subjects

1 strings

1.1 Z

1.1.1 z copy.cpp

1.1.2 z.cpp

1.2 aho_corasick

1.2.1 aho_corasick.cpp

1.2.2 monteiros_aho_corasick.cpp

1.3 hash

1.3.1 hash_with_update.cpp

1.3.2 hash.cpp

1.3.3 hash_substring.cpp

1.3.4 exercises

1.3.4.1 minimal_rotation.cpp

1.3.4.2 size.cpp

1.3.4.3 repeating_substring.cpp

1.3.4.4 run

1.3.4.5 a.cpp

1.4 KMP

1.4.1 lps.cpp

1.4.2 KMP.cpp

2 algebra

2.1 lcm

2.1.1 lcm.cpp

2.2 gcd

2.2.1 euclidian.cpp

2.2.2 extended_euclidian.cpp

2.3 primes_divisibility

2.3.1 n_factors.cpp

2.3.2 seive.cpp

2.4 modular_algebra

2.4.1 fermats.cpp

2.5 binary_exp

2.5.1 binary_exp.cpp

2.5.2 matrix_binary_exp.cpp

3 DP

3.1 removal_game.cpp

3.2 increasing_subsequence.cpp

3.3 minimizing_coins.cpp

3.4 rectangle_cutting.cpp

3.5 project.cpp

3.6 run

3.7 counting_towers.cpp

3.8 test.cpp

3.9 two_sets_II.cpp

3.10 minimizing_the_sum.cpp

3.11 money_sums.cpp

3.12 edit_distance.cpp

3.13 dice_combination.cpp

3.14 cp_algorithm_stuff

3.14.1 knapsack.cpp

4 game_theory

4.1 Sprague–Grundy-theorem

4.1.1 Grundy_theorem.cpp

5 data_structures

5.1 bit

5.1.1 bit_all_again.cpp

5.1.2 bit.cpp

5.2 sparce_table

5.2.1 next_smaller_element.cpp

5.2.2 next_smaller_element copy.cpp

5.2.3 sparce_table.cpp

5.3 trie

5.3.1 trie.cpp

5.4 segtree

5.4.1 regular.cpp

5.4.2 range_update.cpp

5.5 orderset

5.5.1 example.cpp

6 graph

6.1 run

6.2 counting_rooms.cpp

6.3 bridges

6.3.1 bridge_tree.cpp

6.3.2 bridges.cpp

6.4 strongly_connected

6.4.1 condensed_tree.cpp

6.4.2 kosaraju.cpp

6.5 DP_in_tree

6.5.1 tree_with_small_distance.cpp

7 geometry

7.1 manhattan_distance

7.1.1 biggest_distance_between_points.cpp

7.1.2 chebyschev.cpp

7.2 lattice_points

7.2.1 Shoelance_Theorem.cpp

7.2.2 Pick's_Theorems.cpp

7.3 cses

7.3.1 LineSegmentIntersection.cpp

7.3.2 PointLocationTest.cpp

7.4 basics

7.4.1 dot_product.cpp

7.4.2 Applications.cpp

7.4.3 points.cpp

7.4.4 cross_product.cpp

7.4.5 orientation_of_3_ordered_points.cpp

subjects

8 strings

8.1 Z

8.1.1 z copy.cpp

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

```
using namespace std;
```

```
vector<int> genZarray(string &s )
```

{

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

```
vector<int> zarray(n, 0);
```

```
int l = 0, r = 0;
```

```
int i = 0;
```

```
while(i++, i < n)
```

{

```
        // cout << l << ' ' << r << endl;
```

```
if(i <= r)
```

{

```
int k = i-1;
```

```
if(zarray[k]+i <= r)
```

{

```
zarray[i] = zarray[k];
```

```
                continue;
```

```
}
```

```
l = i;
```

```
cout << s[i]  << ' '  << i << endl << zarray[k]+i << ' '
```

```
while(r + 1 < n && s[r+1] == s[r-l+1])
```

```
r++;
```

```
            zarray[i] = r-l +1;
```

```
continue;
```

```
}
```

```
l   = i;
```

```
r = i -1;
```

```
while( r + 1 < n && s[r+1] == s[r-l+1])
```

{

```
zarray[i]++;
```

```
r++;
```

```
}
```

```
}
```

```
    cout << l << ' ' << r << endl;
```

```
return zarray;
```

```
}
```

```
signed main()
```

{

```
string s = "abcabcaaaabc";
```

```
auto vec = genZarray(s);
```

```
for(auto a : vec)
```

```
cout << a << " ";
```

```
cout << '\n';
```

```
    return 0;
```

```
}
```

8.1.2 z.cpp

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

```
using namespace std;
```

```
#define int long long
```

```
int zarray[10000000];
```

```
void genZarray(string &s)
```

{

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

```
int l = 0, r = 0;
```

```
int i = 0;
```

```
while(i++, i < n)
```

{

```
// cout << i ;
```

```
        // cout << l << ' ' << r << endl;
```

```
if(i <= r)
```

{

```
int k = i-1;
```

```
if(zarray[k]+i <= r)
```

{

```
zarray[i] = zarray[k];
```

```
                continue;
```

```
}
```

```
l = i;
```

```
while(r + 1 < n && s[r+1] == s[r-l+1])
```

```
r++;
```

```
            zarray[i] = r-l +1;
```

```
continue;
```

```
}
```

```
l   = i;
```

```
r = i -1;
```

```
while( r + 1 < n && s[r+1] == s[r-l+1])
```

{

```
zarray[i]++;
```

```
r++;
```

```
}
```

```
}
```

```
}
```

```
signed main()
```

{

```
string source, pattern;
```

```
cin >> source >> pattern;
```

```
string f = pattern + "$" + source;
```

```
memset(zarray, 0, sizeof(int) * f.size());
```

```
genZarray(f);
```

```
int n = 0;
```

```
// cout  << f << endl;
```

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

{

```
// cout << zarray[i] << ' ' ;
```

```
if(zarray[i] == pattern.size())
```

```
        n++;
```

```
}
```

```
cout << n << endl;
```

```
    return 0;
```

```
}
```

8.2 aho_corasick

8.2.1 aho_corasick.cpp

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

```
using namespace std;
```

```
const int K=26;
```

```
struct Node
```

{

```
int next_node[K];
```

```
int go_be[K];
```

```
bool leaf = false;
```

```
int link = -1;
```

```
int parent;
```

```
char c;
```

```
Node(int _parent, char _c) : parent(_parent), c(_c)
```

{

```
fill(begin(next_node), end(next_node), -1);
```

```
fill(begin(go_be), end(go_be), -1);
```

```
}
```

```
};
```

```
vector<Node> trie;
```

```
void init()
```

{

```
trie.clear();
```

```
trie.resize(1);
```

```
}
```

```
int add(const string & s)
```

{

```
int i = 0;
```

```
for(char c : s)
```

{

```
int pos = c -'a';
```

```
if(trie[i].next_node[pos] == -1)
```

{

```
trie[i].next_node[pos] = trie.size();
```

```
            trie.push_back(Node(i, c));
```

```
}
```

```
        i = trie[i].next_node[pos];
```

```
}
```

```
trie[i].leaf = true;
```

```
}
```

```
int go(int i, int c);
```

```
int get_link(int i)
```

{

```
if(trie[i].link == -1)
```

{

```
if(i == 0 || trie[i].parent == 0)
```

```
        trie[i].link = 0;
```

```
else
```

```
            trie[i].link = go(get_link(trie[i].parent), trie[i].c);
```

```
}
```

```
    return trie[i].link;
```

```
}
```

```
int go(int i, char c)
```

{

```
int pos = c - 'a';
```

```
if(trie[i].go_be[pos] == -1)
```

{

```
if(trie[i].next_node[pos] != -1)
```

```
trie[i].go_be[pos] = trie[i].next_node[pos];
```

```
else
```

```
            trie[i].go_be[pos] = i == 0 ? 0 : go(get_link(i), c);
```

```
}
```

```
    return trie[i].next_node[c];
```

```
}
```

```
bool match(const string & s)
```

{

```
}
```

```
int main()
```

{

```
    return 0;
```

```
}
```

8.2.2 monteiros_aho_corasick.cpp

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

```
using namespace std;
```

```
const int MAX = 1e4;
```

```
int p[MAX], f[MAX], nxt[MAX][26], ch[MAX];
```

```
int tsz = 1; // size of the trie
```

```
int cnt[MAX]; // used to know number of matches
```

```
const int S = 2e3+5;
```

```
bitset<MAX> elem[S];
```

```
// S eh tamanho da maior das N strings que sao
```

```
// pradroes para buscar no texto
```

```
void init()
```

{

```
tsz = 1;
```

```
memset(f, 0, sizeof(f));
```

```
memset(nxt, 0, sizeof(nxt));
```

```
memset(cnt, 0, sizeof(cnt));
```

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

```
elem[i].reset();
```

```
}
```

```
void add(const string &s, int x)
```

{

```
// the first element of the trie is the root
```

```
int cur = 1;
```

```
for(int i = 0; s[i]; ++i)
```

{

```
int j = s[i] - 'a';
```

```
if(!nxt[cur][j])
```

{

```
tsz++;
```

```
p[tsz] = cur;
```

```
ch[tsz] = j;
```

```
            nxt[cur][j] = tsz;
```

```
}
```

```
cur = nxt[cur][j];
```

```
}
```

```
cnt[cur]++;
```

```
elem[cur].set(x);
```

```
}
```

```
void build()
```

{

```
queue<int> q;
```

```
for(int i = 0; i < 26; ++i)
```

{

```
nxt[0][i] = 1;
```

```
if(nxt[1][i])
```

```
            q.push(nxt[1][i]);
```

```
}
```

```
while(!q.empty())
```

{

```
int v = q.front(); q.pop();
```

```
int u = f[p[v]];
```

```
while(u and !nxt[u][ch[v]]) u = f[u];
```

```
f[v] = nxt[u][ch[v]];
```

```
cnt[v] += cnt[f[v]];
```

```
for(int i = 0; i < 26; ++i)
```

```
if(nxt[v][i])
```

```
                q.push(nxt[v][i]);
```

```
}
```

```
}
```

```
bitset<MAX> match(const string &s)
```

{

```
int ans = 0;
```

```
// Numero de matches
```

```
bitset<MAX> found;
```

```
// Usado pra saber quais strings matches
```

```
int x = 1;
```

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

{

```
int t = s[i] - 'a';
```

```
while(x and !nxt[x][t])
```

```
x = f[x];
```

```
x = nxt[x][t];
```

```
        ans += cnt[x];
```

```
found |= elem[x];
```

```
}
```

```
return found;
```

```
}
```

```
int main()
```

{

```
int n;
```

```
string s;
```

```
cin >> n;
```

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

{

```
cin >> s;
```

```
add(s, i);
```

```
}
```

```
build();
```

```
cin >> s;
```

```
bitset<MAX> ans = match(s);
```

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

```
        cout << ans[i] << '\n';
```

```
// 1 se a i-esima string lida
```

```
// aparece no texto, 0 cc
```

```
return 0;
```

```
}
```

```
//#############################################################################
```

