



Chiang Mai University

PramorKangDamCPP

Suthana Kwaonueng, Theerada Siri, Nitikhon Chantatham

2024-09-06

- 1 Contest
- 2 Tanya
- 3 cellul4r
- 4 Mathematics
- 5 Data structures
- 6 Graph
- 7 Strings

Contest (1)

template.cpp86 lines

```
#include<bits/stdc++.h>
using namespace std;
void __print(int x) {cerr << x;}
void __print(long x) {cerr << x;}
void __print(long long x) {cerr << x;}
void __print(unsigned x) {cerr << x;}
void __print(unsigned long x) {cerr << x;}
void __print(unsigned long long x) {cerr << x;}
void __print(float x) {cerr << x;}
void __print(double x) {cerr << x;}
void __print(long double x) {cerr << x;}
void __print(char x) {cerr << '\'' << x << '\'';}
void __print(const char *x) {cerr << '\"' << x << '\"'};
void __print(const string &x) {cerr << '\"' << x << '\"'};
void __print(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>
void __print(const pair<T, V> &x);
template<typename T>
void __print(const T &x) {int f = 0; cerr << '{'; for (auto &i:
    x) cerr << (f++ ? ", " : ""), __print(i); cerr << "}";}
template<typename T, typename V>
void __print(const pair<T, V> &x) {cerr << '{'; __print(x.first
    ); cerr << ", "; __print(x.second); cerr << '}'};
void __print() {cerr << "]\n";}
template <typename T, typename... V>
void __print(T t, V... v) {__print(t); if (sizeof...(v)) cerr <<
    ", "; __print(v...);}

//#ifdef DEBUG
#define dbg(x...) cerr << "\e[91m"<<__func__<<":"<<__LINE__<<"
    [" << #x << "] = ["; __print(x); cerr << "\e[39m" << endl;
//#else
//#define dbg(x...)
//#endif

typedef long long ll;
typedef long double ld;
typedef complex<ld> cd;

typedef pair<int, int> pi;
typedef pair<ll,ll> pl;
typedef pair<ld,ld> pd;

typedef vector<int> vi;
typedef vector<ld> vd;
typedef vector<ll> vl;
```

11621

```
typedef vector<pi> vpi;
typedef vector<pl> vpl;
typedef vector<cd> vcd;

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

#define rep(i, a) for(int i=0;i<a;++i)
#define FOR(i, a, b) for (int i=a; i<(b); i++)
#define FORi(i, a) for (int i=0; i<(a); i++)
#define FORd(i,a,b) for (int i = (b)-1; i >= a; i--)
#define FORd(i,a) for (int i = (a)-1; i >= 0; i--)
#define trav(a,x) for (auto& a : x)
#define uid(a, b) uniform_int_distribution<int>(a, b)(rng)

#define sz(x) (int)(x).size()
#define mp make_pair
#define pb push_back
// #define f first
// #define s second
#define lb lower_bound
#define ub upper_bound
#define all(x) x.begin(), x.end()
#define ins insert

template<class T> bool ckmin(T& a, const T& b) { return b < a ?
    a = b, 1 : 0; }
template<class T> bool ckmax(T& a, const T& b) { return a < b ?
    a = b, 1 : 0; }

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());

const char nl = '\n';
const int N = 2e5+1;
const int INF = 1e9+7;
const long long LINF = 1e18+7;

void solve(){
}

int main(){
    ios::sync_with_stdio(false);cin.tie(nullptr);
    int t = 1;
    cin>>t;
    while(t--) solve();
}

.vimrc21 lines

"General editor settings
set tabstop=4
set nocompatible
set shiftwidth=4
set expandtab
set autoindent
set smartindent
set ruler
set showcmd
set incsearch
set shellslash
set number
set relativenumber
set cino+=L0

"keybindings for { completion, "jk" for escape, ctrl-a to
    select all
inoremap {<CR> {<CR><Esc>O
```

inoremap {} {}
imap jk <Esc>
map <C-a> <esc>ggVG<CR>
set belloff=all

.bashrc1 lines

export PATH=\$PATH:~/scripts/

build.sh1 lines

g++ -static -DLOCAL -lm -s -x c++ -Wall -Wextra -O2 -std=c++17
 -o \$1 \$1.cpp

stress.sh22 lines

#!/usr/bin/env bash

for ((testNum=0;testNum<\$4;testNum++))
do
 ./\$3 > input
 ./\$2 < input > outSlow
 ./\$1 < input > outWrong
 H1='md5sum outWrong'
 H2='md5sum outSlow'
 if !(cmp -s "outWrong" "outSlow")
 then
 echo "Error found!"
 echo "Input:"
 cat input
 echo "Wrong Output:"
 cat outWrong
 echo "Slow Output:"
 cat outSlow
 exit
 fi
done
echo Passed \$4 tests

Tanya (2)

SegmentTree.h
Description: Segment tree with point update for range sum
Time: O(log N)efd738, 29 lines

//TODO: use 0 base indexing
vector<long long>tree;
void update(int node,int n_l,int n_r,int q_i,long long value){
 if(n_r<q_i || q_i<n_l)return;
 if(q_i==n_l && n_r==q_i){
 tree[node] = value;
 return;
 }
 int mid = (n_r+n_l)/2;
 update(2*node,n_l,mid,q_i,value);
 update(2*node+1,mid+1,n_r,q_i,value);
 tree[node] = tree[2*node] + tree[2*node+1];
}

long long f(int node,int n_l,int n_r,int q_l,int q_r){
 if(n_r<q_l || q_r<n_l)return 0;
 if(q_l<=n_l && n_r<=q_r)return tree[node];
 int mid = (n_l+n_r)/2;
 return f(2*node,n_l,mid,q_l,q_r) + f(2*node+1,mid+1,n_r,q_l
 ,q_r);
}

void build_tree(vi &a,int n){

```
tree.clear();
int m=n;
while(__builtin_popcount(m)!=1)++m;
tree.resize(2*m+1,0);
for(int i=0;i<n;++i)tree[i+m]=a[i];
for(int i=m-1;i>=1;--i)tree[i]=tree[2*i]+tree[2*i+1];
}
```

LazySegmentTree.h

Description: Segment tree with lazy propagation update for range sum
Time: $\mathcal{O}(\log N)$.

2f108e, 51 lines

```
//TODO: use 0 base indexing
vector<long long> tree,lazy;
void update(int node,int n_l,int n_r,int q_l,int q_r,int value)
{
    if(lazy[node]!=0){
        tree[node]+=(long long)(n_r-n_l+1)*lazy[node];
        // for range + update
        if(n_l!=n_r){
            lazy[2*node]+=lazy[node];
            lazy[2*node+1]+=lazy[node];
        }
        lazy[node] = 0;
    }
    if(n_r<q_l || q_r<n_l)return;
    if(q_l<=n_l && n_r<=q_r){
        tree[node]+=(long long)(n_r-n_l+1)*value;
        // for range + update
        if(n_l!=n_r){
            lazy[2*node]+=value;
            lazy[2*node+1]+=value;
        }
        return;
    }
    int mid = (n_r+n_l)/2;
    update(2*node,n_l,mid,q_l,q_r,value);
    update(2*node+1,mid+1,n_r,q_l,q_r,value);
    tree[node] = tree[2*node] + tree[2*node+1];
}

long long f(int node,int n_l,int n_r,int q_l,int q_r){
    if(lazy[node]!=0){
        tree[node]+=(long long)(n_r-n_l+1)*lazy[node];
        if(n_l!=n_r){
            lazy[2*node] += lazy[node];
            lazy[2*node+1] += lazy[node];
        }
        lazy[node] = 0;
    }
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l<=n_l && n_r<=q_r)return tree[node];
    int mid = (n_l+n_r)/2;
    return f(2*node,n_l,mid,q_l,q_r) + f(2*node+1,mid+1,n_r,q_l,q_r);
}

void build_tree(vi &a,int n){
    tree.clear(); lazy.clear();
    int m=n;
    while(__builtin_popcount(m)!=1)++m;
    tree.resize(2*m+1,0); lazy.resize(2*m+1,0);
    for(int i=0;i<n;++i)tree[i+m]=a[i];
    for(int i=m-1;i>=1;--i)tree[i]=tree[2*i]+tree[2*i+1];
}
```

OrderStatisticTree.h

Description: find nth largest element, count elements strictly less than x
Time: $\mathcal{O}(\log N)$

<ext/pb.ds/assoc.container.hpp>, <ext/pb.ds/tree.policy.hpp> 033b41, 7 lines

```
typedef __gnu_pbds::tree<int, __gnu_pbds::null_type, less<int>,
__gnu_pbds::rb_tree_tag, __gnu_pbds::
tree_order_statistics_node_update> ordered_set;

ordered_set st;
//st.order_of_key(x) - find # of elements in st strictly less
than x
//st.size() - size of st
//st.find_by_order(x) - return iterator to the x-th largest
element
//st.clear() - clear container
```

MergeSortTree.h

Description: do the same with orderstatistic tree but now over interval can be used/modify for some possible interval/subbarray queries

Time: $\mathcal{O}(\log N^3)$
<ext/pb.ds/assoc.container.hpp>, <ext/pb.ds/tree.policy.hpp>, <ext/pb.ds/assoc.container.hpp>, <ext/pb.ds/tree.policy.hpp> 2d0612, 94 lines

```
//merge sort tree with fenwick tree
typedef __gnu_pbds::tree<pair<int,int>, __gnu_pbds::null_type,
less<pair<int,int>>, __gnu_pbds::rb_tree_tag, __gnu_pbds::
tree_order_statistics_node_update> ordered_set;
ordered_set st;
//st.order_of_key({x,-1}) - find # of elements in st stricly
less than x
//st.clear() - clear container

vector<ordered_set> mtree;

ordered_set merge(ordered_set &a, ordered_set &b){
    ordered_set result;
    for(auto&p:a){
        result.ins(p);
    }
    for(auto&p:b){
        result.ins(p);
    }
    return result;
}

void update(int node,int n_l,int n_r,int q_i,int id,int old_val
,int value){
    if(n_r<q_i || q_i<n_l)return;
    if(q_i==n_l && n_r==q_i){
        auto it=mtree[node].find({old_val,id});
        mtree[node].erase(it);
        mtree[node].ins({value,id});
        return;
    }
    int mid = (n_r+n_l)/2;
    update(2*node,n_l,mid,q_i,id,old_val,value);
    update(2*node+1,mid+1,n_r,q_i,id,old_val,value);
    auto it=mtree[node].find({old_val,id});
    if(it!=mtree[node].end()){
        mtree[node].erase(it);
        mtree[node].ins({value,id});
    }
}

int f(int node,int n_l,int n_r,int q_l,int q_r, int value){
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l<=n_l && n_r<=q_r){
        return mtree[node].order_of_key({value,-1});
    }
}
```

```
int mid = (n_l+n_r)/2;
return f(2*node,n_l,mid,q_l,q_r,value)+f(2*node+1,mid+1,n_r,q_l,q_r,value);
}

void build_mtree(vi &a){
    int n=(int)a.size();
    int m=n; while(__builtin_popcount(m)!=1)++m;
    //for(int i=0;i<2*m;++i)mtree[i].clear();
    mtree.resize(2*m);
    for(int i=0;i<n;++i)mtree[i+m].ins({a[i],i});
    for(int i=m-1;i>=1;--i)mtree[i]=merge(mtree[2*i],mtree[2*i+1]);
}
//merge sort tree with fenwick tree(BIT) (4 times less space)

typedef __gnu_pbds::tree<pair<int,int>, __gnu_pbds::null_type,
less<pair<int,int>>, __gnu_pbds::rb_tree_tag, __gnu_pbds::
tree_order_statistics_node_update> ordered_set;
ordered_set st;
//st.order_of_key(x) - find # of elements in st stricly less
than x

//TODO: use 1 base indexing
vector<ordered_set>bit;

void update(int i,int k,int old_value, int new_value){
    while(i<(int)bit.size()){
        auto it=bit[i].find({old_value,k});
        assert(it!=bit[i].end());
        if(it!=bit[i].end()){
            bit[i].erase(it);
        }
        bit[i].ins({new_value,k});
        i+=i&-i;//add last set bit
    }
}

int F(int i, int k){//culmulative sum to ith data
    int sum=0;
    while(i>0){
        sum+=bit[i].order_of_key({k,-1});
        i-=i&-i;
    }
    return sum;
}

void build_bit(vi &a){
    bit.resize((int)a.size());
    for(int i=1;i<(int)a.size();++i)bit[i].ins({a[i],i});
    for(int i=1;i<(int)bit.size();++i){
        int p=i+(i&-i);//index to parent
        if(p<(int)bit.size()){
            for(auto&x:bit[i])bit[p].ins(x);
        }
    }
}
```

MoQueries.h

Description: answering offline quries
Time: $\mathcal{O}\left((N+Q)\sqrt{N}\right)$

9df2fb, 46 lines

```
/* TODO: use 0 based indexing*/
void remove(int idx); // TODO: remove value at idx from data
structure
void add(int idx); // TODO: add value at idx from data
structure
int get_answer(); // TODO: extract the current answer of the
data structure
```

```
int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
            make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query>& queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range
    // [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

```
FenwickTree.h
Description: find culmulative sum to ith element
Time: O(log N)
a9def9, 27 lines

//TODO: use 1 base indexing
vector<long long>bit;

//range sum point update(k=new_val-old_val)
void add(int i, int k){//add k to ith data and it's parent
    range so on
    while(i<(int)bit.size()){
        bit[i]+=k;
        i+=i&-i;//add last set bit
    }
}

11 sum(int i){//culmulative sum to ith data
    11 sum=0;
    while(i>0){
        sum+=bit[i];
        i-=i&-i;
    }
    return sum;
}
```

```
void build_bit(vl &a){
    bit=a;
    for(int i=1;i<(int)bit.size();++i){
        int p=i+(i&-i);//index to parent
        if(p<(int)bit.size())bit[p]+=bit[i];
    }
}

RMQ.h
Description: Range Minimum Queries on an array. solving offline queries
in O(1)
Time: build O(N log N) query O(1)
e37d08, 24 lines

int rmq[N][20];

void build_rmq(vi &a){
    for(int j=0;j<20;++j){
        for(int i=0;i<(int)a.size();++i){
            if(j==0){
                rmq[i][0]=a[i];
            } else if(i+(1<<(j-1))<(int)a.size()){
                rmq[i][j]=min(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]);
            }
        }
    }
}

int query(int l, int r){
    int i=l,sub_array_size=r-l+1, ans=INF;
    for(int j=0;j<30;++j){
        if((1<<j)&(sub_array_size)){
            ans=min(ans,rmq[i][j]);
            i+=(1<<j);
        }
    }
    return ans;
}
```

