;

}

bool isEmpty(Node& temp) {

for (int i = 0; i < MAX; i++)

if (temp.child[i] != nullptr)

return false;

return true;

}

bool remove(Node\* root, string s, int depth = 0) {

if (!root)

return false;

if (depth == s.length()){

if (root->count > 0) {

root->count--;

return true;

}

return false;

}

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

if (!root->child[c])

return false;

bool isRemoved = remove(root->child[c], s, depth + 1);

Node& temp = \*(root->child[c]);

if (isRemoved && temp.count == 0 && isEmpty(temp)) {

delete root->child[c];

root->child[c] = nullptr;

}

return isRemoved;

}

int main() {

ios::sync\_with\_stdio(0);

Node\* root = new Node();

add(root, "bigo");

add(root, "complete");

add(root, "algo");

add(root, "algorithm");

cout << search(root, "competitive") << "\n";

cout << search(root, "big") << "\n";

cout << search(root, "algorithm") << "\n";

cout << remove(root, "algorithm") << "\n";

cout << remove(root, "comp") << "\n";

cout << remove(root, "competitive") << "\n";

return 0;

}

**KMP**

vector<int> kmpPreprocess(string p)

{

int m = p.length();

vector<int> pref(m, 0);

int j = 0;

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

{

while (j > 0 && p[j] != p[i])

j = pref[j - 1];

if (p[i] == p[j])

j++;

pref[i] = j;

}

return pref;

}

vector<int> kmpSearch(string t, string p, vector<int>& pref)

{

int n = t.length(), m = p.length();

vector<int> found;

int j = 0;

for (int i = 0; i < n; i++)

{

while (j > 0 && t[i] != p[j])

j = pref[j - 1];

if (t[i] == p[j])

j++;

if (j == m)

{

found.push\_back(i - m + 1);

j = pref[j - 1];

}

}

return found;

}

int main()

{

ios::sync\_with\_stdio(0);

string t, p;

cin >> t >> p;

vector<int> pref = kmpPreprocess(p);

vector<int> founds = kmpSearch(t, p, pref);

cout << "Found at: ";

for (int f : founds)

{

cout << f << " ";

}

return 0;

}

**Z-Function**

vector<int> z\_function(string s)

{

int n = (int)s.length();

vector<int> z(n);

for (int i = 1, l = 0, r = 0; i < n; ++i)

{

if (i <= r)

z[i] = min(r - i + 1, z[i - l]);

while (i + z[i] < n && s[z[i]] == s[i + z[i]])

++z[i];

if (i + z[i] - 1 > r)

l = i, r = i + z[i] - 1;

}

return z;

}

**Catalan**

ll fastPow(ll b, int p)

{

ll res = 1;

while (p)

{

if (p & 1)

res = res \* b % MOD;

b = b \* b % MOD;

p >>= 1;

}

return res;

}

vector<ll> computeCatalans(int n)

{

vector<ll> c(n + 1, 1);

for (int i = 0; i < n; i++)

c[i+1] = ((c[i] \* (4\*i+2) % MOD) \* fastPow(i+2, MOD-2)) % MOD;

return c;

}

**Suffix Array**

const int MAX\_DIGIT = 256;

void countingSort(vector<int>& SA, vector<int>& RA, int k = 0)

{

int n = SA.size();

vector<int> cnt(max(MAX\_DIGIT, n), 0);

for (int i = 0; i < n; i++)

if (i + k < n)

cnt[RA[i + k]]++;

else

cnt[0]++;

for (int i = 1; i < cnt.size(); i++)

cnt[i] += cnt[i - 1];

vector<int> tempSA(n);

for (int i = n - 1; i >= 0; i--)

if (SA[i] + k < n)

tempSA[--cnt[RA[SA[i] + k]]] = SA[i];

else

tempSA[--cnt[0]] = SA[i];

SA = tempSA;

}

vector<int> constructSA(string s)

{

int n = s.length();

vector<int> SA(n);

vector<int> RA(n);

vector<int> tempRA(n);

for (int i = 0; i < n; i++)

{

RA[i] = s[i];

