

Team notebook

Nicolas Alba

July 26, 2023

Contents

1	1. Template	1
1.1	CLIONmain	1
1.2	CMakeLists	2
1.3	debug	2
1.4	vscode-template	2
2	dp	3
2.1	Traveling Sales Man	3
2.2	coin _{change}	4
2.3	edit _{distance}	4
2.4	elevator _{problem}	4
2.5	lcs	5
2.6	lcs3	5
2.7	lis	6
2.8	max _{sum} _{3d}	6
2.9	max _{sum} _{array}	7
2.10	max _{sum} _{array2d}	7
3	flows	8
3.1	Hungarian	8
3.2	MaxFlow	8
3.3	max _{flow}	9
3.4	min _{cost} _{flow}	10
4	geometry	11
4.1	area	11
4.2	convex-hull	11
4.3	heron formula	12
4.4	segment-intersection	12

4.5	sin cos law	13
5	graph	14
5.1	1 - DFS	14
5.2	2 - BFS	14
5.3	3 - Dijkstra	14
5.4	4 - BellmanFord	14
5.5	5 - Floyd Warshall	15
5.6	6 - Euler Path and Cycle	15
5.7	7 - Topological Sort	15
5.8	8 - Transitive Closure	15
5.9	9 - Kruskal	16
5.10	A - Union Find	16
5.11	B - SCC	16
5.12	C-Cycle _{Detection}	17
5.13	Z-Extra-OrStatements2Sat	18
6	math	18
6.1	Chinese Remainder	18
6.2	Combinatorics	19
6.3	Count _{primes}	19
6.4	Erdős-Szekeres _{theorem}	20
6.5	Extended Euclides	20
6.6	FFT	20
6.7	Floor _{sum}	21
6.8	Greatest Common Divisor	22
6.9	Lowest Common Multiple	22
6.10	MatrixExponentiation	22
6.11	Modular Aritmethics	23
6.12	Modular Combinatorics	24
6.13	Ternary Search	24

6.14	catalan	24
6.15	factorization	24
6.16	fermat	26
6.17	primes	26
7	query	27
7.1	1 - Segment Tree	27
7.2	MergeSortTree	28
7.3	Min Segment Tree	29
7.4	Mo's	29
7.5	SegTree Max Sum sub arrays	30
7.6	fenwicktree	31
7.7	min _{stable}	31
7.8	struct lazy tree	31
7.9	struct segment tree	32
7.10	sum _{stable}	33
8	string	33
8.1	1 - KMP	33
8.2	Hashing	33
9	tree	34
9.1	1 K-th Parent	34
9.2	NearestSelectedNodesProblem	34
9.3	TwoPiecesOnTree	35
9.4	lca	35
9.5	moetree	37
9.6	simple-lca	39
10	util	39
10.1	PI	39
10.2	custom _{hash}	39
10.3	custom _{hash} _{pair}	40
10.4	exponential _{notation}	40
10.5	io-int128	40
10.6	macros	41
10.7	multi _{set}	41
10.8	oset	42
10.9	pragmas	42
10.10	priority _{queue}	43
10.11	random	43
10.12	util builtin functions	43

1. Template

1.1 CLIONmain

```
// Practice Every Day :)
#include <bits/stdc++.h>
using namespace std;

#define pb push_back
#define F first
#define S second
#define all(x) (x).begin(), (x).end()
#define sortt(x) sort(all(x))
#define sortn(x, n) sort((x), (x) + (n))
#define sq(a) ((a) * (a))
#define MP make_pair

#define each(x, xs) for (auto &x : (xs))
#define rep(i, be, en) for (__typeof(en) i = (be) - ((be) > (en)); i !=
    (en) - ((be) > (en)); i += 1 - 2 * ((be) > (en)))
// old loops
#define FOR(i, a, b) for (int (i) = (a); (i) < (b); (i)++)
#define ROF(i, a, b) for (int (i) = (a); (i) >= (b); (i)--)
#define REP(i, a, b) for (int (i) = (a); (i) <= (b); (i)++)
#define EACH(a, x) for (auto &(a) : (x))

using ll = long long;
using ld = long double;
using pi = pair<int, int>;
using pl = pair<ll, ll>;
using ti = tuple<long long, long long, long long>;
using vi = vector<int>;
using vb = vector<bool>;
using vl = vector<ll>;
using vs = vector<string>;
using vvl = vector<vl>;
using vpl = vector<pl>;
template<class T> using pql = priority_queue<T, vector<T>, greater<T>>;
template<class T> using pqg = priority_queue<T>;

// >>>>>>>>> debugging >>>>>>>>>
#ifdef DEBUG_NICO
#include "debug.h"
#define LINE cout << "-----" << endl;
```

```

#else
#define deb(x...)
#define LINE
#endif
// <<<<<<<<< debugging <<<<<<<<<

void cfgIO() {
#ifdef NICOLAS
    freopen("../input.txt", "r", stdin);
    freopen("../output.txt", "w", stdout);
    // freopen("../error.txt", "w", stderr);
#endif
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
}
// END DEBUG

void solve();
void init();

int testId = 0;
int main() {
    cfgIO();
    init();
    // int t; cin >> t; while (t--)
    // cout << "Case #" << ++testId << ": ",
    solve(), ++testId;
}
const int N = 1e5 + 10;

void init(){}

void solve() {}

```

1.2 CMakeLists

```

cmake_minimum_required(VERSION 3.22)
project(competitive)

set(CMAKE_CXX_STANDARD 11) # This could different

set(A main.cpp C.cpp) # Add file names here

```

```

foreach(X IN LISTS A)
    add_executable("${X}" "${X}")
    target_compile_definitions("${X}" PRIVATE NICOLAS=1) # add ENV_VAR
    target_compile_definitions("${X}" PRIVATE DEBUG_NICO=1)
endforeach()

```

1.3 debug

```

#include <bits/stdc++.h>
using namespace std;

#ifdef DEBUG_H
#define DEBUG_H

void __print(int x) {cerr << x;}
void __print(long x) {cerr << x;}
void __print(long long x) {cerr << x;}
void __print(unsigned x) {cerr << x;}
void __print(unsigned long x) {cerr << x;}
void __print(unsigned long long x) {cerr << x;}
void __print(float x) {cerr << x;}
void __print(double x) {cerr << x;}
void __print(long double x) {cerr << x;}
void __print(char x) {cerr << '\'' << x << '\'';}
void __print(const char *x) {cerr << "\"" << x << "\"";}
void __print(const string &x) {cerr << "\"" << x << "\"";}
void __print(bool x) {cerr << (x ? "true" : "false");}

template<typename T>
void __print(priority_queue<T> xs)
{cerr << "[ "; while (xs.size()) {__print(xs.top()); xs.pop(); cerr << '
'; }cerr << ']';}

template<typename T, typename V>
void __print(const pair<T, V> &x)
{__print(x.first); cerr << ':'; __print(x.second);}

template<typename T> // for data structures (vector, set, map, etc)
void __print(const T &xs)
{cerr << "[ "; for (auto &x : xs) {__print(x);cerr << ' ';}cerr << ']';}

void _print()
{cerr << "]" << endl;}

```

```

template <typename T, typename... V>
void _print(T t, V... v)
{ _print(t); if (sizeof...(v)) cerr << ", "; _print(v...);}

#define deb(x...) cerr << "[" << #x << "]" = [", _print(x)

#endif /* DEBUG_H */

```

1.4 vscode-template

```

#include <bits/stdc++.h>
using namespace std;

#define pb push_back
#define F first
#define S second
#define all(x) (x).begin(), (x).end()
#define sortt(x) sort(all(x))
#define sq(a) ((a) * (a))

#define each(x, xs) for (auto &x : (xs))
#define rep(i, be, en) for (__typeof(en) i = (be) - ((be) > (en)); i !=
    (en) - ((be) > (en)); i += 1 - 2 * ((be) > (en)))
#define FOR(i, a, b) for (ll (i) = (a); (i) < (b); (i)++)

using ll = long long;
using ld = long double;
using pi = pair<int, int>;
using pl = pair<ll, ll>;
using ti = tuple<long long, long long, long long>;
using vi = vector<int>;
using vb = vector<bool>;
using vl = vector<ll>;
using vs = vector<string>;
using vvl = vector<vl>;
using vpl = vector<pl>;
template<class T> using pql = priority_queue<T, vector<T>, greater<T>>;
template<class T> using pqg = priority_queue<T>;

// >>>>>>>>> debugging >>>>>>>>>
// Change this part to local file or put the debug here
#ifdef ONLINE_JUDGE

```

```

#include "/home/fundacion/templates/debug.h"
#define LINE cerr << "-----" << endl;
#else
#define deb(x...)
#define LINE
#endif
// <<<<<<<<<< debugging <<<<<<<<<<

void test_case();

const ll INF = INT64_MAX;
const int inf = INT32_MAX;
const ld PI = acos(-1);
const int MOD = 1e9 + 7;
const int DX[4]{1,0,-1,0}, DY[4]{0,1,0,-1};

int testId;
int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    int T;
    T = 1;
    cin >> T; // if one test case, comment this line
    rep (t, 0, T) { testId++; test_case(); }

    return 0;
}

void test_case() {

```

2 dp

2.1 Traveling Sales Man

```

// Given directed weighted graph, gets the minimum hamilton cycle.
// Use dfs(0, 1), if 1e9 then it is impossible, otherwise get the min.
const int MAX_SIZE = 15;
const ll IMPOSSIBLE = 1e9;

```

```

ll INITIAL = 0; // initial node
vpl adj[MAX_SIZE];
vvl dp(MAX_SIZE, vl(1 << MAX_SIZE, -1));
ll n, m;
ll target; // init as (1 << n) - 1, full visited

ll dfs(ll x, ll mask) {
    if (dp[x][mask] != -1) {
        return dp[x][mask];
    }
    if (mask == target) {
        each(yy, adj[x]) {
            if (yy.F == INITIAL) {
                return yy.S;
            }
        }
        return dp[x][mask] = IMPOSSIBLE;
    }
    ll ans = IMPOSSIBLE;
    each(yy, adj[x]) {
        ll y, d;
        tie(y, d) = yy;
        if ((mask >> y) & 1) continue;
        ll actual = dfs(y, mask | (1 << y)) + d;
        ans = min(ans, actual);
    }
    return dp[x][mask] = ans;
}

```

2.2 coin_{change}

```

// infinite number of coins
// Get the minimum number of coins that sum a value.
void test_case() {
    ll n, x;
    cin >> n >> x;
    vl dp(x + 1, inf - 1);
    vl coin(n);
    rep(i, 0, n) cin >> coin[i];
    dp[0] = 0;
    rep(i, 0, x) {
        each(c, coin) {
            if (c + i > x) continue;

```

```

                dp[i + c] = min(dp[i + c], dp[i] + 1);
            }
        }
        if (dp[x] + 1 == inf) {
            cout << "-1\n";
        } else {
            cout << dp[x] << "\n";
        }
    }
}

```

2.3 edit_{distance}

```

// editDistance(a, b, a.size(), b.size());

// Cuantas operaciones, (insert, remove, remplazar) necesito
// para que string a y b sean iguales.
int editDistance(string a, string b, int m, int n)
{
    if (m == 0) return n;
    if (n == 0) return m;
    if (a[m - 1] == b[n - 1])
        return editDistance(a, b, m - 1, n - 1);

    return 1 + min({editDistance(a, b, m, n - 1), // Insert
                    editDistance(a, b, m - 1, n), // Remove
                    editDistance(a, b, m - 1, n - 1) // Replace
    });
}

// My own
ll editDistance(string &s, string &t) {
    ll n = s.size();
    ll m = t.size();
    vvl dp(n+1, vl(m+1, 0));
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= m; j++) {
            if (min(i, j) == 0) dp[i][j] = max(i, j);
            else if (s[i-1] == t[j-1]) dp[i][j] = dp[i-1][j-1];
            else dp[i][j] = min(dp[i-1][j], min(dp[i][j-1], dp[i-1][j-1]))
                + 1;
        }
    }
    return dp[n][m];
}