```
HashMap.h
Description: Hash map with mostly the same API as unordered_map, but
~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if
provided).
<ext/pb_ds/assoc.container.hpp>
3b295b, 7 lines

using namespace __gnu_pbds;

const int RANDOM = chrono::high_resolution_clock::now().
    time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;

DSU.h
Description: Disjoint-set data structure.
Time: O(log N)
7efb0c, 22 lines

//TODO: initialized parent[] and _rank[] array
int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void make_set(int v) {
    parent[v] = v;
    _rank[v] = 1;
}
```

```
void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (_rank[a] < _rank[b])
            swap(a, b);
        parent[b] = a;
        _rank[a] += _rank[b];
    }
}

Geometry.h
Description: pt, line, polygon, circle
963fd3, 167 lines

//Geometry
#pragma GCC target("avx2")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")

typedef long long int ll;
typedef long double ld;
typedef complex<ld> pt;
struct line {
    pt P, D; bool S = false;
    line(pt p, pt q, bool b = false) : P(p), D(q - p), S(b) {}
    line(pt p, ld th) : P(p), D(polar((ld)1, th)) {}
};
struct circ { pt C; ld R; };

#define X real()
#define Y imag()
#define CRS(a, b) (conj(a) * (b)).Y //scalar cross product
#define DOT(a, b) (conj(a) * (b)).X //dot product
#define U(p) ((p) / abs(p)) //unit vector in direction of p (
//don't use if Z(p) == true)
#define Z(x) (abs(x) < EPS)
#define A(a) (a).begin(), (a).end() //shortens sort(),
//upper_bound(), etc. for vectors

//constants (INF and EPS may need to be modified)
ld PI = acosl(-1), INF = 1e20, EPS = 1e-12;
pt I = {0, 1};

//true if d1 and d2 parallel (zero vectors considered parallel
//to everything)
bool parallel(pt d1, pt d2) { return Z(d1) || Z(d2) || Z(CRS(U(
d1), U(d2))); }

//"above" here means if l & p are rotated such that l.D points
//in the +x direction, then p is above l. Returns arbitrary
//boolean if p is on l
bool above_line(pt p, line l) { return CRS(p - l.P, l.D) > 0; }

//true if p is on line l
bool on_line(pt p, line l) { return parallel(l.P - p, l.D) &&
    (l.S || DOT(l.P - p, l.P + l.D - p) <= EPS); }

//returns 0 for no intersection, 2 for infinite intersections,
//1 otherwise. p holds intersection pt
11 intsct(line l1, line l2, pt& p) {
    if(parallel(l1.D, l2.D)) //note that two parallel segments
        sharing one endpoint are considered to have infinite
        intersections here
        return 2 * (on_line(l1.P, l2) || on_line(l1.P + l1.D, l2)
            || on_line(l2.P, l1) || on_line(l2.P + l2.D, l1));
    pt q = l1.P + l1.D * CRS(l2.D, l2.P - l1.P) / CRS(l2.D, l1.D)
        ;
    if(on_line(q, l1) && on_line(q, l2)) { p = q; return 1; }
    return 0;
}
```

```
}

//closest pt on l to p
pt cl_pt_on_l(pt p, line l) {
    pt q = l.P + DOT(U(l.D), p - l.P) * U(l.D);
    if(on_line(q, l)) return q;
    return abs(p - l.P) < abs(p - l.P - l.D) ? l.P : l.P + l.D;
}

//distance from p to l
ld dist_to(pt p, line l) { return abs(p - cl_pt_on_l(p, l)); }

//p reflected over l
pt refl_pt(pt p, line l) { return conj((p - l.P) / U(l.D)) * U(l.D) + l.P; }

//ray r reflected off l (if no intersection, returns original ray)
line reflect_line(line r, line l) {
    pt p; if(intsct(r, l, p) - 1) return r;
    return line(p, p + INF * (p - refl_pt(r.P, l)), l);
}

//altitude from p to l
line alt(pt p, line l) { l.S = 0; return line(p, cl_pt_on_l(p, l)); }

//angle bisector of angle abc
line ang_bis(pt a, pt b, pt c) { return line(b, b + INF * (U(a - b) + U(c - b)), l); }

//perpendicular bisector of l (assumes l.S == 1)
line perp_bis(line l) { return line(l.P + l.D / (ld)2, arg(l.D * I)); }

//orthocenter of triangle abc
pt orthocent(pt a, pt b, pt c) { pt p; intsct(alt(a, line(b, c)), alt(b, line(a, c)), p); return p; }

//incircle of triangle abc
circ incirc(pt a, pt b, pt c) {
    pt cent; intsct(ang_bis(a, b, c), ang_bis(b, a, c), cent);
    return {cent, dist_to(cent, line(a, b))};
}

//circumcircle of triangle abc
circ circumcirc(pt a, pt b, pt c) {
    pt cent; intsct(perp_bis(line(a, b, l)), perp_bis(line(a, c, l)), cent);
    return {cent, abs(cent - a)};
}

//is pt p inside the (not necessarily convex) polygon given by poly
bool in_poly(pt p, vector<pt>& poly) {
    line l = line(p, {INF, INF * PI}, l);
    bool ans = false;
    pt lst = poly.back(), tmp;
    for(pt q : poly) {
        line s = line(q, lst, l); lst = q;
        if(on_line(p, s)) return false; //change if border included
        else if(intsct(l, s, tmp)) ans = !ans;
    }
    return ans;
}

//area of polygon, vertices in order (cw or ccw)
ld area(vector<pt>& poly) {
    ld ans = 0;
```

```
pt lst = poly.back();
for(pt p : poly) ans += CRS(lst, p), lst = p;
return abs(ans / 2);
}

//perimeter of polygon, vertices in order (cw or ccw)
ld perim(vector<pt>& poly) {
    ld ans = 0;
    pt lst = poly.back();
    for(pt p : poly) ans += abs(lst - p), lst = p;
    return ans;
}

//centroid of polygon, vertices in order (cw or ccw)
pt centroid(vector<pt>& poly) {
    ld area = 0;
    pt lst = poly.back(), ans = {0, 0};
    for(pt p : poly) {
        area += CRS(lst, p);
        ans += CRS(lst, p) * (lst + p) / (ld)3;
        lst = p;
    }
    return ans / area;
}

//invert a point over a circle (doesn't work for center of circle)
pt circInv(pt p, circ c) {
    return c.R * c.R / conj(p - c.C) + c.C;
}

//vector of intersection pts of two circs (up to 2) (if circles same, returns empty vector)
vector<pt> intsctCC(circ c1, circ c2) {
    if(c1.R < c2.R) swap(c1, c2);
    pt d = c2.C - c1.C;
    if(Z(abs(d) - c1.R - c2.R)) return {c1.C + polar(c1.R, arg(c2.C - c1.C))};
    if(!Z(d) && Z(abs(d) - c1.R + c2.R)) return {c1.C + c1.R * U(d)};
    if(abs(abs(d) - c1.R) >= c2.R - EPS) return {};
    ld th = acos1((c1.R * c1.R + norm(d) - c2.R * c2.R) / (2 * c1.R * abs(d)));
    return {c1.C + polar(c1.R, arg(d) + th), c1.C + polar(c1.R, arg(d) - th)};
}

//vector of intersection pts of a line and a circ (up to 2)
vector<pt> intsctCL(circ c, line l) {
    vector<pt> v, ans;
    if(parallel(l.D, c.C - l.P)) v = {c.C + c.R * U(l.D), c.C - c.R * U(l.D)};
    else v = intsctCC(c, circ(refl_pt(c.C, l), c.R));
    for(pt p : v) if(on_line(p, l)) ans.push_back(p);
    return ans;
}

//external tangents of two circles (negate c2.R for internal tangents)
vector<line> circTangents(circ c1, circ c2) {
    pt d = c2.C - c1.C;
    ld dr = c1.R - c2.R, d2 = norm(d), h2 = d2 - dr * dr;
    if(Z(d2) || h2 < 0) return {};
    vector<line> ans;
    for(ld sg : {-1, 1}) {
        pt u = (d * dr + d * I * sqrt(h2) * sg) / d2;
        ans.push_back(line(c1.C + u * c1.R, c2.C + u * c2.R, l));
    }
    if(Z(h2)) ans.pop_back();
}
```

```
return ans;
}

KMP.h
Description: pattern searching
Time:  $\mathcal{O}(N + M)$ 
4dfec5, 37 lines

int b[N];
int cnt = 0;

void knp_proc(string t, string p){
    int i=0, j=-1; b[0] = -1;
    while(i<(int)p.length()){
        while(j>=0 && p[i]!=p[j]) j =b[j];
        ++i; ++j;
        b[i] = j;
    }
}

void knp_search(string t, string p){ //count number of occurence of p in t
    int i=0, j=0;
    while(i<(int)t.length()){
        while(j>=0 && t[i] != p[j]) j = b[j];
        ++i; ++j;
        if(j==(int)p.length()){
            ++cnt;
            j = b[j];
        }
    }
}

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

StringHashing.h
Description: check equality of two substrings
Time:  $\mathcal{O}(N)$  preprocessing  $\mathcal{O}(1)$  query
276f01, 25 lines

typedef long long ll;
typedef pair<ll, ll> pl;
#define M 1000000321
#define OP(x, y) pl operator x (pl a, pl b){return {a.first x b.first, (a.second y b.second) % M }; }
OP(+, +) OP(*, *) OP(-, + M -)
//random number generator
mt19937 gen(chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<ll> dist(256, M - 1);
//queries - check if  $S[i:i+l] == S[j:j+l]$  (inclusive), S is a string, [:] is slice
#define H(i, l) (h[i] + (l)) - h[i] * p[l]
#define EQ(i, j, l) (H(i, l) == H(j, l))
//preprocessing
const int N = 2e5;
string s;
pl p[N], h[N];
ll n;
```

```
int main(){
    cin >> n >> s;
    p[0] = {1,1}, p[1] = {dist(gen) | 1, dist(gen) };
    for(ll i=1;i<=(ll)s.length();++i){
        p[i] = p[i-1] * p[1];
        h[i] = h[i-1] * p[1] + make_pair(s[i-1], s[i-1]);
    }
}
```

DnQDp.h

Description: find best k consecutive subarray partition

Time: $\mathcal{O}(N \log N)$

108b88, 55 lines

```
//TODO: initialize dp , initialize cost() function of use slide
()
//ll dp[N][M];
```

```
//generic implementation for sliding range technique for logn
cost()
//(persistent segtree alternative)
ll ccost = 0;
int cl = 0, cr = -1;
void slide(int l, int r){
    while(cr < r){
        ++cr;
        //add();
        //...
    }
    while(cl > l){
        --cl;
        //add();
        //...
    }
    while(cr > r){
        //remove();
        //...
        --cr;
    }
    while(cl < l){
        //remove();
        //...
        ++cl;
    }
}
```

```
void compute(int l, int r, int optl, int optr, int j){
    if(l>r) return;

    int mid = (l+r)>>1;
    //pair<ll, int> best = {0,-1};
    //pair<ll, int> best = {LINF,-1};

    //dp is satisfy quadrangle IE if cost() satisfy qudrangle
    IE
    //if cost() is QF => opt() is nondecreasing
    for(int k=optl;k<=min(mid,optr);++k){
        slide(k,mid);
        //best = max(best, {(k>0)?dp[k-1][j-1]:0} + ccost, k );
        //best = min(best, {(k>0)?dp[k-1][j-1]:0} + ccost, k );
    }

    //dp[mid][j] = max(dp[mid][j], best.first);
    //dp[mid][j] = min(dp[mid][j], best.first);
    int opt = best.second;
    if(l!=r){
        compute(l,mid-1,optl,opt,j);
    }
```

```
        compute(mid+1,r,opt,optr,j);
    }
}

//TODO: set dp to LINF or -LINF
```

CHT.h

Description: convex-hull trick

Time: $\mathcal{O}(N)$ or $\mathcal{O}(N \log N)$ if sort the slope

87ad47, 28 lines

```
struct line {
    long long m, c;
    long long eval(long long x) { return m * x + c; }
    long double intersectX(line l) { return (long double) (c -
        l.c) / (l.m - m); }
};

deque<line> dq;
dq.push_front({0, 0});//cant be put in global, remove this to
    local function
//if query ask for minimum remove this line after 1st insertion

//TODO NOTE***: maximum and minimum value exist in bot left
    most and rightmost of convex hull so do search on both l
    to r and r to l

//constructing hull from l to r, maintain correct hull at
    rightmost
/* ****inserting line (maximum hull)
    line cur = line{...some m, ...some c}
    while(dq.size()>=2&&cur.intersectX(dq.back())
        <=cur.intersectX(dq[dq.size()-2]))dq.pop_back();
    dq.pb(cur);
*/

//constructing hull from r to l, maintain correct hull at
    leftmost
/* inserting line (maximum hull)
    line cur = line{...some m, ...some c}
    while(dq.size()>=2&&cur.intersectX(dq[0])>=cur.intersectX(
        dq[1])){
        dq.pop_front();
    }
    dq.push_front(cur);
*/
```

KthAncestor.h

Description: find kth-ancestor of a tree-node

Time: $\mathcal{O}(\log N)$

62aeec, 9 lines

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

LCA.h

Description: find lowest common ancestor of two tree-nodes

Time: $\mathcal{O}(\log N)$

300c4f, 36 lines

```
//TODO: initialize tree(adj list)
const int LOG = 20;
int depth[N], parent[N];
int up[N][LOG]; // 2^j-th ancestor of n
```

```
void dfs(int a,int e){
    for(auto b:adj[a]){
        if(b == e) continue;
        depth[b] = depth[a] + 1;
        parent[b] = a;
        up[b][0] = parent[b];
        for(int i=1;i<LOG;++i){
            up[b][i] = up[up[b][i-1]][i-1];
        }
        dfs(b,a);
    }
}

int lca(int a, int b){
    if(depth[a]<depth[b])swap(a,b);
    int k = depth[a] - depth[b];
    for(int i=LOG-1;i>=0;--i){
        if(k & (1<<i)){
            a = up[a][i];
        }
    }

    if(a == b) return a;
    for(int i=LOG-1;i>=0;--i){
        if(up[a][i] != up[b][i]){
            a = up[a][i];
            b = up[b][i];
        }
    }
    return up[a][0];
}
```

Sieve.h

Description: prime sieve

Time: $\mathcal{O}(N \log \log N)$

abb3a3, 21 lines

