PS Team Note

pysunn

Setting

< template >

```
// optimize
#pragma GCC optimize("O3")
#pragma GCC optimize("Ofast")
#pragma GCC optimize("unroll-loops")

// main
#include <bits/stdc++.h>
#define fio() ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
typedef long long ll;
#define mid (lo + hi) / 2
#define all(v) v.begin(), v.end()
#define pii pair<int, int>

using namespace std;

// util
#define test(a) int _; cin >> _; FOR(a, _, 0)
#define FOR(a, b, c) for (int a = c; a < b; a++)
#define FORE(a, b) for (const auto &a : b)

// 임의 정밀도
__float128 float_number;
```

Data Structures

< segment tree >

```
const int sz = 101010;
ll A[sz];
ll tree[4 * sz];

ll init(int lo, int hi, int node)
{
    if (lo == hi) return tree[node] = A[lo];
    return tree[node] = init(lo, mid, node * 2) + init(mid + 1, hi, node * 2 + 1);
}

ll query(int lo, int hi, int node, int left, int right)
{
    if (left > hi || right < lo) return 0;
    if (left <= lo && hi <= right) return tree[node];
    return
query(lo, mid, node * 2, left, right) + query(mid + 1, hi, node * 2 + 1, left, right);
}
```

```
void update(int lo, int hi, int node, int idx, ll value)
{
    if (idx < lo || idx > hi) return;
    if (lo == hi) tree[node] = value;
    else
    {
        update(lo, mid, node * 2, idx, value);
        update(mid + 1, hi, node * 2 + 1, idx, value);
        tree[node] = tree[node*2] + tree[node*2+1];
    }
}

ll count_inversion(int n)
{
    FOR(i, n + 1, 1) cin >> A[i];
    vector<pii> v;
    FOR(i, n + 1, 1)
    v.push_back({A[i], i});

    sort(all(v));
    FOR(i, n + 1, 1)
    v[i].first = i;

    reverse(all(v));

    ll cnt = 0;
    FORE(e, v)
    {
        int value = e.second;
        cnt += query(1, n, 1, 1, value - 1);
        update(1, n, 1, value, 1);
    }
    return cnt;
}
```

< PBDS (GNU) >

```
//PBDS
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
                                    // less_equal : multi set
#define ordered_set tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update>

ordered_set OS;
void m_erase(ordered_set &OS, int val)
{
    int index = OS.order_of_key(val);
    auto it = OS.find_by_order(index);
    if (*it == val) OS.erase(it);
}
```

Math

< fast binomial coefficient >

```
const ll mod = 1e9 + 7;
const int sz = 400000;

ll fact[sz];
ll inv_fact[sz];

// fast mul
ll fp(ll a, ll b)
{
    if(b == 1) return a % mod;
    if(b % 2 == 0)
    {
        ll res = fp(a, b/2) % mod;
        return ( res % mod * res % mod ) % mod;
    }
    if(b % 2 == 1) return ((a % mod) * (fp(a, b - 1) % mod)) % mod;
}

//nCr
ll binomial(int n, int r)
{
    return (((fact[n] % mod ) * (inv_fact[r] % mod))) % mod * (inv_fact[n-r] % mod) % mod;
}

void fastFactorial(int n)
{
    fact[0] = fact[1] = 1;
    for(ll i = 2 ; i <= n ; i++)
    {
        fact[i] = fact[i-1] * i;
        fact[i] %= mod;
    }

    inv_fact[0] = 1;
    inv_fact[n] = fp(fact[n], mod-2) % mod;

    for(ll i = n ; i >= 1 ; i--)
    {
        inv_fact[i-1] = (inv_fact[i] * i) % mod;
        inv_fact[i-1] %= mod;
    }
}
```

< sieve >

```
// find prime numbers in 1 ~ n
// ret[x] = false  ->  x is prime
// O(n*loglogn)
void sieve(int n, bool prime[])
{
    // prime : init to all true
    prime[1] = false;
    for (int i = 2; i * i <= n; ++i)
    if (prime[i]) for (int j = i * i; j <= n; j += i) prime[j] = false;
}

// calculate number of divisors for 1 ~ n
// when you need to calculate sum, change += 1 to += i
// O(n*logn)
void num_of_divisors(int n, int ret[])
{
    for (int i = 1; i <= n; ++i)
        for (int j = i; j <= n; j += i)
            ret[j] += 1;
}

// calculate euler totient function for 1 ~ n
// phi(n) = number of x s.t. 0 < x < n && gcd(n, x) = 1
// O(n*loglogn)
void euler_phi(int n, int ret[])
{
    for (int i = 1; i <= n; ++i) ret[i] = i;
    for (int i = 2; i <= n; ++i)
        if (ret[i] == i)
            for (int j = i; j <= n; j += i)
                ret[j] -= ret[j] / i;
}
```

Graph

< SCC : Kosaraju >

```
const int SIZE = 10000;

vector<int> graph[2 * SIZE + 1];
vector<int> rev_graph[2 * SIZE + 1];
vector<int> order;

bool vis[2 * SIZE + 1];
int SCC[2 * SIZE + 1];
int SCC_IDX;

void dfs(int n)
{
    vis[n] = true;
    FORE(next, graph[n]) if(!vis[next]) dfs(next);
    order.push_back(n);
```

```
}

void rev_dfs(int n)
{
    vis[n] = true;
    FORE(next, rev_graph[n]) if(!vis[next]) rev_dfs(next);
    SCC[n] = SCC_IDX;
}

bool kosaraju(int N, int M)
{
    FOR(i,M+1,1) if(!vis[i]) dfs(i);
    FOR(i,SIZE + M + 1, SIZE + 1) if(!vis[i]) dfs(i);

    fill(vis, vis + 2*SIZE+1, 0);
    reverse(all(order));

    //reverse dfs
    FORE(i, order)
    {
        if(!vis[i])
        {
            // SCC그룹 설정
            rev_dfs(i);
            SCC_IDX++;
        }
    }
}
```