```

```
}

```

2.4 elevator_{problem}

```
// Given n <= 20 persons, print the minimum number of travels
// to move everyone in a elevator with capacity k.

ll n, k;
vl nums;
vector<pair<ll,ll>> dp;

// minimum travels, last travel with minimum weight.
// use f((1 << n) - 1).F
pair<ll,ll> f(ll mask) {
    if (dp[mask] != make_pair(-1ll, -1ll)) {
        return dp[mask];
    }
    if (mask == 0) {
        return dp[mask] = {0, k};
    }
    dp[mask] = {n + 1, 0}; // one person in a travel, or use popcount.
    for (int i = 0; i < n; i++) {
        // person i is the last to enter to elevator.
        if ((mask >> i) & 1) {
            auto actual = f(mask ^ (1 << i)); // best option without this
            // last person.
            if (actual.S + nums[i] <= k) {
                actual.S += nums[i];
                // what happened if there are a better minimum.
                // well in that case the last person should be other one.
                // so we are trying all options that last person will be
                // better.
            } else {
                actual.S = nums[i];
                actual.F++;
            }
            dp[mask] = min(dp[mask], actual);
        }
    }
    return dp[mask];
}
```

```
// Iterative
void test_case() {
    ll n, k;
    cin >> n >> k;
    vl nums(n);
    vector<pair<ll,ll>> dp(1 << n, {n+1, 0});
    for (int i = 0; i < n; i++) cin >> nums[i];
    dp[0] = {0, k};
    for (int i = 1; i < (1 << n); i++) {
        for (int j = 0; j < n; j++) {
            if (i & (1 << j)) {
                auto actual = dp[i ^ (1 << j)];
                if (actual.S + nums[j] <= k) {
                    actual.S += nums[j];
                } else {
                    actual.F++;
                    actual.S = nums[j];
                }
                dp[i] = min(dp[i], actual);
            }
        }
    }
    cout << dp[(1 << n) - 1].F << "\n";
}
```

2.5 lcs

```
const int M_MAX = 20;
const int N_MAX = 20;
int m, n;
string X;
string Y;
int memo[M_MAX + 1][N_MAX + 1];

// Encuentra el Longest Common Subsequence de string X e Y. m y n son sus
// tamaos
// lcs de abfgh aeeeeiiiiigh = agh
int lcs (int m, int n) {
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            if (i == 0 || j == 0) memo[i][j] = 0;
            else if (X[i - 1] == Y[j - 1]) memo[i][j] = memo[i - 1][j - 1] + 1;
        }
    }
}
```

```

    else memo[i][j] = max(memo[i - 1][j], memo[i][j - 1]);
}
}
return memo[m][n];
}

```

2.6 lcs3

```

string X = "AGGT12";
string Y = "12TXAYB";
string Z = "12XBA";
bool calc[100][100][100];
int dp[100][100][100];
//lcsOf3(X.size() - 1, Y.size() - 1, Z.size() - 1);
int lcsOf3(int i, int j, int k) {
    if(i== -1 || j== -1 || k== -1) // outbounds
        return 0;
    if(calc[i][j][k]) //memo
        return dp[i][j][k];
    calc[i][j][k] = true;
    if(X[i]==Y[j] && Y[j]==Z[k]) // same
        return dp[i][j][k] = 1+lcsOf3(i-1,j-1,k-1);
    else // best of reduce any
        return dp[i][j][k] = max(max(lcsOf3(i-1,j,k),
                                     lcsOf3(i,j-1,k)),lcsOf3(i,j,k-1));
}

```

2.7 lis

```

// TODO: O(n^2)

// nlog(n)
// 1 2 3 5 10 2 -1 100 500
// 1 2 3 5 10 100 500
int lis(vi& v) {
    if (v.size() == 0) // boundary case
        return 0;

    vi tail(v.size(), 0);
    int length = 1; // always points empty slot in tail

```

```

tail[0] = v[0];

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

    // Do binary search for the element in
    // the range from begin to begin + length
    auto start = tail.begin(), end = tail.begin() + length;
    auto it = lower_bound(start, end, v[i]);

    // If not present change the tail element to v[i]
    if (it == tail.begin() + length)
        tail[length++] = v[i];
    else
        *it = v[i];
}

return length;
}

// My own LIS
int lis(vl &nums) {
    vl best;
    int n = nums.size();
    for (int i = 0; i < n; i++) {
        // For non-decreasing
        // int idx = upper_bound(all(best), nums[i]) - best.begin();

        // For increasing
        int idx = lower_bound(all(best), nums[i]) - best.begin();
        if (idx == best.size()) {
            best.pb(nums[i]);
        } else {
            best[idx] = min(best[idx], nums[i]);
        }
    }

    return best.size();
}

```

// Also LIS with Segment Tree

2.8 $\max_s um_3 d$

```

long long a=20, b=20, c=20;
long long acum[a][b][c];
long long INF = -100000000007;

long long max_range_3D(){
    for(int x=0; x<a; x++){
        for(int y = 0; y<b; y++){
            for(int z = 0; z<c; z++){
                if(x>0) acum[x][y][z] += acum[x-1][y][z];
                if(y>0) acum[x][y][z] += acum[x][y-1][z];
                if(z>0) acum[x][y][z] += acum[x][y][z-1];
                if(x>0 && y>0) acum[x][y][z] -=
                    acum[x-1][y-1][z];
                if(x>0 && z>0) acum[x][y][z] -=
                    acum[x-1][y][z-1];
                if(y>0 && z>0) acum[x][y][z] -=
                    acum[x][y-1][z-1];
                if(x>0 && y>0 && z>0) acum[x][y][z] +=
                    acum[x-1][y-1][z-1];
            }
        }
    }
    long long max_value = INF;
    for(int x=0; x<a; x++){
        for(int y = 0; y<b; y++){
            for(int z = 0; z<c; z++){
                for(int h = x; h<a; h++){
                    for(int k = y; k<b; k++){
                        for(int l = z; l<c; l++){
                            long long aux =
                                acum[h][k][l];
                            if(x>0) aux -=
                                acum[x-1][k][l];
                            if(y>0) aux -=
                                acum[h][y-1][l];
                            if(z>0) aux -=
                                acum[h][k][z-1];
                            if(x>0 && y>0) aux +=
                                acum[x-1][y-1][l];
                            if(x>0 && z>0) aux +=
                                acum[x-1][k][z-1];
                            if(z>0 && y>0) aux +=
                                acum[h][y-1][z-1];
                            if(x>0 && y>0 && z>0)
                                aux -=

```

```

                                acum[x-1][y-1][z-1];
                                max_value =
                                    max(max_value,
                                        aux);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return max_value;
}

```

2.9 \max_{sum_array}

```

int maxRangeSum(vector<int> a){
    int sum = 0, ans = 0;
    for (int i = 0; i < a.size(); i++){
        if (sum + a[i] >= 0) {
            sum += a[i];
            ans = max(ans, sum);
        } else sum = 0;
    }
    return ans;
}

```

2.10 $\max_{sum_array2d}$

```

int INF = -1000000007; // minimo valor
int n, m; //filas y columnas
const int MAX_N = 105, MAX_M = 105;
int values[MAX_N][MAX_M];

int max_range_sum2D(){
    for(int i=0; i<n;i++){
        for(int j=0; j<m; j++){
            if(i>0) values[i][j] += values[i-1][j];
            if(j>0) values[i][j] += values[i][j-1];
            if(i>0 && j>0) values[i][j] -= values[i-1][j-1];
        }
    }
}

```



```

int max_mat = INF;
for(int i=0; i<n;i++){
    for(int j=0; j<m; j++){
        for(int h = i; h<n; h++){
            for(int k = j; k<m; k++){
                int sub_mat = values[h][k];
                if(i>0) sub_mat -= values[i-1][k];
                if(j>0) sub_mat -= values[h][j-1];
                if(i>0 && j>0) sub_mat +=
                    values[i-1][j-1];
                max_mat = max(sub_mat, max_mat);
            }
        }
    }
}
return max_mat;
}

```

3 flows

3.1 Hungarian

Halla el mximo match en un grafo bipartito con pesos (min cost) $O(V^3)$

```

typedef ll T;
const T inf = 1e18;

struct hung {
    int n, m;
    vector<T> u, v; vector<int> p, way;
    vector<vector<T>> g;

    hung(int n, int m):
        n(n), m(m), g(n+1, vector<T>(m+1, inf-1)),
        u(n+1), v(m+1), p(m+1), way(m+1) {}

    void set(int u, int v, T w) { g[u+1][v+1] = w; }

    T assign() {
        for (int i = 1; i <= n; ++i) {
            int j0 = 0; p[0] = i;
            vector<T> minv(m+1, inf);

```

```

        vector<char> used(m+1, false);
        do {
            used[j0] = true;
            int i0 = p[j0], j1; T delta = inf;
            for (int j = 1; j <= m; ++j) if (!used[j]) {
                T cur = g[i0][j] - u[i0] - v[j];
                if (cur < minv[j]) minv[j] = cur, way[j] = j0;
                if (minv[j] < delta) delta = minv[j], j1 = j;
            }
            for (int j = 0; j <= m; ++j)
                if (used[j]) u[p[j]] += delta, v[j] -= delta;
                else minv[j] -= delta;
            j0 = j1;
        } while (p[j0]);
        do {
            int j1 = way[j0]; p[j0] = p[j1]; j0 = j1;
        } while (j0);
    }
    return -v[0];
}
};

```

3.2 MaxFlow

// N <= 5000, M <= 30000, C <= 1e9, 300ms

```

const int INF = INT32_MAX;

struct flowEdge{
    ll to, rev, f, cap;
};

struct max_flow {

    vector<vector<flowEdge>> G;

    max_flow(int n) : G(n) {
        nodes = n;
    }

    // Aade arista (st -> en) con su capacidad
    void addEdge(int st, int en, int cap) {

```

```

    flowEdge A = {en, (int)G[en].size(), 0, cap};
    flowEdge B = {st, (int)G[st].size(), 0, 0};
    G[st].pb(A);
    G[en].pb(B);
}

ll nodes, S, T; // asignar estos valores al armar el grafo G
                // nodes = nodos en red de flujo. Hacer G.clear();
                G.resize(nodes);
vl work, lvl;

bool bfs() {
    int qt = 0;
    queue<ll> q;
    q.push(S);
    lvl.assign(nodes, -1);
    lvl[S] = 0;
    while (q.size()) {
        int v = q.front(); q.pop();
        for (flowEdge &e : G[v]) {
            int u = e.to;
            if (e.cap <= e.f || lvl[u] != -1) continue;
            lvl[u] = lvl[v] + 1;
            q.push(u);
        }
    }
    return lvl[T] != -1;
}

ll dfs(ll v, ll f) {
    if (v == T || f == 0) return f;
    for (ll &i = work[v]; i < G[v].size(); i++) {
        flowEdge &e = G[v][i];
        ll u = e.to;
        if (e.cap <= e.f || lvl[u] != lvl[v] + 1) continue;
        ll df = dfs(u, min(f, e.cap - e.f));
        if (df) {
            e.f += df;
            G[u][e.rev].f -= df;
            return df;
        }
    }
    return 0;
}

```

```

ll maxFlow(ll s, ll t) {
    S = s;
    T = t;
    ll flow = 0;
    while (bfs()) {
        work.assign(nodes, 0);
        while (true) {
            ll df = dfs(S, INF);
            if (df == 0) break;
            flow += df;
        }
    }
    return flow;
}
};

```

3.3 \max_{flow}

```

struct Dinitz{
    const int INF = 1e9 + 7;
    Dinitz(){}
    Dinitz(int n, int s, int t) {init(n, s, t);}

    void init(int n, int s, int t)
    {
        S = s, T = t;
        nodes = n;
        G.clear(), G.resize(n);
        Q.resize(n);
    }
    struct flowEdge
    {
        int to, rev, f, cap;
    };

    vector<vector<flowEdge>> G;

    // Aade arista (st -> en) con su capacidad
    void addEdge(int st, int en, int cap) {
        flowEdge A = {en, (int)G[en].size(), 0, cap};
        flowEdge B = {st, (int)G[st].size(), 0, 0};
        G[st].pb(A);