```
//can use to find all prime factor of a number in  $\mathcal{O}(\log n)$ 
const int M = 2e5+1;
vector<bool> is_prime(M+1, true);
void sieve(){
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i * i <= M; i++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= M; j += i)
                is_prime[j] = false;
        }
    }
    /* log n sieve (use sieve to find all prime factors in  $\mathcal{O}(\log n)$  )
    for (int i = 2; i <= M; i++)is_prime[i] = i;
    for (int i = 2; i * i <= M; i++) {
        if (is_prime[i] == i) {
            for (int j = i * i; j <= M; j += i)
                is_prime[j] = i;
        }
    }
    */
}
```

IntFact.h

Description: integer factorization algorithm

Time: $\mathcal{O}(\sqrt{N})$

497201, 40 lines

```
//in general any natural number n has at most  $n^{1/3}$  divisors in
    practice
bool isPrime(ll x) {
    for (ll d = 2; d * d <= x; d++) {
        if (x % d == 0)
            return false;
    }
```

```
    }
    return x >= 2;
}

void decompose(ll x){
    vl temp;
    while(x % 2 == 0){
        temp.pb(2);
        x/=2;
    }

    for(ll i=3;i*i <= x;i+=2){
        if(x % i == 0){
            while(x % i == 0){
                x/=i;
                temp.pb(i);
            }
        }
    }
    if(x>1)temp.pb(x);
    //do something
}

void find_all_divisors(ll x){
    vl temp;
    for(int i=1;(ll)i*i<=x;++i){
        if(x%i==0){
            if(i==x/i)temp.pb(i);
            else {
                temp.pb(i); temp.pb(x/i);
            }
        }
    }
    //temp == all divisors of x
}
```

PhiSieve.h

Description: euler totient function precal

Time: $\mathcal{O}(N \log \log N)$

0815d0, 12 lines

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

PhiFunction.h

Description: euler totient function

Time: $\mathcal{O}(\sqrt{N})$

fb8d57, 13 lines

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
```

GcdExtended.h

Description: find Bezout's coefficient

Time: $\mathcal{O}(\log N)$

af07ae, 12 lines

```
int gcd(int a, int b, int& x, int& y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}
```

Matrix.h

Description: mainly for matrix exponentation and find nth recurrence

Time: $\mathcal{O}(\log N)$

1e389f, 28 lines

```
struct Matrix{
    double a[2][2] = {{0,0},{0,0}};
    Matrix operator *(const Matrix& other){
        Matrix product;
        for(int i=0;i<2;++i){
            for(int j=0;j<2;++j){
                for(int k=0;k<2;++k){
                    product.a[i][j] += a[i][k] * other.a[k][j];
                }
            }
        }
        return product;
    }
};
//for calculating nth recurrence of function
//entry Matrix[i][j] is probablility/number of way ith state
change to jth state
Matrix expo_power(Matrix a, int n){
    Matrix product;
    for(int i=0;i<2;++i)product.a[i][i]=1;
    while(n>0){
        if(n&1){
            product = product * a;
        }
        a = a * a;
        n>>=1;
    }
    return product;
}
```

Binpow.h

Description: binary exponentation

Time: $\mathcal{O}(\log N)$

d4debd, 11 lines

```
long long binpow(long long a, int n){
    long long res = 1;
    while(n>0){
        if(n&1){
            res = res * a;
        }
        a = a * a;
        n>>=1;
    }
    return res;
}
```

Ceil2.h

Description: integer ceiling

Time: $\mathcal{O}(1)$

60a42a, 7 lines

Manacher.h

Description: find if substrng a palindrome each queries in $\mathcal{O}(1)$

Time: $\mathcal{O}(N)$ preprocessing $\mathcal{O}(1)$ query

785cc5, 37 lines

```
//sub-palindrome queries
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}

vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher_odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}

bool is_palindrome(int l, int r){
    if(v[l + r] < (r - l + 1))return false;
    return true;
}

vector<int> build_manacher(string s){ //0 base index string
    auto v = manacher(s);
    for(auto&x:v)--x;
    return v;
}
```

cellul4r (3)

ChineseRemainder.h

Description: Chinese Remainder Theorm

b509e8, 11 lines

```
ll CRT(vector<ll> &a, vector<ll> &n) {
    ll prod = 1;
    for (auto ni:n) prod*=ni;

    ll sm = 0;
    for (int i=0; i<n.size(); i++) {
        ll p = prod/n[i];
        sm += a[i]*inv(p, n[i])*p;
    }
}
```

```
    return sm % prod;
}
```

matrixMul.h

Description: matrix multiplication a*b=c081502, 10 lines

```
typedef vector<vector<ll>> mat;
mat mul(mat &a, mat &b) {
    mat c(a.size(), vector<ll>(b[0].size(), 0));
    for (ll i=0; i<a.size(); ++i)
        for (ll j=0; j<b[0].size(); ++j)
            for (ll k=0; k<b.size(); ++k)
                ( c[i][j] += a[i][k]*b[k][j] )%=M;
            // or no mod if ld
    return c;
}
```

Gaussian.h

Description: Gaussian elimination with partial pivoting Also calculates determinant046106, 28 lines

```
ld elim(vector<vector<ld> > &A, vector<ld> &b) {
    int n=A.size();
    ld det=1; //OPTIONAL CALCULATE DET, return ld, not void
    //REF
    for (int i=0;i<n-1;i++) {
        //PIVOT
        int bigi=max_element(A.begin()+i, A.end(), [i](vector<ld> &r1, vector<ld> &r2)
            {return fabs(r1[i])<fabs(r2[i]);})-A.begin();
        swap(A[i], A[bigi]);
        swap(b[i], b[bigi]);
        if (i!=bigi) det*=-1; //DET PART
        for (int j=i+1;j<n;j++) {
            ld m=A[j][i]/A[i][i];
            for (int k=i;k<n;k++)
                A[j][k]-=m*A[i][k];
            b[j]-=m*b[i];
        }
    }
    //DET PART
    for (int i=0;i<n;i++) det*=A[i][i];
    //BACKSUB
    for (int i=n-1;i>=0;i--) {
        for (int j=i+1;j<n;j++)
            b[i]-=A[i][j]*b[j];
        b[i]/=A[i][i];
    }
    return det;
}
```

modularInverse.h

Time: $\mathcal{O}()$ 04162a, 1 lines

```
ll inv(ll a, ll b){return 1<a ? b - inv(b%a,a)*b/a : 1;}
```

nCk.h

Description: nCk4bb9fe, 6 lines

```
ll comb(ll n, ll k) {
    ld res = 1;
    ld w = 0.01;
    for (ll i = 1; i <= k; ++i) res = res * (n-k+i)/i;
    return (int)(res + w);
}
```

powMod.h

Description: pow mod manul3373be, 6 lines

```
ll powmod(ll x, ll y){
    if(y==0) return 1LL;
    ll t=powmod(x,y/2);
    if (y%2==0) return (t*t)%M;
    return ((x*t)%M)*t)%M;
}
```

primeSievephi.h

761494, 16 lines

```
//prime seive
for (ll i=2; i<NN; i++)
    if (prime[i]==0) {
        prime[i] = i;
        for (ll j=i*i;j<NN;j+=i) if(!prime[j]) prime[j]=i;
    }
// phi, uses seive and power from above. The formula is phi(p^i)
//=(p-1)*p^(i-1).
ll phi(ll n) {
    ll ans = n;
    while (n>1) {
        ll p = prime[n];
        while (n%p==0) n/=p;
        ans = ans/p*(p-1);
    }
    return ans;
}
```

DSU.h

Description: disjoint union set with rank union and path compression5d985a, 10 lines

```
ll parent[NN], sz[NN];
ll find(ll a){ return a == parent[a] ? a : parent[a] = find(
    parent[a]); }
void merge(ll u, ll v) {
    u = find(u), v=find(v);
    if (u!=v) {
        if (sz[u]<sz[v]) swap(u, v);
        sz[u] += sz[v];
        parent[v] = u;
    }
}
```

FenwickTree.h

Description: New tree update and find prefix.e62fac, 21 lines

```
struct FT {
    vector<ll> s;
    FT(int n) : s(n) {}
    void update(int pos, ll dif) { // a [ pos ] += d i f
        for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
    }
    ll query(int pos) { // sum of values in [0 , pos)
        ll res = 0;
        for (; pos > 0; pos &= pos - 1) res += s[pos-1];
        return res;
    }
    int lower_bound(ll sum) {
        if (sum <= 0) return -1;
        int pos = 0;
        for (int pw = 1 << 25; pw; pw >= 1) {
            if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
                pos += pw, sum -= s[pos-1];
        }
        return pos;
    }
};
```

LazySegmentTree.h

Description: Segment tree with ability to add or set values of large intervals, and compute max of intervals. Can be changed to other things. Use with a bump allocator for better performance, and SmallPtr or implicit indices to save memory.

Usage: Node* tr = new Node(v, 0, sz(v));

Time: $\mathcal{O}(\log N)$.

```
"../Various/BumpAllocator.h"34ecf5, 50 lines
const int inf = 1e9;
struct Node {
    Node *l = 0, *r = 0;
    int lo, hi, mset = inf, madd = 0, val = -inf;
    Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
    Node(vi& v, int lo, int hi) : lo(lo), hi(hi) {
        if (lo + 1 < hi) {
            int mid = lo + (hi - lo)/2;
            l = new Node(v, lo, mid); r = new Node(v, mid, hi);
            val = max(l->val, r->val);
        }
        else val = v[lo];
    }
}
int query(int L, int R) {
    if (R <= lo || hi <= L) return -inf;
    if (L <= lo && hi <= R) return val;
    push();
    return max(l->query(L, R), r->query(L, R));
}
void set(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) mset = val = x, madd = 0;
    else {
        push(), l->set(L, R, x), r->set(L, R, x);
        val = max(l->val, r->val);
    }
}
void add(int L, int R, int x) {
    if (R <= lo || hi <= L) return;
    if (L <= lo && hi <= R) {
        if (mset != inf) mset += x;
        else madd += x;
        val += x;
    }
    else {
        push(), l->add(L, R, x), r->add(L, R, x);
        val = max(l->val, r->val);
    }
}
void push() {
    if (!l) {
        int mid = lo + (hi - lo)/2;
        l = new Node(lo, mid); r = new Node(mid, hi);
    }
    if (mset != inf)
        l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
    else if (madd)
        l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
}
};
```

DigitDP.h

Description: can hold the number of digit to find that satisfy p(x) for each digit.c7aa80, 11 lines

```
#define DP dp[pos][is_eq]
ll solve(int pos, bool is_eq) {
    if (~DP) return DP;
    if (pos==n)
        //check for predicate (here it is p(x)=True)
        return DP=1;
```



```
DP = 0;
for (int i=0;i<=(is_eq?r[pos]:9);i++)
    DP += solve(pos+1, is_eq && i==r[pos]);
return DP;
}
```

2SAT.h

Description: solve 2 sat form of or to implies

Time: $\mathcal{O}(N + M)$ 025007, 70 lines

```
struct TwoSatSolver {
    int n_vars;
    int n_vertices;
    vector<vector<int>> adj, adj_t;
    vector<bool> used;
    vector<int> order, comp;
    vector<bool> assignment;
    TwoSatSolver(int _n_vars) : n_vars(_n_vars), n_vertices(2 *
        n_vars), adj(n_vertices), adj_t(n_vertices), used(
            n_vertices), order(), comp(n_vertices, -1), assignment
            (n_vars) {
        order.reserve(n_vertices);
    }
    void dfs1(int v) {
        used[v] = true;
        for (int u : adj[v]) {
            if (!used[u])
                dfs1(u);
        }
        order.push_back(v);
    }
    void dfs2(int v, int cl) {
        comp[v] = cl;
        for (int u : adj_t[v]) {
            if (comp[u] == -1)
                dfs2(u, cl);
        }
    }
    bool solve_2SAT() {
        order.clear();
        used.assign(n_vertices, false);
        for (int i = 0; i < n_vertices; ++i) {
            if (!used[i])
                dfs1(i);
        }
        comp.assign(n_vertices, -1);
        for (int i = 0, j = 0; i < n_vertices; ++i) {
            int v = order[n_vertices - i - 1];
            if (comp[v] == -1)
                dfs2(v, j++);
        }

        assignment.assign(n_vars, false);
        for (int i = 0; i < n_vertices; i += 2) {
            if (comp[i] == comp[i + 1])
                return false;
            assignment[i / 2] = comp[i] > comp[i + 1];
        }
        return true;
    }
    void add_disjunction(int a, bool na, int b, bool nb) {
        // na and nb signify whether a and b are to be negated
        a = 2 * a ^ na;
        b = 2 * b ^ nb;
        int neg_a = a ^ 1;
        int neg_b = b ^ 1;
        adj[neg_a].push_back(b);
        adj[neg_b].push_back(a);
        adj_t[b].push_back(neg_a);
    }
}
```

```
adj_t[a].push_back(neg_b);
}
static void example_usage() {
    TwoSatSolver solver(3);
    solver.add_disjunction(0, false, 1, true);
    // a or not b
    solver.add_disjunction(0, true, 1, true); // not a
    // or not b
    solver.add_disjunction(1, false, 2, false); // b
    // or c
    solver.add_disjunction(0, false, 0, false); // a
    // or a
    assert(solver.solve_2SAT() == true);
    auto expected = vector<bool>(True, False, True);
    assert(solver.assignment == expected);
}
}
```

Bellmen.h

Description: find shortest path handle negative weight.

Time: $\mathcal{O}(V * E)$ f303ca, 26 lines

```
// Edge List
struct Edge {
    int u, v, w;
};
vector<Edge> edges;
// bellman-ford:
vector<int> dist(n+1, INF);
dist[start] = 0;
int count = 0;
bool found_changes = false;
bool neg_cycle = false;
do {
    found_changes = false;
    for (auto edge : edges) {
        int u = edge.u, v = edge.v, w = edge.w;
        if (dist[u]+w < dist[v]) {
            dist[v] = dist[u]+w;
            found_change = true;
        }
    }
    ++count;
    if (count > n-1 && found_changes) {
        neg_cycle = true;
        break;
    }
} while (found_changes);
```

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Time: $\mathcal{O}(V + E)$ 7fd551, 52 lines

```
vector<bool> visited; // keeps track of which vertices are
                      // already visited

// runs depth first search starting at vertex v.
// each visited vertex is appended to the output vector when
// dfs leaves it.
void dfs(int v, vector<vector<int>> const& adj, vector<int> &
    output) {
    visited[v] = true;
    for (auto u : adj[v])
        if (!visited[u])
            dfs(u, adj, output);
    output.push_back(v);
}
```

```
// input: adj — adjacency list of G
// output: components — the strongly connected components in G
// output: adj_cond — adjacency list of G^SCC (by root
//          vertices)
void strongly_connected_components(vector<vector<int>> const&
    adj,
                                vector<vector<int>> &
                                components,
                                vector<vector<int>> &adj_cond
                                ) {

    int n = adj.size();
    components.clear(), adj_cond.clear();
    vector<int> order; // will be a sorted list of G's vertices
    // by exit time
    visited.assign(n, false);
    // first series of depth first searches
    for (int i = 0; i < n; i++)
        if (!visited[i])
            dfs(i, adj, order);
    // create adjacency list of G^T
    vector<vector<int>> adj_rev(n);
    for (int v = 0; v < n; v++)
        for (int u : adj[v])
            adj_rev[u].push_back(v);
    visited.assign(n, false);
    reverse(order.begin(), order.end());
    vector<int> roots(n, 0); // gives the root vertex of a
    // vertex's SCC
    // second series of depth first searches
    for (auto v : order)
        if (!visited[v]) {
            std::vector<int> component;
            dfs(v, adj_rev, component);
            sort(component.begin(), component.end());
            components.push_back(component);
            int root = component.front();
            for (auto u : component)
                roots[u] = root;
        }
    // add edges to condensation graph
    adj_cond.assign(n, {});
    for (int v = 0; v < n; v++)
        for (int u : adj[v])
            if (roots[v] != roots[u])
                adj_cond[roots[v]].push_back(roots[u]);
}
```

topo.h

Description: to find order on DAG.

