



Universidade de Brasília

É só Fazer

Ruan Petrus, Eduardo Freire, Arthur Botelho

1 Contest

2 Data structures

3 Math

4 String

5 Graph

6 DP

Contest (1)

template.cpp26 lines

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

using namespace std;

#define int long long
#define endl '\n'
#define rep(i, a, b) for(int i = (a); i < (b); ++i)
#define all(x) begin(x), end(x)
#define sz(x) (int)(x).size()
#define debug(var) cerr << #var << ": " << var << endl
#define pb push_back
#define eb emplace_back
typedef long long ll;
typedef pair<int, int> pii;
typedef vector<int> vi;

void solve() {
}

int32_t main() {
    ios_base::sync_with_stdio(0); cout.tie(0); cin.tie(0);
    int t = 1;
    while(t--> solve());

    return 0;
}
```

.bashrc3 lines

```
alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \
-fsanitize=undefined,address'
xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps =<
```

.vimrc4 lines

```
set cin ai is ts=4 sw=4 nu rnu
" Select a region and then type :Hash
ca Hash w !cpp -dD -P -fpreprocessed \| tr -d '[:space:]' \
\| md5sum \| cut -c-6
```

hash.sh3 lines

```
# Hashes a file, ignoring all whitespace and comments. Use for
# verifying that code was correctly typed.
cpp -dD -P -fpreprocessed | tr -d '[:space:]' | md5sum |cut -c-6
```

troubleshoot.txt52 lines

```
Pre-submit:
Write a few simple test cases if sample is not enough.
Are time limits close? If so, generate max cases.
Is the memory usage fine?
Could anything overflow?
Make sure to submit the right file.

Wrong answer:
Print your solution! Print debug output, as well.
Are you clearing all data structures between test cases?
Can your algorithm handle the whole range of input?
Read the full problem statement again.
Do you handle all corner cases correctly?
Have you understood the problem correctly?
Any uninitialized variables?
Any overflows?
Confusing N and M, i and j, etc.?
Are you sure your algorithm works?
What special cases have you not thought of?
Are you sure the STL functions you use work as you think?
Add some assertions, maybe resubmit.
Create some testcases to run your algorithm on.
Go through the algorithm for a simple case.
Go through this list again.
Explain your algorithm to a teammate.
Ask the teammate to look at your code.
Go for a small walk, e.g. to the toilet.
Is your output format correct? (including whitespace)
Rewrite your solution from the start or let a teammate do it.

Runtime error:
Have you tested all corner cases locally?
Any uninitialized variables?
Are you reading or writing outside the range of any vector?
Any assertions that might fail?
Any possible division by 0? (mod 0 for example)
Any possible infinite recursion?
Invalidated pointers or iterators?
Are you using too much memory?
Debug with resubmits (e.g. remapped signals, see Various).
```

```
Time limit exceeded:
Do you have any possible infinite loops?
What is the complexity of your algorithm?
Are you copying a lot of unnecessary data? (References)
How big is the input and output? (consider scanf)
Avoid vector, map. (use arrays/unordered_map)
What do your teammates think about your algorithm?
```

```
Memory limit exceeded:
What is the max amount of memory your algorithm should need?
Are you clearing all data structures between test cases?
```

Data structures (2)

```
SegTree.h
Description: Iterative SegTree Can be changed by modifying Spec
Time: O(log N)0386e5, 37 lines
```

```
template<typename Spec>
struct SegTree {
    using LS = Spec;
    using S = typename LS::S;
    using K = typename LS::K;
    int n;
    vector<S> seg;
```

```
SegTree(const vector<S> & v)
    : n(sz(v)), seg(2*n) {
    rep(i, 0, n) seg[i+n] = v[i];
    for(int i = n-1; i >= 1; i--) seg[i] = LS::op(seg[i*2], seg[i*2+1]);
}

void update(int no, K val) {
    no += n;
    seg[no] = LS::update(val, seg[no]);
    while (no > 1) no /= 2, seg[no] = LS::op(seg[no*2], seg[no*2+1]);
}