```

```

    G[en].pb(B);
}

int nodes, S, T; // asignar estos valores al armar el grafo G
                // nodes = nodos en red de flujo. Hacer G.clear();
                G.resize(nodes);

vi work, lvl;
vi Q;

bool bfs() {
    int qt = 0;
    Q[qt++] = S;
    lvl.assign(nodes, -1);
    lvl[S] = 0;
    for (int qh = 0; qh < qt; qh++) {
        int v = Q[qh];
        for (flowEdge &e : G[v]) {
            int u = e.to;
            if (e.cap <= e.f || lvl[u] != -1) continue;
            lvl[u] = lvl[v] + 1;
            Q[qt++] = u;
        }
    }
    return lvl[T] != -1;
}

int dfs(int v, int f) {
    if (v == T || f == 0) return f;
    for (int &i = work[v]; i < G[v].size(); i++) {
        flowEdge &e = G[v][i];
        int u = e.to;
        if (e.cap <= e.f || lvl[u] != lvl[v] + 1) continue;
        int df = dfs(u, min(f, e.cap - e.f));
        if (df) {
            e.f += df;
            G[u][e.rev].f -= df;
            return df;
        }
    }
    return 0;
}

int maxFlow() {
    int flow = 0;
    while (bfs()) {

```

```

        work.assign(nodes, 0);
        while (true) {
            int df = dfs(S, INF);
            if (df == 0) break;
            flow += df;
        }
    }
    return flow;
}
};

```

3.4 $\min_{cost} flow$

```

//  $O(\min(E^2 V^2, EVFLOW))$ 
struct CheapDinitz{
    const int INF = 1e9 + 7;

    CheapDinitz() {}
    CheapDinitz(int n, int s, int t) {init(n, s, t);}

    int nodes, S, T;
    vi dist;
    vi pot, curFlow, prevNode, prevEdge, Q, inQue;

    struct flowEdge{
        int to, rev, flow, cap, cost;
    };
    vector<vector<flowEdge>> G;
    void init(int n, int s, int t)
    {
        nodes = n, S = s, T = t;
        curFlow.assign(n, 0), prevNode.assign(n, 0), prevEdge.assign(n, 0);
        Q.assign(n, 0), inQue.assign(n, 0);
        G.clear();
        G.resize(n);
    }

    void addEdge(int s, int t, int cap, int cost)
    {
        flowEdge a = {t, (int)G[t].size(), 0, cap, cost};
        flowEdge b = {s, (int)G[s].size(), 0, 0, -cost};
        G[s].pb(a);
        G[t].pb(b);
    }
}

```

```

}

void bellmanFord()
{
    pot.assign(nodes, INF);
    pot[S] = 0;
    int qt = 0;
    Q[qt++] = S;
    for (int qh = 0; (qh - qt) % nodes != 0; qh++)
    {
        int u = Q[qh % nodes];
        inQue[u] = 0;
        for (int i = 0; i < (int)G[u].size(); i++)
        {
            flowEdge &e = G[u][i];
            if (e.cap <= e.flow) continue;
            int v = e.to;
            int newDist = pot[u] + e.cost;
            if (pot[v] > newDist)
            {
                pot[v] = newDist;
                if (!inQue[v])
                {
                    Q[qt++ % nodes] = v;
                    inQue[v] = 1;
                }
            }
        }
    }
}

ii MinCostFlow()
{
    bellmanFord();
    int flow = 0;
    int flowCost = 0;
    while (true) // always a good start for an algorithm :v
    {
        set<ii> s;
        s.insert({0, S});
        dist.assign(nodes, INF);
        dist[S] = 0;
        curFlow[S] = INF;
        while (s.size() > 0)
        {

```

```

            int u = s.begin() -> s;
            int actDist = s.begin() -> f;
            s.erase(s.begin());
            if (actDist > dist[u]) continue;
            for (int i = 0; i < (int)G[u].size(); i++)
            {
                flowEdge &e = G[u][i];
                int v = e.to;
                if (e.cap <= e.flow) continue;
                int newDist = actDist + e.cost + pot[u] - pot[v];
                if (newDist < dist[v])
                {
                    dist[v] = newDist;
                    s.insert({newDist, v});
                    prevNode[v] = u;
                    prevEdge[v] = i;
                    curFlow[v] = min(curFlow[u], e.cap - e.flow);
                }
            }
        }
        if (dist[T] == INF)
            break;
        for (int i = 0; i < nodes; i++)
            pot[i] += dist[i];
        int df = curFlow[T];
        flow += df;
        for (int v = T; v != S; v = prevNode[v])
        {
            flowEdge &e = G[prevNode[v]][prevEdge[v]];
            e.flow += df;
            G[v][e.rev].flow -= df;
            flowCost += df * e.cost;
        }
    }
    return {flow, flowCost};
}
};

```

4 geometry

4.1 area

```
// Glass Area
// p is the height of water
// r2 the small radio of base
// r3 the big radio of water ceil
((p * PI)*(sq(r1) + sq(r2) + r1 * r2))/3
```

4.2 convex-hull

```
// lineal or nlogn
struct pt {
    ll x, y;

    pt operator - (pt p) { return {x-p.x, y-p.y}; }

    bool operator == (pt b) { return x == b.x && y == b.y; }
    bool operator != (pt b) { return !((*this) == b); }
    bool operator < (const pt &o) const { return y < o.y || (y == o.y &&
        x < o.x); }
};

ll cross(pt a, pt b) { return a.x*b.y - a.y*b.x; } // x = 180 -> sin = 0
ll orient(pt a, pt b, pt c) { return cross(b-a, c-a); } // clockwise = -
ld norm(pt a) { return a.x*a.x + a.y*a.y; }
ld abs(pt a) { return sqrt(norm(a)); }

struct polygon {
    vector<pt> p;
    polygon(int n) : p(n) {}

    void delete_repetead() {
        vector<pt> aux;
        sort(p.begin(), p.end());
        for(pt &i : p)
            if(aux.empty() || aux.back() != i)
                aux.push_back(i);
        p.swap(aux);
    }

    int top = -1, bottom = -1;
    void normalize() { /// polygon is CCW
        bottom = min_element(p.begin(), p.end()) - p.begin();
        vector<pt> tmp(p.begin()+bottom, p.end());
```

```
tmp.insert(tmp.end(), p.begin(), p.begin()+bottom);
p.swap(tmp);
bottom = 0;
top = max_element(p.begin(), p.end()) - p.begin();
}
```

```
void convex_hull() {
    sort(p.begin(), p.end());
    vector<pt> ch;
    ch.reserve(p.size()+1);
    for(int it = 0; it < 2; it++) {
        int start = ch.size();
        for(auto &a : p) {
            /// if colineal are needed, use < and remove repeated
            points
            while(ch.size() >= start+2 && orient(ch[ch.size()-2],
                ch.back(), a) <= 0)
                ch.pop_back();
            ch.push_back(a);
        }
        ch.pop_back();
        reverse(p.begin(), p.end());
    }
    if(ch.size() == 2 && ch[0] == ch[1]) ch.pop_back();
    /// be careful with CH of size < 3
    p.swap(ch);
}

ld perimeter() {
    ld per = 0;
    for(int i = 0, n = p.size(); i < n; i++)
        per += abs(p[i] - p[(i+1)%n]);
    return per;
}
};
```

4.3 heron formula

```
ld triangle_area(ld a, ld b, ld c) {
    ld s = (a + b + c) / 2;
    return sqrtl(s * (s - a) * (s - b) * (s - c));
}
```

4.4 segment-intersection

```
// LINE they are parallel
// They never be touched because
// other wise provise the point
// The correct name is segment
struct line {
    ld a, b;
    ld x, y;

    ld m() {
        return (a - x)/(b - y);
    }

    bool horizontal() {
        return b == y;
    }

    bool vertical() {
        return a == x;
    }

    void intersects(line &o) {

        if (horizontal() && o.horizontal()) {
            if (y == o.y) {
                cout << "LINE\n";
            } else {
                cout << "NONE\n";
            }
            return;
        }

        if (vertical() && o.vertical()) {
            if (x == o.x) {
                cout << "LINE\n";
            } else {
                cout << "NONE\n";
            }
            return;
        }

        if (!horizontal() && !o.horizontal()) {
```

```
ld ma = m();
ld mb = o.m();

if (ma == mb) {
    ld someY = (o.x - x)/ma + y;
    if (abs(someY - o.y) <= 0.000001) {
        cout << "LINE\n";
    } else {
        cout << "NONE\n";
    }
} else {
    ld xx = (x*mb - o.x*ma + ma*mb*(o.y - y))/(mb - ma);
    ld yy = (xx - x)/ma + y;
    cout << "POINT " << fixed << setprecision(2) << xx << " "
        << yy << "\n";
}
} else {
    if (!horizontal()) {
        ld xx;
        if (x == a) {
            xx = x;
        } else {
            xx = (o.y - y)/m() + x;
        }
        ld yy = o.y;
        cout << "POINT " << fixed << setprecision(2) << xx << " "
            << yy << "\n";
    } else {
        ld xx;
        if (x == a) {
            xx = x;
        } else {
            xx = (y - o.y)/o.m() + o.x;
        }
        ld yy = y;
        cout << "POINT " << fixed << setprecision(2) << xx << " "
            << yy << "\n";
    }
}

};

void test_case() {
```

```

line l[2];
for (int i = 0; i < 2; i++) {
    ld x, y, a, b;
    cin >> x >> y >> a >> b;
    l[i].a = x;
    l[i].b = y;
    l[i].x = a;
    l[i].y = b;
}

l[0].intersects(l[1]);
}

```

4.5 sin cos law

$$a/\sin A == b/\sin B == c/\sin C$$

$$c^2 = a^2 + b^2 - 2ab\cos C$$

5 graph

5.1 1 - DFS

```

const int n = 1e6;
vector<int> adj[n + 1];
bool visited[n + 1];

void dfs(int x) {
    if (visited[x]) return;
    visited[x] = true;
    for (int &a : adj[x]) {
        dfs(x);
    }
}

```

5.2 2 - BFS

2. BFS

```

vector<int> adj[n + 1];
bool visited[n + 1];

void bfs() {
    queue<int> q;
    q.push(0); // initial node
    visited[0] = true;
    while(q.size() > 0) {
        int c = q.front();
        q.pop();
        for (int a : adj[c]) {
            if (visited[a]) continue;
            q.push(a);
            visited[a] = true;
        }
    }
}

```

5.3 3 - Dijkstra

3. Dijkstra

```

const int inf = 1e9;
vector<pair<int, int>> adj[n];
bool processed[n];
ll distance[n];

void dijkstra() {
    priority_queue<pair<int, int>> q;
    for (int i = 0; i < n; i++) {
        distance[i] = inf;
    }
    distance[start] = 0;
    q.push({0, start});
    while (q.size() > 0) {
        int c = q.top().second;
        q.pop();
        if (processed[c]) continue;
        processed[c] = true;
        for (auto& a : adj[c]) {
            int u = a.first;
            int w = a.second;

```

```

        if (distance[c] + w < distance[u]) {
            distance[u] = distance[c] + w;
            q.push({-distance[u], u});
        }
    }
}

```

5.4 4 - BellmanFord

4. BellmanFord

```

const int inf = 1e9;
vector<tuple<int, int, int>> edges;
ll distance[n];

void bellmanFord() {
    for (int i = 0; i < n; i++) {
        distance[i] = inf;
    }
    distance[start] = 0;
    for (int i = 0; i < n - 1; i++) {
        //bool changed = false; add one iteration (i < n) to
        //validate negative cycles
        for (auto& edge : edges) {
            int a, b, w;
            tie(a, b, w) = edge;
            if (distance[a] + w < distance[b]) {
                distance[b] = distance[a] + w;
                //changed = true;
            }
        }
    }
}