Time: $\mathcal{O}(V + E)$ 7c07c6, 18 lines

```
int indeg[N];
// edge(u,v) ++indeg[v]
queue<int> Q;
for (int u = 1; u <= n; ++u) {
    if (indeg[u] == 0)
        Q.push(u);
}
vector<int> seq; // sequence
while (!Q.empty()) {
    int u = Q.front();
    Q.pop();
    seq.push_back(u);
    for (auto v : G[u]) {
        --indeg[v];
        if (indeg[v] == 0)
            Q.push(v);
    }
}
```

```
}
```

primAlgo.h

Description: find minimum spanning tree with Prim Algorithm.

Time: $\mathcal{O}(\log(V) * E)$ c93060, 23 lines

```
using pii = pair<int, int>;
vector<int> dist(n+1, INF);
vector<bool> visited(n+1, false);
priority_queue<pii, vector<pii>, greater<pii>> Q;
dist[start] = 0;
Q.push({dist[start], start});
int sum = 0;
while (!Q.empty()) {
    int u = Q.top().second, d = Q.top().first;
    Q.pop();
    if (visited[u]) continue;
    visited[u] = true;
    sum += dist[u];
    for (auto vw : G[u]) {
        int v = vw.first;
        int w = vw.second;
        if (!visited[v] && w < dist[v]) {
            dist[v] = w;
            Q.push({dist[v], v});
        }
    }
}
```

tarjan.h

Description: find the bridge of the graph and articulation point

Time: $\mathcal{O}(V^2)$ b3792, 27 lines

```
vector<int> G[N];
bool visited[N];
int disc[N], low[N];
set<int> ap; // answer: articulation points
set<pii> bridge; // answer: bridges
int counter = 0;
void tarjan(int u, int p) { // p = parent of u
    visited[u] = true;
    low[u] = disc[u] = ++counter;
    int child = 0;
    for (auto v : G[u]) {
        if (!visited[v]) {
            ++child;
            tarjan(v, u);
            low[u] = min(low[u], low[v]);
            // articulation point
            // parent of root is 0.
            if ((p != 0 && low[v] >= disc[u]) || (p == 0 && child > 1))
                ap.insert(u);
            // bridge
            if (low[v] > disc[u])
                bridge.insert(pii(u, v));
        } else if (v != p) {
            low[u] = min(low[u], disc[v]);
        }
    }
}
```

Dinic.h

Description: In Bipartite graph,Maximum Independent Set a set of vertices such that any two vertices in the set do not have a direct edge between them. Minimum Vertex cover Set of vertices that touches every edge MIS = N - MVC (MVC = MAX FLOW (maximum matching))

Time: $\mathcal{O}(E * V^2)$ 9b8492, 38 lines

primAlgo tarjan Dinic KuhnAlgo Hull stringHash

//define S for MAXN T is S+1 and use add_edge

```
struct dinic {
    struct edge {ll b, cap, flow, flip;;};
    vector<edge> g[S+2];
    ll ans=0, d[S+2], ptr[S+2];
    void add_edge (ll a, ll b, ll cap) {
        g[a].push_back({b, cap, 0, g[b].size()});
        g[b].push_back({a, 0, 0, g[a].size()-1});
    }
    ll dfs (ll u, ll flow=LLONG_MAX) {
        if (u==S+1 || !flow) return flow;
        while (++ptr[u] < g[u].size()) {
            edge &e = g[u][ptr[u]];
            if (d[e.b] != d[u]+1) continue;
            if (ll pushed = dfs(e.b, min(flow, e.cap-e.flow))) {
                e.flow += pushed;
                g[e.b][e.flip].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
    void calc() {
        do {
            vector<ll> q {S};
            memset(d, 0, sizeof d);
            ll i = -(d[S] = 1);
            while (++i<q.size() && !d[S+1])
                for (auto e: g[q[i]])
                    if (!d[e.b] && e.flow<e.cap) {
                        q.push_back(e.b);
                        d[e.b] = d[q[i]]+1;
                    }
            memset(ptr, -1, sizeof ptr);
            while (ll pushed=dfs(S)) ans+=pushed;
        } while (d[S+1]);
    }
};
```

KuhnAlgo.h

Description: mat[right node] is a left node or -1 edges from left to right

Time: $\mathcal{O}(abs(left) * M)$ 709547, 13 lines

```
bool dfs(ll u) {
    if (used[u]) return 0;
    used[u] = 1;
    for (ll v: edges[u])
        if (mat[v]==-1 || dfs(mat[v]))
            return mat[v] = u,1;
    return 0;
}
memset(mat, -1, sizeof mat);
for (ll u=0; u<n; ++u) {
    memset(used, 0, sizeof used);
    flow += dfs(u);
}
```

Hull.h

Description: Convex Hull trick

<bits/stdc++.h> 5966c1, 55 lines

```
#pragma GCC target("avx2")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
using namespace std;
typedef long long int ll;
typedef long double ld;
typedef pair<ll, ll> pl;
typedef vector<ll> vl;
typedef complex<ll> pt;
```

```
#define G(x) ll x; cin >> x;
#define F(i, l, r) for(ll i = l; i < (r); ++i)
#define A(a) (a).begin(), (a).end()
#define CRS(a, b) (conj(a) * (b)).Y
#define K first
#define V second
#define X real()
#define Y imag()
#define N 100010
namespace std {
    bool operator<(pt a, pt b) { return a.X == b.X ? a.Y < b.Y : a.X < b.X; }
}
bool in_hull(pt p, vector<pt>& hu, vector<pt>& hd) {
    if (p == *hu.begin() || p == *hd.begin()) return false; //
        change to true if border counts as inside
    if (p < *hu.begin() || *hd.begin() < p) return false;
    auto u = upper_bound(A(hu), p);
    auto d = lower_bound(hd.rbegin(), hd.rend(), p);
    return CRS(*u - p, *(u - 1) - p) > 0 && CRS(*(d - 1) - p, *d - p) > 0; //change to >= if border counts as "inside"
}
void do_hull(vector<pt>& pts, vector<pt>& h) {
    for (pt p : pts) {
        while (h.size() > 1 && CRS(h.back() - p, h[h.size() - 2] - p) <= 0) //change to < 0 if border points included
            h.pop_back();
        h.push_back(p);
    }
}
pair<vector<pt>, vector<pt>> get_hull(vector<pt>& pts) {
    vector<pt> hu, hd;
    sort(A(pts)), do_hull(pts, hu);
    reverse(A(pts)), do_hull(pts, hd);
    return {hu, hd};
}
vector<pt> full_hull(vector<pt>& pts) {
    auto h = get_hull(pts);
    h.K.pop_back(), h.V.pop_back();
    for (pt p : h.V) h.K.push_back(p);
    return h.K;
}
int main() {
    G(n) vector<pt> v;
    F(i, 0, n) {
        G(x) G(y)
            v.push_back({x, y});
    }
    vector<pt> h = full_hull(v);
}
```

stringHash.h

Description: Hack with string hash.

Time: $\mathcal{O}(N)$ 60d530, 18 lines

```
typedef long long int ll;
typedef pair<ll, ll> pl;
#define M 1000000321
#define OP(x, y) pl operator x (pl a, pl b) { return { a.first
    x b.first, (a.second y b.second) % M }; }
OP(+, +) OP(*, *) OP(-, + M -)
mt19937 gen(chrono::steady_clock::now().time_since_epoch().
    count());
uniform_int_distribution<ll> dist(256, M - 1);
#define H(i, l) (h[(i) + (l)] - h[i] * p[l])
#define EQ(i, j, l) (H(i, l) == H(j, l))
#define N 100010
string s;
pl p[N], h[N];
// EQ is string s in range [i,L), and range [j,L)
```

```
p[0] = { 1, 1 }, p[1] = { dist(gen) | 1, dist(gen) };
for(int i = 1; i <= (ll)s.size(); i++) {
    p[i] = p[i - 1] * p[1];
    h[i] = h[i - 1] * p[1] + make_pair(s[i - 1], s[i - 1]);
}
```

suffixArray.h

Description: find suffix array with string hashing. 2f2c82, 33 lines

```
typedef long long int ll;
typedef pair<ll, ll> pl;
#define M 1000000321
#define OP(x, y) pl operator x (pl a, pl b) { return { a.first
    x b.first, (a.second y b.second) % M }; }
OP(+, +) OP(*, *) OP(-, + M -)
mt19937 gen(chrono::steady_clock::now().time_since_epoch()).
    count());
uniform_int_distribution<ll> dist(256, M - 1);
#define H(i, l) (h[(i) + (l)] - h[i] * p[l])
#define EQ(i, j, l) (H(i, l) == H(j, l))
#define N 100010
string s;
pl p[N], h[N];
ll n, suff[N];
ll lcp(ll i, ll j, ll l, ll r) { //can use any binary search
    function here
    if(l == r) return l;
    ll m = (l + r + 1) / 2;
    return EQ(i, j, m) ? lcp(i, j, m, r) : lcp(i, j, l, m - 1);
}
bool lexLess(ll i, ll lI, ll j, ll lJ) {
    if(EQ(i, j, min(lI, lJ))) return lI < lJ;
    ll m = lcp(i, j, 0, min(lI, lJ) - 1);
    return s[i + m] < s[j + m];
}
p[0] = { 1, 1 }, p[1] = { dist(gen) | 1, dist(gen) };
for(int i = 1; i <= (ll)s.size(); i++) {
    p[i] = p[i - 1] * p[1];
    h[i] = h[i - 1] * p[1] + make_pair(s[i - 1], s[i - 1]);
}
iota(suff, suff + n, 0); //sets suff[i] = i for all i
sort(suff, suff + n, [](ll i, ll j) { return lexLess(i, n - i,
    j, n - j); });
for(int i = 0; i < n; i++) cout << suff[i] << ' ';
    cout << '\n';
}
```

SuffixTree.h

Description: suffix tree, NN here is number of nodes, which is like 2n+10
to[] is edges, root is idx 1 lf[] and rt[] are edge info as half open interval into
s 6f215c, 24 lines

```
map<char, ll> to[NN], lk[NN];
ll lf[NN], rt[NN], par[NN], path[NN];
#define att(a, b, c) to[par[a]=b][s[lf[a]=c]]=a;
void build(string &s) {
    ll n=s.size(), z=2;
    lf[1]--;
    for (ll i=n-1; i+1; i--) {
        ll v, V=n, o=z-1, k=0;
        for (v=o; !lk[v].count(s[i]) && v; v=par[v])
            V -= rt[path[k++]=v]-lf[v];
        ll w = lk[v][s[i]]+1;
        if (to[w].count(s[V])) {
            ll u = to[w][s[V]];
            for (rt[z]=lf[u]; s[rt[z]]==s[V]; rt[z]+=rt[v]-lf[v])
                v=path[--k], V+=rt[v]-lf[v];
            att(z, w, lf[u])
            att(u, z, rt[z])
        }
    }
}
```

```
lk[v][s[i]] = (w = z++)-1;
}
lk[o][s[i]] = z-1;
att(z, w, V)
rt[z++] = n;
}
}
```

ZString.h

Description: Z Algo for string. 9a2512, 18 lines

```
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```

MoAlgo.h

Description: q is query idx's [l,r] closed sort the query first and for each
current we update the value and find the total for each query.
Time: $\mathcal{O}(q * S + n * n / S)$ 5e88f5, 24 lines

```
map<ll,ll> cnt;
set<pair<ll,ll>> best;
ll tot;
void update(ll i, ll d) {
    ll a = x[i];
    best.erase({cnt[a],a});
    cnt[a] += d;
    best.insert({cnt[a],a});
    tot = best.rbegin()->second;
}
ll S = sqrtl(n);
sort(q.begin(), q.end(), [&](ll a, ll b) {
    if (l[a]/S != l[b]/S) return l[a]/S < l[b]/S;
    return l[a]/S%2 ? r[a]>r[b] : r[a]<r[b];
});
ll curl=0, curr=-1;
for (auto i:q) {
    while(curr<r[i]) update(++curr, 1);
    while(curr>r[i]) update(curr--, -1);
    while(curl<l[i]) update(curl++, -1);
    while(curl>l[i]) update(--curl, 1);

    ans[i] = tot;
}
```

ConvexHullTrick.h

Description: Convex hull trick for y = ax+b
Time: $\mathcal{O}()$
<bits/stdc++.h> 71af1b, 35 lines
//Convex Hull Trick
#pragma GCC target("avx2")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
using namespace std;

```
typedef long long int ll;
typedef long double ld;
typedef pair<ll, ll> pl;
typedef vector<ll> vl;
```

```
#define G(x) ll x; cin >> x;
#define F(i, l, r) for(ll i = l; i < (r); ++i)
#define A(a) (a).begin(), (a).end()
#define N 100010
```

```
struct seg { ld x; ll m, b; };
vector<seg> hull;
```

```
void insert(ll m, ll b) { //lines inserted in inc. slope order
    for mathull, dec. for minhull
    while(hull.size()) {
        seg s = hull.back();
        if(s.b + s.m * s.x > b + m * s.x) { //< for minhull
            if(s.m - m) hull.push_back({(b - s.b) / (ld)(s.m - m), m,
                b});
            return;
        }
        hull.pop_back();
    }
    hull = {{LLONG_MIN, m, b}};
}
```