SA[i] = i;

}

for (int step = 1; step < n; step <<= 1)

{

countingSort(SA, RA, step);

countingSort(SA, RA, 0);

tempRA[SA[0]] = 0;

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

{

if (RA[SA[i]] == RA[SA[i - 1]]

&& RA[SA[i] + step] == RA[SA[i - 1] + step])

tempRA[SA[i]] = tempRA[SA[i - 1]];

else

tempRA[SA[i]] = tempRA[SA[i - 1]] + 1;

}

RA = tempRA;

if (RA[SA[n - 1]] == n - 1)

break;

}

return SA;

}

vector<int> computeLCP(const string& s, const vector<int>& SA)

{

int n = SA.size();

vector<int> LCP(n), PLCP(n), c(n, 0);

for (int i = 0; i < n; i++)

c[SA[i]] = i;

int k = 0;

for (int j, i = 0; i < n-1; i++)

{

j = SA[c[i] - 1];

k = max(k - 1, 0);

while (s[i + k] == s[j + k])

k++;

PLCP[i] = k;

}

for (int i = 0; i < n; i++)

LCP[i] = PLCP[SA[i]];

return LCP;

}

int main(){

ios::sync\_with\_stdio(false);

string s;

cin >> s;

s += "$";

vector<int> SA = constructSA(s);

vector<int> LCP = computeLCP(s, SA);

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

cout << SA[i] << " " << s.substr(SA[i]) << "\n";

return 0;

}

**Manacher**

vector<int> manacher(string s)

{

int n = s.length();

vector<int> p(n);

int l = 0, r = -1;

for (int i = 0; i < n; i++)

{

int k = i > r ? 0 : min(p[l + r - i], r - i);

while (0 <= i-k && i+k < n && s[i-k] == s[i+k])

k++;

p[i] = --k;

if (i + k > r)

{

l = i - k;

r = i + k;

}

}

return p;

}

int main()

{

ios::sync\_with\_stdio(false);

string s;

cin >> s;

string t(2 \* s.length() + 1, '#');

for (int i = 0; i < s.length(); i++)

t[2 \* i + 1] = s[i];

vector<int> p = manacher(t);

for (int i = 1; i < p.size() - 1; i++)

{

if (p[i] > 0)

cout << p[i] << " " << s.substr((i-p[i])/2, p[i]) << "\n";

}

return 0;

}

**Maximum Flow**

#include <bits/stdc++.h>

using namespace std;

const int MAX = 100;

const int INF = 1e9;

struct Edge

{

int u, v, cap, flow;

Edge(int u = 0, int v = 0, int cap = 0)

: u(u), v(v), cap(cap), flow(0) {}

};

int level[MAX];

int iter[MAX];

vector<Edge> edges;

vector<int> graph[MAX];

void addEdge(int u, int v, int cap)

{

graph[u].push\_back(edges.size());

edges.push\_back({ u, v, cap });

graph[v].push\_back(edges.size());

edges.push\_back({ v, u, 0 });

}

// build Layered Network

bool bfs(int s, int t)

{

queue<int> q;

q.push(s);

memset(level, -1, sizeof(level));

level[s] = 0;

while (!q.empty())

{

int u = q.front();

q.pop();

for (int i : graph[u])

{

if (level[edges[i].v] != -1)

continue;

if (edges[i].cap <= edges[i].flow)

continue;

level[edges[i].v] = level[u] + 1;

q.push(edges[i].v);

}

}

return level[t] != -1;

}

// find Augmenting Path

int dfs(int s, int t, int pushed)

{

if (pushed == 0)

return 0;

if (s == t)

return pushed;

for (int &i = iter[s]; i < graph[s].size(); i++)

{

int id = graph[s][i];

int v = edges[id].v;

if (level[v] != level[s] + 1)

continue;

if (edges[id].cap <= edges[id].flow)

continue;

int f = min(pushed, edges[id].cap - edges[id].flow);

f = dfs(v, t, f);

if (f == 0)

continue;

edges[id].flow += f;

edges[id ^ 1].flow -= f;

return f;

}

return 0;