```

5.5 5 - Floyd Warshall

5. Floyd Warshall

```

const int inf = 1e9;

```

```

vector<pair<int, int>> adj[n];
ll distance[n][n];

void floydWarshall() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            distance[i][j] = inf;
        }
    }
    for (int i = 0; i < n; i++) {
        for (auto p : adj[i]) {
            int b = p.first;
            int w = p.second;
            distance[i][b] = w;
        }
    }
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                distance[i][j] = min(distance[i][j],
                                      distance[i][k] + distance[k][j]);
            }
        }
    }
}

```

5.6 6 - Euler Path and Cycle

6. Euler Path and Cycle

```

// TODO

```

5.7 7 - Topological Sort

7. Topological Sort

```

stack<int> topo;
vector<int> adj[n + 1];
bool visited[n + 1];

void dfs(int x) {

```



```

    if (visited[x]) return;
    visited[x] = true;
    for (int a : adj[x]) {
        dfs(a);
    }
    topo.push(x);
}

```

5.8 8 - Transitive Closure

8. Transitive Closure

```

const int inf = 1e9;
vector<int> adj[n];
ll distance[n][n];

void floydWarshall() {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            distance[i][j] = false;
        }
    }
    for (int i = 0; i < n; i++) {
        for (int b : adj[i]) {
            distance[i][b] = true;
        }
    }
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                distance[i][j] |= distance[i][k] &
                    distance[k][j];
            }
        }
    }
}

```

5.9 9 - Kruskal

9. Kruskal

```

vector<tuple<int, int, int> edges;

void kruskal() {
    ll res = 0;
    sort(edges.begin(), edges.end()); // sorted by weight
    for (auto edge : edges) {
        int a, b, w;
        tie(w, a, b) = edge;
        if (find(a) != find(b)) {
            group(a, b);
            res += w;
        }
    }
}

```

5.10 A - Union Find

10. Union Find

```

struct union_find {
    vi link;
    vi score;
    vi size;
    int n;
    void init(int nn) {
        link.resize(nn);
        score.resize(nn);
        size.resize(nn);
        this->n = nn;
        for (int i = 0; i < n; i++) {
            link[i] = i;
            score[i] = 0;
            size[i] = 1;
        }
    }
    int find(int x) {
        if (link[x] == x) return x;
        return (link[x] = find(link[x]));
    }
    void group(int a, int b) {
        int pa = find(a);
        int pb = find(b);
    }
}

```

```

    if (pa != pb) {
        if (score[pa] >= score[pb]) {
            link[pb] = pa;
            size[pa] += size[pb];
            if (score[pa] == score[pb]) score[pa]++;
        } else {
            link[pa] = pb;
            size[pb] += size[pa];
        }
    }
}
};

```

5.11 B - SCC

Dado un grafo dirigido halla las componentes fuertemente conexas (SCC).

```

const int inf = 1e9;
const int MX = 1e5+5; //Cantidad maxima de nodos
vector<int> g[MX]; //Lista de adyacencia
stack<int> st;
int low[MX], pre[MX], cnt;
int comp[MX]; //Almacena la componente a la que pertenece cada nodo
int SCC; //Cantidad de componentes fuertemente conexas
int n, m; //Cantidad de nodos y aristas

void tarjan(int u) {
    low[u] = pre[u] = cnt++;
    st.push(u);

    for (auto &v : g[u]) {
        if (pre[v] == -1) tarjan(v);
        low[u] = min(low[u], low[v]);
    }
    if (low[u] == pre[u]) {
        while (true) {
            int v = st.top(); st.pop();
            low[v] = inf;
            comp[v] = SCC;
            if (u == v) break;
        }
        SCC++;
    }
}

```

```

}

void init() {
    cnt = SCC = 0;
    for (int i = 0; i <= n; i++) {
        g[i].clear();
        pre[i] = -1; //no visitado
    }
}

// example
void test_case() {
    cin >> n >> m;
    init();
    rep(i, 0, m) {
        int x, y;
        cin >> x >> y;
        g[x].pb(y);
    }
    rep(i, 1, n + 1) {
        if (pre[i] == -1) {
            tarjan(i);
        }
    }
}

```

5.12 C-Cycle_{Detection}

```

const int N = 1e5 + 10;
vpl adj[N];
int vis[N];
vpl res;
vpl edge;

void dfs(int x) {
    if (vis[x] == 2) return;
    vis[x] = 1;
    each(z, adj[x]) {
        int y, i;
        tie(y, i) = z;
        if (vis[y] == 1) {
            pl a = {-1, -1};
            if (edge[i] == a) {

```

```

        edge[i] = {y, x};
    }
} else {
    pl a = {-1, -1};
    if (edge[i] == a) {
        edge[i] = {x, y};
    }
}
if (vis[y] == 0) dfs(y);
}
vis[x] = 2;
}

void test_case() {
    int n, m;
    cin >> n >> m;
    edge = vpl(m);
    rep(i, 0, m) {
        int x, y;
        cin >> x >> y;
        adj[x].pb({y, i});
        adj[y].pb({x, i});
        edge[i] = {-1, -1};
    }
    rep(i, 1, n + 1) {
        dfs(i);
    }
    each(r, edge) {
        cout << r.F << " " << r.S << "\n";
    }
}

```

5.13 Z-Extra-OrStatements2Sat

```

// Return the smaller lexicographic array of size n that satities a_i |
// a_j = z
// a_i | a_i = z is allowed.
// there must exists a solution.
vector<ll> f(ll n, vector<tuple<ll,ll,ll>> &statements) {
    ll m = statements.size();
    vector<vector<pair<ll,ll>>> adj(n + 1);

```

```

    const ll bits = 30;
    vector<ll> taken(n+1, (1 << bits) - 1), answer(n+1, (1 << bits) - 1);
    for (int i = 0; i < m; i++) {
        ll x, y, z;
        tie(x, y, z) = statements[i];

        answer[x] &= z;
        answer[y] &= z;
        if (x == y) {
            taken[x] = 0;
            continue;
        }
        taken[x] &= z;
        taken[y] &= z;
        adj[x].pb({y, z});
        adj[y].pb({x, z});
    }
    for (int x = 1; x <= n; x++) {
        for (int i = 0; i < bits; i++) {
            if (!((taken[x] >> i) & 1)) continue;
            ll allHave = true;
            for (auto y : adj[x]) {
                if ((y.S >> i) & 1) {
                    allHave &= ((taken[y.F] >> i) & 1) || ((answer[y.F] >>
                        i) & 1);
                }
            }
            taken[x] -= 1 << i;
            if (allHave) {
                answer[x] -= 1 << i;
                for (auto y : adj[x]) {
                    if ((y.S >> i) & 1) {
                        taken[y.F] |= 1 << i;
                        taken[y.F] ^= 1 << i;
                    }
                }
            }
        }
    }
    answer.erase(answer.begin());
    return answer;
}

```

6 math

6.1 Chinese Remainder

```

ll x, y;
/// O(log(max(a, b)))
ll euclid(ll a, ll b) {
    if(b == 0) { x = 1; y = 0; return a; }
    ll d = euclid(b, a%b);
    ll aux = x;
    x = y;
    y = aux - a/b*y;
    return d;
}

pair<ll, ll> crt(vector<ll> A, vector<ll> M) {
    ll n = A.size(), ans = A[0], lcm = M[0];
    for (int i = 1; i < n; i++) {
        ll d = euclid(lcm, M[i]);
        if ((A[i] - ans) % d) return {-1, -1};
        ll mod = lcm / d * M[i];
        ans = (ans + x * (A[i] - ans) / d % (M[i] / d) * lcm) % mod;
        if (ans < 0) ans += mod;
        lcm = mod;
    }
    return {ans, lcm};
}

```

6.2 Combinatorics

```

// if k == 0 then 1
// if k negative or no enough choices then 0
// O(min(n, n-k)) lineal
ll nck(ll n, ll k) {
    if (k < 0 || n < k) return 0;
    k = min(k, n-k);
    ll ans = 1;
    for (int i = 1; i <= k; i++) {
        ans = ans * (n-i+1) / i;
    }
    return ans;
}

```

6.3 Count_{primes}

```

// sprime.count_primes(n);
// O(n^{2/3})
// PI(n) = Count prime numbers until n inclusive

struct count_primers_struct {
    vector<int> primes;
    vector<int> mnprimes;
    ll ans;
    ll y;
    vector<pair<pair<ll, int>, char>> queries;

    ll count_primes(ll n) {
        // this y is actually n/y
        // also no logarithms, welcome to reality, this y is the best for
        // n=10^12 or n=10^13
        y = pow(n, 0.64);
        if (n < 100) y = n;

        // linear sieve
        primes.clear();
        mnprimes.assign(y + 1, -1);
        ans = 0;
        for (int i = 2; i <= y; ++i) {
            if (mnprimes[i] == -1) {
                mnprimes[i] = primes.size();
                primes.push_back(i);
            }
            for (int k = 0; k < primes.size(); ++k) {
                int j = primes[k];
                if (i * j > y) break;
                mnprimes[i * j] = k;
                if (i % j == 0) break;
            }
        }
        if (n < 100) return primes.size();
        ll s = n / y;

        for (int p : primes) {
            if (p > s) break;
            ans++;
        }
        // pi(n / y)
        int ssz = ans;
    }
}

```

```

// F with two pointers
int ptr = primes.size() - 1;
for (int i = ssz; i < primes.size(); ++i) {
    while (ptr >= i && (ll)primes[i] * primes[ptr] > n)
        --ptr;
    if (ptr < i) break;
    ans -= ptr - i + 1;
}

// phi, store all queries
phi(n, ssz - 1);

sort(queries.begin(), queries.end());
int ind = 2;
int sz = primes.size();

// the order in fenwick will be reversed, because prefix sum in a
// fenwick is just one query
fenwick fw(sz);
for (auto qq : queries) {
    auto na = qq.F;
    auto sign = qq.S;
    auto n = na.F;
    auto a = na.S;
    while (ind <= n)
        fw.add(sz - 1 - mnprimes[ind++], 1);
    ans += (fw.ask(sz - a - 2) + 1) * sign;
}
queries.clear();
return ans - 1;
}

void phi(ll n, int a, int sign = 1) {
    if (n == 0) return;
    if (a == -1) {
        ans += n * sign;
        return;
    }
    if (n <= y) {
        queries.emplace_back(make_pair(n, a), sign);
        return;
    }
    phi(n, a - 1, sign);
    phi(n / primes[a], a - 1, -sign);
}

```

```

}

struct fenwick {
    vector<int> tree;
    int n;

    fenwick(int n = 0) : n(n) {
        tree.assign(n, 0);
    }

    void add(int i, int k) {
        for (; i < n; i = (i | (i + 1)))
            tree[i] += k;
    }

    int ask(int r) {
        int res = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            res += tree[r];
        return res;
    }
};

};

count_primers_struct sprime;

```

6.4 Erdős–Szekeres_theorem

Suppose a, b in \mathbb{N} , $n=ab+1$, and x_1, \dots, x_n is a sequence of n real numbers. Then **this** sequence contains a monotonic increasing (decreasing) subsequence of $a+1$ terms **or** a monotonic decreasing (increasing) subsequence of $b+1$ terms. Dilworth's lemma is a **generalization of this theorem**.

