General

Template CPP

#pragma GCC optimize("Ofast")

#pragma GCC optimize("unroll-loops")

#include "bits/stdc++.h"

using **namespace** std;

using ll = **long** **long**;

using ld = **long** **double**;

#define each(i, a) for (**auto** &i : a)

#define all(x) (x).begin(), (x).end()

#define cmin(a, b) a = min(a, b)

#define cmax(a, b) a = max(a, b)

*// #ifndef ONLINE\_JUDGE*

*//     #include "debug.hpp"*

*// #endif*

mt19937 mt(chrono::high\_resolution\_clock::now().time\_since\_epoch().count());

**int** getRand() {

    return mt();

}

**void** readMain() {}

*// #define READ\_MAIN*

*// #define TESTCASES*

*// #define FILES*

*// #define TIME*

**void** solve(**int** **&**t) {

}

int32\_t main(**void**) {

    cin.tie(0)->sync\_with\_stdio(0);

**int** t = 1;

    clock\_t tt = clock();

    #ifdef FILES

        freopen("input.txt", "r", stdin);

        freopen("output.txt", "w", stdout);

    #endif

    #ifdef READ\_MAIN

        readMain();

    #endif

    #ifdef TESTCASES

        cin >> t;

    #endif

    for (**int** i = 1; i <= t; i++) {

        solve(i);

        cout << "\n";

    }

    #ifdef TIME

        cout << setprecision(3) << fixed << (ld)(clock() - tt) / CLOCKS\_PER\_SEC;

    #endif

    return 0;

}

Includes cpp

#include <iostream>

#include <iomanip>

#include <fstream>

#include <cstdio>

#include <cstdlib>

#include <cassert>

#include <climits>

#include <cmath>

#include <algorithm>

#include <cstring>

#include <string>

#include <vector>

#include <list>

#include <stack>

#include <set>

#include <bitset>

#include <queue>

#include <map>

#include <sstream>

#include <functional>

#include <unordered\_map>

#include <unordered\_set>

#include <complex>

#include <random>

#include <chrono>

Python template

import sys

import re

from math import ceil, log, sqrt, floor

\_\_local\_run\_\_ = False

if \_\_local\_run\_\_:

    sys.stdin = open('input.txt', 'r')

    sys.stdout = open('output.txt', 'w')

**def** main():

    a = int(input())

    b = int(input())

    print(a\*b)

main()

Compilation

g++ $1.cpp -o $1

Compile and run

g++ $1.cpp -o $1 && ./$1

Autotests

for ((i = 1; i <= 5; i++)); do

  echo "Checking test #$i"

  ./gen > input

  ./main < input > outputMain

  ./brute < input > outputBrute

  diff -w outputMain outputBrute || break

*# diff -w <(./main < input) <(./brute < input) || break*

  echo "Done #$i"

done;

*# -w ignoring white spaces*

Debug

#ifndef DEBUG\_H

#define DEBUG\_H

#include <iostream>

#include <string>

#include <utility>

using **namespace** std;

**inline** **void** \_\_print(**bool** x) { cerr << (x ? "true" : "false"); }

**inline** **void** \_\_print(**char** x) { cerr << '\'' << x << '\''; }

**inline** **void** \_\_print(**const** **char\*** x) { cerr << "\"" << x << "\""; }

**inline** **void** \_\_print(**const** std::string**&** x) { cerr << "\"" << x << "\""; }

**inline** **void** \_\_print(**int** x) { cerr << x; }

**inline** **void** \_\_print(**long** **long** x) { cerr << x; }

**inline** **void** \_\_print(**unsigned** **long** **long** x) { cerr << x; }

**inline** **void** \_\_print(**float** x) { cerr << x; }

**inline** **void** \_\_print(**double** x) { cerr << x; }

**inline** **void** \_\_print(**long** **double** x) { cerr << x; }

**template** <**typename** T>

**inline** **void** \_\_print(**const** T**&** x) {

    cerr << "{"; **int** f = 0;

    for (**auto**& i : x) cerr << (f++ ? ", " : ""), \_\_print(i);

    cerr << "}";