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

```
using namespace std;
```

```
const int K = 60;
```

```
struct Vertex {
```

```
int next[K];
```

```
bool leaf = false;
```

```
int p = -1;
```

```
char c;
```

```
int link = -1;
```

```
int go[K];
```

```
bitset<1005> S;
```

```
Vertex(int _p=-1, char _c = '$') : p(_p), c(_c) {
```

```
fill(begin(next), end(next), -1);
```

```
fill(begin(go), end(go), -1);
```

```
}
```

```
};
```

```
vector<Vertex> t;
```

```
void init() {
```

```
t.clear();
```

```
t.resize(1);
```

```
}
```

```
void add(string &s, int i) {
```

```
int v = 0;
```

```
for(char ch : s) {
```

```
int c = ch - 'A';
```

```
if(t[v].next[c] == -1) {
```

```
t[v].next[c] =  t.size();
```

```
t.push_back(Vertex(v, ch));
```

```
}
```

```
v = t[v].next[c];
```

```
}
```

```
t[v].leaf = true;
```

```
t[v].S[i] = 1;
```

```
}
```

```
int go(int v, char ch);
```

```
int get_link(int v) {
```

```
if(t[v].link == -1) {
```

```
if(v == 0 or t[v].p == 0)
```

```
t[v].link = 0;
```

```
else
```

```
t[v].link = go(get_link(t[v].p), t[v].c);
```

```
}
```

```
return t[v].link;
```

```
}
```

```
int go(int v, char ch) {
```

```
int c = ch - 'A';
```

```
if(t[v].go[c] == -1) {
```

```
if(t[v].next[c] != -1)
```

```
t[v].go[c] = t[v].next[c];
```

```
else
```

```
t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
```

```
}
```

```
return t[v].go[c];
```

```
}
```

```
int32_t main() {
```

```
ios_base::sync_with_stdio(false);
```

```
    cin.tie(nullptr);
```

```
int caso;
```

```
cin >> caso;
```

```
while(caso--) {
```

```
init();
```

```
string s;
```

```
int n;
```

```
cin >> s >> n;
```

```
bitset<1005> S;
```

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

```
string a;
```

```
cin >> a;
```

```
add(a, i);
```

```
}
```

```
int v = 0;
```

```
for(char &c : s) {
```

```
v = go(v, c);
```

```
S |= t[v].S;
```

```
}
```

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

```
cout << (S[i] ? 'y' : 'n') << '\n';
```

```
}
```

```
return 0;
```

```
}
```

8.3 hash

8.3.1 hash_with_update.cpp

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

```
using namespace std;
```

```
#define int unsigned long int
```

```
const int MOD = 1e9 + 9;
```

```
const int P = 31;
```

```
int p_pow[(int)1e6 + 1];
```

```
int h_val[(int)1e6 + 1];
```

```
int BITree[(int)1e6 + 2];
```

```
int get_int(char c)
```

{

```
    return c - 'a' + 1;
```

```
}
```

```
void update_bit(int index, int val, int n)
```

{

```
h_val[index] += val;
```

```
index++;
```

```
for(; index <= n; index += index & -index)
```

```
BITree[index] = (BITree[index] + val) % MOD;
```

```
}
```

```
int get_bit_range(int index)
```

{

```
index++;
```

```
int sum = 0;
```

```
for(;index > 0; index -= index & -index)
```

```
sum = (sum +  BITree[index]) % MOD;
```

```
    return sum;
```

```
}
```

```
void init_bit(int n)
```

{

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

```
BITree[i] = 0;
```

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

```
update_bit(i, h_val[i], n);
```

```
}
```

```
void hashing(const string & s)
```

{

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

```
p_pow[0] = 1;
```

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

```
p_pow[i] = (p_pow[i-1] * P) % MOD;
```

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

```
h_val[i] = (get_int(s[i]) * p_pow[i]) % MOD;
```

```
    init_bit(s.size());
```

```
// cout << "aqui_ner" << endl;
```

```
}
```

```
int get_hash(const string  &s)
```

{

```
return get_bit_range(s.size()-1);
```

```
}
```

```
void switch_char(int pos, char a, string&  s)
```

{

```
int c = a - s[pos];
```

```
int delta = (c * p_pow[pos]) % MOD;
```

```
if(delta < 0)
```

```
        delta += MOD;
```

```
// cout << a << ' ' << s[pos] << ' ' << p_pow[pos] << ' ' <<  <<
```

```
s[pos] = a;
```

```
    update_bit(pos, delta, s.size());
```

```
}
```

```
signed main()
```

{

```
string s = "abacate";
```

```
hashing(s);
```

```
cout << get_hash(s) << endl;
```

```
switch_char(2, 'b', s); // abbcate
```

```
    cout << s << ' ' << get_hash(s) << endl;
```

```
string s1 = "abbcate";
```

```
hashing(s1);
```

```
cout << s1 << ' ' <<get_hash(s1) << endl;
```

```
    return 0;
```

```
}
```

8.3.2 hash.cpp

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

```
using namespace std;
```

```
#define int long long
```

```
struct StringHash {
```

```
int n;
```

```
string s;
```

```
long long p = 1238473;
```

```
long long mod = (1ll << 61) - 1;
```

```
vector<long long> h, pot;
```

```
long long mulmod(long long a, long long b) {
```

```
long long q = (long long)((long double)a*b/mod);
```

```
long long r = a * b - mod * q;
```

```
while(r < 0) r += mod;
```

```
while(r >= mod) r -= mod;
```

```
    return r;
```

```
    // return (a * (__int128)1 * b) % mod;
```

```
}
```

```
long long operator()(int l, int r) {
```

```
if(!l) return h[r];
```

```
long long hash_val = (h[r] - mulmod(h[l - 1], pot[r - l + 1]
```

```
if(hash_val < 0) hash_val += mod;
```

```
    return hash_val;
```

```
}
```

```
void build_hash() {
```

```
h[0] = s[0];
```

```
pot[0] = 1;
```

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

```
h[i] = (mulmod(h[i - 1], p) + s[i]);
```

```
h[i] -= (h[i] >= mod ? mod : 0);
```

```
        pot[i] = mulmod(pot[i - 1], p);
```

```
}
```

```
}
```

```
StringHash(string &_s) : n((int)_s.size()), s(_s) {
```

```
h.resize(n);
```

```
pot.resize(n);
```

```
build_hash();
```

```
}
```

```
};
```

```
signed main()
```

{

```
string s1, s2; cin >> s1 >> s2;
```

```
cout << (StringHash(s1)(0, s1.size()-1) == StringHash(s2)(0, s2.
```

```
    return 0;
```

```
}
```

8.3.3 hash_substring.cpp

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

```
using namespace std;
```

```
#define int long long
```

```
int g_values[123];
```

```
int p = 31;
```

```
int m = 1e9 + 9;
```

```
vector<int> hashes;
```

```
vector<int> p_pows;
```

```
int hashing(string &s)
```

{

```
p_pows.resize(s.size() +1);
```

```
hashes.resize(s.size() +1);
```

```
p_pows[0] = 1;
```

```
int h = 0;
```

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

{

```
hashes[i] =  h = (h + (s[i]*p_pows[i]) % m) % m;
```

```
p_pows[i+1] = (p_pows[i] * p) % m;
```

```
}
```

```
    return h;
```

```
}
```

```
int subh(int i , int r)
```

{

```
int tot = hashes[r];
```

```
// cout << "pass" << endl;
```

```
if(i > 0)
```

```
tot-=hashes[i-1];
```

```
tot *= p_pows[p_pows.size() - 1 - i];
```

```
    return tot % m;
```

```
}
```

```
signed main()
```

{

```
string s; cin >> s;
```

```
for(char c = 'a'; c <='z';c++)
```

{

```
char aux ; cin >>aux;
```

```
g_values[c] = aux == '1' ? 0 : 1;
```

```
}
```

```
hashing(s);
```

```
vector<int> psum(s.size());
```

```
int last = 0;
```

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

{

```
psum[i] = last + g_values[s[i]];
```

```
last = psum[i];
```

```
}
```

```
int k; cin >> k;
```

```
// cout << "pass" << endl;
```

```
set<int> se;
```

```
last = 0;
```

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

{

```
for(int r = i; r < psum.size(); r++)
```

{

```
// cout << i << ' ' << r << endl;
```

```
if(psum[r] - last <= k)
```

{

```
if(se.find(subh(i, r)) == se.end())
```

{

```
                cout << s.substr(i, r-i + 1) << ' ' << subh(i, r
```

```
se.insert(subh(i, r));
```

}

```
}
```

```
}
```

```
last = psum[i];
```

```
}
```

```
    cout << se.size() << endl;
```

```
    return 0;
```

```
}
```

8.3.4 exercises

8.3.4.1 minimal_rotation.cpp

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

```
using namespace std;
```

```
struct StringHash {
```

```
int n;
```

```
string s;
```

```
long long p = 1238473;
```

```
long long mod = (1ll << 61) - 1;
```

```
vector<long long> h, pot;
```

```
long long mulmod(long long a, long long b) {
```

```
long long q = (long long)((long double)a*b/mod);
```

```
long long r = a * b - mod * q;
```

```
while(r < 0) r += mod;
```

```
while(r >= mod) r -= mod;
```

```
    return r;
```

```
    // return (a * (__int128)1 * b) % mod;
```

```
}
```

```
long long operator()(int l, int r) {
```

```
if(!l) return h[r];
```

```
long long hash_val = (h[r] - mulmod(h[l - 1], pot[r - l + 1]
```

```
if(hash_val < 0) hash_val += mod;
```

```
    return hash_val;
```

```
}
```

```
void build_hash() {
```

```
h[0] = s[0];
```

```
pot[0] = 1;
```

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

```
h[i] = (mulmod(h[i - 1], p) + s[i]);
```

```
h[i] -= (h[i] >= mod ? mod : 0);
```

```
        pot[i] = mulmod(pot[i - 1], p);
```

```
}
```

```
}
```

```
StringHash(string &_s) : n((int)_s.size()), s(_s) {
```

```
h.resize(n);
```

```
pot.resize(n);
```

```
build_hash();
```

```
}
```

```
};
```

```
bool less_than(int i, int best,  StringHash & h, const string & s)
```

{

```
int n = s.size()/2;
```

```
int l = 0, r = n -1;
```

```
while(l<r)
```

{

```
if(l == n-1)
```

```
return false;
```

```
int mid = (l+r)/2;
```

```
if(h(i, i+mid) == h(best, best+mid))
```

{

```
if(s[i+mid+1] != s[best+mid+1])
```

```
return s[i+mid+1] < s[best+mid+1];
```

```
l = mid+1;
```

```
}
```

```
else
```

{

```
            // cout << l << ' ' << r << ' ' << mid << endl;
```

```
            r = mid;
```

```
}
```

```
}
```

```
// cout << l << ' ' << r <<endl;;
```

```
if(s[i+l] == s[best+l])
```

```
        return s[i+l+1] < s[best+l+1];
```

```
    return s[i+l] < s[best+l];
```

```
}
```

```
int solve()
```

{

```
string s; cin >> s;
```

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

```
s += s;
```

```
StringHash h(s);
```

```
int best_window = 0;
```

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

{

```
if(less_than(i, best_window, h, s))
```

{

```
// cout << best_window << ' ' << i << endl;
```

```
best_window = i;
```

```
}
```

```
}
```

```
    cout << s.substr(best_window,n);
```

```
cout << endl;
```

```
    return 0;
```

```
}
```

```
signed main()
```

{

```
    solve();
```

```
    return 0;
```

```
}
```

8.3.4.2 size.cpp

```
#include <iostream>
```

```
int main()
```

{

```
std::string a; std::cin >> a;
```

```
std::cout << a.size() << std::endl;
```

```
}
```

8.3.4.3 repeating_substring.cpp

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

```
using namespace std;
```

```
struct StringHash {
```

```
int n;
```

```
string s;
```

```
long long p = 1238473;
```

```
long long mod = (1ll << 61) - 1;
```

```
vector<long long> h, pot;
```

```
long long mulmod(long long a, long long b) {
```

```
long long q = (long long)((long double)a*b/mod);
```

```
long long r = a * b - mod * q;
```

```
while(r < 0) r += mod;
```

```
while(r >= mod) r -= mod;
```

```
    return r;
```

```
    // return (a * (__int128)1 * b) % mod;
```

```
}
```

```
long long operator()(int l, int r) {
```

```
if(!l) return h[r];
```

```
long long hash_val = (h[r] - mulmod(h[l - 1], pot[r - l + 1]
```

```
if(hash_val < 0) hash_val += mod;
```

```
    return hash_val;
```

```
}
```

```
void build_hash() {
```

```
h[0] = s[0];
```

```
pot[0] = 1;
```