6.5 Extended Euclides

```

// It finds X and Y in equation:
// a * X + b * Y = gcd(a, b)

int x, y;

```



```

        unsigned long long b) {
    unsigned long long ans = 0;
    while (true) {
        if (a >= m) {
            ans += n * (n - 1) / 2 * (a / m);
            a %= m;
        }
        if (b >= m) {
            ans += n * (b / m);
            b %= m;
        }

        unsigned long long y_max = a * n + b;
        if (y_max < m) break;
        // y_max < m * (n + 1)
        // floor(y_max / m) <= n
        n = (unsigned long long)(y_max / m);
        b = (unsigned long long)(y_max % m);
        swap(m, a);
    }
    return ans;
}

long long floor_sum(long long n, long long m, long long a, long long b) {
    assert(0 <= n && n < (1LL << 32));
    assert(1 <= m && m < (1LL << 32));
    unsigned long long ans = 0;
    if (a < 0) {
        unsigned long long a2 = safe_mod(a, m);
        ans -= 1ULL * n * (n - 1) / 2 * ((a2 - a) / m);
        a = a2;
    }
    if (b < 0) {
        unsigned long long b2 = safe_mod(b, m);
        ans -= 1ULL * n * ((b2 - b) / m);
        b = b2;
    }
    return ans + floor_sum_unsigned(n, m, a, b);
}

```

6.8 Greatest Common Divisor

```
// Alternative: __gcd(a, b);
```

```

// O(log(max(a, b)))

ll gcd(ll a, ll b) {
    return b == 0 ? a : gcd(b, a % b);
}

```

6.9 Lowest Common Multiple

```

// O(log(max(a, b)))
int lcm(int a, int b) {
    return a/gcd(a, b) * b;
}

```

6.10 Matrix Exponentiation

```

// For Linear recurrences DP in O(log(N)*M^3)

typedef ll T;
const int M = 2;
struct Matrix {
    T a[M][M] = {0};
    Matrix() {}

    Matrix (vector<vector<T>> o) {
        for (int i = 0; i < M; i++)
            for (int j = 0; j < M; j++)
                a[i][j] = o[i][j];
    }

    Matrix operator * (const Matrix &o) {
        Matrix ans;
        for (int i = 0; i < M; i++)
            for (int j = 0; j < M; j++)
                for (int k = 0; k < M; k++)
                    ans.a[i][j] += a[i][k] * o.a[k][j]
                    //,ans.a[i][j] %= MOD
                    ;
        return ans;
    }
};

```

```

Matrix matrixPower(Matrix a, ll power) {
    Matrix ans;
    for (int i = 0; i < M; i++) ans.a[i][i] = 1;

    while (power) {
        if (power & 1) {
            ans = ans * a;
        }

        a = a * a;
        power >>= 1;
    }

    return ans;
}

void test_case() {
    ll n;
    cin >> n;
    Matrix m({
        {1, 1},
        {1, 0}
    });

    auto ans = matrixPower(m, n);
    cout << ans.a[0][1] << "\n";
}

```

6.11 Modular Arithmetics

Modular Arithmetics.cpp

```

ll sum(ll a, ll b) {
    ll c = a + b;
    if (c >= m) c -= m;
    return c;
}

ll sub(ll a, ll b) {
    ll c = a - b;
    if (c < 0) c += m;
    return c;
}

```

```

}

ll mul(__int128 a, __int128 b) {
    return (a * b) % m;
}

ll modexp(ll a, ll n) {
    if (n == 0) return 1;
    ll p = modexp(a, n / 2);
    ll res = mul(p, p);
    if (n & 1) {
        res = mul(res, a);
    }
    return res;
}

// O(sqrt n)
ll phi(ll n) {
    ll ans = n;
    for (int p = 2; p <= n/p; ++p) {
        if (n % p == 0) ans -= ans / p;
        while (n % p == 0) n /= p;
    }
    if (n > 1) ans -= ans / n;
    return ans;
}

ll x, y;
/// O(log(max(a, b)))
ll euclid(ll a, ll b) {
    if (b == 0) { x = 1; y = 0; return a; }
    ll d = euclid(b, a%b);
    ll aux = x;
    x = y;
    y = aux - a/b*y;
    return d;
}

ll invmod(ll a) {
    ll d = euclid(a, m);
    if (d > 1) return -1;
    return (x % m + m) % m;
}

ll divv(ll a, ll b) {

```



```

    ll inv = invmod(b);
    if (inv == -1) return -1;
    ll res = mul(a, inv);
    return res;
}

// a * (b^{euler(m) - 1})
// for primes: a * b ^ (P - 2)
ll divv2(ll a, ll b) {
    if (__gcd(b, m) != 1) return -1;
    ll ex = modexp(b, euler - 1);
    ll res = mul(a, ex);
    return res;
}

```

6.12 Modular Combinatorics

```

// NCK nck(maxN, primeMod)
// ^nC_k How many ways you can choose k items from an array of n items.

struct NCK {
    ll MAX_N;
    ll MOD;
    vl fact;

    explicit NCK(ll maxN, ll mod) : MAX_N(maxN), MOD(mod) {
        fact.resize(MAX_N + 1, 1);
        fact[0] = 1;
        REP(i, 1, MAX_N) {
            fact[i] = fact[i - 1] * (i % MOD);
            fact[i] %= MOD;
        }
    }

    ll inv(ll a){
        return powmod(a, MOD-2); // MOD is prime, otherwise use powmod(a,
            eulerPhi(mod) - 1)
    }

    ll powmod(ll a, ll b){
        if (b == 0) return 1;
        ll mid = powmod(a, b / 2);

```

```

        ll ans = (mid * mid) % MOD;
        if (b & 1) {
            ans *= a;
            ans %= MOD;
        }
        return ans;
    }

    ll nCk(ll n, ll k){
        ll nOverK = (fact[n] * inv(fact[k])) % MOD;
        return (nOverK * inv(fact[n-k])) % MOD;
    }
};

```

6.13 Ternary Search

```

// this is for find minimum point in a parabolic
// O(log3(n))
ll left = 0;
ll right = n - 1;
while (left + 3 < right) {
    ll mid1 = left + (right - left) / 3;
    ll mid2 = right - (right - left) / 3;
    if (f(b, lines[mid1]) <= f(b, lines[mid2])) {
        right = mid2;
    } else {
        left = mid1;
    }
}

ll target = -4 * a * c;
ll ans = -1; // find the answer, in this case any works.
for (ll mid = left; mid <= right; mid++) {
    if (f(b, lines[mid]) + target < 0) {
        ans = mid;
    }
}

```

6.14 catalan

```

static int MAX = 30;
static long catalan[] = new long[MAX+1];

```

```
static void catalanNumbers(){
    catalan[0] = 1;
    for(int i = 1; i <= MAX; i++){
        catalan[i] = (long)(catalan[i-1]*((double)(2*((2 * i)-
            1))/(i + 1)));
    }
}
```

6.15 factorization

```
// Polar rho, miller rabin
// O(log^3(n))
// But I get TLE once in 1e7
ll expmod(ll b, ll e, ll m) {
    ll ans = 1;
    while (e) {
        if (e&1) ans = (1ll*ans*b) % m;
        b = (1ll*b*b) % m;
        e /= 2;
    }
    return ans;
}

ll mulmod(ll a, ll b, ll m) {
    ll r = a*b-(ll)((long double)a*b/m+.5)*m;
    return r < 0 ? r+m : r;
}

/// O(log^3(n))
bool test(ll n, int a) {
    if (n == a) return true;
    ll s = 0, d = n-1;
    while (d%2 == 0) s++, d /= 2;
    ll x = expmod(a, d, n);
    if (x == 1 || x+1 == n) return true;
    for (int i = 0; i < s-1; i++) {
        x = mulmod(x, x, n);
        if (x == 1) return false;
        if (x+1 == n) return true;
    }
    return false;
}
```

```
ll gcd(ll a, ll b) { return a ? gcd(b%a, a) : b; }
```

```
ll rho(ll n) {
    if (!(n&1)) return 2;
    ll x = 2, y = 2, d = 1;
    ll c = rand() % n + 1;
    while (d == 1) {
        x = (mulmod(x, x, n) + c) % n;
        y = (mulmod(y, y, n) + c) % n;
        y = (mulmod(y, y, n) + c) % n;
        d = gcd(abs(x-y), n);
    }
    return d == n ? rho(n) : d;
}
```

```
bool is_prime(ll n) {
    if (n == 1) return false;
    int ar[] = {2,3,5,7,11,13,17,19,23};
    for (auto &p : ar) if (!test(n, p)) return false;
    return true;
}
```

```
/// O(log(n)^3) aprox
void fact(ll n, map<ll, int> &f) {
    if (n == 1) return;
    if (is_prime(n)) { f[n]++; return; }
    ll q = rho(n);
    fact(q, f); fact(n/q, f);
}
```

```
// #####
// Normal algorithm with precomputing primes
// O(sqrt(MAX_N)/log(sqrt(MAX_N))), it worked for 1e9 for me
const ll MAX_N = 1e7;
vl primes;
void init() {
    ll N = sqrt(MAX_N) + 1;
    vector<bool> sieve(N + 1);
    for (ll i = 2; i <= N; i++) {
        if (!sieve[i]) {
            for (ll j = i*i; j <= N; j+=i) {
                sieve[j] = true;
            }
        }
    }
}
```

```

    }
    for (ll i = 2; i <= N; i++) {
        if (!sieve[i]) primes.pb(i);
    }
}

vl fact(ll n) {
    vl ans;
    ll rest = n;
    for (auto &p : primes) {
        if (p * p > n) break;
        if (rest % p == 0) {
            ans.pb(p);
            while (rest % p == 0) rest/=p;
        }
    }
    if (rest != 1) {
        ans.pb(rest);
    }
    return ans;
}

// #####
// Modification of sieve erathostenes
// From CF Faster than previous, but needs more memory
const int N = int(1e7) + 5;
int mind[N];
void init() {
    for (int i = 0; i < N; i++)
        mind[i] = i;

    for (int p = 2; p < N; p++) {
        if (mind[p] != p)
            continue;
        for (int d = 2 * p; d < N; d += p)
            mind[d] = min(mind[d], p);
    }
}

vector<int> getPrimes(int v) {
    vector<int> ps;
    while (v > 1) {
        if (ps.empty() || ps.back() != mind[v])
            ps.push_back(mind[v]);
        v /= mind[v];
    }
}

```

```

    }
    return ps;
}

```

6.16 fermat

```

// ll fermatFactors(ll n) {
//     ll a = ceil(sqrt(n)) ;
//     if(a * a == n){
//         return a;
//     }
//     ll b;
//     while(true) {
//         ll b1 = a * a - n ;
//         b = (ll)sqrt(b1) ;

//         if(b * b == b1)
//             break;
//         else
//             a += 1;
//     }
//     return min(a - b, a + b);
// }

```

6.17 primes

```

// O(sqrt(n))
bool isPrime(int x) {
    for (int d = 2; d * d <= x; d++) {
        if (x % d == 0)
            return false;
    }
    return true;
}

// O(nloglogn)
// sieve[X] == 0 if it is prime
int const N = 1e6;
bool sieve[N + 1];
vector<int> primes;

```

```

void calculate() {
    for (int p = 2; p <= N; p++) {
        if (sieve[p]) continue;
        primes.PB(p);
        for (ll i = 1ll*p*p; i <= N; i += p)
            sieve[i] = true;
    }
}

// For 64-bit integers
// O((ln n)^2)
// 32 bits bases: 2, 3, 5, 7.
// 64 bits bases: 2 ... 37

using u64 = uint64_t;
using u128 = __uint128_t;

u64 binpower(u64 base, u64 e, u64 mod) {
    u64 result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s) {
    u64 x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
}

bool MillerRabin(u64 n) {
    if (n < 2)
        return false;

```

```

    int r = 0;
    u64 d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}

```

7 query

7.1 1 - Segment Tree

1. Segment Tree

```

const int N = 1e6 + 1;

int tree[N * 4 + 4];
int nums[N + 1];

void build(int i, int l, int r) {
    if (l == r) {
        tree[i] = nums[r];
    } else {
        int mid = (l + r) / 2;
        build(i * 2 + 1, l, mid);
        build(i * 2 + 2, mid + 1, r);
        tree[i] = tree[i * 2 + 1] + tree[i * 2 + 2];
        // tree[i] = compare(tree[i * 2 + 1], tree[i * 2 + 2]);
    }
}

void update(int i, int l, int r, int pos, int diff) {
    if (l <= pos && pos <= r) {