S query(int l, int r) { // [l, r)
    S vl = LS::id(), vr = LS::id();
    for (l += n, r += n; l < r; l /= 2, r /= 2) {
        if (l & 1) vl = LS::op(vl, seg[l++]);
        if (r & 1) vr = LS::op(seg[--r], vr);
    }
    return LS::op(vl, vr);
}
};

struct Spec {
    using S = int;
    using K = int;
    static S op(S a, S b) { return max(a, b); }
    static S update(K f, S a) { return f + a; }
    static S id() { return 0; }
};
```

```
LazySeg.h
Description: Iterative Lazy SegTree Can be changed by modifying Spec
Time: O(log N)ee5763, 96 lines
```

```
template<typename Spec>
struct LazySeg {
    using LS = Spec;
    using S = typename LS::S;
    using K = typename LS::K;
    int n;
    vector<S> seg;
    vector<K> lazy;
    vector<bool> has_lazy;
    // vector<int> lx, rx; // Additional info

    LazySeg(vector<S> & v) : n(sz(v)), seg(2*n) , lazy(n),
        has_lazy(n) {
        rep(no, 0, n) seg[no+n] = v[no];
        for (int no = n-1; no >= 1; no--) pull(no);

        // Additional info, n must be power of two
        /*
        lx.assign(2*n, 0); rx.assign(2*n, 0);
        lx[1] = 0; rx[1] = n;
        rep(no, 1, n) {
            int mid = (lx[no] + rx[no])/2;
            lx[no*2] = lx[no]; rx[no*2] = mid;
            lx[no*2+1] = mid; rx[no*2+1] = rx[no];
        }
        */
}
```

```
S query(int l, int r) { // [l, r)
    l += n;
    r += n;
    push_to(l); push_to(r-1);
    S vl = LS::id(), vr = LS::id();
```

```
while(l < r) {
    if (l & 1) vl = LS::op(vl, seg[l++]);
    if (r & 1) vr = LS::op(seg[--r], vr);
    l >>= 1; r >>= 1;
}
return LS::op(vl, vr);
}

void update(int l, int r, K val) {
    l += n;
    r += n;
    push_to(l); push_to(r-1);
    int lo = l, ro = r;
    while(l < r) {
        if (l & 1) lo = max(lo, l), apply(l++, val);
        if (r & 1) ro = max(ro, r), apply(--r, val);
        l >>= 1; r >>= 1;
    }
    pull_from(lo);
    pull_from(ro-1);
}

void apply(int no, K val) {
    seg[no] = LS::update(val, seg[no]);
    // seg[no] = LS::update(val, seg[no], lx[no], rx[no]);

    if (no < n) {
        if (has_lazy[no]) lazy[no] = LS::compose(val, lazy[no]);
        else lazy[no] = val;
        has_lazy[no] = true;
    }
}

void pull_from(int no) {
    while(no > 1) no >>= 1, pull(no);
}

void pull(int no) {
    seg[no] = LS::op(seg[no*2], seg[no*2+1]);
}

void push_to(int no) {
    int h = 0; int p2 = 1;
    while(p2 < no) p2 *= 2, h++;
    for (int i = h; i >= 1; i--) push(no >> i);
}

void push(int no) {
    if (has_lazy[no]) {
        apply(no*2, lazy[no]);
        apply(no*2+1, lazy[no]);
        has_lazy[no] = false;
    }
}

struct Spec {
    using S = int;
    using K = int;
    static S op(S a, S b) { return max(a, b); }
    static S update(K f, S a) { return f + a; }
    static K compose(const K f, const K g) { return f + g; }
    static S id() { return 0; }
};
```

Psum.h

Description: Multidimensional Psum Requires Abelian Group S (op, inv, id)

Memory: $\mathcal{O}(N^D)$
Time: $\mathcal{O}(1)$
65f259, 29 lines