}

**inline** **void** \_print() { cerr << "]\n"; }

**template** <**typename** T, **typename**... V>

**inline** **void** \_print(T t, V... v) {

    \_\_print(t);

    if (sizeof...(v)) cerr << ", ";

    \_print(v...);

}

#define debug(x...) cerr << "[" << #x << "] = ["; \_print(x);

#endif

Generate file cpp

#pragma GCC optimize("Ofast")

#pragma GCC optimize("unroll-loops")

#include "bits/stdc++.h"

using **namespace** std;

using ll = **long** **long**;

using ld = **long** **double**;

#define each(i, a) for (**auto** &i : a)

#define all(x) (x).begin(), (x).end()

#define cmin(a, b) a = min(a, b)

#define cmax(a, b) a = max(a, b)

*// #ifndef ONLINE\_JUDGE*

*//     #include "debug.hpp"*

*// #endif*

mt19937 mt(chrono::high\_resolution\_clock::now().time\_since\_epoch().count());

**int** rand(**int** a, **int** b) {

  return a + getRand() % (b - a + 1);

}

**void** gen\_array(**int** a, **int** b, **int** n) {

*// cout << n << "\n";*

    for (**int** i = 0; i < n; i++) {

        cout << rand(a, b) << " ";

    }

}

**int** getRand() {

    return mt();

}

**void** readMain() {}

*// #define READ\_MAIN*

*// #define TESTCASES*

*// #define FILES*

*// #define TIME*

**void** solve(**int** **&**t) {

}

int32\_t main(**void**) {

    cin.tie(0)->sync\_with\_stdio(0);

**int** t = 1;

    clock\_t tt = clock();

    #ifdef FILES

        freopen("input.txt", "r", stdin);

        freopen("output.txt", "w", stdout);

    #endif

    #ifdef READ\_MAIN

        readMain();

    #endif

    #ifdef TESTCASES

        cin >> t;

    #endif

    for (**int** i = 1; i <= t; i++) {

        solve(i);

        cout << "\n";

    }

    #ifdef TIME

        cout << setprecision(3) << fixed << (ld)(clock() - tt) / CLOCKS\_PER\_SEC;

    #endif

    return 0;

}

Solution file cpp

#pragma GCC optimize("Ofast")

#pragma GCC optimize("unroll-loops")

#include "bits/stdc++.h"

using **namespace** std;

using ll = **long** **long**;

using ld = **long** **double**;

#define each(i, a) for (**auto** &i : a)

#define all(x) (x).begin(), (x).end()

#define cmin(a, b) a = min(a, b)

#define cmax(a, b) a = max(a, b)

*// #ifndef ONLINE\_JUDGE*

*//     #include "debug.hpp"*

*// #endif*

mt19937 mt(chrono::high\_resolution\_clock::now().time\_since\_epoch().count());

**int** getRand() {

    return mt();

}

**void** readMain() {}

*// #define READ\_MAIN*

*// #define TESTCASES*

*// #define FILES*

*// #define TIME*

**void** solve(**int** **&**t) {

}

int32\_t main(**void**) {

    cin.tie(0)->sync\_with\_stdio(0);

**int** t = 1;

    clock\_t tt = clock();

    #ifdef FILES

        freopen("input.txt", "r", stdin);

        freopen("output.txt", "w", stdout);

    #endif

    #ifdef READ\_MAIN

        readMain();

    #endif

    #ifdef TESTCASES

        cin >> t;

    #endif

    for (**int** i = 1; i <= t; i++) {

        solve(i);

        cout << "\n";

    }

    #ifdef TIME

        cout << setprecision(3) << fixed << (ld)(clock() - tt) / CLOCKS\_PER\_SEC;

    #endif

    return 0;

}

Data structures

Disjoint set union

**struct** DSU {

    vector<**int**> par;

    vector<**int**> sz;

    DSU(**int** n) {

        FOR(i, 0, n) {

            par.pb(i);

            sz.pb(1);

        }

    }

**int** find(**int** a) {

        return par[a] = par[a] == a ? a : find(par[a]);

    }

**bool** same(**int** a, **int** b) {

        return find(a) == find(b);

