**LazyP**

template <typename T>

struct LazySegmentTree {

vector<T> v, t, L;

size\_t size = 0;

LazySegmentTree(const vector<T>& vv) {

v = vv;

size = vv.size();

t.assign(size << 2, 0);

L.assign(size << 2, -1); // Lazy propagation array initialized with -1

if (!vv.empty())

build(1, 0, size - 1);

}

void push(int node, int st, int en) {

if (L[node] != -1) {

t[node] = (en - st + 1) \* L[node]; // Lazy update

if (st != en) {

L[node << 1] = L[node << 1 | 1] = L[node]; // Propagate to children

}

L[node] = -1; // Reset lazy value

}

}

T merge(T a, T b) {

return (a + b); // Change as needed

}

void build(int N, int s, int e) {

if (s == e) {

t[N] = v[s];

return;

}

int mid = (s + e) >> 1;

build(N << 1, s, mid);

build(N << 1 | 1, mid + 1, e);

t[N] = merge(t[N << 1], t[N << 1 | 1]);

}

void update(int l, int r, T val) {

update(1, 0, size - 1, l, r, val);

}

void update(int N, int s, int e, int l, int r, T val) {

push(N, s, e);

if (s > r || e < l)

return;

if (s >= l && e <= r) {

L[N] = val; // Lazy update

push(N, s, e);

return;

}

int mid = (s + e) >> 1;

update(N << 1, s, mid, l, r, val);

update(N << 1 | 1, mid + 1, e, l, r, val);

t[N] = merge(t[N << 1], t[N << 1 | 1]);

}

T query(int l, int r) {

return query(1, 0, size - 1, l, r);

}

T query(int N, int s, int e, int l, int r) {

push(N, s, e);

if (s > r || e < l)

return 0; // Change as needed

if (s >= l && e <= r)

return t[N];

int mid = (s + e) >> 1;

T q1 = query(N << 1, s, mid, l, r);

T q2 = query(N << 1 | 1, mid + 1, e, l, r);

return merge(q1, q2);

}

};

int64\_t any\_to\_anybase( int64\_t num , int64\_t baseA , int64\_t baseB){

int64\_t ans = 0;

int64\_t dec = stoll(to\_string(num), nullptr , baseA);

int64\_t rem = 0;

int64\_t i = 1;

while (dec){

rem = dec % baseB;

dec /= baseB;

ans += rem \* i;

i\*=10;

}

return ans;

}

**COMBI**

struct Combi {

int n; vector<mint> facts, finvs, invs;

Combi(int \_n): n(\_n), facts(\_n), finvs(\_n), invs(\_n) {

facts[0] = finvs[0] = 1;

invs[1] = 1;

for (int i = 2; i < n; i++) invs[i] = invs[mod % i] \* (-mod / i);

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

facts[i] = facts[i - 1] \* i;

finvs[i] = finvs[i - 1] \* invs[i];

}

}

inline mint fact(int n) { return facts[n]; }

inline mint finv(int n) { return finvs[n]; }

inline mint inv(int n) { return invs[n]; }

inline mint ncr(int n, int k) { return n < k ? 0 : facts[n] \* finvs[k] \* finvs[n - k]; }

inline mint npr(int n, int k) { return n < k ? 0 : facts[n] \* facts[n - k]; }

};

int64\_t ith\_num\_in\_a\_seq(int64\_t i){

int64\_t ans = 0;

int64\_t a = 1;

int64\_t digit[] = {0, 1, 2, 3, 5, 6, 7, 8, 9};

while (i){

//ith % total digit

ans += digit[i % 9] \* a;

i /= 9;

//mult by 10 everytime

a \*= 10;

//i divide number of digit

}

return ans;

}

**LENGTH ENCODING**

template<typename T, typename T\_iterable>

vector<pair<T, int>> run\_length\_encoding(const T\_iterable &items) {

vector<pair<T, int>> runs;

T previous;

int count = 0;

for (const T &item : items)

if (item == previous) {

count++;

} else {

if (count > 0)

runs.emplace\_back(previous, count);

previous = item;

count = 1;

}

if (count > 0)

runs.emplace\_back(previous, count);

return runs;

}

**MONO STACK**

std::vector<int> nextGreaterElements(const std::vector<int>& nums) {

int n = nums.size();

std::vector<int> result(n, -1);

std::stack<int> s;

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

while (!s.empty() && nums[s.top()] < nums[i]) {

result[s.top()] = nums[i];

s.pop();

}

s.push(i);

}

return result;

}

**PBDS**

#undef \_GLIBCXX\_HAVE\_ICONV

#include <bits/extc++.h>

template<class T>

using o\_set = \_\_gnu\_pbds::tree<T, \_\_gnu\_pbds::null\_type, less\_equal<T>, \_\_gnu\_pbds::rb\_tree\_tag, \_\_gnu\_pbds::tree\_order\_statistics\_node\_update>;

**SPT**

template <typename T>

class SPT {

private:

std::vector<std::vector<T>> table;

std::vector<int> log2;

int n;

std::function<T(T, T)> operation;