```
ll query(ll x) {
    seg s = *--upper_bound(A(hull), x, [](ll a, seg b) { return a
        < b.x; });
    return s.b + s.m * x;
}
```

Matrices.h

Description: return determinant of matrix matrix multiplication, inverse
matrix.
Time: $\mathcal{O}()$ b1ca6c, 50 lines

```
typedef long long int ll;
typedef long double ld;
typedef pair<ll, ll> pl;
typedef vector<ll> vl;
typedef vector<vector<ld>> mat;
```

```
#define G(x) ll x; cin >> x;
#define F(i, l, r) for(ll i = l; i < (r); ++i)
#define FD(i, r, l) for(ll i = r; i > (l); --i)
#define A(a) (a).begin(), (a).end()
#define MAT(a, n, m) mat a(n, vector<ld>(m, 0));
#define N 100010
```

```
ld elim(mat& a) { //a is n x m augmented matrix (m>= n),
    returns determinant
    ll n = a.size(), m = a[0].size();
    ld det = 1;
    F(i, 0, n) {
        ll bI = i;
        F(j, i, n) if(fabs(a[j][i]) > fabs(a[bI][i])) bI = j;
        swap(a[i], a[bI]);
        det *= i - bI ? -a[i][i] : a[i][i];
        if(!det) return 0;
        FD(k, m - 1, i - 1) a[i][k] /= a[i][i];
        F(j, 0, n) if(j - i) FD(k, m - 1, i - 1) a[j][k] -= a[j
            ][i] * a[i][k];
    }
    return det;
}
```

```
mat mult(mat& a, mat& b) {
    ll n = a.size(), m = b.size(), p = b[0].size();
    MAT(c, n, p)
    F(i, 0, n) F(j, 0, m) F(k, 0, p) c[i][k] += a[i][j] * b[j][k];
    return c;
}

mat pw(mat& a, ll p) { //only works for p > 0
    if(p == 1) return a;
    mat t = pw(a, p / 2);
    t = mult(t, t);
    return p & 1 ? mult(t, a) : t;
}

mat inv(mat& a) { //inverts a, returns empty matrix if non-invertible
    ll n = a.size();
    MAT(b, n, 2 * n)
    F(i, 0, n) copy(A(a[i]), b[i].begin(), b[i][n + i] = 1;
    if(!elim(b)) return {};
    F(i, 0, n) FD(j, n - 1, -1) b[i][j] = b[i].back(), b[i].pop_back();
    return b;
}

typedef long long int ll;
typedef long double ld;
typedef complex<ld> cd;
typedef pair<ll, ll> pl;
typedef vector<ll> vl;

#define G(x) ll x; cin >> x;
#define F(i, l, r) for(ll i = l; i < (r); ++i)
#define N 100010
#define L 20

vector<cd> rt(2, 1);

void fft(vector<cd>& a) {
    ll n = a.size(); vl rev(n);
    F(i, 0, n) if(i < (rev[i] = (rev[i / 2] | (i & 1 ? n : 0)) / 2)) swap(a[i], a[rev[i]]);
    for(ll k = 1; k < n; k *= 2)
        for(ll i = 0; i < n; i += 2 * k) F(j, 0, k) {
            cd z = rt[j + k] * a[i + j + k];
            a[i + j + k] = a[i + j] - z;
            a[i + j] += z;
        }
}

vector<ld> mult(vector<ld> a, vector<ld> b) {
    ll n = a.size() + b.size();
    while(__builtin_popcountll(n) > 1) n++;
    vector<cd> in(n), out(n);
    a.resize(n), b.resize(n);
    F(i, 0, n) in[i] = {a[i], b[i]};
    fft(in);
    for(cd& x : in) x *= x;
    F(i, 0, n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    vector<ld> res(n);
    F(i, 0, n) res[i] = out[i].imag() / 4 / n;
    return res;
}
```

FFT.h
Description: Fast Fourier Transform
Time: $\mathcal{O}()$

```
int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    for(ll k = 2; k < (1 << L); k *= 2) {
        cd x = polar(1.1, acos(-1.1) / k);
        F(i, k, 2 * k) rt.push_back(rt[i / 2] * (i & 1 ? x : 1));
    }
}

backtracking.h
Time:  $\mathcal{O}()$ 
<bits/stdc++.h> dda661, 67 lines

//Backtracking
#pragma GCC target("avx2")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")
using namespace std;

#define F(i, k) for(int i = 0; i < (k); ++i)
#define G(x) int x; cin >> x;
#define D F(i, n) F(j, m) cout << gr[i][j] << " \n"[j == m - 1];
#define OK(r, c) (r >= 0 && c >= 0 && r < n && c < m)
#define SQ(r, c) ((r) / 3 + 3 * ((c) / 3)) //UPDATE?
#define MT(r, c, i) (!i ? r : (i == 1 ? c : SQ(r, c))) //UPDATE?
#define ST(r, c, i) (!i ? r : (i == 1 ? c : SQ(r, c))) //UPDATE?
#define N 100 //UPDATE?
#define M 3 //UPDATE?
#define S 3 //UPDATE?

int n = -1, m = -1, dfV = -1, smV = -1, mnV = -1, mxV = -1; //IMPLEMENT
int gr[N][N], mk[M][N], em[S][N], sm[S][N];

bool lV(int r, int c) { return true; } //IMPLEMENT

void add(int r, int c, int x, int s) {
    gr[r][c] = s > 0 ? x : dfV;
    F(i, S) {
        sm[i][ST(r, c, i)] -= s * x;
        em[i][ST(r, c, i)] -= s;
    }
    F(i, M) mk[i][MT(r, c, i)] ^= 1 << (x - mnV);
}

int getX(int r, int c, bool mn) { //distinct element optimization?
    int ans = mn ? mnV : mxV;
    F(i, S) {
        int t1 = sm[i][ST(r, c, i)], t2 = em[i][ST(r, c, i)] - 1;
        ans = mn ? max(ans, t1 - t2 * mxV) : min(ans, t1 - t2 * mnV);
    }
    return ans;
}

bool bT(int r, int c) {
    if(r == n) return true;
    int rI = r + (c + 1) / m, cI = (c + 1) % m;
    if(gr[r][c] - dfV) return lV(r, c) && bT(rI, cI);
    int mask = mk[0][r] & mk[1][c] & mk[2][SQ(r, c)]; //UPDATE?
    int mnX = getX(r, c, true), mxX = getX(r, c, false);
    for(int x = mnX; x <= mxX && mask >> (x - mnV); ++x) if((mask >> (x - mnV)) & 1) {
```

```
        add(r, c, x, 1);
        if(lV(r, c) && bT(rI, cI)) return true;
        add(r, c, x, -1);
    }
    return false;
}

void init() { //are you indexing properly with the input?
    F(i, M) fill_n(mk[i], N, (1 << (mxV - mnV + 1)) - 1); //UPDATE?
    F(i, S) {
        fill_n(em[i], N, n); //UPDATE?
        fill_n(sm[i], N, smV); //UPDATE?
    }
    F(r, n) F(c, m) { //need to reset board for test cases?
        G(x) //are you using add() vs gr[][] properly?
        if(x - dfV) add(r, c, x, 1); //need to check if input is consistent?
    }
}

int main() { init(); bT(0, 0); D } //need to see if bT returns false?
```

Centroid.h
Description: Centroid decomposition
Time: $\mathcal{O}()$

```
vl tree[N];
ll sz[N], cPar[N], lvl[N]; //lvl is 1-indexed

ll getSize(ll i, ll p) {
    sz[i] = 1;
    for(ll j : tree[i])
        if(j - p && !lvl[j])
            sz[i] += getSize(j, i);
    return sz[i];
}

ll centroid(ll i, ll p, ll n) {
    for(ll j : tree[i])
        if(j - p && !lvl[j] && sz[j] > n / 2)
            return centroid(j, i, n);
    return i;
}

ll decomp(ll i, ll l) {
    ll cent = centroid(i, -1, getSize(i, -1));
    lvl[cent] = 1;
    for(ll j : tree[cent]) if(!lvl[j])
        cPar[decomp(j, 1 + 1)] = cent;
    return cent;
}

int main() {
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    G(n)
    F(i, 0, n - 1) {
        G(u) G(v)
        tree[u].push_back(v);
        tree[v].push_back(u);
    }
    ll root = decomp(1, 1);
}
```

dfsTree.h

Time: $\mathcal{O}()$

09b6b6, 14 lines

```
v1 graph[N], backIn[N], backOut[N], tree[N];
l1 dep[N], par[N];

void dfs(l1 i, l1 p) {
    par[i] = p;
    dep[i] = dep[p] + 1;
    for(l1 j : graph[i]) if(!dep[j]) {
        tree[i].push_back(j);
        dfs(j, i);
    } else if(dep[j] < dep[i] - 1) {
        backIn[j].push_back(i);
        backOut[i].push_back(j);
    }
}
```

LCA.h

Time: $\mathcal{O}()$

98433f, 20 lines

```
#define N 100010
#define L 20

l1 dep[N], par[N][L];
v1 tree[N];

void dfs(l1 i, l1 p) {
    dep[i] = dep[p] + 1;
    par[i][0] = p;
    F(1, 1, L) par[i][l] = par[par[i][l-1]][l-1];
    for(l1 j : tree[i]) if(j - p) dfs(j, i);
}

l1 lca(l1 a, l1 b) {
    if(dep[a] < dep[b]) swap(a, b);
    FD(1, L - 1, -1) if((dep[a] - dep[b]) >> 1) a = par[a][l];
    if(a == b) return a;
    FD(1, L - 1, -1) if(par[a][l] - par[b][l]) a = par[a][l], b = par[b][l];
    return par[a][0];
}
```

Geometry.h

Time: $\mathcal{O}()$

92b95d, 162 lines

```
typedef long long int l1;
typedef long double ld;
typedef complex<ld> pt;
struct line {
    pt P, D; bool S = false;
    line(pt p, pt q, bool b = false) : P(p), D(q - p), S(b) {}
    line(pt p, ld th) : P(p), D(polar((ld)1, th)) {}
};
struct circ { pt C; ld R; };

#define X real()
#define Y imag()
#define CRS(a, b) (conj(a) * (b)).Y //scalar cross product
#define DOT(a, b) (conj(a) * (b)).X //dot product
#define U(p) ((p) / abs(p)) //unit vector in direction of p (
    don't use if Z(p) == true)
#define Z(x) (abs(x) < EPS)
#define A(a) (a).begin(), (a).end() //shortens sort(),
    upper_bound(), etc. for vectors

//constants (INF and EPS may need to be modified)
ld PI = acosl(-1), INF = 1e20, EPS = 1e-12;
pt I = {0, 1};
```

```
//true if d1 and d2 parallel (zero vectors considered parallel
to everything)
bool parallel(pt d1, pt d2) { return Z(d1) || Z(d2) || Z(CRS(U(
d1), U(d2))); }

//"above" here means if l & p are rotated such that l.D points
in the +x direction, then p is above l. Returns arbitrary
boolean if p is on l
bool above_line(pt p, line l) { return CRS(p - l.P, l.D) > 0; }

//true if p is on line l
bool on_line(pt p, line l) { return parallel(l.P - p, l.D) &&
(!l.S || DOT(l.P - p, l.P + l.D - p) <= EPS); }

//returns 0 for no intersection, 2 for infinite intersections,
1 otherwise. p holds intersection pt
l1 intsct(line l1, line l2, pt& p) {
    if(parallel(l1.D, l2.D)) //note that two parallel segments
        sharing one endpoint are considered to have infinite
        intersections here
        return 2 * (on_line(l1.P, l2) || on_line(l1.P + l1.D, l2)
            || on_line(l2.P, l1) || on_line(l2.P + l2.D, l1));
    pt q = l1.P + l1.D * CRS(l2.D, l2.P - l1.P) / CRS(l2.D, l1.D)
        ;
    if(on_line(q, l1) && on_line(q, l2)) { p = q; return 1; }
    return 0;
}

//closest pt on l to p
pt cl_pt_on_l(pt p, line l) {
    pt q = l.P + DOT(U(l.D), p - l.P) * U(l.D);
    if(on_line(q, l)) return q;
    return abs(p - l.P) < abs(p - l.P - l.D) ? l.P : l.P + l.D;
}

//distance from p to l
ld dist_to(pt p, line l) { return abs(p - cl_pt_on_l(p, l)); }

//p reflected over l
pt refl_pt(pt p, line l) { return conj((p - l.P) / U(l.D)) * U(
l.D) + l.P; }

//ray r reflected off l (if no intersection, returns original
ray)
line reflect_line(line r, line l) {
    pt p; if(intsct(r, l, p) - 1) return r;
    return line(p, p + INF * (p - refl_pt(r.P, l)), 1);
}

//altitude from p to l
line alt(pt p, line l) { l.S = 0; return line(p, cl_pt_on_l(p,
l)); }

//angle bisector of angle abc
line ang_bis(pt a, pt b, pt c) { return line(b, b + INF * (U(a
- b) + U(c - b)), 1); }

//perpendicular bisector of l (assumes l.S == 1)
line perp_bis(line l) { return line(l.P + l.D / (ld)2, arg(l.D
* I)); }

//orthocenter of triangle abc
pt orthocent(pt a, pt b, pt c) { pt p; intsct(alt(a, line(b, c)
), alt(b, line(a, c)), p); return p; }

//incircle of triangle abc
circ incirc(pt a, pt b, pt c) {
    pt cent; intsct(ang_bis(a, b, c), ang_bis(b, a, c), cent);
    return {cent, dist_to(cent, line(a, b))};
}
```

```
}

//circumcircle of triangle abc
circ circumcirc(pt a, pt b, pt c) {
    pt cent; intsct(perp_bis(line(a, b, 1)), perp_bis(line(a, c,
1)), cent);
    return {cent, abs(cent - a)};
}

//is pt p inside the (not necessarily convex) polygon given by
poly
bool in_poly(pt p, vector<pt>& poly) {
    line l = line(p, {INF, INF * PI}, 1);
    bool ans = false;
    pt lst = poly.back(), tmp;
    for(pt q : poly) {
        line s = line(q, lst, 1); lst = q;
        if(on_line(p, s)) return false; //change if border included
        else if(intsct(l, s, tmp)) ans = !ans;
    }
    return ans;
}

//area of polygon, vertices in order (cw or ccw)
ld area(vector<pt>& poly) {
    ld ans = 0;
    pt lst = poly.back();
    for(pt p : poly) ans += CRS(lst, p), lst = p;
    return abs(ans / 2);
}

//perimeter of polygon, vertices in order (cw or ccw)
ld perim(vector<pt>& poly) {
    ld ans = 0;
    pt lst = poly.back();
    for(pt p : poly) ans += abs(lst - p), lst = p;
    return ans;
}