```
#define MAS template<class... As> //multiple arguments
template<int D, class S>
struct Psum{ using T = typename S::T;
    int n;
    vector<Psum<D-1, S>> v;
    MAS Psum(int s, As... ds):n(s+1),v(n,Psum<D-1, S>(ds...)){
    MAS void set(T x, int p, As... ps){v[p+1].set(x, ps...);}
    void push(Psum& p){rep(i, 1, n)v[i].push(p.v[i]);}
    void init(){rep(i, 1, n)v[i].init(),v[i].push(v[i-1]);}
    MAS T query(int l, int r, As... ps){
        return S::op(v[r+1].query(ps...),S::inv(v[l].query(ps...))
        ;
    }
};

template<class S>
struct Psum<0, S>{ using T = typename S::T;
    T val=S::id;
    void set(T x){val=x;}
    void push(Psum& a){val=S::op(a.val,val);}
    void init(){}
    T query(){return val;}
};

struct G{
    using T = int;
    static constexpr T id = 0;
    static T op(T a, T b){return a+b;}
    static T inv(T a){return -a;}
};

MultiDSegTree.h
Description: Multidimensional SegTree Requires Monoid S (op, id)
Memory:  $\mathcal{O}(N^D)$ 
Time:  $\mathcal{O}((\log N)^D)$ 
53621d, 37 lines
#pragma once

#define MAS template<class... As> //multiple arguments
template<int D, class S>
struct SegTree{ using T = typename S::T;
    int n;
    vector<SegTree<D-1, S>> seg;
    MAS SegTree(int s, As... ds):n(s),seg(2*n, SegTree<D-1, S>(ds
        ...)){}
    MAS T get(int p, As... ps){return seg[p+n].get(ps...);}
    MAS void update(T x, int p, As... ps){
        p+=n; seg[p].update(x, ps...);
        for(p>>=1;p>=1;p>>=1)
            seg[p].update(S::op(seg[2*p].get(ps...),seg[2*p+1].get(ps
                ...)), ps...);
    }
    MAS T query(int l, int r, As... ps){
        T lv=S::id,rv=S::id;
        for(l+=n,r+=n+1;l<r;l>>=1,r>>=1){
            if (l&1)lv = S::op(lv,seg[l++].query(ps...));
            if (r&1)rv = S::op(seg[--r].query(ps...),rv);
        }
        return S::op(lv,rv);
    }
};

template<class S>
struct SegTree<0, S>{ using T = typename S::T;
    T val=S::id;
```

```
T get(){return val;}
void update(T x){val=x;}
T query(){return val;}
};

struct M{ //monoid
    using T = int;
    static constexpr T id = 0;
    static T op(T a, T b){return max(a,b);}
};

SparseTable.h
Description: Multidimensional Sparse Table Requires Idempotent Monoid S (op, inv, id)
Memory:  $\mathcal{O}((n \log n)^D)$ 
Time:  $\mathcal{O}(1)$  query,  $\mathcal{O}((n \log n)^D)$  build
c900f0, 39 lines
#define MAS template<class...As> //multiple arguments
template<int D, class S>
struct SpTable{ using T = typename S::T;
    using isp = SpTable<D-1, S>;
    inline int lg(signed x){return __builtin_clz(1)-__builtin_clz
        (x);}

    int n;
    vector<vector<isp>> tab;
    MAS SpTable(int s, As... ds):n(s),
        tab(1+lg(n),vector<isp>(n,isp(ds...))){}
    MAS void set(T x, int p, As... ps){tab[0][p].set(x, ps...);}
    void join(SpTable& a, SpTable& b){
        rep(i, 0, 1+lg(n))rep(j, 0, n)
            tab[i][j].join(a.tab[i][j], b.tab[i][j]);
    }
    void init(){
        rep(i, 0, n)tab[0][i].init();
        rep(i, 0, lg(n))rep(j, 0, n-(1<<i))
            tab[i+1][j].join(tab[i][j], tab[i][j+(1<<i)]);
    }
    MAS T query(int l, int r, As... ps){
        int k = lg(r-l+1); r+=1-(1<<k);
        return S::op(tab[k][l].query(ps...),tab[k][r].query(ps...))
        ;
    }
};