T identity;

public:

SPT(const std::vector<T>& arr, std::function<T(T, T)> op, T id) {

n = arr.size();

operation = op;

identity = id;

int logn = static\_cast<int>(std::log2(n)) + 1;

log2.resize(n + 1);

table.assign(logn, std::vector<T>(n, identity));

log2[1] = 0;

for (int i = 2; i <= n; ++i) {

log2[i] = log2[i / 2] + 1;

}

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

table[0][i] = arr[i];

}

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

for (int i = 0; i + (1 << k) - 1 < n; ++i) {

table[k][i] = operation(table[k - 1][i], table[k - 1][i + (1 << (k - 1))]);

}

}

}

T query(int l, int r) {

int len = r - l + 1;

int k = log2[len];

return operation(table[k][l], table[k][r - (1 << k) + 1]);

}

};

**STRING HASH**

constexpr int base1 = 121, base2 = 263;

constexpr int mod1 = 1E9 + 7, mod2 = 1E9 + 9;

constexpr int N = 1E6 + 5; // modify

array<int, 2> pw[N], invpw[N];

int p(int a, int b, int M) {

int ans = 1 % M;

while (b) {

if (b & 1) {

ans = 1LL \* ans \* a % M;

}

a = 1LL \* a \* a % M;

b >>= 1;

}

return ans;

}

int inv(int n, int m) {

return p(n, m - 2, m);

}

void pre() {

pw[0][0] = 1;

pw[0][1] = 1;

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

pw[i][0] = (1LL \* pw[i - 1][0] \* base1) % mod1;

pw[i][1] = (1LL \* pw[i - 1][1] \* base2) % mod2;

}

invpw[0][0] = 1;

invpw[0][1] = 1;

int iv1 = inv(base1, mod1);

int iv2 = inv(base2, mod2);

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

invpw[i][0] = (1LL \* invpw[i - 1][0] \* iv1) % mod1;

invpw[i][1] = (1LL \* invpw[i - 1][1] \* iv2) % mod2;

}

}

struct Hashing {

string s;

array<int, 2> pref[N];

array<int, 2> get\_hash(string &s) {

int n = s.size();

array<int, 2> hash{0, 0};

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

hash[0] = (hash[0] + 1LL \* s[i] \* pw[i][0]) % mod1;

hash[1] = (hash[1] + 1LL \* s[i] \* pw[i][1]) % mod2;

}

return hash;

}

void build(string &t) {

s = t;

int n = s.size();

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

pref[i][0] = (1LL \* s[i] \* pw[i][0]) % mod1;

if (i) pref[i][0] = (pref[i][0] + pref[i - 1][0]) % mod1;

pref[i][1] = (1LL \* s[i] \* pw[i][1]) % mod2;

if (i) pref[i][1] = (pref[i][1] + pref[i - 1][1]) % mod2;

}

}

array<int, 2> get\_sub\_hash(int l, int r) {

array<int, 2> hs{0, 0};

hs[0] = pref[r][0];

if (l) hs[0] = (hs[0] - pref[l - 1][0] + mod1) % mod1;

hs[0] = (1LL \* hs[0] \* invpw[l][0]) % mod1;

hs[1] = pref[r][1];

if (l) hs[1] = (hs[1] - pref[l - 1][1] + mod2) % mod2;

hs[1] = (1LL \* hs[1] \* invpw[l][1]) % mod2;

return hs;

}

};

STRING HASH UPDATE

constexpr int base1 = 121, base2 = 263;

constexpr int mod1 = 1E9 + 7, mod2 = 1E9 + 9;

constexpr int N = 2E5 + 5; // modify

int p(int a, int b, int M) {

int ans = 1 % M;

while (b) {

if (b & 1) {

ans = 1LL \* ans \* a % M;

}

a = 1LL \* a \* a % M;

b >>= 1;

}

return ans;

}

int inv(int n, int m) {

return p(n, m - 2, m);

}

array<int,2>pw[N],invpw[N];

void pre() {

pw[0][0] = 1;

pw[0][1] = 1;

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

pw[i][0] = 1LL \* pw[i - 1][0] \* base1 % mod1;

pw[i][1] = 1LL \* pw[i - 1][1] \* base2 % mod2;

}

invpw[0][0] = 1;

invpw[0][1] = 1;

int iv1 = inv(base1, mod1);

int iv2 = inv(base2, mod2);

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

invpw[i][0] = 1LL \* invpw[i - 1][0] \* iv1 % mod1;

invpw[i][0] %= mod1;

invpw[i][1] = 1LL \* invpw[i - 1][1] \* iv2 % mod2;

invpw[i][1] %= mod2;

}

}

using Node = array<int, 2>;

struct hashing {

string s;

int n;

vector<Node> t;

hashing() {}

hashing(string \_s) {

s = \_s;

n = s.size();

t.resize(n << 2);

build(1, 0, n - 1);

}

Node combine(Node a, Node b) {

Node x = Node{(a[0] + b[0]) % mod1, (a[1] + b[1]) % mod2};

return x;