//centroid of polygon, vertices in order (cw or ccw)
pt centroid(vector<pt>& poly) {
    ld area = 0;
    pt lst = poly.back(), ans = {0, 0};
    for(pt p : poly) {
        area += CRS(lst, p);
        ans += CRS(lst, p) * (lst + p) / (ld)3;
        lst = p;
    }
    return ans / area;
}

//invert a point over a circle (doesn't work for center of
circle)
pt circInv(pt p, circ c) {
    return c.R * c.R / conj(p - c.C) + c.C;
}

//vector of intersection pts of two circs (up to 2) (if circles
same, returns empty vector)
vector<pt> intsctCC(circ c1, circ c2) {
    if(c1.R < c2.R) swap(c1, c2);
    pt d = c2.C - c1.C;
    if(Z(abs(d) - c1.R - c2.R)) return {c1.C + polar(c1.R, arg(c2
.C - c1.C))};
    if(!Z(d) && Z(abs(d) - c1.R + c2.R)) return {c1.C + c1.R * U(
d)};
    if(abs(abs(d) - c1.R) >= c2.R - EPS) return {};
    ld th = acosl((c1.R * c1.R + norm(d) - c2.R * c2.R) / (2 * c1
.R * abs(d)));
}
```

```
    return {c1.C + polar(c1.R, arg(d) + th), c1.C + polar(c1.R,
        arg(d) - th)};
}

//vector of intersection pts of a line and a circ (up to 2)
vector<pt> intsetCL(circ c, line l) {
    vector<pt> v, ans;
    if(parallel(l.D, c.C - l.P)) v = {c.C + c.R * U(l.D), c.C - c
        .R * U(l.D)};
    else v = intsetCC(c, circ{refl_pt(c.C, l), c.R});
    for(pt p : v) if(on_line(p, l)) ans.push_back(p);
    return ans;
}

//external tangents of two circles (negate c2.R for internal
    tangents)
vector<line> circTangents(circ c1, circ c2) {
    pt d = c2.C - c1.C;
    ld dr = c1.R - c2.R, d2 = norm(d), h2 = d2 - dr * dr;
    if(Z(d2) || h2 < 0) return {};
    vector<line> ans;
    for(ld sg : {-1, 1}) {
        pt u = (d * dr + d * I * sqrt(h2) * sg) / d2;
        ans.push_back(line(c1.C + u * c1.R, c2.C + u * c2.R, l));
    }
    if(Z(h2)) ans.pop_back();
    return ans;
}
```

Mathematics (4)

4.1 Equations

$ax^2 + bx + c = 0 \Rightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

The extremum is given by $x = -b/2a$.

$$\begin{matrix} ax + by = e \\ cx + dy = f \end{matrix} \Rightarrow \begin{matrix} x = \frac{ed - bf}{ad - bc} \\ y = \frac{af - ec}{ad - bc} \end{matrix}$$

In general, given an equation $Ax = b$, the solution to a variable x_i is given by

$$x_i = \frac{\det A'_i}{\det A}$$

where A'_i is A with the i 'th column replaced by b .

Non-distinct roots r become polynomial factors, e.g.
 $a_n = (d_1n + d_2)r^n$.

4.2 Trigonometry

$$\begin{aligned} \sin(v + w) &= \sin v \cos w + \cos v \sin w \\ \cos(v + w) &= \cos v \cos w - \sin v \sin w \end{aligned}$$

$$\begin{aligned} \tan(v + w) &= \frac{\tan v + \tan w}{1 - \tan v \tan w} \\ \sin v + \sin w &= 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2} \\ \cos v + \cos w &= 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2} \end{aligned}$$

$$(V + W) \tan(v - w)/2 = (V - W) \tan(v + w)/2$$

where V, W are lengths of sides opposite angles v, w .

$$\begin{aligned} a \cos x + b \sin x &= r \cos(x - \phi) \\ a \sin x + b \cos x &= r \sin(x + \phi) \end{aligned}$$

where $r = \sqrt{a^2 + b^2}, \phi = \text{atan2}(b, a)$.

4.3 Geometry

4.3.1 Triangles

Side lengths: a, b, c

Semiperimeter: $p = \frac{a + b + c}{2}$

Area: $A = \sqrt{p(p - a)(p - b)(p - c)}$

Circumradius: $R = \frac{abc}{4A}$

Inradius: $r = \frac{A}{p}$

Length of median (divides triangle into two equal-area triangles):

$m_a = \frac{1}{2} \sqrt{2b^2 + 2c^2 - a^2}$

Length of bisector (divides angles in two):

$$s_a = \sqrt{bc \left[1 - \left(\frac{a}{b + c} \right)^2 \right]}$$

Law of sines: $\frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$

Law of cosines: $a^2 = b^2 + c^2 - 2bc \cos \alpha$

Law of tangents: $\frac{a + b}{a - b} = \frac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

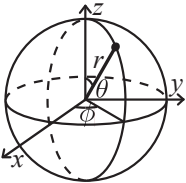
4.3.2 Quadrilaterals

With side lengths a, b, c, d , diagonals e, f , diagonals angle θ , area A and magic flux $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is 180° ,
 $ef = ac + bd$, and $A = \sqrt{(p - a)(p - b)(p - c)(p - d)}$.

4.3.3 Spherical coordinates



$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \text{acos}(z / \sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \text{atan2}(y, x) \end{aligned}$$

4.4 Derivatives/Integrals

$$\begin{aligned} \frac{d}{dx} \arcsin x &= \frac{1}{\sqrt{1 - x^2}} & \frac{d}{dx} \arccos x &= -\frac{1}{\sqrt{1 - x^2}} \\ \frac{d}{dx} \tan x &= 1 + \tan^2 x & \frac{d}{dx} \arctan x &= \frac{1}{1 + x^2} \\ \int \tan ax &= -\frac{\ln |\cos ax|}{a} & \int x \sin ax &= \frac{\sin ax - ax \cos ax}{a^2} \\ \int e^{-x^2} &= \frac{\sqrt{\pi}}{2} \text{erf}(x) & \int x e^{ax} dx &= \frac{e^{ax}}{a^2} (ax - 1) \end{aligned}$$

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

4.5 Sums

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \dots + n = \frac{n(n + 1)}{2}$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n + 1)(n + 1)}{6}$$

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n + 1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n + 1)(2n + 1)(3n^2 + 3n - 1)}{30}$$

4.6 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)$$

$$\sqrt{1 + x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, \quad (-\infty < x < \infty)$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, \quad (-\infty < x < \infty)$$

4.7 Probability theory

Let X be a discrete random variable with probability $p_X(x)$ of assuming the value x . It will then have an expected value (mean) $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$ and variance $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$ where σ is the standard deviation. If X is instead continuous it will have a probability density function $f_X(x)$ and the sums above will instead be integrals with $p_X(x)$ replaced by $f_X(x)$.

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent X and Y ,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

4.7.1 Discrete distributions

Binomial distribution

The number of successes in n independent yes/no experiments, each which yields success with probability p is $\text{Bin}(n, p)$, $n = 1, 2, \dots$, $0 \leq p \leq 1$.

$$p(k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1 - p)$$

$\text{Bin}(n, p)$ is approximately $\text{Po}(np)$ for small p .

First success distribution

The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability p is $\text{Fs}(p)$, $0 \leq p \leq 1$.

$$p(k) = p(1 - p)^{k-1}, \quad k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1 - p}{p^2}$$

Poisson distribution

The number of events occurring in a fixed period of time t if these events occur with a known average rate κ and independently of the time since the last event is $\text{Po}(\lambda)$, $\lambda = t\kappa$.

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, \quad k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

4.7.2 Continuous distributions

Uniform distribution

If the probability density function is constant between a and b and 0 elsewhere it is $\text{U}(a, b)$, $a < b$.

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

Exponential distribution

The time between events in a Poisson process is $\text{Exp}(\lambda)$, $\lambda > 0$.

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Normal distribution

Most real random values with mean μ and variance σ^2 are well described by $\mathcal{N}(\mu, \sigma^2)$, $\sigma > 0$.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$ and $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$ then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

Data structures (5)

SegmentTree.h

Description: Segment tree with point update for range sum

Time: $\mathcal{O}(\log N)$

efd738, 29 lines

//TODO: use 0 base indexing

vector<long long>tree;

void update(int node,int n_l,int n_r,int q_i,long long value){

if(n_r<q_i || q_i<n_l)return;

if(q_i==n_l && n_r==q_i){

tree[node] = value;

return;

}

int mid = (n_r+n_l)/2;

update(2*node,n_l,mid,q_i,value);

update(2*node+1,mid+1,n_r,q_i,value);

tree[node] = tree[2*node] + tree[2*node+1];

}

long long f(int node,int n_l,int n_r,int q_l,int q_r){

if(n_r<q_l || q_r<n_l)return 0;

if(q_l<=n_l && n_r<=q_r)return tree[node];

int mid = (n_l+n_r)/2;

return f(2*node,n_l,mid,q_l,q_r) + f(2*node+1,mid+1,n_r,q_l

,q_r);

}

void build_tree(vi &a,int n){

tree.clear();

int m=n;

while(__builtin_popcount(m)!=1)++m;

tree.resize(2*m+1,0);

for(int i=0;i<n;++i)tree[i+m]=a[i];

for(int i=m-1;i>=1;--i)tree[i]=tree[2*i]+tree[2*i+1];

}

LazySegmentTree.h

Description: Segment tree with lazy propagation update for range sum

Time: $\mathcal{O}(\log N)$

2f108e, 51 lines

//TODO: use 0 base indexing

vector<long long> tree,lazy;

void update(int node,int n_l,int n_r,int q_l,int q_r,int value)

{

if(lazy[node]!=0){

tree[node]+=(long long)(n_r-n_l+1)*lazy[node];

// for range + update

if(n_l!=n_r){

lazy[2*node]+=lazy[node];

lazy[2*node+1]+=lazy[node];

}

lazy[node] = 0;

}

if(n_r<q_l || q_r<n_l)return;

if(q_l<=n_l && n_r<=q_r){

tree[node]+=(long long)(n_r-n_l+1)*value;

// for range + update

if(n_l!=n_r){

lazy[2*node]+=value;

lazy[2*node+1]+=value;

}

return;

}

int mid = (n_r+n_l)/2;

update(2*node,n_l,mid,q_l,q_r,value);

update(2*node+1,mid+1,n_r,q_l,q_r,value);

tree[node] = tree[2*node] + tree[2*node+1];

}

long long f(int node,int n_l,int n_r,int q_l,int q_r){

if(lazy[node]!=0){

tree[node]+=(long long)(n_r-n_l+1)*lazy[node];

if(n_l!=n_r){

lazy[2*node] += lazy[node];

lazy[2*node+1] += lazy[node];

}

lazy[node] = 0;

}

if(n_r<q_l || q_r<n_l)return 0;

if(q_l<=n_l && n_r<=q_r)return tree[node];

int mid = (n_l+n_r)/2;

return f(2*node,n_l,mid,q_l,q_r) + f(2*node+1,mid+1,n_r,q_l

,q_r);

}

void build_tree(vi &a,int n){

tree.clear(); lazy.clear();

int m=n;

while(__builtin_popcount(m)!=1)++m;

tree.resize(2*m+1,0); lazy.resize(2*m+1,0);

for(int i=0;i<n;++i)tree[i+m]=a[i];

for(int i=m-1;i>=1;--i)tree[i]=tree[2*i]+tree[2*i+1];

}

OrderStatisticTree.h

Description: find nth largest element, count elements strictly less than x
Time: $\mathcal{O}(\log N)$

<ext/pb.ds/assoc.container.hpp>, <ext/pb.ds/tree.policy.hpp>

033b41, 7 lines

```
typedef __gnu_pbds::tree<int, __gnu_pbds::null_type, less<int>,
    __gnu_pbds::rb_tree_tag, __gnu_pbds::
    tree_order_statistics_node_update> ordered_set;
```

```
ordered_set st;
//st.order_of_key(x) - find # of elements in st strictly less
    than x
//st.size() - size of st
//st.find_by_order(x) - return iterator to the x-th largest
    element
//st.clear() - clear container
```

MergeSortTree.h

Description: do the same with orderstatistic tree but now over interval can be used/modify for some possible interval/subbarray queries
Time: $\mathcal{O}(\log N^3)$

<ext/pb.ds/assoc.container.hpp>, <ext/pb.ds/tree.policy.hpp>,
<ext/pb.ds/assoc.container.hpp>, <ext/pb.ds/tree.policy.hpp>

2d0612, 94 lines

```
//merge sort tree with fenwick tree
typedef __gnu_pbds::tree<pair<int,int>, __gnu_pbds::null_type,
    less<pair<int,int>>, __gnu_pbds::rb_tree_tag, __gnu_pbds::
    tree_order_statistics_node_update> ordered_set;
ordered_set st;
//st.order_of_key({x,-1}) - find # of elements in st stricly
    less than x
//st.clear() - clear container
```

```
vector<ordered_set> mtree;

ordered_set merge(ordered_set &a, ordered_set &b){
    ordered_set result;
    for(auto&p:a){
        result.ins(p);
    }
    for(auto&p:b){
        result.ins(p);
    }
    return result;
}
```

```
void update(int node,int n_l,int n_r,int q_i,int id,int old_val
    ,int value){
    if(n_r<q_i || q_i<n_l)return;
    if(q_i==n_l && n_r==q_i){
        auto it=mtree[node].find({old_val,id});
        mtree[node].erase(it);
        mtree[node].ins({value,id});
        return;
    }
    int mid = (n_r+n_l)/2;
    update(2*node,n_l,mid,q_i,id,old_val,value);
    update(2*node+1,mid+1,n_r,q_i,id,old_val,value);
    auto it=mtree[node].find({old_val,id});
    if(it!=mtree[node].end()){
        mtree[node].erase(it);
        mtree[node].ins({value,id});
    }
}
```

```
int f(int node,int n_l,int n_r,int q_l,int q_r, int value){
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l==n_l && n_r==q_r){
        return mtree[node].order_of_key({value,-1});
    }
}
```

```
int mid = (n_l+n_r)/2;
return f(2*node,n_l,mid,q_l,q_r,value)+f(2*node+1,mid+1,n_r
    ,q_l,q_r,value);
}
```

```
void build_mtree(vi &a){
    int n=(int)a.size();
    int m=n; while(__builtin_popcount(m)!=1)++m;
    //for(int i=0;i<2*m;++i)mtree[i].clear();
    mtree.resize(2*m);
    for(int i=0;i<n;++i)mtree[i+m].ins({a[i],i});
    for(int i=m-1;i>1;--i)mtree[i]=merge(mtree[2*i],mtree[2*i
        +1]);
}
//merge sort tree with fenwick tree(BIT) (4 times less space)
```