template<class S>
struct SpTable<0, S>{ using T = typename S::T;
    T val=S::id;
    void set(T x){val=x;}
    void join(SpTable& a, SpTable& b){val=S::op(a.val,b.val);}
    void init(){}
    T query(){return val;}
};

struct IM{
    using T = int;
    static constexpr T id = 0;
    static T op(T a, T b){return max(a, b);}
};

BIT.h
Description: Multidimensional BIT/Fenwick Tree Requires Abelian Group "S" (op, inv, id)
Memory:  $\mathcal{O}(N^D)$ 
Time:  $\mathcal{O}((\log N)^D)$ 
778135, 31 lines
#define MAS template<class... As> //multiple arguments
template<int D, class S>
```

```
struct BIT{ using T = typename S::T;
    int n;
    vector<BIT<D-1, S>> bit;
    MAs BIT(int s, As... ds):n(s),bit(n+1, BIT<D-1, S>(ds...)){}
    inline int lastbit(int x){return x&(-x);}
    MAs void add(T x, int p, As... ps){
        for(p++;p<=n;p+=lastbit(p))bit[p].add(x, ps...);
    }
    MAs T query(int l, int r, As... ps){
        T lv=S::id,rv=S::id; r++;
        for(;r>=1;r-=lastbit(r))rv=S::op(rv,bit[r].query(ps...));
        for(;l>=1;l-=lastbit(l))lv=S::op(lv,bit[l].query(ps...));
        return S::op(rv,S::inv(lv));
    }
};
```

```
template<class S>
struct BIT<0, S>{ using T = typename S::T;
    T val=S::id;
    void add(T x){val=S::op(val,x);}
    T query(){return val;}
};
```

```
struct AG{ //abelian group analogous to int addition
    using T = int;
    static constexpr T id = 0;
    static T op(T a, T b){return a+b;}
    static T inv(T a){return -a;}
};
```

MoQueries.h

Description: Solve queries offline Can be changed by modifying Spec

Time: $\mathcal{O}\left(n \times \sqrt{q}\right)$

```
cf6ecd, 23 lines
vi mo(vector<pii> Q, vector<int> V) { // Queries in Q are [l, r
    )
    int L = 0, R = 0, blk = 350; // N/sqrt (Q)

    vi s(sz(Q)), res = s;
    auto K = [&](pii x) {return pii(x.first/blk, x.second ^ -(x.
        first/blk & 1)); };
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });

    int sum = 0;
    auto add = [&](int ind, int end) { sum += V[ind]; }; // add
    a [ ind ] (end = 0 or 1)
    auto del = [&](int ind, int end) { sum -= V[ind]; }; //
    remove a [ ind ]
    auto calc = [&]() { return sum; }; //
    compute current answer

    for (int qi : s) {
        pii q = Q[qi];
        while (L > q.first) add(--L, 0);
        while (R < q.second) add(R++, 1);
        while (L < q.first) del(L++, 0);
        while (R > q.second) del(--R, 1);
        res[qi] = calc();
    }
    return res;
}
```

Math (3)

LinearDiophantineEquation.h

Description: Find a solution to equation $a \cdot x + b \cdot y = c$

Time: $\mathcal{O}\left(\log(a)\right)$