}

int dinic(int s, int t)

{

int newFlow, sumFlow = 0;

while (bfs(s, t))

{

memset(iter, 0, sizeof(iter));

while (newFlow = dfs(s, t, INF))

{

sumFlow += newFlow;

}

}

return sumFlow;

}

int main()

{

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int n, m, s, t;

cin >> n >> m >> s >> t;

for (int u, v, w, i = 0; i < m; i++)

{

cin >> u >> v >> w;

addEdge(u, v, w);

}

cout << dinic(s, t);

return 0;

}

**Mincost - MaxFlow**

#include <bits/stdc++.h>

using namespace std;

using ll = long long;

const int MAX = 1010;

const ll INF = 1e18;

struct Edge

{

int u, v;

ll cap, cost;

Edge(int u = 0, int v = 0, ll cap = 0, ll cost = 0)

: u(u), v(v), cap(cap), cost(cost) {}

};

vector<Edge> edges;

ll dist[MAX];

int pre[MAX];

vector<int> graph[MAX];

int n, m;

void addEdge(int u, int v, ll cap, ll cost)

{

graph[u].push\_back(edges.size());

edges.push\_back({ u, v, cap, cost });

graph[v].push\_back(edges.size());

edges.push\_back({ v, u, 0, -cost });

}

bool spfa(int s, int t)

{

fill(dist, dist + n + 1, INF);

vector<bool> inQueue(n + 1, false);

queue<int> q;

dist[s] = 0;

inQueue[s] = true;

q.push(s);

while (!q.empty())

{

int u = q.front();

q.pop();

inQueue[u] = false;

for (int id : graph[u])

{

int v = edges[id].v;

ll cap = edges[id].cap, cost = edges[id].cost;

if (cap <= 0 || dist[v] <= dist[u] + cost)

continue;

dist[v] = dist[u] + cost;

pre[v] = id;

if (!inQueue[v])

{

q.push(v);

inQueue[v] = true;

}

}

}

return dist[t] != INF;

}

pair<ll, ll> mcmf(int s, int t)

{

ll minCost = 0, maxFlow = 0;

while (spfa(s, t))

{

ll flow = INF;

int id, cur = t;

while (cur != s)

{

id = pre[cur];

flow = min(flow, edges[id].cap);

cur = edges[id].u;

}

maxFlow += flow;

minCost += flow \* dist[t];

cur = t;

while (cur != s)

{

id = pre[cur];

edges[id].cap -= flow;

edges[id ^ 1].cap += flow;

cur = edges[id].u;

}

}

return { minCost, maxFlow };

}

int main()

{

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int s, t;

cin >> n >> m >> s >> t;

for (int u, v, cap, cost, i = 0; i < m; i++)

{

cin >> u >> v >> cap >> cost;

addEdge(u, v, cap, cost);

}

auto res = mcmf(s, t);

cout << res.second << " " << res.first << "\n";

return 0;

}

**Matrix**

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

int MOD = 1e9 + 7;

struct Matrix

{

vector<vector<ll>> a;

int n, m;

Matrix(int n = 0, int m = 0) : n(n), m(m)

{

a.resize(n);

for (int i = 0; i < n; i++)

{

a[i].resize(m);

for (int j = 0; j < m; j++)

a[i][j] = 0;

}

}

};

Matrix operator\*(const Matrix& a, const Matrix& b)

{

Matrix c(a.n, b.m);

for (int i = 0; i < a.n; i++)

for (int j = 0; j < b.m; j++)

for (int k = 0; k < a.m; k++)

c.a[i][j] = (a.a[i][k] \* b.a[k][j] % MOD

+ c.a[i][j]) % MOD;

return c;

}

ostream& operator<<(ostream& os, const Matrix& mt)

{

for (int i = 0; i < mt.n; i++)

{

for (int j = 0; j < mt.m; j++)

os << mt.a[i][j] << " ";

os << "\n";

}

return os;

}

Matrix Identity(int n)

{

Matrix mat(n, n);

for (int i = 0; i < n; i++)

mat.a[i][i] = 1;

return mat;

}

Matrix pow(const Matrix& a, ll k)