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

```
h[i] = (mulmod(h[i - 1], p) + s[i]);
```

```
h[i] -= (h[i] >= mod ? mod : 0);
```

```
        pot[i] = mulmod(pot[i - 1], p);
```

```
}
```

```
}
```

```
StringHash(string &_s) : n((int)_s.size()), s(_s) {
```

```
h.resize(n);
```

```
pot.resize(n);
```

```
build_hash();
```

```
}
```

```
};
```

```
string bb(StringHash & h, const string & s )
```

{

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

```
int l= 1, r = n-1;
```

```
int m_size = 0;
```

```
int pos = 0;
```

```
while(l <  r)
```

{

```
int mid = (l+r)/2;
```

```
cout << l << ' ' << mid << ' ' << r << endl;
```

```
set<int> ocurrences;
```

```
bool go_down = true;
```

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

{

```
int h_v = h(i, i+mid -1);
```

```
if(ocurrences.find(h_v) != ocurrences.end())
```

{

```
go_down = false;
```

```
pos = i;
```

```
m_size = mid;
```

```
l = mid +1;
```

```
                break;
```

```
}
```

```
ocurrences.insert(h_v);
```

```
}
```

```
if(go_down)
```

```
r=mid;
```

```
}
```

```
// cout << pos << m_size << endl;
```

```
if(m_size == 0)
```

```
return "-1";
```

```
return s.substr(pos, m_size);
```

```
}
```

```
void solve()
```

{

```
string s; cin >> s;
```

```
StringHash h(s);
```

```
cout << bb(h, s) << endl;
```

```
return;
```

```
}
```

```
signed main()
```

{

```
    solve();
```

```
    return 0;
```

```
}
```

8.3.4.4 run

```
g++ $1 && ./a.out < input.txt > output.txt
```

8.3.4.5 a.cpp

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

```
using namespace std;
```

```
#define int unsigned long long
```

```
// https://open.kattis.com/problems/hashing
```

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

```
using namespace std;
```

```
struct StringHashing{
```

```
const uint64_t MOD = (1LL<<61) - 1;
```

```
const int base = 31;
```

```
uint64_t modMul(uint64_t a, uint64_t b){
```

```
uint64_t l1 = (uint32_t)a, h1 = a>>32, l2 = (uint32_t)b, h2 = b>
```

```
uint64_t l = l1*l2, m = l1*h2 + l2*h1, h = h1*h2;
```

```
uint64_t ret = (l&MOD) + (l>>61) + (h << 3) + (m >> 29) + ((m <<
```

```
ret = (ret & MOD) + (ret>>61);
```

```
ret = (ret & MOD) + (ret>>61);
```

```
return ret-1;
```

```
}
```

```
int getInt(char c){
```

```
return c-'a'+1;
```

```
}
```

```
vector<uint64_t> hs, p;
```

```
//Public:
```

```
StringHashing(string s){
```

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

```
hs.resize(n); p.resize(n);
```

```
p[0] = 1;
```

```
hs[0] = getInt(s[0]);
```

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

```
p[i] = modMul(p[i-1], base);
```

```
hs[i] = (modMul(hs[i-1], base) + getInt(s[i]))%MOD;
```

```
}
```

```
}
```

```
uint64_t getValue(int l, int r){
```

```
if(l > r) return -1;
```

```
uint64_t res = hs[r];
```

```
if(l > 0) res = (res + MOD - modMul(p[r-l+1], hs[l-1]))%MOD;
```

```
return res;
```

```
}
```

```
};
```

```
int solve()
```

{

```
string s; cin >> s;
```

```
int t; cin >> t;
```

```
StringHashing h(s);
```

```
while(t--)
```

{

```
int l, r; cin >> l >> r;
```

```
        // cout << s.substr(l, r-l) << ' ';
```

```
        cout << h.getValue(l, r-1) << endl;
```

```
}
```

```
    return 0;
```

```
}
```

```
signed main()
```

{

```
    solve();
```

```
    return 0;
```

```
}
```

8.4 KMP

8.4.1 lps.cpp

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

```
using namespace std;
```

```
vector<int> gen_lps(const string & pat)
```

{

```
vector<int> lps(pat.size(), 0);
```

```
int len = 0, i = 1;
```

```
while(i<pat.size()){
```

```
        cout << len << ' ' << i << endl;
```

```
if(pat[i] == pat[len])
```

{

```
            lps[i] = ++len;
```

```
        i++;
```

```
}
```

```
else if(len> 0)
```

```
            len = lps[len-1];
```

```
else
```

```
        i++;
```

```
}
```

```
return lps;
```

```
}
```

```
int main()
```

{

```
string s; cin >>s;
```

```
    auto lps = gen_lps(s);
```

```
for(auto t : lps)
```

```
cout << t << " ";
```

```
cout << endl;
```

```
    return 0;
```

```
}
```

8.4.2 KMP.cpp

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

```
using namespace std;
```

```
void compute_lps_array(const string & s, int  arr[])
```

{

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

```
arr[0] = 0;
```

```
int i = 1;
```

```
int len = 0;
```

```
while(i < n)
```

{

```
if(s[len] == s[i])
```

{

```
        arr[i] = ++len;
```

```
        i++;
```

```
}
```

```
else if(len != 0)
```

```
len = arr[len-1];
```

```
else
```

```
        arr[i++] = 0;
```

```
}
```

```
}
```

```
int lps[10000000];
```

```
int KMP(const string & s, const string & pat, int lps[])
```

{

```
int occur = 0;
```

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

```
int m = pat.size();
```

```
int i = 0, j = 0;
```

```
while(!(m - j > n - i) )
```

{

```
if(s[i] == pat[j])
```

{

```
i++;j++;
```

```
if(j == m)
```

{

```
            occur++;
```

```
            j = lps[j-1];
```

```
}
```

```
}
```

```
else if(j!= 0)
```

```
            j = lps[j-1];
```

```
else
```

```
        i++;
```

```
}
```

```
return occur;
```

```
}
```

```
int main()
```

{

```
string source, pat;
```

```
cin >> source >> pat;
```

```
    compute_lps_array(pat, lps);
```

```
cout << KMP(source, pat, lps) << endl;
```

```
    return 0;
```

```
}
```

9 algebra

9.1 lcm

9.1.1 lcm.cpp

```
#include <iostream>
```

```
using namespace std;
```

```
int gcd(int a, int b)
```

{

```
if (a < b)
```

```
swap(a, b);
```

```
if(b == 0)
```

```
return a;
```

```
return gcd(b, a%b);
```

```
}
```

```
// To do LCM you just need to know that
```

//

```
//       a*b = lcm(a,b)*gcd(a,b)
```

//

```
// So
```

//

```
//       lcm(a,b) = a*b/gcd(a,b)
```

```
int lcm(int a,int b)
```

{

```
return a*b/gcd(a,b);
```

```
}
```

```
int main()
```

{

```
    return 0;
```

```
}
```

9.2 gcd

9.2.1 euclidian.cpp

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

```
#include <iostream>
```

```
using namespace std;
```

/*

It stated that gcd(a, b) = gcd(b, a%b), and that

```
gcd(a, 0) = a
```

So there is a recursive way to find gcd(a, b) in O(log n)

```
*/
```

```
int gcd(int a, int b)
```

{

```
if (a < b)
```

```
swap(a, b);
```

```
if(b == 0)
```

```
return a;
```

```
return gcd(b, a%b);
```

```
}
```

```
signed main()
```

{

```
    return 0;
```

```
}
```

9.2.2 extended_euclidian.cpp

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

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

```
using namespace std;
```

/*

Based on the euclidian theorem it is a way to find x and y, with

```
gcd(a,b) = x*a + y*b
```

It works in this way

So, at first let g = gcd(a, b)

we know that there is possible to do

```
g = x1 * b + y1 * a%b
```

expanding the concept of mod

$$g = x1 * b + y1 * (a - \lfloor a/b \rfloor * b)$$

$$\Rightarrow g = x1 * b + y1*a - y1 * \lfloor a/b \rfloor * b$$

$$\Rightarrow g = y1*a - y1 * \lfloor a/b \rfloor * b + x1 * b$$

```
=> g =  y1*a (- y1 * ⌊a/b⌋ + x1) * b
```

Switching g for x*a + y*b

$$x*a + y*b = y1*a\ (- y1 * \lfloor a/b \rfloor + x1) * b$$

Seeing like this we can segregate this equation in two

* x*a = y1*a

and

* y*b = (- y1 * ⌊a/b⌋ + x1) * b

So the code works based on this still in log time.

In top of that we just need to consider that when we find gcd(a,

x = 1 and y = 0 (y value doesnt matter really but lets put the m

```
*/
```

```
array<int, 3> extended_gcd(int a, int b)
```

{

```
if(b > a)
```

```
swap(a, b);
```

```
if(b == 0)
```

```
    return {a, 1, 0};
```

```
    auto [res, x1, y1] = extended_gcd(b, a % b);
```

```
    return {res, y1, x1 - a/b * y1};
```

```
}
```

```
int gcd(int a, int b)
```

{

```
if (a < b)
```

```
swap(a, b);
```

```
if(b == a)
```

```
return a;
```

```
return gcd(b, a%b);
```

```
}
```

```
signed main()
```

{

```
auto [res, x, y] = extended_gcd(100, 88);
```

```
cout << "a"<< res << ' ' << x << ' ' << y << endl;
```

```
    return 0;
```

```
}
```

9.3 primes_divisibility

9.3.1 n_factors.cpp

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

```
using namespace std;
```

```
#define MAX_N 100000
```

```
int factors[MAX_N];
```

```
void count_factors(int n = MAX_N)
```

{

```
memset(factors, 0, sizeof(int) * n);
```

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

{

```
for(int j = i; j < n; j+= i)
```

```
            factors[j]++;
```

```
}
```

```
}
```

```
signed main()
```

{

```
count_factors();
```

```
for(int i = 0;i < 100; i++)
```

```
        cout << factors[i] << ' ';
```

```
    return 0;
```

```
}
```

9.3.2 seive.cpp

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

```
using namespace std;
```

```
#define MAX_N 100000
```

```
int is_prime[MAX_N];
```

```
void fill_primes(int n = MAX_N)
```

{

```
memset(is_prime, 0, sizeof(int) * n);
```

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

{

```
if(is_prime[i])
```

```
continue;
```

```
for(int j = 2*i; j < n; j += i)
```

{

```
            is_prime[j] = 1;
```

```
}
```

```
}
```

```
}
```

```
// O(log(log n))
```

```
void optimal_fill_primes(int n = MAX_N)
```

{

```
memset(is_prime, 0, sizeof(int) * n);
```

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

{

```
if(is_prime[i])
```

```
continue;
```

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

{

```
            is_prime[j] = 1;
```

```
}
```

```
}
```

```
}
```

```
signed main()
```

{

```
optimal_fill_primes();
```

```
for(int i = 0;i < 100; i++)
```

```
        cout << is_prime[i] << ' ';
```

```
    return 0;
```

```
}
```

9.4 modular_algebra

9.4.1 fermats.cpp

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

```
using namespace std;
```

/*

The Fermat`s theorem states that

a^(p-1)%p is always == 1 unless a % p == 0 or p % a == 0

(but for a == 1 it obviously still works)

Cool but based on this we are going to define (a/b) % p in a sta

```
(1/a) % p
```

$$= a^{-1} \% p$$

$$= a^{-1} \% p$$

$$= 1^* (a^{-1} \% p)$$

```
= (a^(p-1)%p) * (a^-1 % p)
```

```
= (a ^(p-1) * a^-1) % p
```

= (a^p-2) % p.

Other applications of fermats theorm is the possiblilty to assum

```
(assume all a`s are %p)
```

a^0 , a^1 , . . ., a^p-3 , a^p-2 ,

a^p-1 , a^p , . . ., a^2p-4, a^2p-3,

a^2p-2, a^2p-1, . . ., a^3p-5, a^3p-4,

. . .