```
538f05, 14 lines
array<ll, 3> exgcd(ll a, ll b) {
    if (a == 0) return {0, 1, b};
    auto [x, y, g] = exgcd(b % a, a);
    return {y - b / a * x, x, g};
}

array<ll, 4> find_any_solution(ll a, ll b, ll c) {
    assert(a != 0 || b != 0);
    auto [x, y, g] = exgcd(a, b);
    if (c % g) return {false, 0, 0, 0};
    x *= c / g;
    y *= c / g;
    return {true, x, y, g};
}
```

String (4)

Hash.h

Description: String hashing

Memory: $\mathcal{O}(n)$

Time: $\mathcal{O}(1)$ query, $\mathcal{O}(n)$ build

```
<sys/time.h>
2d1ad6, 43 lines
/*
    Arithmetic mod two primes and 2^32 simultaneously.
    C can be inititalize to a number less than MOD or random

    timeval tp;
    gettimeofday(&tp, 0);
    C = (int)tp.tv_usec; // (less than modulo)
    assert((ull)(H(1)*2+1-3) == 0);
*/

typedef uint64_t ull;
template<int M, class B>
struct A {
    int x; B b; A(int x=0) : x(x), b(x) {}
    A(int x, B b) : x(x), b(b) {}
    A operator+(A o) const {int y = x+o.x; return{y - (y>=M)*M, b
        +o.b};}
    A operator-(A o) const {int y = x-o.x; return{y + (y< 0)*M, b
        -o.b};}
    A operator*(A o) const { return {(int)((ll)x*o.x % M), b*o.b}
        ; }
    explicit operator ull() const { return x ^ (ull) b << 21; }
    bool operator==(A o) const { return (ull)*this == (ull)o; }
    bool operator<(A o) const { return (ull)*this < (ull)o; }
};
typedef A<1000000007, A<1000000009, unsigned>> H;
```

static int C;

```
struct HashInterval {
    int n;
    vector<H> ha, pw;
    template<typename S>
    HashInterval(const S & str) : n(sz(str)), ha(n+1), pw(n+1) {
        pw[0] = 1;
        rep(i,0,n)
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    }
    H query(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
}
```

```
H queryI(int a, int b) {
    return query(n - b, n - a);
}
};
```

DinamicHashing.h

Description: Dinamic string hashing

Memory: $\mathcal{O}(n)$

Time: $\mathcal{O}(1)$ query, $\mathcal{O}(n)$ build

cf8a3c, 41 lines

```
typedef uint64_t ull;

template<int M, class B>
struct A {
    int x; B b;
    constexpr A(int x=0) : x(x), b(x) {}
    constexpr A(int x, B b) : x(x), b(b) {}
    A operator+(A o) const {int y = x+o.x; return{y - (y>=M)*M, b
        +o.b};}
    A operator-(A o) const {int y = x-o.x; return{y + (y< 0)*M, b
        -o.b};}
    A operator*(A o) const { return {(int)(1LL*x*o.x % M), b*o.b}
        ; }
    explicit operator ull() const { return x ^ (ull) b << 21; }
    bool operator==(A o) const { return (ull)*this == (ull)o; }
    bool operator<(A o) const { return (ull)*this < (ull)o; }
};
```

typedef A<989831283, A<912391239 , unsigned>> H;

const static int C = 12312;

```
struct DinamicHash {
    int n;
    vector<int> s;
    vector<H> p;
    SegTree seg;
    DinamicHash(const vector<int> & v) : n(v.size()), s(v), p(n
        +1) {
        p[0] = 1;
        vector<H> values(n);
        rep(i, 0, n) {
            values[i] = p[i] * s[i];
            p[i+1] = p[i] * C;
        }
        seg = SegTree(values);
    };
    H query(int l, int r) const { // [l, r)
        return seg.query(l, r) * p[n-r];
    }
    void update(int idx, int v) {
        s[idx] = v;
        seg.update(idx, p[idx] * s[idx]);
    }
};
```

Automata.h

Description: Suffix automata

Memory: $\mathcal{O}(n * 26)$

Time: $\mathcal{O}(n)$ build

92d90c, 49 lines