```

```

        if (l == r) { // leaf
            tree[i] += diff;
        } else { // node
            int mid = (l + r) / 2;
            update(i * 2 + 1, l, mid, pos, diff);
            update(i * 2 + 2, mid + 1, r, pos, diff);
            tree[i] = tree[i * 2 + 1] + tree[i * 2 + 2];
            // tree[i] = compare(...)
        }
    }

}

int query(int i, int sl, int sr, int l, int r) {
    if (l <= sl && sr <= r) { // overlap
        return tree[i];
    } else if (sr < l || r < sl) { // no overlap
        return 0;
    } else { // partially overlap
        int mid = (sl + sr) / 2;
        return query(i * 2 + 1, sl, mid, l, r) + query(i * 2 + 2,
            mid + 1, sr, l, r);
        // return compare(a, b);
    }
}
}

```

7.2 Merge_{sort}Tree

```

// usage
// vector<node*> nodes;
// tree.query(l, r, nodes);

// returns log(n) sorted segments in a range (l, r)

struct node {
    ll l, r;
    vl nums;
    vl prefix;
};

struct segtree {
    int n;
    vector<node> tree;
    void init(int nn, vl& nodes) {

```

```

        tree.clear();
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        tree.resize(size * 2);
        build(0, 0, n - 1, nodes);
    }

void query(ll i, ll sl, ll sr, ll l, ll r, vector<node*> &ans) {
    if (l <= sl && sr <= r) {
        ans.pb(&tree[i]);
    } else if (sr < l || r < sl) {

    } else {
        int mid = (sl + sr) >> 1;
        query(i * 2 + 1, sl, mid, l, r, ans);
        query(i * 2 + 2, mid + 1, sr, l, r, ans);
    }
}

void query(ll l, ll r, vector<node*> &ans) {
    return query(0, 0, n - 1, l, r, ans);
}

void build(int nodei, int l, int r, vl &nums) {
    if (l == r) {
        tree[nodei].nums = { nums[l] };
        tree[nodei].prefix = { nums[l] };
        tree[nodei].l = l;
        tree[nodei].r = r;
    } else {
        ll mid = (l + r) >> 1;
        build(nodei * 2 + 1, l, mid, nums);
        build(nodei * 2 + 2, mid + 1, r, nums);
        ll a = tree[nodei*2+1].nums.size();
        ll b = tree[nodei*2+2].nums.size();
        tree[nodei].nums.reserve(a + b);
        tree[nodei].prefix.resize(a+b);
        ll i = 0;
        ll j = 0;
        while (i < a && j < b) {
            ll simon = tree[nodei*2+1].nums[i];

```

```

    ll simon2 = tree[nodei*2+2].nums[j];
    if (simon <= simon2) {
        tree[nodei].nums.pb(simon);
        i++;
    } else {
        tree[nodei].nums.pb(simon2);
        j++;
    }
}
while (i < a) {
    tree[nodei].nums.pb(tree[nodei*2+1].nums[i]);
    i++;
}
while (j < b) {
    tree[nodei].nums.pb(tree[nodei*2+2].nums[j]);
    j++;
}
tree[nodei].prefix[0] = tree[nodei].nums[0];
for (int i = 1; i < a + b; i++) {
    tree[nodei].prefix[i] = tree[nodei].prefix[i - 1] +
        tree[nodei].nums[i];
}
tree[nodei].l = l;
tree[nodei].r = r;
}
}
};

```

7.3 Min Segment Tree

```

// Max segment tree
struct segtree {
    int n;
    vl tree;

    void init(int nn) {
        tree.clear();
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        tree.resize(size * 2);
    }
};

```

```

}

void update(int i, int sl, int sr, int pos, ll diff) {
    if (sl <= pos && pos <= sr) {
        if (sl == sr) {
            tree[i] += diff;
        } else {
            int mid = (sl + sr) / 2;
            update(i * 2 + 1, sl, mid, pos, diff);
            update(i * 2 + 2, mid + 1, sr, pos, diff);
            tree[i] = max(tree[i * 2 + 1], tree[i * 2 + 2]);
        }
    }
}

void update(int pos, ll diff) {
    update(0, 0, n - 1, pos, diff);
}

ll query(int i, int sl, int sr, int l, int r) {
    if (l <= sl && sr <= r) {
        return tree[i];
    } else if (sr < l || r < sl) {
        return INT64_MIN;
    } else {
        int mid = (sl + sr) / 2;
        auto a = query(i * 2 + 1, sl, mid, l, r);
        auto b = query(i * 2 + 2, mid + 1, sr, l, r);
        return max(a, b);
    }
}

ll query(int l, int r) {
    return query(0, 0, n - 1, l, r);
}
};

```

7.4 Mo's

```

const int BLOCK_SIZE = 430; // 1e5=310 2e5=430

struct query {
    int l, r, idx;
};

```

```

    bool operator <(query &other) const {
        return MP(1 / BLOCK_SIZE, r) < MP(other.l / BLOCK_SIZE, other.r);
    }
};

void add(int idx);
void remove(int idx);
ll getAnswer();

vector<ll> mo(vector<query> queries) {
    vector<ll> answers(queries.size());
    int l = 0;
    int r = -1;
    sort(all(queries));
    EACH(q, queries) {
        while (q.l < l) add(--l);
        while (r < q.r) add(++r);
        while (l < q.l) remove(l++);
        while (q.r < r) remove(r--);
        answers[q.idx] = getAnswer();
    }
    return answers;
}

v1 nums; //init
ll ans = 0;
int cnt[1000001];
void add(int idx) {}
void remove(int idx) {}
ll getAnswer() {
    return ans;
}

```

7.5 SegTree Max Sum sub arrays

```

// Segmenttree to calculate the maximum sum of all possible sub arrays.
// assing value is the initial default value
// set the values with modif
// get the answer with tree.query(0, n - 1).val

struct DynamicMaxSubarraySum {

```

```

    struct node {
        ll pref, suf, val, sum;
    };
    int N;
    ll neutral;
    vector<node> t;
    DynamicMaxSubarraySum(int _N, ll assign_value) {
        neutral = assign_value;
        N = _N;
        t.resize(4 * N);
        FOR(i, 0, 4 * N) t[i] = {0, 0, 0, 0};
        build(1, 0, N - 1);
    }
    void build(int i, int l, int r) {
        if(l == r) {
            t[i].pref = t[i].suf = t[i].val = t[i].sum = neutral;
            return;
        }
        int mid = (l + r) >> 1;
        build(2 * i, l, mid);
        build(2 * i + 1, mid + 1, r);
        t[i] = merge(t[2 * i], t[2 * i + 1]);
    }
    node merge(node a, node b) {
        node c;
        c.pref = max(a.pref, a.sum + b.pref);
        c.suf = max(b.suf, b.sum + a.suf);
        c.val = max({a.val, b.val, a.suf + b.pref});
        c.sum = a.sum + b.sum;
        return c;
    }

    void modif(int i, int l, int r, int pos, ll val) {
        if(l > pos || r < pos) return;
        if(l == pos && r == pos) {
            t[i].pref = t[i].suf = t[i].val = t[i].sum = val;
            return;
        }
        int mid = (l + r) >> 1;
        modif(2 * i, l, mid, pos, val);
        modif(2 * i + 1, mid + 1, r, pos, val);
        t[i] = merge(t[2 * i], t[2 * i + 1]);
    }
    node query(int i, int l, int r, int tl, int tr) {
        if(l > tr || r < tl) return {0, 0, 0, 0};

```

```

    if(l >= tl && r <= tr) return t[i];
    int mid = (l + r) >> 1;
    return merge(query(2 * i, l, mid, tl, tr), query(2 * i + 1, mid +
        1, r, tl, tr));
}

void modif(int pos, ll val) {
    modif(1, 0, N - 1, pos, val);
}
node query(int l, int r) {
    return query(1, 0, N - 1, l, r);
}
node query(int pos) {
    return query(1, 0, N - 1, pos, pos);
}
};

```

7.6 fenwicktree

```

struct FenwickTree {
    vector<int> bit;
    int n;

    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }

    FenwickTree(vector<int> a) : FenwickTree(a.size()) {
        for (size_t i = 0; i < a.size(); i++)
            add(i, a[i]);
    }

    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) - 1)
            ret += bit[r];
        return ret;
    }

    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
}

```

```

void add(int idx, int delta) {
    for (; idx < n; idx = idx | (idx + 1))
        bit[idx] += delta;
}
};

```

7.7 min_sparse_table

```

using Type = int;

struct min_sparse {

    int log;
    vector<vector<Type>>> sparse;

    void init(vector<Type> &nums) {
        int n = nums.size();
        log = 0;
        while (n) log++, n/=2;
        n = nums.size();
        sparse.assign(n, vector<Type>(log, 0));
        for (int i = 0; i < n; i++) sparse[i][0] = nums[i];
        for (int l = 1; l < log; l++) {
            for (int j = 0; j + (1 << l) - 1 < n; j++) {
                sparse[j][l] = min(sparse[j][l-1], sparse[j+(1 <<
                    (l-1))][l-1]);
            }
        }
    }

    Type query(int x, int y) {
        int n = y - x + 1;
        int logg = -1;
        while (n) logg++, n/=2;
        return min(sparse[x][logg], sparse[y-(1 << logg)+1][logg]);
    }
};

```

7.8 struct lazy tree


```

struct lazytree {
    int n;
    vl sum;
    vl lazySum;

    void init(int nn) {
        sum.clear();
        n = nn;
        int size = 1;
        while (size < n) {
            size *= 2;
        }
        sum.resize(size * 2);
        lazySum.resize(size * 2);
    }

    void update(int i, int sl, int sr, int l, int r, ll diff) {
        if (lazySum[i]) {
            sum[i] += (sr - sl + 1) * lazySum[i];
            if (sl != sr) {
                lazySum[i * 2 + 1] += lazySum[i];
                lazySum[i * 2 + 2] += lazySum[i];
            }
            lazySum[i] = 0;
        }
        if (l <= sl && sr <= r) {
            sum[i] += (sr - sl + 1) * diff;
            if (sl != sr) {
                lazySum[i * 2 + 1] += diff;
                lazySum[i * 2 + 2] += diff;
            }
        } else if (sr < l || r < sl) {
        } else {
            int mid = (sl + sr) >> 1;
            update(i * 2 + 1, sl, mid, l, r, diff);
            update(i * 2 + 2, mid + 1, sr, l, r, diff);
            sum[i] = sum[i * 2 + 1] + sum[i * 2 + 2];
        }
    }

    void update(int l, int r, ll diff) {
        assert(l <= r);
        assert(r < n);
        update(0, 0, n - 1, l, r, diff);
    }
}

```

```

ll query(int i, int sl, int sr, int l, int r) {
    if (lazySum[i]) {
        sum[i] += lazySum[i] * (sr - sl + 1);
        if (sl != sr) {
            lazySum[i * 2 + 1] += lazySum[i];
            lazySum[i * 2 + 2] += lazySum[i];
        }
        lazySum[i] = 0;
    }
    if (l <= sl && sr <= r) {
        return sum[i];
    } else if (sr < l || r < sl) {
        return 0;
    } else {
        int mid = (sl + sr) >> 1;
        return query(i * 2 + 1, sl, mid, l, r) + query(i * 2 + 2, mid
            + 1, sr, l, r);
    }
}

ll query(int l, int r) {
    assert(l <= r);
    assert(r < n);
    return query(0, 0, n - 1, l, r);
}
};

```

7.9 struct segment tree

// Segment Tree for Sum in ranges, also gives you the quantity of numbers greater than zero (present numbers)

// segtree tree;
 // tree.init(N);
 // update values
 // uses queries

```

struct segtree {
    int n;
    vl sum;
    vl present;
}