}

Node mul(Node a, int x) {

Node P = {int(1LL \* a[0] \* x % mod1), int(1LL \* a[1] \* x % mod2)};

return P;

}

Node mul(Node a,Node b){

a[0] = 1LL \* a[0] \* b[0] % mod1;

a[1] = 1LL \* a[1] \* b[1] % mod2;

return a;

}

void build(int node, int st, int en) {

if (st == en) {

t[node] = mul(pw[st], s[st]);

return;

}

int mid = (st + en) >> 1;

build(node << 1, st, mid);

build(node << 1 | 1, mid + 1, en);

t[node] = combine(t[node << 1], t[node << 1 | 1]);

}

void update(int node, int st, int en, int i, char v) {

if (i < st || i > en)

return;

if (st == en) {

t[node] = mul(pw[i], v);

s[i] = v;

return;

}

int mid = (st + en) >> 1;

update(node << 1, st, mid, i, v);

update(node << 1 | 1, mid + 1, en, i, v);

t[node] = combine(t[node << 1], t[node << 1 | 1]);

}

Node query(int node, int st, int en, int l, int r) {

if (l > en || r < st)

return Node{0, 0};

if (st >= l && en <= r)

return t[node];

int mid = (st + en) >> 1;

Node p = query(node << 1, st, mid, l, r);

Node q = query(node << 1 | 1, mid + 1, en, l, r);

return combine(p, q);

}

void upd(int pos, char ch) {

update(1, 0, n - 1, pos, ch);

}

Node qry(int l, int r) {

return query(1, 0, n - 1, l, r);

}

Node get\_hash(int l,int r){

return mul(qry(l,r),invpw[l]);

}

};

**Kahn’s algo**

indeg[b]++;

if(indeg[i]==0)q.push

While(!q)

order.push(cur)

for(int next: g[cur]){

order[next]--;

if(order[next] ==0 ){

q.push(next);

}

}

if(order.size() == n) possible

**SCC**

constexpr int N = 1E5 + 5;

vector<int>v[N] , v\_rev[N];

vector<bool>used(N);

vector<int>comp;

vector<int>order;

void dfs1(int ver) {

used[ver] = true;

for (int child : v[ver]) {

if (!used[child])

dfs1(child);

}

order.emplace\_back(ver); // order

}

void dfs2(int ver) {

used[ver] = true;

comp.emplace\_back(ver); // pushing comp

for (int child : v\_rev[ver]) {

if (!used[child])

dfs2(child);

}

}

void tcase() {

int n; cin >> n;

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

int x, y; cin >> x >> y;

v[x].emplace\_back(y);

v\_rev[y].emplace\_back(x);

}

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

int x, y; cin >> x >> y;

v[x].emplace\_back(y);

v\_rev[y].emplace\_back(x);

}

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

if (!used[i])

dfs1(i);

}

used.assign(N, false);

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

for (int x : order) {

if (!used[x])

dfs2(x);

for (int c : comp)

cout << c << ' ';

cout << '\n';

comp.clear();

}

}

int legendre(i64 n, i64 p) { // only if p is prime -> max x such that p^x|n!

i64 ans = 0;

while (n) {

ans += n / p;

n /= p;

}

return ans;

}

auto dijkstra = [&](int source)->i64{

min\_heap<array<i64,2>>q;

vector<bool>vis(N,false);

vector<i64>dis(N,INF);

q.push({0,source});

dis[source] = 0;

while(!q.empty()){

auto [cur\_w,cur\_v] = q.top();

q.pop(); if(vis[cur\_v])continue;

vis[cur\_v] = true;

for(auto [c\_v,c\_w]:v[cur\_v]){

if(dis[cur\_v] + c\_w < dis[c\_v]){

dis[c\_v] = dis[cur\_v] + c\_w;

q.push({dis[c\_v],c\_v});

}

}

}

return dis[n];

};

**Kth Ancistor**

const int N = 1E5 + 5;

const int LOG = \_\_bit\_width(N);

vector<int>v[N];

vector<int>depth(N);

vector<vector<int>>up(N, vector<int>(LOG));

void dfs(int a) {

for (int b : v[a]) {

depth[b] = depth[a] + 1;

up[b][0] = a;

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

up[b][i] = up[up[b][i - 1]][i - 1];

}

dfs(b);

}

}

int get\_lca(int a, int b) {

if (depth[a] < depth[b])

swap(a, b);

int dif = depth[a] - depth[b];

for (int k = LOG - 1; k >= 0; --k) {

if (dif & (1 << k))

a = up[a][k];

}

if (a == b)

return a;

for (int k = LOG - 1; k >= 0; --k) {

if (up[a][k] != up[b][k]) {

a = up[a][k];

b = up[b][k];

}

}

return up[a][0];

}

int get\_kth\_ancestor(int node, int k) {

for (int i = 0; i < LOG; ++i) {

if (k & (1 << i)) {

node = up[node][i];

if (node == 0)

break;

}

}

return node == 0 ? -1 : node;

}