```
struct Automata {
    int saID = 1, last = 1;
    int n;
    vector<int> len, lnk;
    vector<array<int,27>> to;
    vector<int> occ, fpos;
    vector<int> states;
```

```
Automata(const string & s, const char a = 'a')
: n(s.size()), len(2*n+2), lnk(2*n+2), to(2*n+2, {0}), occ
(2*n+2), fpos(2*n+2) {
    for (const auto & c: s) push(c-a);

    states.assign(saID, 0);
    iota(all(states), 1);
    sort(all(states), [&](const auto & u, const auto & v) {
        return len[u] > len[v]; });
    for (auto st: states) {
        occ[lnk[st]] += occ[st];
    }
}

void push(int c) {
    int a = ++saID;
    int p = last;
    last = a;

    len[a] = len[p] + 1;
    occ[a] = 1;
    fpos[a] = len[a] - 1;

    for (; p > 0 && !to[p][c]; p = lnk[p]) to[p][c] = a;
    int q = to[p][c];
    if (p == 0) {
        lnk[a] = 1;
    }
    else if (len[p] + 1 == len[q]) {
        lnk[a] = q;
    }
    else {
        int clone = ++saID;
        lnk[clone] = lnk[q];
        to[clone] = to[q];
        fpos[clone] = fpos[q];

        len[clone] = len[p] + 1;
        lnk[a] = lnk[q] = clone;
        for (; to[p][c] == q; p = lnk[p]) to[p][c] = clone;
    }
}
};
```

KMP.h

Description: KMP automaton

Memory: $\mathcal{O}(26n)$

Time: $\mathcal{O}(n)$ build, $\mathcal{O}(1)$ query

32c2a7, 22 lines

```
struct KMP{
    const int inic = 'a';
    int n; vi pi; vector<vi> aut;
    KMP(string s):n(s.size()),pi(n),aut(n+1, vi(26,0)){
        rep(i, 1, n){
            int j = pi[i-1];
            while(j > 0 and s[j]!=s[i])j = pi[i-1];
            pi[i] = j + (s[i]==s[j]);
        }
        s.pb('#');
        rep(i, 0, n+1){
            rep(c, 0, 26){
                if (i > 0 and c+inic!=s[i])aut[i][c] = aut[pi[i-1]][c];
                else aut[i][c] = i + (c+inic == s[i]);
            }
        }
    }
    pair<bool, int> nxt(int cur, char c){
        cur = aut[cur][c];
        return {cur==n, cur};
    }
};
```

```

    }
};

Graph (5)

Kosaraju.h
Description: Kosaraju
Memory:  $\mathcal{O}(n)$ 
Time:  $\mathcal{O}(n+m)$  query,  $\mathcal{O}(n+m)$  build
eb04c1, 43 lines

struct Kosaraju {
    int n;
    vector<vector<int>>> g, rg;
    vector<bool> vis;
    vector<int> id;
    vector<vector<int>>> dag, comp;
    int cc = 0;
    vector<int> S;

    Kosaraju(int _n)
        : n(_n), g(n), rg(n), vis(n), id(n) {}

    void add_edge(int a, int b) {
        g[a].eb(b);
        rg[b].eb(a);
    }
    void dfs(int a) {
        vis[a] = true;
        for (auto b: g[a]) if (!vis[b]) dfs(b);
        S.eb(a);
    }
    void scc(int a, int c) {
        vis[a] = true;
        id[a] = c;
        for (auto b: rg[a]) if (!vis[b]) scc(b, c);
    }
    void run() {
        rep(a, 0, n) if (!vis[a]) dfs(a);
        fill(all(vis), 0);
        reverse(all(S));
        for (auto a: S) if (!vis[a]) scc(a, cc++);

        dag.resize(cc); comp.resize(cc);
        vector<pair<int, int>> edges;
        rep(a, 0, n) {
            comp[id[a]].eb(a);
            for (int b: g[a]) if (id[a] != id[b]) edges.eb(id[a], id[
                b]);
        }
        sort(all(edges));
        edges.erase(unique(all(edges)), edges.end());
        for (const auto & [a, b]: edges) dag[a].eb(b);
    }
};