    }

**void** unite(**int** a, **int** b) {

        a = find(a);

        b = find(b);

        if(sz[a] > sz[b]) swap(a, b);

        sz[b] += sz[a];

        par[a] = b;

    }

};

Segment tree

**struct** SegmentTree {

**int** n;

    vector<ll> t;

**const** ll IDENTITY = 0; *// OO for min, -OO for max, ...*

    ll f(ll a, ll b) {

        return a+b;

    }

    SegmentTree(**int** \_n) {

        n = \_n; t = vector<ll>(4\*n, IDENTITY);

    }

    SegmentTree(vector<ll>**&** arr) {

        n = arr.size(); t = vector<ll>(4\*n, IDENTITY);

        build(arr, 1, 0, n-1);

    }

**void** build(vector<ll>**&** arr, **int** v, **int** tl, **int** tr) {

        if(tl == tr) { t[v] = arr[tl]; }

        else {

**int** tm = (tl+tr)/2;

            build(arr, 2\*v, tl, tm);

            build(arr, 2\*v+1, tm+1, tr);

            t[v] = f(t[2\*v], t[2\*v+1]);

        }

    }

*// sum(1, 0, n-1, l, r)*

    ll sum(**int** v, **int** tl, **int** tr, **int** l, **int** r) {

        if(l > r) return IDENTITY;

        if (l == tl && r == tr) return t[v];

**int** tm = (tl+tr)/2;

        return f(sum(2\*v, tl, tm, l, min(r, tm)), sum(2\*v+1, tm+1, tr, max(l, tm+1), r));

    }

*// update(1, 0, n-1, i, v)*

**void** update(**int** v, **int** tl, **int** tr, **int** pos, ll newVal) {

        if(tl == tr) { t[v] = newVal; }

        else {

**int** tm = (tl+tr)/2;

            if(pos <= tm) update(2\*v, tl, tm, pos, newVal);

            else update(2\*v+1, tm+1, tr, pos, newVal);

            t[v] = f(t[2\*v],t[2\*v+1]);

        }

    }

};

Trie

**struct** Trie {

**const** **int** ALPHA = 26;

**const** **char** BASE = 'a';

    vector<vector<**int**>> nextNode;

    vector<**int**> mark;

**int** nodeCount;

    Trie() {

        nextNode = vector<vector<**int**>>(MAXN, vector<**int**>(ALPHA, -1));

        mark = vector<**int**>(MAXN, -1);

        nodeCount = 1;

    }

**void** insert(**const** string**&** s, **int** id) {

**int** curr = 0;

        FOR(i, 0, (**int**)s.length()) {

**int** c = s[i] - BASE;

            if(nextNode[curr][c] == -1) {

                nextNode[curr][c] = nodeCount++;

            }

            curr = nextNode[curr][c];

        }

        mark[curr] = id;

    }

**bool** exists(**const** string**&** s) {

**int** curr = 0;

        FOR(i, 0, (**int**)s.length()) {

**int** c = s[i] - BASE;

            if(nextNode[curr][c] == -1) return false;

            curr = nextNode[curr][c];

        }

        return mark[curr] != -1;

    }

};

Fenwick tree

**struct** Fenwick {

    vector<ll> tree;

**int** n;

    Fenwick(){}

    Fenwick(**int** \_n) {

        n = \_n;

        tree = vector<ll>(n+1, 0);

    }

**void** add(**int** i, ll val) { *// arr[i] += val*

        for(; i <= n; i += i&(-i)) tree[i] += val;

    }

    ll get(**int** i) { *// arr[i]*

        return sum(i, i);

    }

    ll sum(**int** i) { *// arr[1]+...+arr[i]*

        ll ans = 0;

        for(; i > 0; i -= i&(-i)) ans += tree[i];

        return ans;

    }

    ll sum(**int** l, **int** r) {*// arr[l]+...+arr[r]*

        return sum(r) - sum(l-1);

    }

};

Bellman Ford

**struct** Edge

{

**int** a, b, cost;

};

**int** n, m, v; *// v - starting vertex*

vector<Edge> e;

*/\* Finds SSSP with negative edge weights.*

*\* Possible optimization: check if anything changed in a relaxation step. If not - you can break early.*