{

if (k == 0)

return Identity(a.n);

if (k == 1)

return a;

Matrix temp = pow(a, k / 2);

temp = temp \* temp;

if (k & 1)

temp = temp \* a;

return temp;

}

int main()

{

Matrix mat(2, 2), base(2, 1), temp;

mat.a[0][0] = 0;

mat.a[0][1] = mat.a[1][0] = mat.a[1][1] = 1;

base.a[0][0] = 0;

base.a[1][0] = 1;

int t, n;

cin >> t;

while (t--)

{

cin >> n;

temp = pow(mat, n);

temp = temp \* base;

cout << temp.a[0][0] << "\n";

}

return 0;

}

**Convex Hull**

#include <algorithm>

#include <cstdio>

#include <vector>

using namespace std;

const int N = 20000;

struct Point

{

long long x, y;

bool operator<(const Point &v) const

{

return x == v.x ? y < v.y : x < v.x;

}

long long cross(const Point &p, const Point &q) const

{

return (p.x - x) \* (q.y - y) - (p.y - y) \* (q.x - x);

}

} p[N], poly[N];

int n;

void enter()

{

scanf("%d", &n);

for (int i = 0; i < n; ++i)

scanf("%lld%lld", &p[i].x, &p[i].y);

}

long long size(Point poly[], int k)

{

long long S = (poly[k - 1].x - poly[0].x) \* (poly[k - 1].y + poly[0].y);

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

S += (poly[i - 1].x - poly[i].x) \* (poly[i - 1].y + poly[i].y);

return S;

printf("%lld\n", S);

}

void solve()

{

sort(p, p + n);

int k = 0;

for (int i = 0; i < n; ++i)

{

while (k >= 2 && poly[k - 2].cross(poly[k - 1], p[i]) <= 0)

--k;

poly[k++] = p[i];

}

for (int i = n - 2, t = k + 1; i >= 0; --i)

{

while (k >= t && poly[k - 2].cross(poly[k - 1], p[i]) <= 0)

--k;

poly[k++] = p[i];

}

printf("%lld\n", size(poly, k));

}

int main()

{

enter();

solve();

return 0;

}

**DP Convex**

#include <bits/stdc++.h>

using namespace std;

struct Line

{

double a, b;

Line(double a = 0, double b = 0) : a(a), b(b) {}

bool operator < (const Line& l) const

{

if (a != l.a)

return a < l.a;

return b > l.b;

}

double intersectX(const Line& l) const

{

return (l.b - b) / (a - l.a);

}

double eval(double x) const

{

return a \* x + b;

}

};

vector<Line> buildConvex(const vector<Line>& lines)

{

vector<Line> res;

int n = 0;

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

{

while (res.size() >= 2 &&

res[n - 2].intersectX(res[n - 1]) >=

res[n - 1].intersectX(lines[i])

)

{

res.pop\_back();

n--;

}

res.push\_back(lines[i]);

n++;

}

return res;

}

double query(const vector<Line>& lines, double x)

{

int l = 0, r = lines.size() - 1;

double res = lines[r].eval(x);

while (l < r)

{

int m = (l + r) / 2;

if (lines[m].intersectX(lines[m + 1]) >= x)

{

res = lines[m].eval(x);

r = m;

}

else

{

l = m + 1;

}

}

return res;

}

int main()

{

int n, q;

cin >> n >> q;

vector<Line> lines(n);

for (int i = 0; i < n; i++)

{

cin >> lines[i].a >> lines[i].b;

}

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

lines = buildConvex(lines);

while (q--)

{

double x;

cin >> x;

cout << query(lines, x) << "\n";

}

return 0;

}

**LCA**

#include <iostream>

#include <algorithm>

#include <vector>

using namespace std;

const int MAX = 1e5 + 10;

const int MAX\_LOG = 18;

int n, l;

vector<int> graph[MAX];

int counter;

int tin[MAX], tout[MAX];

int parent[MAX][MAX\_LOG];

void dfs(int u, int p)