```

```

void init(int nn) {
    sum.clear();
    present.clear();
    n = nn;
    int size = 1;
    while (size < n) {
        size *= 2;
    }
    sum.resize(size * 2);
    present.resize(size * 2);
}

void update(int i, int sl, int sr, int pos, ll diff) {
    if (sl <= pos && pos <= sr) {
        if (sl == sr) {
            sum[i] += diff;
            present[i] = sum[i] > 0;
        } else {
            int mid = (sl + sr) / 2;
            update(i * 2 + 1, sl, mid, pos, diff);
            update(i * 2 + 2, mid + 1, sr, pos, diff);
            sum[i] = sum[i * 2 + 1] + sum[i * 2 + 2];
            present[i] = present[i * 2 + 1] + present[i * 2 + 2];
        }
    }
}

void update(int pos, ll diff) {
    update(0, 0, n - 1, pos, diff);
}

pl query(int i, int sl, int sr, int l, int r) {
    if (l <= sl && sr <= r) {
        return {sum[i], present[i]};
    } else if (sr < l || r < sl) {
        return {0, 0};
    } else {
        int mid = (sl + sr) / 2;
        auto a = query(i * 2 + 1, sl, mid, l, r);
        auto b = query(i * 2 + 2, mid + 1, sr, l, r);
        return {a.F + b.F, a.S + b.S};
    }
}

pl query(int l, int r) {

```

```

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

```

7.10 $\text{sum}_{\text{parseable}}$

8 string

8.1 1 - KMP

1. KMP.cpp

```

struct KMP {
    int kmp(vector<ll> &s, vector<ll> &p) {
        int n = s.size(), m = p.size(), cnt = 0;
        vector<int> pf = prefix_function(p);
        for(int i = 0, j = 0; i < n; i++) {
            while(j && s[i] != p[j]) j = pf[j-1];
            if(s[i] == p[j]) j++;
            if(j == m) {
                cnt++;
                j = pf[j-1];
            }
        }
        return cnt;
    }
}

vector<int> prefix_function(vector<ll> &s) {
    int n = s.size();
    vector<int> pf(n);
    pf[0] = 0;
    for (int i = 1, j = 0; i < n; i++) {
        while (j && s[i] != s[j]) j = pf[j-1];
        if (s[i] == s[j]) j++;
        pf[i] = j;
    }
    return pf;
}

```

8.2 Hashing

```

ll pot(ll b, ll e, ll m) {
    ll res = 1;
    while (e > 0) {
        if (e&1) res = res * b % m;
        e >>= 1;
        b = b * b % m;
    }
    return res;
}

struct Hash
{
    int p = 997, m[2], in[2];
    vector<int> h[2], inv[2];
    Hash(string s)
    {
        m[0] = 998244353, m[1] = 1000000009;
        for(int i = 0; i < 2; i++)
        {
            in[i] = pot(p, m[i]-2, m[i]);
            h[i].resize(s.size() + 1);
            inv[i].resize(s.size() + 1);
            ll acu = 1;
            h[i][0] = 0, inv[i][0] = 1;
            for(int j = 0; j < s.size(); j++)
            {
                h[i][j + 1] = (h[i][j] + acu * s[j]) % m[i];
                inv[i][j + 1] = (1ll * inv[i][j] * in[i]) %
                    m[i];
                acu = (acu * p) % m[i];
            }
        }
    }

    // Return the hash of the the substring of 's' from index 'b' to
    // 'e' inclusive.
    // Note that ABCABC, the hash of 0 to 2 is the same as 3 to 5.
    ll get(int b, int e)
    {
        e++; // Important to make this inclusive
        ll ha[2];
        for(int i = 0; i < 2; i++)

```

```

        ha[i] = (((h[i][e] - h[i][b]) * (1ll)inv[i][b]) %
            m[i] + m[i]) % m[i];
        return((ha[0] << 32) | ha[1]);
    }
};

```

9 tree

9.1 1 K-th Parent

1. K-th Parent.cpp

```

class TreeAncestor {
    int LOG = 20;
    int up[50000][20];
public:
    TreeAncestor(int n, vector<int>& parent) {
        memset(up, -1, 50000 * LOG * 4);
        for (int i = 0; i < n; i++) {
            up[i][0] = parent[i];
        }
        for (int k = 1; k < LOG; k++) {
            for (int i = 0; i < n; i++) {
                if (up[i][k-1] != -1)
                    up[i][k] = up[up[i][k-1]][k-1];
            }
        }
    }

    int getKthAncestor(int node, int k) {
        for (int i = 0; i < LOG; i++) {
            if (k & 1<<i) {
                node = up[node][i];
            }
            if (node == -1) return -1;
        }
        return node;
    }
};

```

9.2 Nearest_{selectedNodes}problem

```
// Given an order of selected nodes in a tree, you should print the
// minimum distance between two selected nodes after each operation.

// O(nlogn or n*sqrt(n)); n <= 2*10^5, 2.7 seconds.
// adj is the adjacency list, order is the selected nodes in order
// n is the number of nodes, returns the minimum after each operation
// note that operation 0 answer is 1e9
v1 f(vv1 &adj, v1 &order, ll n) {
    v1 answer;
    v1 dist(n + 1, 1e9);
    ll best = 1e9;
    v1 q(n + 1);
    ll sz = 0;
    for (int i = 0; i < n; i++) {
        best = min(best, dist[order[i]]);
        sz = 0;
        dist[order[i]] = 0;
        q[sz++] = order[i];
        ll idx = 0;
        while (idx < sz) {
            ll x = q[idx++];
            if (dist[x] + 1 >= best) break;
            for (auto &y : adj[x]) {
                if (dist[x] + 1 < dist[y]) {
                    dist[y] = dist[x] + 1;
                    q[sz++] = y;
                }
            }
        }
        answer.pb(best);
    }
    return answer;
}
```

9.3 Two_{pieces}_{onTree}

```
// In a tree with 'n' nodes where 2 pieces starting from root 1
// must go to certain nodes each one and must not exceed 'd' between
// the two pieces, after they have to return to root 1
// two_pieces_on_tree() find the minimum quantity of moves
```

```
// My submission in CF:
https://codeforces.com/contest/1774/submission/189071201
```

```
ll n, d; // quantity of nodes, maximum distance between pieces
vv1 children; // tree
vector<int> a, b; // nodes that must visit first and second piecesS

void dfs(ll x, v1 &route) {
    route.pb(x);
    ll kParent = 1; //route
    if (route.size() - 1 >= d) {
        kParent = route[route.size() - 1 - d];
    }
    b[kParent] |= a[x];
    a[kParent] |= b[x];
    each(y, children[x]) {
        dfs(y, route);
        a[x] |= a[y];
        b[x] |= b[y];
    }
    route.pop_back();
}

ll two_pieces_on_tree() {
    ll root = 1;
    v1 emptyRoute = v1();
    dfs(root, emptyRoute);
    ll total = 0;
    for (int i = 1; i <= n; i++) {
        total += a[i] + b[i];
    }
    return total * 2 - 4;
}
```

9.4 lca

```
#include<bits/stdc++.h>
//#include<cmath>
//#include<bitset>

using namespace std;

#define MP make_pair
```

```

#define MT make_tuple
#define PB push_back
#define F first
#define S second
#define all(x) (x).begin(), (x).end()
#define sortt(x) sort(all(x))
#define sortn(x, n) sort((x), (x) + (n));
#define SQ(a) ((a) * (a))
#define max3(a, b, c) max((a), max((b), (c)))
#define max4(a, b, c, d) max(max3(a, b, c), d)
#define min3(a, b, c) min((a), min((b), (c)))
#define min4(a, b, c, d) min(min3(a, b, c), d)
#define fastIO() cin.tie(0); ios::sync_with_stdio(0);

// loops
#define FOR(i, a, b) for (int (i) = (a); (i) < (b); (i)++)
#define ROF(i, a, b) for (int (i) = (a); (i) >= (b); (i)--)
#define REP(i, a, b) for (int (i) = (a); (i) <= (b); (i)++)
#define EACH(a, x) for (auto &(a) : (x))

typedef long long ll;
typedef pair<int, int> pii;
typedef tuple<long long, long long, long long> tiii;
typedef pair<long long, long long> pll;
typedef unsigned long long ull;
typedef long double ld;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vl;
typedef vector<string> vs;

const int dx[4]{1,0,-1,0}, dy[4]{0,1,0,-1};
const int MOD = 1e9 + 7;

template <typename... V>
void funcDebug(string vars, V... v) {
    cout << vars << " = ";
    string delim = "";
    (... , (cout << delim << v, delim = ", "));
    cout << endl;
}

// #define ONLINE_JUDGE
#ifndef ONLINE_JUDGE
    #define deb(x...) funcDebug(#x, x);

```

```

#define debug(x) (cout << #x << ": " << x << endl);
#define LINE cout << "-----" << endl;
#define LINE2 cout << " ~ ~ ~ ~ ~ " << endl;
#define LINE3 cout << "- - - - -" << endl;
#define debugA(x, n) cout << #x << ": "; for (int zabz = 0; zabz < n;
    zabz++) cout << (x)[zabz] << " "; cout << endl;
#define debugI(x) cout << #x << ": "; EACH(y, (x)) cout << y << " ";
    cout << endl;

#else
    #define deb(x...)
    #define debug(x)
    #define debugA(x, n)
    #define LINE
    #define LINE2
    #define LINE3
    #define debugI(x)
#endif

const ll inf1 = INT64_MAX;
const int inf = INT32_MAX;

// const int N = 1e5 + 10;

// const int LOG = 16;

const int N = 50000;
const int LOG = 16;

vector<pii> children[N];
int up[N][LOG];
int dist[N][LOG];
int depth[N];
bool visited[N];

void dfs(int x, int level = 0) {
    if (visited[x]) return;
    visited[x] = true;
    depth[x] = level;
    EACH(y, children[x]) {
        if (!visited[y.F]) {
            up[y.F][0] = x;
            dist[y.F][0] = y.S;

```

```

        dfs(y.F, level + 1);
    }
}

int query(int x, int y) {
    if (depth[y] > depth[x]) swap(x, y);
    int toUp = depth[x] - depth[y];
    int bit = 0;
    int res = 0;
    while (toUp) {
        if (toUp & 1) res += dist[x][bit], x = up[x][bit];
        bit++;
        toUp >>= 1;
    }
    if (x == y) return res;
    ROF(i, LOG - 1, 0) {
        if (up[x][i] != up[y][i]) {
            res += dist[x][i] + dist[y][i];
            x = up[x][i];
            y = up[y][i];
        }
    }
    return dist[x][0] + dist[y][0] + res;
}

void solve() {
    int n;
    cin >> n;
    FOR(i, 0, n - 1) {
        int a, b, w;
        cin >> a >> b >> w;
        children[a].PB({b, w});
        children[b].PB({a, w});
    }
    int root = 0;
    dfs(root);
    FOR(i, 1, LOG) {
        FOR(j, 0, n) {
            int ancestor = up[j][i - 1];
            up[j][i] = up[ancestor][i - 1];
            dist[j][i] = dist[ancestor][i - 1] + dist[j][i - 1];
        }
    }
    int q;

```

```

    cin >> q;
    while (q--) {
        int a, b;
        cin >> a >> b;
        cout << query(a, b) << "\n";
    }
}

int main() {
    fastIO();
    solve();
}

```

9.5 moetree

```

#include <bits/stdc++.h>
using namespace std;
typedef vector<int> vi;
typedef vector<vi> vvi;

map<int, int> getID;
map<int, int>::iterator it;

const int LOGN = 20;
int id, bs, N;
int counter[50050];
int A[50050], P[100050];
int res[100050];
int st[50050], ed[50050];
int DP[20][50050], level[50050];
bool flag[50050];
bool seen[50050];
vvi edges;

struct Q {
    int l, r, p, id;
    bool operator < (const Q& other) const {
        return (1 / bs < other.l / bs || (1 / bs == other.l / bs && r <
            other.r));
    } //operator <
} q[100050];