*\* To find a negative cycle: perform one more relaxation step. If anything changes - a negative cycle exists.*

*\*/*

**void** solve() {

    vector<**int**> d (n, oo);

    d[v] = 0;

    for (**int** i=0; i<n-1; ++i)

        for (**int** j=0; j<m; ++j)

            if (d[e[j].a] < oo)

                d[e[j].b] = min (d[e[j].b], d[e[j].a] + e[j].cost);

*// display d, for example, on the screen*

}

Dijkstra

vector<vector<pair<**int**, **int**>>> adj;

**void** dijkstra(**int** s, vector<**int**> **&** d, vector<**int**> **&** p) {

**int** n = adj.size();

    d.assign(n, oo);

    p.assign(n, -1);

    d[s] = 0;

    min\_heap<pii> q;

    q.push({0, s});

    while (!q.empty()) {

**int** v = q.top().second;

**int** d\_v = q.top().first;

        q.pop();

        if (d\_v != d[v]) continue;

        for (**auto** edge : adj[v]) {

**int** to = edge.first;

**int** len = edge.second;

            if (d[v] + len < d[to]) {

                d[to] = d[v] + len;

                p[to] = v;

                q.push({d[to], to});

            }

        }

    }

}

Pick’s theorem

We are given a lattice polygon with non-zero area. Let's denote its area by S, the number of points with integer coordinates

lying strictly inside the polygon by I and the number of points lying on the sides of the polygon by B. Then:

S = I + (B / 2) – 1

Sieve

**void** eratosfen(**int** n) {

    vector<**bool**> sieve(n + 1, true);

    sieve[0] = sieve[1] = false;

    for (**int** i = 2; i \* i; i <= n) {

        if (sieve[i]) {

            for (**int** j = i \* i; j <= n; j += i) {

                sieve[j] = false;

            }

        }

    }

}

Euler totient function

**int** phi(**int** n) {

**int** result = n;

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

        if (n % i == 0) {

            while (n % i == 0)

                n /= i;

            result -= result / i;

        }

    }

    if (n > 1)

        result -= result / n;

    return result;

}

Big Integer

**struct** Int{

    string s;

    Int() : s("0") {}

    Int(string s) : s(s) {}

**bool** operator<(**const** Int**&** x){

        if(s.size() != x.s.size()) return s.size() < x.s.size();

        return s < x.s;

    }

    string stripZeros(string s){

        for(**int** i = 0; i < size(s); i++)

            if(s[i] != '0'){

                return s.substr(i);

            }

        return "0";

    }

    Int operator+(**const** Int **&**obj){

        string a = s, b = obj.s;

**int** n = size(a), m = size(b);

        reverse(aint(a));

        reverse(aint(b));

        if(n < m) a += string(m - n, '0'), n = m;

        else b += string(n - m, '0'), m = n;

        string c = "";

**int** dilda = 0;

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

**int** num = (dilda + a[i] - '0' + b[i] - '0');

            dilda = num / 10;

            num %= 10;

            c += **char**(num + '0');

        }

        if(dilda) c += **char**(dilda + '0');

        reverse(aint(c));

        return Int(stripZeros(c));

    }

    Int operator-(**const** Int**&** obj){

        string a = s, b = obj.s;

        if(a == b) return Int("0");

        reverse(aint(a));

        reverse(aint(b));

        string c = "";

**int** n = a.size(), m = b.size();

        if(n < m) a += string(m - n, '0'), n = m;

        else b += string(n - m, '0'), m = n;

**int** dilda = 0;

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

**int** num = dilda + a[i] - b[i];

            if(num < 0){

                dilda = -1;

                num += 10;

            }else dilda = 0;

            c += **char**(num + '0');

        }

        reverse(aint(c));

        return Int(stripZeros(c));

    }

    Int operator\*(**const** Int **&**obj){

        string a = s, b = obj.s;

        reverse(aint(a));

        reverse(aint(b));

**int** n = size(a), m = size(b);

        vector<**int**>ans(n + m, 0);

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

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

                ans[i + j] += (a[i] - '0') \* (b[j] - '0');

            }

        }

        string c = "";