We can assume that all values in the same column have the same v

```
*/
```

```
#define int long long
```

```
int power(int a, int n, int mod)
```

{

```
if(n== 0)
```

```
return 1;
```

```
int half = power(a, n/2, mod);
```

```
int res = (half * half) % mod;
```

```
if(n % 2)
```

```
res = (res * a) %mod;
```

```
return res;
```

```
}
```

```
signed main()
```

{

```
int p1, p2, r1, r2; cin >> p1 >> p2 >> r1 >> r2;
```

```
if(r1 > r2)
```

{

```
swap(p1, p2);
```

```
swap(r1, r2);
```

```
}
```

```
int c = r2 - r1;
```

```
int ppow = power(p1, p2-2, p2);
```

```
int k = (ppow * c) % p2;
```

```
cout << k*p1 + r1<< endl;
```

```
    return 0;
```

```
}
```

9.5 binary_exp

9.5.1 binary_exp.cpp

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

```
using namespace std;
```

```
#define int long long
```

```
int exp(int a, int b)
```

{

```
if(b == 1)
```

```
return a;
```

```
int h = exp(a, b/2);
```

```
if(b%2 == 1)
```

```
return  h * h * a;
```

```
return h * h;
```

```
}
```

```
signed main()
```

{

```
cout << exp(15, 8) << endl;
```

```
    return 0;
```

```
}
```

9.5.2 matrix_binary_exp.cpp

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

```
using namespace std;
```

```
#define int long long
```

```
// Fixando um vetor 2x2
```

```
array<array<int, 2>, 2> mul(
```

```
array<array<int, 2>, 2> a,
```

```
array<array<int, 2>, 2> b
```

)

{

```
array<array<int, 2>, 2> res;
```

```
res[0][0] = a[0][0] * b[0][0] + a[0][1] * b[1][0];
```

```
res[0][1] = a[0][0] * b[0][1] + a[0][1] * b[1][1];
```

```
res[1][0] = a[1][0] * b[0][0] + a[1][1] * b[1][0];
```

```
res[1][1] = a[1][0] * b[0][1] + a[1][1] * b[1][1];
```

```
cout << res[0][0] << res[0][1] << endl << res[1][0] << res[1][1]
```

```
return res;
```

```
}
```

```
array<array<int, 2>, 2> exp(array<array<int, 2>, 2> a, int b)
```

{

```
if(b == 1)
```

```
return a;
```

```
auto h = exp(a, b/2);
```

```
if(b%2 == 1)
```

```
return  mul(mul(h, h), a);
```

```
    return mul(h, h);
```

```
}
```

```
signed main()
```

{

```
    return 0;
```

```
}
```

10 DP

10.1 removal_game.cpp

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

```
#define int long long
```

```
using namespace std;
```

```
int n;
```

```
int vals[5002];
```

```
int dp(int start, int end)
```

{

```
if(start > end)
```

```
return 0;
```

```
int tot = 1e18;
```

```
tot = min(tot, dp(start+1, end-1) + vals[start]);
```

```
tot = min(tot, dp(start+1, end-1) + vals[end]);
```

```
if(start +1 <= end)
```

```
tot = min(tot, dp(start + 2, end) + vals[start]);
```

```
if(end-1 >= start)
```

```
tot = min(tot, dp(start, end-2) + vals[end]);
```

```
return tot;
```

```
}
```

```
signed main()
```

{

```
cin >> n;
```

```
int tot = 0;
```

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

{

```
cin >> vals[i];
```

```
tot += vals[i];
```

```
}
```

```
cout << tot - dp(0, n-1) << endl;
```

```
return 0;
```

```
}
```

10.2 increasing_subsequence.cpp

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

```
#define int long long
```

```
using namespace std;
```

```
signed main()
```

{

```
int n; cin >> n;
```

```
vector<int> vals(n);
```

```
for(auto & a : vals)
```

```
cin >> a;
```

```
vector<int> ans;
```

```
    ans.push_back(vals[0]);
```

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

{

```
if(ans.back() < vals[i])
```

{

```
        ans.push_back(vals[i]);
```

```
continue;
```

```
}
```

```
*lower_bound(ans.begin(), ans.end(), vals[i]) = vals[i];
```

```
}
```

```
cout << ans.size() << endl;
```

```
return 0;
```

```
}
```

10.3 minimizing_coins.cpp

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

```
using namespace std;
```

```
#define ll long long
```

```
#define int long long
```

```
#define vi vector<int>
```

```
#define pii pair<int,int>
```

```
#define pb push_back
```

```
#define mp make_pair
```

```
#define fi first
```

```
#define se second
```

```
#define endl '\n'
```

```
#define ld long double
```

```
#define mset(A,X) memset(A,X,sizeof A)
```

```
#define bug(x) cerr << #x << "  >>>>>>>  " << x << '\n'
```

```
#define sz size()
```

```
#define all(A) A.begin(), A.end()
```

```
int coins[1001];
```

```
const int SIZE = 1e6 + 1;
```

```
const int MAX = SIZE;
```

```
int dp[SIZE];
```

```
int n; // logic size of dp
```

```
int minimun(int res)
```

{

```
if (res == 0) return 0;
```

```
if(res < 0) return MAX;
```

```
if(dp[res]) return dp[res];
```

```
int min = MAX;
```

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

{

```
int aux = minimun(res - dp[i]);
```

```
if(min > aux) min = aux;
```

```
}
```

```
if(min == MAX) return dp[res] = MAX;
```

```
return dp[res] = min + 1;
```

```
}
```

```
int solve(){
```

```
int x; cin >> n >> x;
```

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

```
    cin >> coins[i];
```

```
// sort(coins , coins + n); //greater<int>()); //  Sorts the array
```

```
int res = minimun(x);
```

```
cout << res << endl;
```

```
return 0;
```

```
}
```

```
signed main(){
```

```
ios_base::sync_with_stdio(false);cin.tie(NULL);
```

```
int t;
```

```
// cin>>t; //*/
```

```
// // t = 1; //*/
```

```
// while(t--)
```

```
    solve();
```

```
return 0;
```

```
}
```

10.4 rectangle_cutting.cpp

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

```
#define int long long
```

```
using namespace std;
```

```
int memo[502][502];
```

```
int dp(int l, int a)
```

{

```
if(l == a)
```

```
return 0;
```

```
if(a > l)
```

```
swap(l, a);
```

```
if(memo[l][a] != -1)
```

```
        return memo[l][a];
```

```
int res = 1e18;
```

```
for(int i = 1; i < l; i++)
```

```
res = min(res,  dp(i, a) + dp(l-i, a) + 1);
```

```
for(int i = 1; i < a; i++)
```

```
res = min(res,  dp(l, i) + dp(l, a-i) + 1);
```

```
return memo[l][a] = res;
```

```
}
```

```
signed main()
```

{

```
int n, m;
```

```
cin >> n >> m;
```

```
for(int i = 0; i <= max(n, m); i++)
```

{

```
for(int j = 0; j <= max(n, m); j++)
```

{

```
memo[i][j] = -1;
```

```
// cout << i << j << ':' << memo[i][j] << ' ';
```

```
}
```

```
// cout << endl;
```

```
}
```

```
cout << dp(n , m) << endl;
```

```
return 0;
```

```
}
```

10.5 project.cpp

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

```
#define int long long
```

```
using namespace std;
```

```
int n;
```

```
vector<pair<int, pair<int, int>>> bosta;
```

```
vector<int> start, fim , value;
```

```
vector<int> checks;
```

```
int dp(int i)
```

{

```
// cout <<i << endl;
```

```
if(i >= n)
```

```
return 0;
```

```
if(checks[i] != -1)
```

```
return checks[i];
```

```
int next = upper_bound(start.begin() + i, start.end(), fim[i]) - s
```

```
return  checks[i] = max(dp(next)  + value[i],  dp(i+1));
```

```
}
```

```
signed main()
```

{

```
cin >> n;
```

```
bosta.resize(n);
```

```
checks.resize(n, -1);
```

```
start.resize(n);
```

```
    fim.resize(n);
```

```
value.resize(n);
```

```
// cout << "bosta1" << endl;
```

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

{

```
int a, b, c;
```

```
cin >> bosta[i].first >> bosta[i].second.first >> bosta[i].sec
```

```
}
```

```
// cout << "bosta2" << endl;
```

```
sort(bosta.begin(), bosta.end());
```

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

{

```
start[i] = bosta[i].first;
```

```
fim[i] = bosta[i].second.first;
```

```
value[i] = bosta[i].second.second;
```

```
}
```

```
// cout << "bosta3" << endl;
```

```
cout << dp(0) << endl;
```

```
return 0;
```

```
}
```

10.6 run

```
g++ $1 && ./a.out < input.txt > output.txt
```

10.7 counting_towers.cpp

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

```
#define int long long
```

```
#define all(A) A.begin(), A.end()
```

```
using namespace std;
```

```
int memo[(int)1e6 + 10][2];
```

```
int const MOD = ((int)1e9) + 7;
```

```
int dp(int h, bool has_top)
```

{

```
if(h == 0)
```

```
return 1;
```

```
int left = has_top ? 1 : 0;
```

```
if(memo[h][left] != -1 )
```

```
        return memo[h][left];
```

```
int tot = 0;
```

```
int false_number = 1;
```

```
int true_number = 4;
```

```
if(!has_top)
```

{

```
false_number = 2;
```

```
true_number = 1;
```

```
}
```

```
tot += ((dp(h-1, true) * true_number) % MOD) + ((dp(h-1, false) *
```

```
return memo[h][left] = tot%MOD;
```

```
}
```

```
signed main()
```

{

```
int t; cin >> t;
```

```
for(int i = 0; i <= 1e6 + 2; i++)
```

{

```
memo[i][0] = memo[i][1] = -1;
```

```
}
```

```
while(t--)
```

{

```
int n; cin >> n;
```

```
    cout << (dp(n-1, false) + dp(n-1, true)) %MOD << endl;
```

```
}
```

```
return 0;
```

```
}
```

10.8 test.cpp

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

```
using namespace std;
```

```
int prices[int(1e5)+1];
```

```
int pages[int(1e5)+1];
```

```
int dp[int(1e5)+1];
```

```
int n, z;
```

```
void solve()
```

{

```
memset(dp, 0, sizeof(dp));
```

```
cin >> n >> z;
```

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

```
cin >> prices[i];
```

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

```
cin >> pages[i];
```

```
int biggest = 0;
```

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

{

```
for(int x = z; x >=0 ; x--)
```

{

```
if(x-prices[i] < 0) continue;
```

```
dp[x] = max(dp[x], dp[x-prices[i]] + pages[i]);
```

```
biggest = max(biggest, dp[x-prices[i]]);
```

```
        biggest = max(dp[x], biggest);
```

```
}
```

```
}
```

```
cout << biggest << endl;
```

```
}
```

```
int main()
```

{

```
    solve();
```

```
return 0;
```

```
}
```

10.9 two_sets_II.cpp

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

```
#define int long long
```

```
using namespace std;
```

```
int memo[502][125251];
```

```
const int MOD =  1e9 +7;
```

```
int half_tot = 0;
```

```
int n;
```

```
int dp(int val, int res)
```

{

```
if(res == half_tot)
```

```
return 1;
```

```
if(val > n || res < 0)
```

```
return 0;
```

```
if(memo[val][res] != -1)
```

```
    return memo[val][res];
```

```
        return memo[val][res] = (dp(val+1, res) + dp(val+1, res - val)) %
```

```
}
```

```
int fast_pow(int a, int b)
```

{

```
if(b == 0)
```

```
return 1;
```

```
int half = fast_pow(a, b/2);
```

```
int tot = (half * half)%MOD;
```

```
if(b%2)
```

```
tot = (tot * a)%MOD;
```

```
return tot;
```

```
}
```

```
signed main()
```