```

```

void DFS0(const int u) {
    seen[u] = 1;
    P[id] = u;
    st[u] = id++;
    for (auto& e : edges[u]) {
        if (!seen[e]) {
            DP[0][e] = u;
            level[e] = level[u] + 1;
            DFS0(e);
        }
    }
    P[id] = u;
    ed[u] = id++;
}

void prep(const int r) {
    level[r] = 0;
    for (int i = 0; i < LOGN; i++)
        DP[i][r] = r;
    id = 0;
    DFS0(r);
    for (int i = 1; i < LOGN; i++)
        for (int j = 1; j <= N; j++)
            DP[i][j] = DP[i - 1][DP[i - 1][j]];
}

int LCA(int a, int b) {
    if (level[a] > level[b])
        swap(a, b);
    int diff = level[b] - level[a];
    for (int i = 0; i < LOGN; i++)
        if (diff & (1 << i))
            b = DP[i][b]; //move 2^i parents upwards
    if (a == b)
        return a;
    for (int i = LOGN - 1; i >= 0; i--)
        if (DP[i][a] != DP[i][b])
            a = DP[i][a], b = DP[i][b];
    return DP[0][a];
}

int main() {
    int Q, n1, n2, L, R, a, v = 1, tot;
    scanf("%d %d", &N, &Q);
    edges.assign(N + 5, vi());

```

```

    bs = sqrt(N);
    for (int i = 1; i <= N; i++) {
        scanf("%d", &a);
        A[i] = ((it = getID.find(a)) != getID.end()) ? it->second :
            (getID[a] = v++);
    }
    for (int i = 0; i < N - 1; i++) {
        scanf("%d %d", &n1, &n2);
        edges[n1].push_back(n2);
        edges[n2].push_back(n1);
    }
    prep(1);
    for (int i = 0; i < Q; i++) {
        scanf("%d %d", &n1, &n2);
        if (st[n1] > st[n2])
            swap(n1, n2);
        q[i].p = LCA(n1, n2);
        if (q[i].p == n1)
            q[i].l = st[n1], q[i].r = st[n2];
        else
            q[i].l = ed[n1], q[i].r = st[n2];
        q[i].id = i;
    }
    sort(q, q + Q);
    L = 0; R = -1; tot = 0;
    for (int i = 0; i < Q; i++) {
        while (R < q[i].r) {
            if (!flag[P[++R]])
                tot += (++counter[A[P[R]]] == 1);
            else
                tot -= (--counter[A[P[R]]] == 0);
            flag[P[R]] = !flag[P[R]];
        }
        while (R > q[i].r) {
            if (!flag[P[R]])
                tot += (++counter[A[P[R]]] == 1);
            else
                tot -= (--counter[A[P[R]]] == 0);
            flag[P[R]] = !flag[P[R]];
            R--;
        }
        while (L < q[i].l) {
            if (!flag[P[L]])
                tot += (++counter[A[P[L]]] == 1);
            else

```

```

        tot -= (--counter[A[P[L]]] == 0);
        flag[P[L]] = !flag[P[L]];
        L++;
    }//while
    while (L > q[i].l) {
        if (!flag[P[--L]])
            tot += (++counter[A[P[L]]] == 1);
        else
            tot -= (--counter[A[P[L]]] == 0);
        flag[P[L]] = !flag[P[L]];
    }//while
    res[q[i].id] = tot + (q[i].p != P[q[i].l] && !counter[A[q[i].p]]);
} //for
for (int i = 0; i < Q; i++)
    printf("%d\n", res[i]);
return 0;
} //main

```

9.6 simple-lca

// view: <https://cses.fi/problemset/task/1688/>

```

vector<vector<ll>> children;
vector<vector<ll>> up;
const int LOG = 18; // 2e5
vector<ll> depth;

void dfs(ll x, ll d = 0) {
    depth[x] = d;
    for (auto y : children[x]) {
        dfs(y, d + 1);
    }
}

ll kParent(ll x, ll k) {
    ll i = 0;
    while (k) {
        if (k & 1) {
            x = up[i][x];
        }
        k >>= 1;
        i++;
    }
    return x;
}

```

```

}

ll query(ll x, ll y) {
    if (depth[x] < depth[y]) {
        swap(x, y);
    }

    x = kParent(x, depth[x] - depth[y]);
    if (x == y) {
        return x;
    }
    for (int i = LOG - 1; i >= 0; i--) {
        if (up[i][x] != up[i][y]) {
            x = up[i][x];
            y = up[i][y];
        }
    }
    return up[0][x];
}

void test_case() {
    ll n, q;
    cin >> n >> q;
    children = vector<vector<ll>>(n);
    up = vector<vector<ll>>(LOG, vector<ll>(n, 0));
    depth = vector<ll>(n);
    for (int i = 1; i < n; i++) {
        ll p;
        cin >> p;
        p--;
        children[p].pb(i);
        up[0][i] = p;
    }
    dfs(0);
    for (int i = 1; i < LOG; i++) {
        for (int j = 0; j < n; j++) {
            up[i][j] = up[i-1][up[i-1][j]];
        }
    }
    for (int i = 0; i < q; i++) {
        ll x, y;
        cin >> x >> y;
        x--, y--;
        cout << query(x, y) + 1 << "\n";
    }
}

```

}

10 util

10.1 PI

```
const ld PI = acos(-1);
```

10.2 custom_hash

```
struct custom_hash {
    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();
        x ^= FIXED_RANDOM;
        return x ^ (x >> 16);
    }
};

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

10.3 custom_hash_pair

```
// Use: unordered_set<pair<ll,ll>, HASH> exists;
```

```
struct HASH{
    size_t operator()(const pair<ll,ll>&x) const{
        return hash<ll>()(((ll)x.first)^(((ll)x.second)<<32));
    }
};
```

10.4 exponential_notation

```
// O(n) convert numbers to Exponential Notation
// (e.g 0102.150 -> 1.0215E2)
// only float numbers > 0
string exponential_notation(string s) {
    int firstPos = find_if(all(s), [&](char c) {
        return c != '0' && c != '.';
    }) - s.begin();
    int dotPos = find(all(s), '.') - s.begin();
    ll base = dotPos - (firstPos + (firstPos <= dotPos));
    s.erase(dotPos, 1);
    for (int i = 0; i < 2; i++) { //erase traveling zeros
        while (s.back() == '0') s.pop_back();
        reverse(all(s));
    }
    if (s.size() > 1) s.insert(1, ".");
    if (base != 0) s+= "E" + to_string(base);
    return s;
}
```

10.5 io-int128

```
__int128 read() {
    __int128 x = 0, f = 1;
    char ch = getchar();
    while (ch < '0' || ch > '9') {
        if (ch == '-') f = -1;
        ch = getchar();
    }
    while (ch >= '0' && ch <= '9') {
        x = x * 10 + ch - '0';
        ch = getchar();
    }
}
```

```

    return x * f;
}
void print(__int128 x) {
    if (x < 0) {
        putchar('-');
        x = -x;
    }
    if (x > 9) print(x / 10);
    putchar(x % 10 + '0');
}

void print(__int128 x) {
    if (x < 0) {
        cout << "-";
        x = -x;
    }
    if (x > 9) print(x / 10);
    cout << char((int)(x % 10) + '0');
}

```

10.6 macros

```

#define MP make_pair
#define MT make_tuple
#define PB push_back
#define F first
#define S second
#define all(x) (x).begin(), (x).end()
#define sortt(x) sort(all(x))
#define sortn(x, n) sort((x), (x) + (n));
#define SQ(a) ((a) * (a))
#define max3(a, b, c) max((a), max((b), (c)))
#define max4(a, b, c, d) max(max3(a, b, c), d)
#define min3(a, b, c) min((a), min((b), (c)))
#define min4(a, b, c, d) min(min3(a, b, c), d)
#define fastIO() cin.tie(0); ios::sync_with_stdio(0);

// loops
#define FOR(i, a, b) for (ll (i) = (a); (i) < (b); (i)++)
#define ROF(i, a, b) for (ll (i) = (a); (i) >= (b); (i)--)
#define REP(i, a, b) for (ll (i) = (a); (i) <= (b); (i)++)
#define EACH(a, x) for (auto &(a) : (x))

```

```

typedef long long ll;
typedef pair<int, int> pii;
typedef tuple<long long, long long, long long> tiii;
typedef pair<long long, long long> pll;
typedef unsigned long long ull;
typedef long double ld;
typedef vector<int> vi;
typedef vector<bool> vb;
typedef vector<ll> vl;
typedef vector<pll> vppll;
typedef vector<vl> vv1;
typedef vector<vi> vvi;
typedef vector<string> vs;
typedef vector<ld> vld;

template<class T> using pql = priority_queue<T, vector<T>, greater<T>>;
template<class T> using pqg = priority_queue<T>;

const ld DINF=1e100;
const ld EPS = 1e-9;
const ld PI = acos(-1);
const ll inf1 = INT64_MAX;
const int inf = INT32_MAX;
const int dx[4]={1,0,-1,0}, dy[4]={0,1,0,-1};
const int MOD = 1e9 + 7;

```

10.7 multi_oset

```

#include <bits/stdc++.h>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/assoc_container.hpp>

using namespace __gnu_pbds;

struct multiordered_set {
    tree<ll,
        null_type,
        less_equal<ll>, // this is the trick
        rb_tree_tag,
        tree_order_statistics_node_update> oset;

    //this function inserts one more occurrence of (x) into the set.
    void insert(ll x) {

```

```

    oset.insert(x);
}

//this function checks weather the value (x) exists in the set or not.
bool exists(ll x) {
    auto it = oset.upper_bound(x);
    if (it == oset.end()) {
        return false;
    }
    return *it == x;
}

//this function erases one occurrence of the value (x).
void erase(ll x) {
    if (exists(x)) {
        oset.erase(oset.upper_bound(x));
    }
}

//this function returns the value at the index (idx)..(0 indexing).
ll find_by_order(ll pos) {
    return *(oset.find_by_order(pos));
}

//this function returns the first index of the value (x)..(0
indexing).
int first_index(ll x) {
    if (!exists(x)) {
        return -1;
    }
    return (oset.order_of_key(x));
}

//this function returns the last index of the value (x)..(0 indexing).
int last_index(ll x) {
    if (!exists(x)) {
        return -1;
    }
    if (find_by_order(size() - 1) == x) {
        return size() - 1;
    }
    return first_index(*oset.lower_bound(x)) - 1;
}

//this function returns the number of occurrences of the value (x).

```

```

int count(ll x) {
    if (!exists(x)) {
        return -1;
    }
    return last_index(x) - first_index(x) + 1;
}

//this function clears all the elements from the set.
void clear() {
    oset.clear();
}

//this function returns the size of the set.
ll size() {
    return (ll)oset.size();
}
};

```

10.8 oset

```

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

#define oset tree<ll, null_type, less<ll>,
    rb_tree_tag, tree_order_statistics_node_update>
//find_by_order(k) order_of_key(k)

```

10.9 pragmas

```

#pragma GCC target("popcnt")
//It's worth noting that after adding __builtin_popcount() is replaced to
corresponding machine instruction (look at the difference). In my
test this maked x2 speed up. bitset::count() use __builtin_popcount()
call in implementation, so it's also affected by this.
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")
#pragma GCC target("popcnt")
#pragma GCC target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,tune=native")
#pragma GCC optimize(3)

```

```

#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vc")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")

```

```

#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")

```

10.10 *priority_queue*

```

template<class T> using pql = priority_queue<T,vector<T>,greater<T>>;//
    less
template<class T> using pqg = priority_queue<T>; // greater

```

10.11 random

```

mt19937 mt_rng(chrono::steady_clock::now().time_since_epoch().count());
// also for ll exists mt19937_64
ll randint(ll a, ll b) {
    return uniform_int_distribution<ll>(a, b)(mt_rng);
}

```

10.12 util builtin functions

```

# Sum the values of a iterable
# Very important to put 0ll to avoid overflows
accumulate(v.begin(),v.end(),0ll)/n;

```