```
typedef __gnu_pbds::tree<pair<int,int>, __gnu_pbds::null_type,
    less<pair<int,int>>, __gnu_pbds::rb_tree_tag, __gnu_pbds::
    tree_order_statistics_node_update> ordered_set;
ordered_set st;
//st.order_of_key(x) - find # of elements in st stricly less
    than x
```

```
//TODO: use 1 base indexing
vector<ordered_set>bit;
```

```
void update(int i,int k,int old_value, int new_value){
    while(i<(int)bit.size()){
        auto it=bit[i].find({old_value,k});
        assert(it!=bit[i].end());
        if(it!=bit[i].end()){
            bit[i].erase(it);
        }
        bit[i].ins({new_value,k});
        i+=i&-i;//add last set bit
    }
}
```

```
int F(int i, int k){//culmulative sum to ith data
    int sum=0;
    while(i>0){
        sum+=bit[i].order_of_key({k,-1});
        i-=i&-i;
    }
    return sum;
}
```

```
void build_bit(vi &a){
    bit.resize((int)a.size());
    for(int i=1;i<(int)a.size();++i)bit[i].ins({a[i],i});
    for(int i=1;i<(int)bit.size();++i){
        int p=i+(i&-i);//index to parent
        if(p<(int)bit.size()){
            for(auto&x:bit[i])bit[p].ins(x);
        }
    }
}
```

MoQueries.h

Description: answering offline queries
Time: $\mathcal{O}\left((N+Q)\sqrt{N}\right)$

```
/* TODO: use 0 based indexing*/
void remove(int idx); // TODO: remove value at idx from data
    structure
void add(int idx); // TODO: add value at idx from data
    structure
int get_answer(); // TODO: extract the current answer of the
    data structure
```

```
int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
            make_pair(other.l / block_size, other.r);
    }
};
```

```
vector<int> mo_s_algorithm(vector<Query>& queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());
```

```
// TODO: initialize data structure

int cur_l = 0;
int cur_r = -1;
// invariant: data structure will always reflect the range
    [cur_l, cur_r]
for (Query q : queries) {
    while (cur_l > q.l) {
        cur_l--;
        add(cur_l);
    }
    while (cur_r < q.r) {
        cur_r++;
        add(cur_r);
    }
    while (cur_l < q.l) {
        remove(cur_l);
        cur_l++;
    }
    while (cur_r > q.r) {
        remove(cur_r);
        cur_r--;
    }
    answers[q.idx] = get_answer();
}
return answers;
}
```

FenwickTree.h

Description: find culmulative sum to ith element
Time: $\mathcal{O}(\log N)$

//TODO: use 1 base indexing

vector<long long>bit;

```
//range sum point update(k=new_val-old_val)
void add(int i, int k){//add k to ith data and it's parent
    range so on
    while(i<(int)bit.size()){
        bit[i]+=k;
        i+=i&-i;//add last set bit
    }
}
```

```
11 sum(int i){//culmulative sum to ith data
11 sum=0;
    while(i>0){
        sum+=bit[i];
        i-=i&-i;
    }
    return sum;
}
```



```
void build_bit(vl &a){
    bit=a;
    for(int i=1;i<(int)bit.size();++i){
        int p=i+(i&-i);//index to parent
        if(p<(int)bit.size())bit[p]+=bit[i];
    }
}
```

RMQ.h
Description: Range Minimum Queries on an array. solving offline queries in $\mathcal{O}(1)$
Time: build $\mathcal{O}(N \log N)$ query $\mathcal{O}(1)$

```
e37d08, 24 lines

int rmq[N][20];

void build_rmq(vi &a){
    for(int j=0;j<20;++j){
        for(int i=0;i<(int)a.size();++i){
            if(j==0){
                rmq[i][0]=a[i];
            } else if(i+(1<<(j-1))<(int)a.size()){
                rmq[i][j]=min(rmq[i][j-1],rmq[i+(1<<(j-1))][j-1]);
            }
        }
    }
}

int query(int l, int r){
    int i=l,sub_array_size=r-l+1, ans=INF;
    for(int j=0;j<30;++j){
        if((1<<j)&(sub_array_size)){
            ans=min(ans,rmq[i][j]);
            i+=(1<<j);
        }
    }
    return ans;
}
```

HashMap.h
Description: Hash map with mostly the same API as unordered_map, but ~3x faster. Uses 1.5x memory. Initial capacity must be a power of 2 (if provided).
<ext/pb_ds/assoc.container.hpp> 3b295b, 7 lines

```
using namespace __gnu_pbds;

const int RANDOM = chrono::high_resolution_clock::now().
    time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;
```

Graph (6)

6.1 Fundamentals

BellmanFord.h
Description: Calculates shortest paths from s in a graph that might have negative edge weights. Unreachable nodes get $\text{dist} = \text{inf}$; nodes reachable through negative-weight cycles get $\text{dist} = -\text{inf}$. Assumes $V^2 \max |w_i| < \sim 2^{63}$.
Time: $\mathcal{O}(VE)$

```
830a8f, 23 lines

const ll inf = LLONG_MAX;
struct Ed { int a, b, w, s() { return a < b ? a : -a; }};
struct Node { ll dist = inf; int prev = -1; };

void bellmanFord(vector<Node>& nodes, vector<Ed>& eds, int s) {
```

```
nodes[s].dist = 0;
sort(all(eds), [](Ed a, Ed b) { return a.s() < b.s(); });

int lim = sz(nodes) / 2 + 2; // /3+100 with shuffled vertices
rep(i,0,lim) for (Ed ed : eds) {
    Node cur = nodes[ed.a], &dest = nodes[ed.b];
    if (abs(cur.dist) == inf) continue;
    ll d = cur.dist + ed.w;
    if (d < dest.dist) {
        dest.prev = ed.a;
        dest.dist = (i < lim-1 ? d : -inf);
    }
}
rep(i,0,lim) for (Ed e : eds) {
    if (nodes[e.a].dist == -inf)
        nodes[e.b].dist = -inf;
}
}
```

FloydWarshall.h
Description: Calculates all-pairs shortest path in a directed graph that might have negative edge weights. Input is an distance matrix m , where $m[i][j] = \text{inf}$ if i and j are not adjacent. As output, $m[i][j]$ is set to the shortest distance between i and j , inf if no path, or $-\text{inf}$ if the path goes through a negative-weight cycle.
Time: $\mathcal{O}(N^3)$

```
531245, 12 lines

const ll inf = 1LL << 62;
void floydWarshall(vector<vector<ll>>& m) {
    int n = sz(m);
    rep(i,0,n) m[i][i] = min(m[i][i], 0LL);
    rep(k,0,n) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) {
            auto newDist = max(m[i][k] + m[k][j], -inf);
            m[i][j] = min(m[i][j], newDist);
        }
    rep(k,0,n) if (m[k][k] < 0) rep(i,0,n) rep(j,0,n)
        if (m[i][k] != inf && m[k][j] != inf) m[i][j] = -inf;
}
```

TopoSort.h
Description: Topological sorting. Given is an oriented graph. Output is an ordering of vertices, such that there are edges only from left to right. If there are cycles, the returned list will have size smaller than n – nodes reachable from cycles will not be returned.
Time: $\mathcal{O}(|V| + |E|)$

```
66a137, 14 lines

vi topoSort(const vector<vi>& gr) {
    vi indeg(sz(gr)), ret;
    for (auto& li : gr) for (int x : li) indeg[x]++;
    queue<int> q; // use priority-queue for lexic. largest ans.
    rep(i,0,sz(gr)) if (indeg[i] == 0) q.push(i);
    while (!q.empty()) {
        int i = q.front(); // top() for priority queue
        ret.push_back(i);
        q.pop();
        for (int x : gr[i])
            if (--indeg[x] == 0) q.push(x);
    }
    return ret;
}
```

6.2 Network flow

PushRelabel.h
Description: Push-relabel using the highest label selection rule and the gap heuristic. Quite fast in practice. To obtain the actual flow, look at positive values only.
Time: $\mathcal{O}(V^2\sqrt{E})$

```
struct PushRelabel {
    struct Edge {
        int dest, back;
        ll f, c;
    };
    vector<vector<Edge>> g;
    vector<ll> ec;
    vector<Edge*> cur;
    vector<vi> hs; vi H;
    PushRelabel(int n) : g(n), ec(n), cur(n), hs(2*n), H(n) {}

    void addEdge(int s, int t, ll cap, ll rcap=0) {
        if (s == t) return;
        g[s].push_back({t, sz(g[t]), 0, cap});
        g[t].push_back({s, sz(g[s])-1, 0, rcap});
    }

    void addFlow(Edge& e, ll f) {
        Edge &back = g[e.dest][e.back];
        if (!ec[e.dest] && f) hs[H[e.dest]].push_back(e.dest);
        e.f += f; e.c -= f; ec[e.dest] += f;
        back.f -= f; back.c += f; ec[back.dest] -= f;
    }

    ll calc(int s, int t) {
        int v = sz(g); H[s] = v; ec[t] = 1;
        vi co(2*v); co[0] = v-1;
        rep(i,0,v) cur[i] = g[i].data();
        for (Edge& e : g[s]) addFlow(e, e.c);

        for (int hi = 0;;) {
            while (hs[hi].empty()) if (!hi--) return -ec[s];
            int u = hs[hi].back(); hs[hi].pop_back();
            while (ec[u] > 0) // discharge u
                if (cur[u] == g[u].data() + sz(g[u])) {
                    H[u] = le9;
                    for (Edge& e : g[u]) if (e.c && H[u] > H[e.dest]+1)
                        H[u] = H[e.dest]+1, cur[u] = &e;
                    if (++co[H[u]], !--co[hi] && hi < v)
                        rep(i,0,v) if (hi < H[i] && H[i] < v)
                            --co[H[i]], H[i] = v + 1;
                    hi = H[u];
                } else if (cur[u]->c && H[u] == H[cur[u]->dest]+1)
                    addFlow(*cur[u], min(ec[u], cur[u]->c));
                else ++cur[u];
            }
        }
        bool leftOfMinCut(int a) { return H[a] >= sz(g); }
};
```

MinCostMaxFlow.h
Description: Min-cost max-flow. If costs can be negative, call setpi before maxflow, but note that negative cost cycles are not supported. To obtain the actual flow, look at positive values only.
Time: $\mathcal{O}(FE \log(V))$ where F is max flow. $\mathcal{O}(VE)$ for setpi.

```
58385b, 79 lines

#include <bits/extc++>.h

const ll INF = numeric_limits<ll>::max() / 4;

struct MCMF {
    struct edge {
        int from, to, rev;
        ll cap, cost, flow;
    };
    int N;
    vector<vector<edge>> ed;
    vi seen;
    vector<ll> dist, pi;
```

```
vector<edge*> par;

MCMF(int N) : N(N), ed(N), seen(N), dist(N), pi(N), par(N) {}

void addEdge(int from, int to, ll cap, ll cost) {
    if (from == to) return;
    ed[from].push_back(edge{ from,to,sz(ed[to]),cap,cost,0 });
    ed[to].push_back(edge{ to,from,sz(ed[from])-1,0,-cost,0 });
}

void path(int s) {
    fill(all(seen), 0);
    fill(all(dist), INF);
    dist[s] = 0; ll di;

    __gnu_pbds::priority_queue<pair<ll, int>> q;
    vector<decltype(q)::point_iterator> its(N);
    q.push({ 0, s });

    while (!q.empty()) {
        s = q.top().second; q.pop();
        seen[s] = 1; di = dist[s] + pi[s];
        for (edge& e : ed[s]) if (!seen[e.to]) {
            ll val = di - pi[e.to] + e.cost;
            if (e.cap - e.flow > 0 && val < dist[e.to]) {
                dist[e.to] = val;
                par[e.to] = &e;
                if (its[e.to] == q.end())
                    its[e.to] = q.push({ -dist[e.to], e.to });
                else
                    q.modify(its[e.to], { -dist[e.to], e.to });
            }
        }
        rep(i,0,N) pi[i] = min(pi[i] + dist[i], INF);
    }

    pair<ll, ll> maxflow(int s, int t) {
        ll totflow = 0, totcost = 0;
        while (path(s), seen[t]) {
            ll fl = INF;
            for (edge* x = par[t]; x; x = par[x->from])
                fl = min(fl, x->cap - x->flow);

            totflow += fl;
            for (edge* x = par[t]; x; x = par[x->from]) {
                x->flow += fl;
                ed[x->to][x->rev].flow -= fl;
            }
        }
        rep(i,0,N) for(edge& e : ed[i]) totcost += e.cost * e.flow;
        return {totflow, totcost/2};
    }

    // If some costs can be negative, call this before maxflow:
    void setpi(int s) { // (otherwise, leave this out)
        fill(all(pi), INF); pi[s] = 0;
        int it = N, ch = 1; ll v;
        while (ch-- && it--)
            rep(i,0,N) if (pi[i] != INF)
                for (edge& e : ed[i]) if (e.cap)
                    if ((v = pi[i] + e.cost) < pi[e.to])
                        pi[e.to] = v, ch = 1;
        assert(it >= 0); // negative cost cycle
    }
};
```

```
EdmondsKarp.h
Description: Flow algorithm with guaranteed complexity  $O(VE^2)$ . To get
edge flow values, compare capacities before and after, and take the positive
values only.
482fe0, 36 lines

template<class T> T edmondsKarp(vector<unordered_map<int, T>>&
    graph, int source, int sink) {
    assert(source != sink);
    T flow = 0;
    vi par(sz(graph)), q = par;

    for (;;) {
        fill(all(par), -1);
        par[source] = 0;
        int ptr = 1;
        q[0] = source;

        rep(i,0,ptr) {
            int x = q[i];
            for (auto e : graph[x]) {
                if (par[e.first] == -1 && e.second > 0) {
                    par[e.first] = x;
                    q[ptr++] = e.first;
                    if (e.first == sink) goto out;
                }
            }
        }
        return flow;
    }
out:
    T inc = numeric_limits<T>::max();
    for (int y = sink; y != source; y = par[y])
        inc = min(inc, graph[par[y]][y]);

    flow += inc;
    for (int y = sink; y != source; y = par[y]) {
        int p = par[y];
        if ((graph[p][y] -= inc) <= 0) graph[p].erase(y);
        graph[y][p] += inc;
    }
}
```

```
MinCut.h
Description: After running max-flow, the left side of a min-cut from  $s$  to  $t$ 
is given by all vertices reachable from  $s$ , only traversing edges with positive
residual capacity.
```