{

tin[u] = ++counter;

parent[u][0] = p;

for (int v : graph[u])

if (v != p)

dfs(v, u);

tout[u] = ++counter;

}

void preprocess()

{

counter = 0;

l = ceil(log2(n));

dfs(1, 1);

for (int k = 1; k <= l; k++)

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

parent[i][k] = parent[parent[i][k - 1]][k - 1];

}

int isAncestor(int u, int v)

{

return tin[u] <= tin[v] && tin[v] <= tout[u];

}

int lca(int u, int v)

{

if (isAncestor(u, v))

return u;

if (isAncestor(v, u))

return v;

for (int k = l; k >= 0; k--)

if (!isAncestor(parent[u][k], v))

u = parent[u][k];

return parent[u][0];

}

int main()

{

ios::sync\_with\_stdio(0);

int q, u, v;

cin >> n;

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

{

cin >> u >> v;

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

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

}

preprocess();

cin >> q;

while (q--)

{

cin >> u >> v;

cout << lca(u, v) << "\n";

}

return 0;

}

**GMC**

class GMC {

public:

long gmu[N], gmv[N], d[N];

bool bfs() {

queue<long> q;

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

if (!gmu[i]) {d[i] = 0; q.push(i);}

else d[i] = 1e9;

}

d[0] = 1e9;

while (q.size()) {

long u =q.front();

q.pop();

foe(it,ke[u])

for (int v : ke[u]) if (d[gmv[v]]==1e9) {

d[gmv[v]] = d[u] + 1;

q.push(gmv[v]);

}

}

return d[0] != 1e9;

}

bool dfs(long u) {

if (!u) return true;

for (int v: ke[u]) if (d[gmv[v]] == d[u] + 1 && dfs(gmv[v])) {

gmv[v] = u;

gmu[u] = v;

return true;

}

d[u] = 1e9;

return false;

}

long find() {

long sol = 0;

for (int i = 1; i <= n; i++) gmu[i] = d[i] = 0;

for (int i = 1; i <= m; i++) gmv[i] = 0;

while (bfs()) {

for (int i = 1; i <= n; i++) if (!gmu[i] && dfs(i)) sol++;

}

return sol;

}

};

**Gauss**

ll pow(ll base, ll p, ll MOD)

{

if(p == 0)

return 1;

if(p % 2 == 0)

{

ll d = pow(base, p / 2, MOD);

return (d \* d) % MOD;

}

return (pow(base, p - 1, MOD) \* base) % MOD;

}

ll inv(ll x, ll MOD)

{

return pow(x, MOD - 2, MOD);

}

ll gauss(vector<vector<ll> > &a, ll MOD)

{

int n = a.size(), m = a[0].size() - 1;

for(int i = 0; i < n; i++)

for(int j = 0; j <= m; j++)

a[i][j] = (a[i][j] % MOD + MOD) % MOD;

vector<int> where(m, -1);

for(int col = 0, row = 0; col < m && row < n; col++)

{

int sel = row;

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

if(a[i][col] > a[sel][col])

sel = i;

if(a[sel][col] == 0)

{

where[col] = -1;

continue;

}

for(int i = col; i <= m; i++)

swap(a[sel][i], a[row][i]);

where[col] = row;

ll c\_inv = inv(a[row][col], MOD);

for(int i = 0; i < n; i++)

if(i != row)

{

if(a[i][col] == 0)

continue;

ll c = (a[i][col] \* c\_inv) % MOD;

for(int j = 0; j <= m; j++)

a[i][j] = (a[i][j] - c \* a[row][j] % MOD + MOD) % MOD;

}

row++;

}

vector<ll> ans(m, 0);

ll result = 1;

for(int i = 0; i < m; i++)

if(where[i] != -1)

ans[i] = (a[where[i]][m] \* inv(a[where[i]][i], MOD)) % MOD;

else

result = (result \* MOD) % mod;

for(int i = 0; i < n; i++)

{

ll sum = a[i][m] % MOD;

for(int j = 0; j < m; j++)

sum = (sum + MOD - (ans[j] \* a[i][j]) % MOD) % MOD;

if(sum != 0)

return 0;

}

return result;

}