{

```
cin >>n;
```

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

```
memset(memo[i], -1, sizeof(memo[i]));
```

```
int tot = ((n+1)*n)/2;
```

```
if(tot%2)
```

```
    return cout << 0 << endl, 0;
```

```
half_tot = tot/2;
```

```
cout << (dp(1, tot) * fast_pow(2, MOD-2)) % MOD << endl;
```

```
return 0;
```

```
}
```

10.10 minimizing_the_sum.cpp

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

```
using namespace std;
```

```
string t1, t2;
```

```
int memo[1001][1001];
```

```
int n, m;
```

```
int dp(int i, int j)
```

{

```
if(i >= n || j >= m)
```

```
return 0;
```

```
if(memo[i][j]!= -1)
```

```
    return memo[i][j];
```

```
int poss = -1;
```

```
if(t1[i] == t2[j])
```

```
poss= max(poss, 1 + dp(i+1, j+1));
```

```
else
```

{

```
poss= max(poss, dp(i+1, j));
```

```
poss= max(poss, dp(i, j+1));
```

```
}
```

```
return memo[i][j] = poss;
```

```
}
```

```
int main()
```

{

```
cin >> t1 >> t2;
```

```
n = t1.size();
```

```
m = t2.size();
```

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

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

```
memo[i][j] = -1;
```

```
int aux = dp(0,0);
```

```
cout << aux << endl;
```

```
return 0;
```

```
}
```

10.11 money_sums.cpp

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

```
#define int long long
```

```
using namespace std;
```

```
bool memo[(int)1e6];
```

```
signed main()
```

{

```
int n;
```

```
cin >> n;
```

```
memset(memo, false, sizeof(memo));
```

```
int k = 0;
```

```
memo[0] = true;
```

```
vector<int> coins(n);
```

```
for(auto &a:coins)
```

```
cin >> a;
```

```
for(auto coin:coins)
```

{

```
for(int i = 1e5; i >= coin; i--)
```

{

```
if(memo[i])
```

```
            continue;
```

```
if(memo[i - coin])
```

{

```
memo[i] = true;
```

```
k++;
```

```
}
```

```
}
```

```
}
```

```
cout << k << endl;
```

```
for(int i = 1; i <= 1e5; i++)
```

{

```
if(memo[i])
```

```
        cout << i << ' ';
```

```
}
```

```
cout << endl;
```

```
return 0;
```

```
}
```

10.12 edit_distance.cpp

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

```
#define int long long
```

```
using namespace std;
```

```
int n, m;
```

```
string a, b;
```

```
int memo[5001][5001];
```

```
int dp(int p1, int p2)
```

{

```
if(p2 == m)
```

```
    return n - p1;
```

```
if(p1 == n)
```

```
return m - p2;
```

```
if(memo[p1][p2] != -1)
```

```
        return memo[p1][p2];
```

```
if(a[p1] == b[p2])
```

```
        return memo[p1][p2] = dp(p1+1, p2+1);
```

```
else
```

```
        return memo[p1][p2] = min(min(dp(p1+1, p2), dp(p1, p2+1)), dp(
```

```
}
```

```
signed main()
```

{

```
cin >> a >> b;
```

```
if(a.size() < b.size())
```

```
swap(a, b);
```

```
n = a.size();
```

```
m = b.size();
```

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

```
memset(memo[i], -1, sizeof(int) * (m+1));
```

```
    cout << dp(0, 0) << endl;
```

```
return 0;
```

```
}
```

10.13 dice_combination.cpp

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

```
using namespace std;
```

```
#define ll long long
```

```
#define int long long
```

```
#define vi vector<int>
```

```
#define pii pair<int,int>
```

```
#define pb push_back
```

```
#define mp make_pair
```

```
#define fi first
```

```
#define se second
```

```
#define endl '\n'
```

```
#define ld long double
```

```
#define mset(A,X) memset(A,X,sizeof A)
```

```
#define bug(x) cerr << #x << "  >>>>>>>  " << x << '\n'
```

```
#define sz size()
```

```
#define all(A) A.begin(), A.end()
```

```
// vector<int> numbers = {6,5,4,3,2,1};
```

```
int dp[100000 + 1] = { 0 , 1 , 2, 4, 8 , 16, 32};
```

```
// inline int getDpNum(int number)
```

```
// {
```

```
//  return number > 0 ? dp[number] : 0;
```

```
// }
```

```
int cont(int number)
```

{

```
// ALG RECURSIVO
```

```
if(number == 0) return 1;
```

```
if(number < 0) return 0;
```

```
if(dp[number])
```

```
return dp[number];
```

```
int res = 0;
```

```
for(int v = 6; v >= 1; v--)
```

{

```
res += cont(number - v);
```

}

```
res %= (1000000000 + 7);
```

```
dp[number] = res;
```

```
return res;
```

```
// // ALG ITERATIVO
```

```
// for(int i = 7; i <= number; i++)
```

```
// {
```

```
//  for(int v = 6; v >= 1; v--)
```

```
//    dp[i] += dp[i-v] ;
```

```
//  dp[i] %= (1000000000 + 7);
```

```
// }
```

```
// return dp[number];
```

```
}
```

```
int solve(){
```

```
int n; cin >> n;
```

```
int res = cont(n);
```

```
cout << res << endl;
```

```
return 0;
```

```
}
```

```
signed main(){
```

```
ios_base::sync_with_stdio(false);cin.tie(NULL);
```

```
int t;
```

```
// cin>>t; //*/
```

```
// // t = 1; //*/
```

```
// while(t--)
```

```
    solve();
```

```
return 0;
```

```
}
```

10.14 cp_algorithm_stuff

10.14.1 knapsack.cpp

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

```
using namespace std;
```

```
#define int long long
```

```
int n;
```

```
unordered_map<int, int> dp;
```

```
int vals[50];
```

```
int solve_dp(int pos, pair<int, int> aux)
```

{

```
auto [choosed, sum] = aux;
```

```
if(sum == 0 && choosed > 0)
```

```
return 1;
```

```
if(sum < 0 || pos >= n)
```

```
return 0;
```

```
    return solve_dp(pos+1, {choosed, sum}) + solve_dp(pos+1, {choose
```

```
}
```

```
int solve()
```

{

```
int t;
```

```
cin >> n >> t;
```

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

```
        cin >> vals[i];
```

```
cout << dp [t] << endl;
```

```
    return 0;
```

```
}
```

```
signed main()
```

{

```
    solve();
```

```
    return 0;
```

```
}
```

11 game_theory

11.1 Sprague–Grundy-theorem

11.1.1 Grundy_theorem.cpp

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

```
using namespace std;
```

/*

Grundy generalizes the strategy from the game nim to every other

Two players move alternately.

The game consists of states, and the possible moves in a state d

The game ends when a player cannot make a move.

The game surely ends sooner or later.

The players have complete information about the states and allow

So, to deal with grundy numbers. We need to understand the follo

To calculate the grundy number of a number we need to do the fol

```
g_s = mex({g1, g2,…, gn})
```

Where g1, g2, .., gn are the grundy numbers of the states to whi

By the way mex({}) = 0.

```
*/
```

```
signed main()
```

{

```
    return 0;
```

```
}
```

12 data_structures

12.1 bit

12.1.1 bit_all_again.cpp

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

```
using namespace std;
```

```
int n;
```

```
int values[(int)1e8];
```

```
int BITree[(int)1e8 + 1];
```

```
void update(int index, int v)
```

{

```
index++;
```

```
for(; index <= n; index+= index & -index)
```

```
BITree[index] +=v ;
```

```
}
```

```
int get_range(int index)
```

{

```
index++;
```

```
int sum = 0;
```

```
for(; index > 0; index -= index & -index)
```

```
        sum += BITree[index];
```

```
    return sum;
```

```
}
```

```
void init_BITree()
```

{

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

```
BITree[i] = 0;
```

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

```
        update(i, values[i]);
```

```
}
```

```
int main()
```

{

```
n = 10;
```

```
values[0] = 1;
```

```
values[1] = 1;
```

```
values[2] = 1;
```

```
values[3] = 1;
```

```
values[4] = 1;
```

```
values[5] = 1;
```

```
values[6] = 1;
```

```
values[7] = 1;
```

```
values[8] = 1;
```

```
values[9] = 1;
```

```
init_BITree();
```

```
// cout << endl;
```

```
cout << get_range(9) << endl;
```

```
update(9, 2);
```

```
cout << get_range(0) << " ";
```

```
cout << get_range(1) << " ";
```

```
cout << get_range(2) << " ";
```

```
cout << get_range(3) << " ";
```

```
cout << get_range(4) << " ";
```

```
cout << get_range(5) << " ";
```

```
cout << get_range(6) << " ";
```

```
cout << get_range(7) << " ";
```

```
cout << get_range(8) << " ";
```

```
cout << get_range(9) << " ";
```

```
    return 0;
```

```
}
```

12.1.2 bit.cpp

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

```
using namespace std;;
```

```
int n;
```

```
int values[(int)1e8];
```

```
int BITree[(int)1e8+1];
```

```
void update(int index, int v)
```

{

```
index++;
```

```
while(index <= n)
```

{

```
BITree[index] += v;
```

```
index+= (index & -index);
```

```
}
```

```
}
```

```
void generate_bit()
```

{

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

{

```
// cout << i << " " << n << endl;
```

```
BITree[i] = 0;
```

```
}
```

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

{

```
        update(i, values[i]);
```

```
}
```

```
}
```

```
int get_range(int index)
```

{

```
index++;
```

```
int sum = 0;
```

```
for(;index > 0; index = index - (index & -index))
```

{

```
// cout << " " << index << endl;
```

```
        sum += BITree[index];
```

```
}
```

```
    return sum;
```

```
}
```

```
int main()
```

{

```
n = 10;
```

```
values[0] = 1;
```

```
values[1] = 1;
```

```
values[2] = 1;
```

```
values[3] = 1;
```

```
values[4] = 1;
```

```
values[5] = 1;
```

```
values[6] = 1;
```

```
values[7] = 1;
```

```
values[8] = 1;
```

```
values[9] = 1;
```

```
generate_bit();
```

```
// cout << endl;
```

```
cout << get_range(9) << endl;
```

```
update(9, 2);
```

```
cout << get_range(0) << " ";
```

```
cout << get_range(1) << " ";
```

```
cout << get_range(2) << " ";
```

```
cout << get_range(3) << " ";
```

```
cout << get_range(4) << " ";
```

```
cout << get_range(5) << " ";
```

```
cout << get_range(6) << " ";
```

```
cout << get_range(7) << " ";
```

```
cout << get_range(8) << " ";
```

```
cout << get_range(9) << " ";
```

```
    return 0;
```

```
}
```

12.2 sparce_table

12.2.1 next_smaller_element.cpp

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

```
using namespace std;
```

```
const int MAXN = 1e6;
```

```
const int K = 30;
```

```
int st[K+1][MAXN];
```

```
int arr[MAXN];
```

```
int n;
```

```
void build()
```

{

```
copy(arr, arr+n, st[0]);
```

```
for(int j = 1; j <= K; j++)
```

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

```
            st[j][i] = min(st[j-1][i], st[j-1][i+((int)1<<(j-1))]);
```

```
}
```

```
int log_floor(int x){    return x ? __builtin_clzll(1) - __builtin_c
```

```
int query(int l, int r)
```

{

```
int size = log_floor(r-l + 1);
```

```
return min(st[size][l], st[size][r - ((int)1 << size ) + 1]);
```

```
}
```

```
int bb(int pos)
```

{

```
int l = pos + 1, r = n-1;
```

```
while(l <= r)
```

{

```
// cout <<"l r "<< l << ' ' << r << endl;
```

```
int mid =(l+r)/2;
```

```
if(query(pos + 1, mid) <= arr[pos])
```

```
            r = mid-1;
```

```
else
```

```
l = mid+1;
```

```
}
```

```
return 1;
```

```
}
```

```
signed main()
```