< Topology Sort >

```
int inDegree[MAX_SIZE];
vector<int> E[MAX_SIZE];
int N, M;

vector<int> result;
void topologySort()
{
    queue<int> q;
    FOR(i,N+1,1) if(!inDegree[i]) q.push(i);
    FOR(i,N+1,1)
    {
        if (q.empty()) return;
        int now = q.front(); q.pop();
        result.push_back(now);
        FORE(next, E[now]) if (--inDegree[next] == 0) q.push(next);
    }
}
```

Shortest Path

< dijkstra : priority queue>

```
vector<pii> v[20202];
int INF = 999999;
int dis[20202];

int V,E;
void dijkstra(int start)
{
    priority_queue<pii> pq; //cost, next
    pq.push({0, start}); // 자기 자신으로 가는 비용은 0
    dis[start] = 0;

    //pq가 빌때까지 반복
    while (!pq.empty())
    {
        // now : 거리가 가장 짧은 정점
        int dist = -pq.top().first; //now로 가는 비용
        int now = pq.top().second; //정점
        pq.pop();

        //이미 최단거리 존재 시 skip
        if (dist > dis[now]) continue;

        // now를 시작점으로 하는 연결된 다른 모든 간선들에 대해서 확인
        for (int i = 0; i < v[now].size() ; i++)
        {
            int nextcost = v[now][i].first;
            int next = v[now][i].second;

            if (dis[next] > dis[now] + nextcost)
            {
                dis[next]  = dis[now] + nextcost;
                //우선순위 큐 삽입
                pq.push({-dis[next], next});
            }
        }
    }

    FOR(i,V+1,1)
    {
        if (dis[i] == INF) cout <<"INF\n";
        else cout << dis[i]<<'\n';
    }
}
```

< Bellman >

```
ll INF = 1e9;

bool bellman(int START, int N, const vector<pair<pii,ll>>& v, ll dist[])
{
    for (int i = 1; i <= N-1; i++)
    {
        for (int j = 0; j < v.size(); j++)
        {
            int from = v[j].first.first;
            int to = v[j].first.second;
            int cost = v[j].second;

            if (dist[to] > dist[from] + cost) dist[to] = dist[from] + cost;
        }
    }

    // more change : minus edge detecting!
    for (int j = 0; j < v.size(); j++)
    {
        int from = v[j].first.first;
        int to = v[j].first.second;
        int cost = v[j].second;

        if (dist[to] > dist[from] + cost) return true;
    }
    return false;
}
```

< floyd >

```
int cost[101][101] = {0};
const int INF = 9999999;

void floyd(int n, int m)
{
    FOR(i,n+1,0)
    {
        FOR(j,n+1,0)
        {
            cost[i][j] = INF;
            //자신에서 자신으로 가는 거리는 0으로 초기화
            if(i == j) cost[i][j] = 0;
        }
    }

    FOR(i,m,0)
    {
        int a,b,c; cin>>a>>b>>c;
        if(cost[a][b] > c) cost[a][b] = c;
    }

    //floyd -> 모든 정점에 대해서 생각
    FOR(k,n+1,1) FOR(i,n+1,1) FOR(j,n+1,1)
      cost[i][j] = min(cost[i][j], cost[i][k] + cost[k][j]);
```

```
    FOR(i,n+1,1)
    {
        FOR(j,n+1,1)
        {
            // 가는 경로가 존재하지 않음 (갈 수 없음)
            if(cost[i][j] >= INF) cout << 0 << " ";
            // 가는 경로가 존재함
            else cout << cost[i][j] << " ";
        }
        cout<<'\n';
    }
}
```

String

< KMP >

```
vector<int> Fail(string pattern)
{
    int m = pattern.length();
    vector<int> pi(m); // partial match table

    int begin = 1;   // 📌 접두사 역할을 하는 검색어에서 일치하기 시작한 "시작" 포인터
    int matched = 0; // 📌 접미사 역할을 하는 검색어의 일치한 개수이자 포인터
    pi[0] = 0;
    while (begin + matched < m)
    {
        if (pattern[begin + matched] == pattern[matched])
        {
            matched++;
            pi[begin + matched - 1] = matched;
        }

        else
        {
            if (matched == 0)
                begin++;
            else
            {
                begin += matched - pi[matched - 1];
                matched = pi[matched - 1];
            }
        }
    }
    return pi;
}
```

```
vector<int> KMP(string pattern, string text)
{
    int m = pattern.length();
    int n = text.length();
    vector<int> pos;
    vector<int> pi = Fail(pattern);

    int begin = 0;
    int matched = 0;
    while (begin + m <= n)
    {
        if (matched < m && text[begin + matched] == pattern[matched])
        {
            matched++;
            if (matched == m)
                pos.push_back(begin);
        }
        else
        {
            if (matched == 0)
                begin++;
            else
            {
                begin += matched - pi[matched - 1];
                matched = pi[matched - 1];
            }
        }
    }
    return pos;
}
```

Geometry

< CCW >

```
#define x first
#define y second
#define pll pair<ll,ll>

ll ccw(pll a, pll b, pll c)
{
    return (a.x * b.y + b.x * c.y + c.x * a.y) - (b.x * a.y + c.x * b.y + a.x * c.y);
}

double dist(pll a, pll b)
{
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}

bool ccw_cmp(pll a, pll b, pll o)
{
    ll c = ccw(o, a, b);
    if (c != 0) return c > 0;
    return dist(o, a) < dist(o, b);
}
```