**int** dilda = 0;

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

**int** num = (dilda + ans[i]);

            c += **char**(num % 10 + '0');

            dilda = num / 10;

        }

        if(dilda) c += dilda;

        reverse(aint(c));

        return Int(stripZeros(c));

    }

    Int operator/(**const** **int** **&**n){

        s = stripZeros(s);

        string ans = "";

**int** num = 0;

        for(**int** i = 0; i < size(s); i++){

            num = 10 \* num + s[i] - '0';

            ans += **char**(num / n + '0');

            num %= n;

        }

        return Int(stripZeros(ans));

    }

    Int operator^(**int** n){

        Int a(s);

        Int c("1");

        while(n){

            if(n & 1) c = c \* a;

            n >>= 1;

            a = a \* a;

        }

        return c;

    }

**friend** istream**&** operator>> (istream**&** out, Int**&** obj);

**friend** ostream**&** operator<< (ostream**&** out, **const** Int**&** obj);

};

istream**&** operator>> (istream**&** out, Int**&** obj){

    out >> obj.s;

    return out;

}

ostream**&** operator<< (ostream**&** out, **const** Int**&** obj){

    out << obj.s;

    return out;

}

Fast power

**int** binpow(**int** a, **int** n, **int** mod = 1e9 + 7) {

    if (!a) {

        return 0;

    }

    if (!n) {

        return 1;

    }

**int** res = 1;

    while (n) {

        if (n & 1) res = res \* a % mod;

        n >>= 1;

        a = a \* a % mod;

    }

    return res;

}

Z function

**void** z\_function(string **&**s) {

**int** n = s.size();

  vector<**int**> z(n);

**int** l = 0, r = 0;

  for (**int** i = 1; i < n; i++) {

    if (i < r) {

      z[i] = min(z[i - l], r - i);

    }

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

      z[i]++;

    }

    if (i + z[i] > r) {

      l = i;

      r = i + z[i];

    }

  }

}

Prefix function

**void** prefix\_function(string **&**s) {

**int** n = s.size();

  vector<**int**> pi(n);

  for (**int** i = 1; i < n; i++) {

**int** j = pi[i - 1];

    while (j > 0 && s[i] != s[j]) {

      j = pi[j - 1];

    }

    if (s[i] == s[j]) {

      j++;

    }

    pi[i] = j;

  }

}

Hashing

**struct** HashedString {

**const** ll A1 = 999999929, B1 = 1000000009, A2 = 1000000087, B2 = 1000000097;

    vector<ll> A1pwrs, A2pwrs;

    vector<pll> prefixHash;

    HashedString(**const** string**&** \_s) {

        init(\_s);

        calcHashes(\_s);

    }

**void** init(**const** string**&** s) {

        ll a1 = 1;

        ll a2 = 1;

        FOR(i, 0, (**int**)s.length()+1) {

            A1pwrs.pb(a1);

            A2pwrs.pb(a2);

            a1 = (a1\*A1)%B1;

            a2 = (a2\*A2)%B2;

        }

    }

**void** calcHashes(**const** string**&** s) {

        pll h = {0, 0};

        prefixHash.pb(h);

        for(**char** c : s) {

            ll h1 = (prefixHash.back().first\*A1 + c)%B1;

            ll h2 = (prefixHash.back().second\*A2 + c)%B2;

            prefixHash.pb({h1, h2});

        }

    }

    pll getHash(**int** l, **int** r) {

        ll h1 = (prefixHash[r+1].first - prefixHash[l].first\*A1pwrs[r+1-l]) % B1;

        ll h2 = (prefixHash[r+1].second - prefixHash[l].second\*A2pwrs[r+1-l]) % B2;

        if(h1 < 0) h1 += B1;

        if(h2 < 0) h2 += B2;

        return {h1, h2};

    }

};

Ternary search

**double** ternary\_search(**double** l, **double** r) {

    while (r - l > eps) {

**double** m1 = l + (r - l) / 3;

**double** m2 = r - (r - l) / 3;

**double** f1 = f(m1);

**double** f2 = f(m2);

        if (f1 < f2)

            l = m1;

        else

            r = m2;

    }

    return f(l); *//return the maximum of f(x) in [l, r]*

}