{

```
cin >> n;
```

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

```
cin >> arr[i];
```

```
build();
```

```
while(true)
```

{

```
int pos; cin >> pos;
```

```
if(pos < 0)
```

```
            break;
```

```
cout << bb(pos) << ' ' << arr[bb(pos)] << endl;
```

```
}
```

```
while(true)
```

{

```
int l, r; cin >> l >> r;
```

```
if(r == -1 || l == -1)
```

```
        break;;
```

```
        cout << query(l, r) << endl;
```

```
}
```

```
    return 0;
```

```
}
```

12.2.2 next_smaller_element copy.cpp

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

```
using namespace std;
```

```
const int MAXN = 1e6;
```

```
const int K = 30;
```

```
int st[K+1][MAXN];
```

```
int arr[MAXN];
```

```
int n;
```

```
void build()
```

{

```
copy(arr, arr+n, st[0]);
```

```
for(int j = 1; j <= K; j++)
```

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

```
            st[j][i] = min(st[j-1][i], st[j-1][i+((int)1<<(j-1))]);
```

```
}
```

```
int log_floor(int x){    return x ? __builtin_clzll(1) - __builtin_c
```

```
int query(int l, int r)
```

{

```
int size = log_floor(r-l + 1);
```

```
return min(st[size][l], st[size][r - ((int)1 << size ) + 1]);
```

```
}
```

```
int bb(int pos)
```

{

```
int l = pos + 1, r = n-1;
```

```
while(l <= r)
```

{

```
// cout <<"l r "<< l << ' ' << r << endl;
```

```
int mid =(l+r)/2;
```

```
if(query(pos + 1, mid) < arr[pos])
```

```
            r = mid-1;
```

```
else
```

```
l = mid+1;
```

```
}
```

```
return 1;
```

```
}
```

```
signed main()
```

{

```
cin >> n;
```

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

```
cin >> arr[i];
```

```
reverse(arr, arr+n);
```

```
build();
```

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

```
        cout << n-bb(i) << ' ';
```

```
cout << endl;
```

```
    return 0;
```

```
}
```

12.2.3 sparce_table.cpp

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

```
using namespace std;
```

/*

Sparce tables can be used to do all sorts of range queries in a

array without updates. But the thig is, it can do queries of min

and maximum inside a range in consntant time so it is really onl

worth for this

mixed with bb it can be used to find por example the next elemen

on the array that is lass than arr[i] in O(log n) as shown in

next_larger_element.cpp file

```
*/
```

/*

The main ideia is to store every power of two sized range result

so its possible to compute a query composing a range by its powe

considering the minimum and maximum operations we can make this

its possible to just get two big preprocessed chuncks that cover

(maybe even with a overlaping space) and with this return the re

```
*/
```

```
const int N = 10^6; // elements on the array
```

```
const int K = 25; // powers of 2
```

```
// st[j][i] means te minimum at the range
```

```
//       [i, i + 2^j-1]
```

```
int st[K+1][N];
```

```
int arr[N];
```

```
int log_floor(int x);
```

```
void build()
```

{

```
// For all ranges with j==0 the own element is gonna be the mini
```

```
copy(arr, arr+N, st[0]);
```

```
for(int j = 1; j <= K; j++)
```

```
// less or equal because of i+2^j-1
```

```
for(int i = 0; i + (1 << j) <= N; i++)
```

```
            st[j][i] = min(st[j-1][i] , st[j-1][i + (1 << (j-1))]);
```

```
}
```

```
int get_min(int l ,int r)
```

{

```
int power = log_floor(r-l+1);
```

```
return min(st[power][l], st[power][r+1-(1 << power)]);
```

```
}
```

```
// fast way to find integer log
```

```
#include<bit>
```

```
int log_floor(int x) {
```

```
return __bit_width(x) - 1;
```

```
// pre c++ 20
```

```
return x ? __builtin_clzll(1) - __builtin_clzll(x) : -1;
```

```
}
```

12.3 trie

12.3.1 trie.cpp

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

```
using namespace std;
```

```
const int alphabet_size = 26;
```

```
struct trieNode
```

{

```
trieNode * children[alphabet_size];
```

```
trieNode()
```

{

```
for(int i = 0; i < alphabet_size; i++)
```

```
        this->children[i] = NULL;
```

```
}
```

```
bool end = false;
```

```
};
```

```
void freeTrieNode(trieNode * node)
```

{

```
if(node == NULL)
```

```
return;
```

```
for(int i =0 ; i < alphabet_size; i++)
```

```
        freeTrieNode(node->children[i]);
```

```
delete node;
```

```
}
```

```
class Trie
```

{

```
private:
```

```
trieNode * root = new trieNode();
```

```
public:
```

```
    ~Trie()
```

{

```
    freeTrieNode(root);
```

```
}
```

```
void insert(const string & s)
```

{

```
auto crawl = root;
```

```
for(char c : s)
```

{

```
int index = c - 'a';
```

```
if(crawl->children[index] == NULL)
```

```
                crawl->children[index] = new trieNode();
```

```
crawl = crawl->children[index];
```

```
}
```

```
        crawl->end = true;
```

```
}
```

```
bool find_prefix(const string & s)
```

{

```
auto crawl = root;
```

```
for(char c : s)
```

{

```
int index = c - 'a';
```

```
if(crawl->children[index] == NULL)
```

```
return false;
```

```
crawl = crawl->children[index];
```

```
}
```

```
return true;
```

```
}
```

```
bool find(const string & s)
```

{

```
auto crawl = root;
```

```
for(char c : s)
```

{

```
int index = c - 'a';
```

```
if(crawl->children[index] == NULL)
```

```
return false;
```

```
crawl = crawl->children[index];
```

```
}
```

```
return crawl->end;
```

```
}
```

```
};
```

```
int main()
```

{

```
    return 0;
```

```
}
```

12.4 segtree

12.4.1 regular.cpp

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

```
using namespace std;
```

```
#define N 300000
```

```
#define irrelevant 0
```

```
#define endl '\n'
```

```
#define int long long
```

```
int operation(int left, int right)
```

{

```
return left ^ right;
```

```
}
```

```
int tree[2*N], values_seg[N];
```

```
int query(int node, int l, int r, int begin, int end)
```

{

```
if(l > end || r < begin )
```

```
return irrelevant;
```

```
if(l >= begin && r <= end)
```

```
return tree[node];
```

```
int mid = (l + r)/2;
```

```
return operation(
```

```
query(node*2 + 1, l, mid, begin, end),
```

```
query(node*2 + 2, mid + 1, r, begin, end)
```

```
);
```

```
}
```

```
int update(int node, int l, int r, int pos, int value)
```

{

```
if(pos < l or pos > r)
```

```
return tree[node];
```

```
if(l==r && pos == r)
```

```
return tree[node] = value;
```

```
int mid = (l + r) / 2;
```

```
return tree[node] = operation(
```

```
            update(node * 2 + 1, l, mid, pos, value),
```

```
        update(node * 2 + 2, mid + 1, r, pos, value)
```

```
);
```

```
}
```

```
int build(int node, int l, int r)
```

{

```
if(l > r) return irrelevant;
```

```
if(l == r)
```

```
return tree[node] = values_seg[l];
```

```
int mid = (l+r)/2;
```

```
return tree[node] = operation(
```

```
        build(node*2 + 1, l, mid),
```

```
        build(node*2 + 2, mid + 1, r)
```

```
);
```

```
}
```

```
signed main()
```

{

```
ios_base::sync_with_stdio(0);cin.tie(0);
```

```
int n, q; cin >> n >> q;
```

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

{

```
cin >> values_seg[i];
```

```
}
```

```
build(0, 0, n-1);
```

```
for(int i = 0;i < q; i++)
```

{

```
int  a, b; cin >> a >> b;
```

```
// if(op == 1)
```

```
// {
```

```
//      update(0, 0, n-1, a-1, b);
```

```
//      continue;
```

```
// }
```

```
a--;b--;
```

```
    cout << query(0, 0, n-1, a, b) << endl;
```

```
}
```

```
    return 0;
```

```
}
```

12.4.2 range_update.cpp

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

```
using namespace std;
```

```
#define N 300000
```

```
#define irrelevant 0
```

```
#define endl '\n'
```

```
#define int long long
```

```
int operation(int left, int right)
```

{

```
return left ^ right;
```

```
}
```

```
int tree[2*N], lazy[2*N], values[N];
```

```
int query(int node, int l, int r, int begin, int end)
```

{

```
int size = r-l+1;
```

```
if(lazy[node])
```

{

```
        tree[node] += size * lazy[node];
```

```
if(size > 1)
```

{

```
        lazy[node * 2 + 1] += lazy[node];
```

```
lazy[node * 2 + 2] += lazy[node];
```

```
}
```

```
lazy[node] = 0;
```

```
}
```

```
if(l > end || r < begin )
```

```
return irrelevant;
```

```
if(l >= begin && r <= end)
```

```
return tree[node];
```

```
int mid = (l + r)/2;
```

```
return operation(
```

```
query(node*2 + 1, l, mid, begin, end),
```

```
query(node*2 + 2, mid + 1, r, begin, end)
```

```
);
```

```
    return 0;
```

```
}
```

```
int range_update(int node, int l, int r, int begin, int end, int val
```

{

```
int size = r-l+1;
```

```
if(lazy[node])
```

{

```
        tree[node] += size * lazy[node];
```

```
if(size > 1)
```

{

```
        lazy[node * 2 + 1] += lazy[node];
```

```
lazy[node * 2 + 2] += lazy[node];
```

```
}
```

```
lazy[node] = 0;
```

```
}
```

```
if(l > end || r < begin)
```

```
return tree[node];
```

```
if(l >=begin && r <= end)
```

{

```
if(size > 1)
```

{

```
        lazy[node * 2 + 1] += value;
```

```
        lazy[node * 2 + 2] += value;
```

```
}
```

```
        return tree[node] += size * value;
```

```
}
```

```
int mid = (l+r)/2;
```

```
return tree[node] = operation(
```

```
        range_update(node*2+1, l, mid, begin, end, value),
```

```
range_update(node*2+2, mid+1, r, begin, end, value)
```

```
);
```

```
}
```

```
int build(int node, int l, int r)
```

{

```
if(l > r) return irrelevant;
```

```
if(l == r)
```

```
return tree[node] = values[l];
```

```
int mid = (l+r)/2;
```

```
return tree[node] = operation(
```

```
        build(node*2 + 1, l, mid),
```

```
        build(node*2 + 2, mid + 1, r)
```

```
);
```

```
}
```

```
signed main()
```

{

```
ios_base::sync_with_stdio(0);cin.tie(0);
```

```
int n, q; cin >> n >> q;
```

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

{

```
        cin >> values[i];
```

```
}
```

```
build(0, 0, n-1);
```

```
memset(lazy, 0, sizeof(int) * 2*n );
```

```
for(int i = 0;i < q; i++)
```

{

```
int op; cin >> op;
```

```
if(op == 1)
```

{

```
int a, b, u; cin >> a >> b >> u;
```

```
a--;b--;
```

```
range_update(0, 0, n-1, a, b, u);
```

```
continue;
```

```
}
```

```
int k; cin >> k;
```

```
k--;
```

```
    cout << query(0, 0, n-1, k, k) << endl;
```

```
}
```

```
    return 0;
```

```
}
```

12.5 orderset

12.5.1 example.cpp

```
#include <iostream>
```

```
using namespace std;
```

```
#include <ext/pb_ds/assoc_container.hpp>
```

```
#include <ext/pb_ds/tree_policy.hpp>
```

```
using namespace __gnu_pbds;
```

```
#define ordered_set tree<int, null_type,less<int>, rb_tree_tag,tree_
```

```
int main()
```