TwoSat.h
Description: Two sat
Memory:  $\mathcal{O}(n)$ 
Time:  $\mathcal{O}(n+m)$  query,  $\mathcal{O}(n+m)$  build
0b8692, 64 lines

struct TwoSat{
    int n;
    vector<vector<int>>> g, gi;
    vector<bool> vis;
    vector<int> vars, comp;
    vector<int> top;
    TwoSat(int _n)
        : n(_n), g(2*n), gi(2*n), vis(2*n), vars(n, -1), comp(2*n)
```

```

    {}

    int neg(int a) {
        if (a >= n) return a-n;
        return a + n;
    }

    void add_or(int a, int b) {
        g[neg(a)].eb(b);
        g[neg(b)].eb(a);

        gi[b].eb(neg(a));
        gi[a].eb(neg(b));
    }

    void add_imp(int a, int b) {
        add_or(neg(a), b);
    }

    void add_either(int a, int b) {
        add_or(a, b);
        add_or(neg(a), neg(b));
    }

    void dfs(int a) {
        vis[a] = true;
        for (auto b: g[a]) if (!vis[b]) dfs(b);
        top.eb(a);
    }

    void idfs(int a, int c){
        vis[a] = true;
        comp[a] = c;
        for (auto b: gi[a]) if (!vis[b]) idfs(b, c);
    }

    bool sat() {
        int c = 0;
        rep(a, 0, 2*n) if (!vis[a]) dfs(a);

        fill(all(vis), 0);
        reverse(all(top));
        for(int a : top) if (!vis[a]) idfs(a, c++);

        for(int a: top){
            if (comp[a] == comp[neg(a)]) return false;

            bool is_neg = a >= n;
            if (is_neg) a = neg(a);

            if (vars[a] == -1) vars[a] = is_neg;
        }

        return true;
    }
};

OnlineMatching.h
Description: Modified khun developed for specific question able to run
2 * 106 queries, in 2 * 106 x 106 graph in 3 seconds codeforces
Time:  $\mathcal{O}(confia)$ 
6ac539, 42 lines

struct OnlineMatching {
    int n = 0, m = 0;
    vector<int> vis, match, dist;
    vector<vector<int>>> g;
    vector<int> last;
    int t = 0;
```

```
OnlineMatching(int n_, int m_) : n(n_), m(m_),
vis(n, 0), match(m, -1), dist(n, n+1), g(n), last(n, -1)
{}

void add(int a, int b) {
    g[a].pb(b);
}

bool kuhn(int a) {
    vis[a] = t;
    for(int b: g[a]) {
        int c = match[b];
        if (c == -1) {
            match[b] = a;
            return true;
        }
        if (last[c] != t || (dist[a] + 1 < dist[c]))
            dist[c] = dist[a] + 1, last[c] = t;
    }
    for (int b: g[a]) {
        int c = match[b];
        if (dist[a] + 1 == dist[c] && vis[c] != t && kuhn(c)) {
            match[b] = a;
            return true;
        }
    }
    return false;
}

bool can_match(int a) {
    t++;
    last[a] = t;
    dist[a] = 0;
    return kuhn(a);
}
};
```

FuncGraph.h
Description: Functional Graph
Memory: $\mathcal{O}(n)$
Time: $\mathcal{O}(n)$

ceceb3, 25 lines

```
struct FuncGraph{
    int n;
    vector<vi> gr, comps;
    vi head;

    FuncGraph(vi& fn):
        n(sz(fn)), gr(n), head(n, -1){
            rep(i, 0, n)gr[fn[i]].pb(i);
            vi visited(n, 0);
            auto dfs = [&](auto rec, int v, int c) -> void{
                head[v] = c; visited[v] = 1;
                for(int f : gr[v])if (head[f]!=f)rec(rec, f, c);
            };
            rep(i, 0, n){
                if (visited[i])continue;
                int l=fn[i], r=fn[fn[i]];
                while(l!=r) l=fn[l],r=fn[fn[r]];
                vi cur = {r}; l = fn[l];
                for(;l!=r;l=fn[l]) cur.pb(l);
                for(int x : cur) head[x] = x;
                for(int x : cur) dfs(dfs, x, x);
                comps.pb(cur);
            }
        }
};
```

DP (6)

```
SOS.h
Description: Sos DP
Time:  $\mathcal{O}(n * 2^n)$ 
5063f0, 19 lines

//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
    dp[mask][~1] = A[mask]; //handle base case separately (leaf states)
    for(int i = 0;i < N; ++i){
        if(mask & (1<<i))
            dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
        else
            dp[mask][i] = dp[mask][i-1];
    }
    F[mask] = dp[mask][N-1];
}

//memory optimized, super easy to code.
for(int i = 0; i<(1<<N); ++i)
    F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N); ++mask){
    if(mask & (1<<i))
        F[mask] += F[mask^(1<<i)];
}
```

Techniques (A)

techniques.txt	117 lines
Recursion	
Divide and conquer	
Finding interesting points in N log N	
Algorithm analysis	
Master theorem	
Amortized time complexity	
Greedy algorithm	
Max contiguous subvector sum	
Invariants	
Graph theory	
DP, com cyclo no dikstra reverso	
Breadth first search	
Depth first search	
DFS trees	
Dijkstra's algorithm	
MST: Prim's algorithm	
Bellman-Ford	
Min-cost max flow	
Floyd-Warshall	
Euler cycles	
Flow networks	
Bipartite matching	
Topological sorting	
Strongly connected components	
2-SAT	
Cut vertices, cut-edges and biconnected components	
Vertex coloring	
Bipartite graphs	
Diameter and centroid	
K'th shortest path	
Shortest cycle	
Dynamic programming	
Knapsack	
Longest common subsequence	
Longest increasing subsequence	
Number of paths in a dag	
Shortest path in a dag	
over intervals	
over subsets	
over probabilities	
over trees	
3^n set cover	
Divide and conquer	
Convex hull optimizations	
RMQ (sparse table a.k.a 2^k-jumps)	
Combinatorics	
Inclusion/exclusion	
Catalan number	
Pick's theorem	
Number theory	
Integer parts (School's excursion)	
Divisibility	
Euclidean algorithm	
Modular inverses	
Modular exponentiation by squaring	
Chinese remainder theorem	
Fermat's little theorem	
Euler's theorem	
Phi function	
Frobenius number	
Quadratic reciprocity	
Pollard-Rho	
Miller-Rabin	
Hensel lifting	
Vieta root jumping	

Game theory
Combinatorial games
Game trees
Mini-max
Nim
Games on graphs
Games on graphs with loops
Grundy numbers
Bipartite games without repetition
General games without repetition
Alpha-beta pruning
Probability theory
Optimization
Binary search
Ternary search (Convex functions)
Binary search on derivative
Numerical methods
Newton's method
Root-finding with binary/ternary search
Matrices
Gaussian elimination
Exponentiation by squaring
Geometry
Cross product
Scalar product
Convex hull
Polygon cut
Closest pair (Distance functions)
Hull diameter (Distance functions)
Sweeping
Discretization (convert to events and sweep)
Angle sweeping
Line sweeping
Strings
Longest common substring
Knuth-Morris-Pratt
Tries
Rolling polynomial hashes
Aho-Corasick
Letter position lists
Combinatorial search
Meet in the middle
Brute-force with pruning
Data structures
LCA (2^k-jumps in trees in general)
HLD
Centroid decomposition
SegTree, LazySeg
Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
Monotone queues / monotone stacks / sliding queues
Sliding queue using 2 stacks
Persistent segment tree