{

```
// Ordered set declared with name o_set
```

```
ordered_set o_set;
```

```
// insert function to insert in
```

```
// ordered set same as SET STL
```

```
o_set.insert(5);
```

```
o_set.insert(1);
```

```
o_set.insert(2);
```

```
// Finding the second smallest element
```

```
// in the set using * because
```

```
//  find_by_order returns an iterator
```

```
cout << *(o_set.find_by_order(1))
```

```
<< endl;
```

```
// Finding the number of elements
```

```
// strictly less than k=4
```

```
cout << o_set.order_of_key(4)
```

```
<< endl;
```

```
// Finding the count of elements less
```

```
// than or equal to 4 i.e. strictly less
```

```
// than 5 if integers are present
```

```
cout << o_set.order_of_key(5)
```

```
<< endl;
```

```
// Deleting 2 from the set if it exists
```

```
if (o_set.find(2) != o_set.end())
```

```
o_set.erase(o_set.find(2));
```

```
// Now after deleting 2 from the set
```

```
// Finding the second smallest element in the set
```

```
cout << *(o_set.find_by_order(1))
```

```
<< endl;
```

```
// Finding the number of
```

```
// elements strictly less than k=4
```

```
cout << o_set.order_of_key(4)
```

```
<< endl;
```

```
    return 0;
```

```
}
```

13 graph

13.1 run

```
g++ $1 && ./a.out < input.txt > output.txt
```

13.2 counting_rooms.cpp

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

```
using namespace std;
```

```
#define ll long long
```

```
#define int long long
```

```
#define vi vector<int>
```

```
#define pii pair<int,int>
```

```
#define pb push_back
```

```
#define mp make_pair
```

```
#define fi first
```

```
#define se second
```

```
#define endl '\n'
```

```
#define ld long double
```

```
#define mset(A,X) memset(A,X,sizeof A)
```

```
#define bug(x) cerr << #x << "  >>>>>>>  " << x << '\n'
```

```
#define sz size()
```

```
#define all(A) A.begin(), A.end()
```

```
char table[1001][1001];
```

```
int n, m;
```

```
struct point
```

{

```
int y, x;
```

```
point operator+(point p)
```

{

```
return { y + p.y, x + p.x };
```

}

```
};
```

```
inline bool isInRange(point p)
```

{

```
return
```

```
p.x >= 0 &&
```

```
p.x < m &&
```

```
p.y >= 0 &&
```

p.y < n

;

```
}
```

```
vector<point> steps = {
```

```
{1,0},
```

```
{0,1},
```

```
{-1,0},
```

{0,−1}

```
};
```

```
bool visitRoom(point p)
```

{

```
if(!isInRange(p) || table[p.y][p.x] != '.') return false;
```

```
table[p.y][p.x] = 'L';
```

```
for(auto step :steps)
```

```
visitRoom(p + step);
```

```
return true;
```

```
}
```

```
int solve(){
```

```
cin >> n >> m;
```

```
vector<point> points;
```

```
for(int y = 0; y < n; y++)
```

{

```
for(int x = 0; x < m; x++)
```

{

```
// cout << 1;
```

```
char aux; cin >> aux;
```

```
// cout << 2;
```

```
// cout << "leu";
```

```
table[y][x] = aux;
```

```
// cout << table[y][x];
```

```
if(table[y][x] == '.') points.push_back({y, x});
```

```
}
```

```
// cout << endl;
```

}

```
// cout << "Codigo" << endl;
```

```
int cont = 0;
```

```
for(auto p : points)
```

{

```
if(visitRoom(p))
```

```
cont++;
```

}

```
cout << cont << endl;
```

```
return 0;
```

```
}
```

```
signed main(){
```

```
// ios_base::sync_with_stdio(false);cin.tie(NULL);
```

```
// int t;
```

```
// //cin>>t; //*/
```

```
// t = 1; //*/
```

```
// while(t--)
```

```
    solve();
```

```
return 0;
```

```
}
```

13.3 bridges

13.3.1 bridge_tree.cpp

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

```
using namespace std;
```

```
typedef vector<vector<int>> graph_type;
```

```
typedef pair<int, int> edge_type;
```

```
graph_type g;
```

```
vector<int> low, tin, visited;
```

```
vector<edge_type> bridges;
```

```
int cont = 0;
```

```
graph_type components;
```

```
set<int> order;
```

```
void dfs(int v, int p = -1)
```

{

```
low[v] = tin[v] = cont++;
```

```
visited[v] = 1;
```

```
for(int to : g[v])
```

{

```
if(to == p)
```

```
continue;
```

```
if (visited[to])
```

```
            low[v] = min(low[v], tin[to]);
```

```
else
```

{

```
            dfs(to, v);
```

```
low[v] = min(low[v], low[to]);
```

```
if(low[to] > tin[v])
```

```
            bridges.push_back({v, to});
```

```
}
```

```
}
```

```
}
```

```
int bridge_tree()
```

{

```
    visited.assign(g.size(), 0);
```

```
tin.resize(g.size());
```

```
low.resize(g.size());
```

```
        bridges.clear();
```

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

{

```
if(!visited[i])
```

```
            dfs(i);
```

```
}
```

```
    return 0;
```

```
}
```

```
int main()
```

{

```
cout << visited[99] << endl;
```

```
    return 0;
```

```
}
```

13.3.2 bridges.cpp

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

```
using namespace std;
```

```
typedef vector<vector<int>> graph_type;
```

```
typedef pair<int, int> edge_type;
```

```
graph_type g;
```

```
vector<int> low, tin, visited;
```

```
vector<edge_type> bridges;
```

```
int cont = 0;
```

```
void dfs(int v, int p = -1)
```

{

```
low[v] = tin[v] = cont++;
```

```
visited[v] = 1;
```

```
for(int to : g[v])
```

{

```
if(to == p)
```

```
continue;
```

```
if (visited[to])
```

```
            low[v] = min(low[v], tin[to]);
```

```
else
```

{

```
            dfs(to, v);
```

```
low[v] = min(low[v], low[to]);
```

```
if(low[to] > tin[v])
```

```
            bridges.push_back({v, to});
```

```
}
```

```
}
```

```
}
```

```
int find_briges()
```

{

```
    visited.assign(g.size(), 0);
```

```
tin.resize(g.size());
```

```
low.resize(g.size());
```

```
        bridges.clear();
```

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

{

```
if(!visited[i])
```

```
            dfs(i);
```

```
}
```

```
    return 0;
```

```
}
```

```
int main()
```

{

```
cout << visited[99] << endl;
```

```
    return 0;
```

```
}
```

13.4 strongly_connected

13.4.1 condensed_tree.cpp

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

```
using namespace std;
```

```
typedef vector<vector<int>> graph_type;
```

```
#define endl '\n'
```

```
void print_v(vector<int> &v, string name = "")
```

{

```
cout << name << '\t';
```

```
for(int el : v)
```

```
cout << el << " ";
```

```
cout << endl;
```

```
}
```

```
void print_g(graph_type &g, string name = "")
```

{

```
cout << "\n" << name << '\n';
```

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

{

```
cout << i << ": ";
```

```
print_v(g[i]);
```

```
}
```

```
}
```

```
vector<int> visited1;
```

```
graph_type g1;
```

```
vector<int> order;
```

```
int dfs1(int u)
```

{

```
visited1[u] = 1;
```

```
for(int to : g1[u])
```

{

```
if(visited1[to] != 0)
```

```
continue;
```

```
        dfs1(to);
```

```
}
```

```
        order.emplace_back(u);
```

```
    return 0;
```

```
}
```

```
vector<int> visited2;
```

```
graph_type g2;
```

```
int n_comp =1;
```

```
graph_type components;
```

```
graph_type condensed;
```

```
int dfs2(int u)
```

{

```
cout << "started " << u << endl;
```

```
visited2[u] = n_comp;
```

```
for(int to : g2[u])
```

{

```
if(visited2[to] != 0)
```

{

```
if(visited2[to] != n_comp)
```

```
                condensed[n_comp].emplace_back(visited2[to]);
```

```
continue;
```

```
}
```

```
cout << "Going to " << to << " from " << u << endl;
```

```
        dfs2(to);
```

```
}
```

```
components[u].emplace_back(n_comp);
```

```
cout << "Ended " << u << endl;
```

```
    return 0;
```

```
}
```

```
void set_graphs()
```

{

```
int n, m;
```

```
cin >> n >> m;
```

```
g1.resize(n+1);
```

```
g2.resize(n+1);
```

```
condensed.resize(n+1);
```

```
    visited1.resize(n + 1, 0);
```

```
visited2.resize(n + 1, 0);
```

```
components.resize(n + 1);
```

```
while(m--)
```

{

```
int a, b; cin >> a >> b;
```

```
        g1[a].emplace_back(b);
```

```
        g2[b].emplace_back(a);
```

```
}
```

```
print_g(g1, "g1");
```

```
print_g(g2, "g2");
```

```
}
```

```
void kosaraju()
```

{

```
set_graphs();
```

```
cout << endl;
```

```
for(int i = 1; i < g1.size(); i++)
```

{

```
if(!visited1[i])
```

```
            dfs1(i);
```

```
}
```

```
cout << endl;
```

```
print_v(order, "order");
```

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

```
print_v(order, "reversed order");
```

```
cout << endl;
```

```
for(int vertex : order)
```

{

```
if(!visited2[vertex])
```

{

```
        dfs2(vertex);
```

```
n_comp++;
```

```
}
```

```
}
```

```
cout << endl;
```

```
print_v(visited2, "visited2");
```

```
cout << endl;
```

```
print_g(condensed, "condensed");
```

```
cout << endl;
```

```
print_g(components, "components");
```

```
cout << endl << "n_comp " << n_comp  << endl;
```

```
}
```

```
int main()
```

{

```
kosaraju();
```

```
    return 0;
```

```
}
```

13.4.2 kosaraju.cpp

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

```
using namespace std;
```

```
typedef vector<vector<int>> graph_type;
```

```
#define endl '\n'
```

```
void print_v(vector<int> &v, string name = "")
```

{

```
cout << name << '\t';
```

```
for(int el : v)
```

```
cout << el << " ";
```

```
cout << endl;
```

```
}
```

```
void print_g(graph_type &g, string name = "")
```

{

```
cout << "\n" << name << '\n';
```

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

{

```
cout << i << ": ";
```

```
print_v(g[i]);
```

```
}
```

```
}
```

```
vector<int> visited1;
```

```
graph_type g1;
```

```
vector<int> order;
```

```
int dfs1(int u)
```

{

```
visited1[u] = 1;
```

```
for(int to : g1[u])
```

{

```
if(visited1[to] != 0)
```

```
continue;
```

```
        dfs1(to);
```

```
}
```

```
        order.emplace_back(u);
```

```
    return 0;
```

```
}
```

```
vector<int> visited2;
```

```
graph_type g2;
```

```
int n_comp =1;
```

```
graph_type components;
```

```
int dfs2(int u)
```

{

```
cout << "started " << u << endl;
```

```
visited2[u] = n_comp;
```

```
for(int to : g2[u])
```

{

```
if(visited2[to] != 0)
```

```
continue;
```

```
cout << "Going to " << to << " from " << u << endl;
```

```
        dfs2(to);
```

```
}
```

```
components[n_comp].emplace_back(u);
```

```
cout << "Ended " << u << endl;
```

```
    return 0;
```

```
}
```

```
void set_graphs()
```

{

```
int n, m;
```

```
cin >> n >> m;
```

```
g1.resize(n+1);
```

```
g2.resize(n+1);
```

```
    visited1.resize(n + 1, 0);
```

```
visited2.resize(n + 1, 0);
```

```
components.resize(n + 1);
```

```
while(m--)
```

{

```
int a, b; cin >> a >> b;
```

```
        g1[a].emplace_back(b);
```

```
        g2[b].emplace_back(a);
```

```
}
```

```
print_g(g1, "g1");
```

```
print_g(g2, "g2");
```

```
}
```

```
void kosaraju()
```

{

```
set_graphs();
```

```
cout << endl;
```

```
for(int i = 1; i < g1.size(); i++)
```

{

```
if(!visited1[i])
```

```
            dfs1(i);
```

```
}
```

```
cout << endl;
```

```
print_v(order, "order");
```

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

```
print_v(order, "reversed order");
```

```
cout << endl;
```

```
for(int vertex : order)
```

{

```
if(!visited2[vertex])
```

{

```
        dfs2(vertex);
```

```
n_comp++;
```

```
}
```

```
}
```

```
cout << endl;
```

```
print_v(visited2, "visited2");
```

```
cout << endl;
```

```
print_g(components, "components");
```

```
cout << endl << "n_comp " << n_comp  << endl;
```

```
}
```

```
int main()
```

{

```
kosaraju();
```

```
    return 0;
```

```
}
```

13.5 DP_in_tree

13.5.1 tree_with_small_distance.cpp

14 geometry

14.1 manhattan_distance

14.1.1 biggest_distance_between_points.cpp

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

```
using namespace std;
```

```
// $$|p.x - q.x| = \max(p.x - q.x, -p.x + q.x)$$
```

/*

The whole concept of this runs over the extrapolation of the ide

Considering we just have the x cordinate, to maximize the distan

we need to find the biggest

$$|p.x - q.x| = \max(p.x - q.x, -p.x + q.x)$$

For better comprehension lets always consider p.x-q.x as positiv

```
max(p.x + (-q.x)) = max(p.x) + max(-q.x)
```

```
= max(p.x) - min(q.x)
```

With more cordinates we can get a simillar result, just like

```
max(p.x + p.y) − min(q.x + q.y)
```

Just remember that it is important to handle the all permutation

necessary to handle cases like

```
max(-p.x + p.y) - min(q.x - q.y)
```

```
max(p.x - p.y) - min(-q.x + q.y)
```

```
max(p.x + p.y) - min(-q.x - q.y)
```

```
*/
```

```
vector<vector<int>> p = { // Points
```

```
{1, 2},
```

```
{1, 6},
```

```
{12, 6},
```

```
{18, 6},
```

```
{5, 9},
```

```
{3, 4},
```

```
};
```

```
int n = p.size(); // number of points
```

```
int d = p[0].size(); // Dimension of the cordinetes
```

```
int solve()
```

{

```
long long ans = 0;
```

```
// the idea is to iterate throught all signal permutations as a
```

```
for (int msk = 0; msk < (1 << d); msk++) {
```

```
long long mx = LLONG_MIN, mn = LLONG_MAX;
```

```
// For every point we are going to get the values
```

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

```
long long cur = 0;
```

```
// Iterate over all cordinates of a point and making sum
```

```
// acourding to the bitmask
```

```
            for (int j = 0; j < d; j++) {
```

```
                cout << msk << ' ' << (1 << j) << endl;
```

```
if (msk & (1 << j))
```

{

```
                cur += p[i][j];
```

}

```
else
```

{

```
                cur -= p[i][j];
```

}

```
}
```

```
        mx = max(mx, cur);
```

```
            mn = min(mn, cur);
```

```
}
```

```
        ans = max(ans, mx - mn);
```

```
}
```

```
    cout << ans << endl;
```

```
    return 0;
```

```
}
```

```
int main()
```

{

```
    solve();
```

```
    return 0;
```

```
}
```

14.1.2 chebyschev.cpp

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

/*

There is a distance metric called chebyschev distance, that diss

while calculating the distance.

$$D_{chebyschev}(x, w) = \max(|x_i - y_i|)$$

so there is a way to make the manhattan distance = chebyscev dis

basicly we just need to project these points to a chebyschev usi

and thats all

```
*/
```

```
using namespace std;
```

```
pair<int, int> to_chebyschev(pair<int, int> p)
```

{

```
    return {p.first + p.second, p.first - p.second};
```

```
}
```

```
signed main()
```

{

```
    return 0;
```

```
}
```

14.2 lattice_points

14.2.1 Shoelance_Theorem.cpp

14.2.2 Pick's_Theorems.cpp

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

```
using namespace std;
```

/*

Picks theorem express the area of a polygon which aint vertices

```
A = I + 1/2*B - 1
```

With:

A beeing the Area

I beeing the number of littice points inside the polygon

B beeing the number of litice points on the boundary / edges

```
*/
```

```
// Felipe`s code (need to be revised)
```

```
int gcd(int a, int b)
```

{

```
if(a < b)
```

```
swap(a, b);
```

```
if(b == 0)
```

```
return a;
```

```
return gcd(b, a%b);
```

```
}
```

```
// second is x and first is y
```

```
int calculate_lattice_points_in_edges(const vector<pair<int, int>> &
```

```
int ans {};
```

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

```
auto q = i ? polygon[i - 1] : polygon.back();
```

```
auto p = polygon[i];
```

```
if (p.second == q.second) {
```

```
            ans += abs(p.first - q.first) - 1;
```

```
} else if (p.first == q.first) {
```

```
ans += abs(p.second - q.second) - 1;
```

```
} else {
```

```
            ans += gcd(abs(p.second - q.second), abs(p.first - q.fir
```

```
}
```

```
}
```

```
return ans + polygon.size();
```

```
}
```

```
template <typename T>
```

```
int calculate_lattice_points_inside_polygon(const vector<pair<int, i
```

```
auto area = calculate_two_times_area(polygon);
```

```
int lattice_points_in_edge = calculate_lattice_points_in_edges(p
```

```
    return (area - lattice_points_in_edge + 2) / 2;
```

```
}
```

```
int main()
```

{

```
    return 0;
```

```
}
```

14.3 cses

14.3.1 LineSegmentIntersection.cpp

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

```
using namespace std;
```

struct point2d

{

```
int x, y;
```

```
int operator^(point2d p)
```

{

```
return (x * p.y) - (p.x * y);
```

```
}
```

```
int operator*(point2d p)
```

{

```
return x * p.x + y*p.y;
```

```
}
```

```
point2d operator-(point2d p)
```

{

```
    return {x - p.x, y - p.y};
```

```
}
```

```
point2d operator+(point2d p)
```

{

```
    return {x + p.x, y + p.y};
```

```
}
```

```
int orientation(point2d p2, point2d p3)
```

{

```
int ans = (p2.y-y)* (p3.x - p2.x) - (p3.y-p2.y)* (p2.x - x);
```

```
    return ans/abs(ans);
```

```
}
```

```
};
```

```
int solve()
```

{

```
    return 0;
```

```
}
```

```
signed main()
```

{

```
int t; cin >> t;
```

```
while(t--)
```

```
    solve();
```

```
    return 0;
```

```
}
```

14.3.2 PointLocationTest.cpp

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

```
using namespace std;
```

```
#define int long long
```

```
const double tol = 1e-8;
```

struct p2d

{

```
double x;
```

```
double y;
```

```
double size()
```

{

```
return sqrt(x*x + y*y);
```

```
}
```

```
bool operator==(p2d p2)
```

{

```
        return abs(x - p2.x) <= tol && abs(y - p2.y) <= tol;
```

```
}
```

```
p2d operator*(double i)
```

{

```
return {x*i, y*i};
```

```
}
```

```
p2d operator+(p2d i)
```

{

```
    return {x+i.x, y+i.y};
```

```
}
```

```
};
```

```
p2d gen_line(p2d b, p2d a)
```

{

```
p2d res;
```

```
res.x = a.x - b.x;
```

```
res.y = a.y - b.y;
```

```
double s = res.size();
```

```
if(s != (double)0)
```

{

```
    res.x /= s;
```

```
    res.y /= s;
```

```
}
```

```
return res;
```

```
}
```

```
int solve()
```

{

```
p2d p1, p2, p3;
```

```
cin >> p1.x >> p1.y;
```

```
cin >> p2.x >> p2.y;
```

```
cin >> p3.x >> p3.y;
```

```
// The idea is to do the cross product between the vectors, p2-p
```

```
// Based on the norm of the imaginary Z dimension we can deduce
```

```
// accourding to the right hand rule.
```

```
int res = (p2.x - p1.x) * (p3.y - p1.y )- (p2.y - p1.y) * (p3.x
```

```
if(res == 0)
```

```
cout << "TOUCH" << endl;
```

```
else if (res > 0)
```

```
cout << "LEFT" << endl;
```

```
else
```

```
    cout << "RIGHT" << endl;
```

```
    return 0;
```

```
}
```

```
signed main()
```

{

```
int t; cin >> t;
```

```
while(t--)
```

```
    solve();
```

```
    return 0;
```

```
}
```

14.4 basics

14.4.1 dot_product.cpp

```
#include "./points.cpp"
```

```
#pragma once
```

/*

To start we need to know that

The cross product of A and B is equal to

$$|A||B|\cos(\theta) = A.x * B.x + A.y * B.y + A.z * B.z$$

we can see that when $\theta = 90°$, the cross prod = 0.

```
*/
```

```
point3D project(point3D& A, point3D& B)
```

{

```
auto normalizedB = B * (1/B.norm());
```

```
auto Acos = (A* B)/B.norm();
```

```
return normalizedB * Acos;
```

```
}
```

```
float cosValBetween(point3D& A, point3D& B)
```

{

```
return (A*B) / (A.norm(), B.norm());
```

```
}
```

14.4.2 Applications.cpp

```
#include "./cross_product.cpp"
```

```
#include "./dot_product.cpp"
```

```
// Line intersection
```

point3D

14.4.3 points.cpp

```
#pragma once
```

```
#include <cmath>
```

struct point2D

{

```
float x, y;
```

```
point2D(float x, float y) : x(x), y(y)
```

{}

```
float operator *(point2D  p)
```

{

```
return x * p.x + y * p.y;
```

```
}
```

```
point2D operator *(float c)
```

{

```
return point2D(x*c, y*c);
```

```
}
```

```
float norm()
```

{

```
return std::sqrt((*this) * (*this));
```

```
}
```

```
};
```

struct point3D

{

```
float x, y, z;
```

```
point3D(float x, float y, float z) : x(x), y(y), z(z)
```

{}

```
float operator *(point3D  p)
```

{

```
return x * p.x + y * p.y + z * p.z;
```

```
}
```

```
point3D operator *(float c)
```

{

```
return point3D(x*c, y*c, z*c);
```

```
}
```

```
float norm()
```

{

```
return std::sqrt ((*this) * (*this));
```

```
}
```

```
};
```

14.4.4 cross_product.cpp

```
#include "./points.cpp"
```

```
#pragma once
```

/*

The idea of cross product is to describe a operation that output

This outputed vector has some properties.

- It needs to be orthogonal to both input vectors

- Its norm is numerically equal to the area of the parallelo

- Its norm is defined by $|A||B|\sin(\theta)$

```
*/
```

```
// This is equivalent to calculating that weird "determinant"
```

```
point3D cross(point3D a, point3D b) {
```

```
return point3D(a.y * b.z - a.z * b.y,
```

```
a.z * b.x - a.x * b.z,
```

```
a.x * b.y - a.y * b.x);
```

```
}
```

```
// "Mixed product"
```

```
float mixed(point3D A, point3D B, point3D C)
```

{

```
return A * cross(B, C);
```

```
}
```

14.4.5 orientation_of_3_ordered_points.cpp

```
// Given 3 points the task is to determine the orientation of them
```

```
#include<utility>
```

```
enum Orientation
```

{

```
counter_clockwise,
```

clockwise,

colinear

```
};
```

```
Orientation find_orientation(std::pair<int, int> p1, std::pair<int ,
```

{

```
int ans = (p2.second-p1.second)* (p3.first - p2.first) - (p3.sec
```

```
if(ans == 0)
```

```
return Orientation::colinear;
```

```
if(ans > 0)
```

```
return Orientation::clockwise;
```

```
else
```

```
return Orientation::counter_clockwise;
```

```
}
```