```
GlobalMinCut.h
Description: Find a global minimum cut in an undirected graph, as repre-
sented by an adjacency matrix.
Time:  $\mathcal{O}(V^3)$ 
8b0e19, 21 lines

pair<int, vi> globalMinCut(vector<vi> mat) {
    pair<int, vi> best = {INT_MAX, {}};
    int n = sz(mat);
    vector<vi> co(n);
    rep(i,0,n) co[i] = {i};
    rep(ph,1,n) {
        vi w = mat[0];
        size_t s = 0, t = 0;
        rep(it,0,n-ph) { //  $O(V^2) \rightarrow O(E \log V)$  with prio. queue
            w[t] = INT_MIN;
            s = t, t = max_element(all(w)) - w.begin();
            rep(i,0,n) w[i] += mat[t][i];
        }
        best = min(best, {w[t] - mat[t][t], co[t]});
        co[s].insert(co[s].end(), all(co[t]));
        rep(i,0,n) mat[s][i] += mat[t][i];
    }
}
```

```
        rep(i,0,n) mat[i][s] = mat[s][i];
        mat[0][t] = INT_MIN;
    }
    return best;
}
```

```
GomoryHu.h
Description: Given a list of edges representing an undirected flow graph,
returns edges of the Gomory-Hu tree. The max flow between any pair of
vertices is given by minimum edge weight along the Gomory-Hu tree path.
Time:  $\mathcal{O}(V)$  Flow Computations
*PushRelabel.h*
0418b3, 13 lines

typedef array<ll, 3> Edge;
vector<Edge> gomoryHu(int N, vector<Edge> ed) {
    vector<Edge> tree;
    vi par(N);
    rep(i,1,N) {
        PushRelabel D(N); // Dinic also works
        for (Edge t : ed) D.addEdge(t[0], t[1], t[2], t[2]);
        tree.push_back({i, par[i], D.calc(i, par[i])});
        rep(j,i+1,N)
            if (par[j] == par[i] && D.leftOfMinCut(j)) par[j] = i;
    }
    return tree;
}
```

6.3 Matching

```
hopcroftKarp.h
Description: Fast bipartite matching algorithm. Graph  $g$  should be a list
of neighbors of the left partition, and  $btoa$  should be a vector full of -1's of
the same size as the right partition. Returns the size of the matching.  $btoa[i]$ 
will be the match for vertex  $i$  on the right side, or  $-1$  if it's not matched.
Usage: vi btoa(m, -1); hopcroftKarp(g, btoa);
Time:  $\mathcal{O}(\sqrt{VE})$ 
f612e4, 42 lines

bool dfs(int a, int L, vector<vi>& g, vi& btoa, vi& A, vi& B) {
    if (A[a] != L) return 0;
    A[a] = -1;
    for (int b : g[a]) if (B[b] == L + 1) {
        B[b] = 0;
        if (btoa[b] == -1 || dfs(btoa[b], L + 1, g, btoa, A, B))
            return btoa[b] = a, 1;
    }
    return 0;
}
```

```
int hopcroftKarp(vector<vi>& g, vi& btoa) {
    int res = 0;
    vi A(g.size()), B(btoa.size()), cur, next;
    for (;;) {
        fill(all(A), 0);
        fill(all(B), 0);
        cur.clear();
        for (int a : btoa) if(a != -1) A[a] = -1;
        rep(a,0,sz(g)) if(A[a] == 0) cur.push_back(a);
        for (int lay = 1;; lay++) {
            bool islast = 0;
            next.clear();
            for (int a : cur) for (int b : g[a]) {
                if (btoa[b] == -1) {
                    B[b] = lay;
                    islast = 1;
                }
            }
            else if (btoa[b] != a && !B[b]) {
                B[b] = lay;
                next.push_back(btoa[b]);
            }
        }
    }
}
```

```
        if (islast) break;
        if (next.empty()) return res;
        for (int a : next) A[a] = lay;
        cur.swap(next);
    }
    rep(a,0,sz(g))
        res += dfs(a, 0, g, btoa, A, B);
}
}
```

DFSMatching.h

Description: Simple bipartite matching algorithm. Graph g should be a list of neighbors of the left partition, and $btoa$ should be a vector full of -1's of the same size as the right partition. Returns the size of the matching. $btoa[i]$ will be the match for vertex i on the right side, or -1 if it's not matched.
Usage: vi btoa(m , -1); dfsMatching(g , btoa);
Time: $\mathcal{O}(VE)$

522b98, 22 lines

```
bool find(int j, vector<vi>& g, vi& btoa, vi& vis) {
    if (btoa[j] == -1) return 1;
    vis[j] = 1; int di = btoa[j];
    for (int e : g[di])
        if (!vis[e] && find(e, g, btoa, vis)) {
            btoa[e] = di;
            return 1;
        }
    return 0;
}
int dfsMatching(vector<vi>& g, vi& btoa) {
    vi vis;
    rep(i,0,sz(g)) {
        vis.assign(sz(btoa), 0);
        for (int j : g[i])
            if (find(j, g, btoa, vis)) {
                btoa[j] = i;
                break;
            }
    }
    return sz(btoa) - (int)count(all(btoa), -1);
}
```

MinimumVertexCover.h

Description: Finds a minimum vertex cover in a bipartite graph. The size is the same as the size of a maximum matching, and the complement is a maximum independent set.

"DFSMatching.h" da4196, 20 lines

```
vi cover(vector<vi>& g, int n, int m) {
    vi match(m, -1);
    int res = dfsMatching(g, match);
    vector<bool> lfound(n, true), seen(m);
    for (int it : match) if (it != -1) lfound[it] = false;
    vi q, cover;
    rep(i,0,n) if (lfound[i]) q.push_back(i);
    while (!q.empty()) {
        int i = q.back(); q.pop_back();
        lfound[i] = 1;
        for (int e : g[i]) if (!seen[e] && match[e] != -1) {
            seen[e] = true;
            q.push_back(match[e]);
        }
    }
    rep(i,0,n) if (!lfound[i]) cover.push_back(i);
    rep(i,0,m) if (seen[i]) cover.push_back(n+i);
    assert(sz(cover) == res);
    return cover;
}
```

WeightedMatching.h

Description: Given a weighted bipartite graph, matches every node on the left with a node on the right such that no nodes are in two matchings and the sum of the edge weights is minimal. Takes $cost[N][M]$, where $cost[i][j] = cost$ for $L[j]$ to be matched with $R[j]$ and returns (min cost, match), where $L[i]$ is matched with $R[match[i]]$. Negate costs for max cost. Requires $N \leq M$.
Time: $\mathcal{O}(N^2M)$

1e0fe9, 31 lines

```
pair<int, vi> hungarian(const vector<vi> &a) {
    if (a.empty()) return {0, {}};
    int n = sz(a) + 1, m = sz(a[0]) + 1;
    vi u(n), v(m), p(m), ans(n - 1);
    rep(i,1,n) {
        p[0] = i;
        int j0 = 0; // add "dummy" worker 0
        vi dist(m, INT_MAX), pre(m, -1);
        vector<bool> done(m + 1);
        do { // dijkstra
            done[j0] = true;
            int i0 = p[j0], j1, delta = INT_MAX;
            rep(j,1,m) if (!done[j]) {
                auto cur = a[i0 - 1][j - 1] - u[i0] - v[j];
                if (cur < dist[j]) dist[j] = cur, pre[j] = j0;
                if (dist[j] < delta) delta = dist[j], j1 = j;
            }
            rep(j,0,m) {
                if (done[j]) u[p[j]] += delta, v[j] -= delta;
                else dist[j] -= delta;
            }
            j0 = j1;
        } while (p[j0]);
        while (j0) { // update alternating path
            int j1 = pre[j0];
            p[j0] = p[j1], j0 = j1;
        }
    }
    rep(j,1,m) if (p[j]) ans[p[j] - 1] = j - 1;
    return {-v[0], ans}; // min cost
}
```

GeneralMatching.h

Description: Matching for general graphs. Fails with probability N/mod .
Time: $\mathcal{O}(N^3)$

"../numerical/MatrixInverse-mod.h" cb1912, 40 lines

```
vector<pii> generalMatching(int N, vector<pii>& ed) {
    vector<vector<ll>> mat(N, vector<ll>(N)), A;
    for (pii pa : ed) {
        int a = pa.first, b = pa.second, r = rand() % mod;
        mat[a][b] = r, mat[b][a] = (mod - r) % mod;
    }

    int r = matInv(A = mat), M = 2*N - r, fi, fj;
    assert(r % 2 == 0);

    if (M != N) do {
        mat.resize(M, vector<ll>(M));
        rep(i,0,N) {
            mat[i].resize(M);
            rep(j,N,M) {
                int r = rand() % mod;
                mat[i][j] = r, mat[j][i] = (mod - r) % mod;
            }
        }
    } while (matInv(A = mat) != M);

    vi has(M, 1); vector<pii> ret;
    rep(it,0,M/2) {
        rep(i,0,M) if (has[i])
            rep(j,i+1,M) if (A[i][j] && mat[i][j]) {
```

```
                fi = i; fj = j; goto done;
            } assert(0); done:
        } if (fj < N) ret.emplace_back(fi, fj);
        has[fi] = has[fj] = 0;
        rep(sw,0,2) {
            ll a = modpow(A[fi][fj], mod-2);
            rep(i,0,M) if (has[i] && A[i][fj]) {
                ll b = A[i][fj] * a % mod;
                rep(j,0,M) A[i][j] = A[i][j] - A[fi][j] * b % mod;
            }
            swap(fi,fj);
        }
    }
    return ret;
}
```

6.4 DFS algorithms

SCC.h

Description: Finds strongly connected components in a directed graph. If vertices u, v belong to the same component, we can reach u from v and vice versa.

Usage: scc(graph, [&](vi& v) { ... }) visits all components in reverse topological order. comp[i] holds the component index of a node (a component only has edges to components with lower index). ncomps will contain the number of components.

Time: $\mathcal{O}(E + V)$

76b5c9, 24 lines

```
vi val, comp, z, cont;
int Time, ncomps;
template<class G, class F> int dfs(int j, G& g, F& f) {
    int low = val[j] = ++Time, x; z.push_back(j);
    for (auto e : g[j]) if (comp[e] < 0)
        low = min(low, val[e] ? dfs(e,g,f));

    if (low == val[j]) {
        do {
            x = z.back(); z.pop_back();
            comp[x] = ncomps;
            cont.push_back(x);
        } while (x != j);
        f(cont); cont.clear();
        ncomps++;
    }
    return val[j] = low;
}
template<class G, class F> void scc(G& g, F f) {
    int n = sz(g);
    val.assign(n, 0); comp.assign(n, -1);
    Time = ncomps = 0;
    rep(i,0,n) if (comp[i] < 0) dfs(i, g, f);
}
```

BiconnectedComponents.h

Description: Finds all biconnected components in an undirected graph, and runs a callback for the edges in each. In a biconnected component there are at least two distinct paths between any two nodes. Note that a node can be in several components. An edge which is not in a component is a bridge, i.e., not part of any cycle.

Usage: int eid = 0; ed.resize(N);

for each edge (a,b) {
ed[a].emplace_back(b, eid);
ed[b].emplace_back(a, eid++); }

bicomps([&](const vi& edgelist) {...});

Time: $\mathcal{O}(E + V)$

c6b7c7, 32 lines

```
vi num, st;
vector<vector<pii>> ed;
int Time;
template<class F>
```

```
int dfs(int at, int par, F& f) {
    int me = num[at] = ++Time, top = me;
    for (auto [y, e] : ed[at]) if (e != par) {
        if (num[y]) {
            top = min(top, num[y]);
            if (num[y] < me)
                st.push_back(e);
        } else {
            int si = sz(st);
            int up = dfs(y, e, f);
            top = min(top, up);
            if (up == me) {
                st.push_back(e);
                f(vi(st.begin() + si, st.end()));
                st.resize(si);
            }
            else if (up < me) st.push_back(e);
            else { /* e is a bridge */
            }
        }
    }
    return top;
}

template<class F>
void bicomps(F f) {
    num.assign(sz(ed), 0);
    rep(i,0,sz(ed)) if (!num[i]) dfs(i, -1, f);
}
```

2sat.h

Description: Calculates a valid assignment to boolean variables a, b, c,... to a 2-SAT problem, so that an expression of the type $(a||b)\&\&(!a||c)\&\&(d||!b)\&\&...$ becomes true, or reports that it is unsatisfiable. Negated variables are represented by bit-inversions (~x).

Usage: TwoSat ts(number of boolean variables);
ts.either(0, ~3); // Var 0 is true or var 3 is false
ts.setValue(2); // Var 2 is true
ts.atMostOne({0,~1,2}); // <= 1 of vars 0, ~1 and 2 are true
ts.solve(); // Returns true iff it is solvable
ts.values[0..N-1] holds the assigned values to the vars

Time: $\mathcal{O}(N + E)$, where N is the number of boolean variables, and E is the number of clauses.

5f9706, 56 lines

```
struct TwoSat {
    int N;
    vector<vi> gr;
    vi values; // 0 = false, 1 = true

    TwoSat(int n = 0) : N(n), gr(2*n) {}

    int addVar() { // (optional)
        gr.emplace_back();
        gr.emplace_back();
        return N++;
    }

    void either(int f, int j) {
        f = max(2*f, -1-2*f);
        j = max(2*j, -1-2*j);
        gr[f].push_back(j^1);
        gr[j].push_back(f^1);
    }

    void setValue(int x) { either(x, x); }

    void atMostOne(const vi& li) { // (optional)
        if (sz(li) <= 1) return;
        int cur = ~li[0];
        rep(i,2,sz(li)) {
            int next = addVar();
```

```
            either(cur, ~li[i]);
            either(cur, next);
            either(~li[i], next);
            cur = ~next;
        }
        either(cur, ~li[1]);
    }

    vi val, comp, z; int time = 0;
    int dfs(int i) {
        int low = val[i] = ++time, x; z.push_back(i);
        for(int e : gr[i]) if (!comp[e])
            low = min(low, val[e] ?: dfs(e));
        if (low == val[i]) do {
            x = z.back(); z.pop_back();
            comp[x] = low;
            if (values[x>>1] == -1)
                values[x>>1] = x&1;
        } while (x != i);
        return val[i] = low;
    }

    bool solve() {
        values.assign(N, -1);
        val.assign(2*N, 0); comp = val;
        rep(i,0,2*N) if (!comp[i]) dfs(i);
        rep(i,0,N) if (comp[2*i] == comp[2*i+1]) return 0;
        return 1;
    }
};
```

EulerWalk.h

Description: Eulerian undirected/directed path/cycle algorithm. Input should be a vector of (dest, global edge index), where for undirected graphs, forward/backward edges have the same index. Returns a list of nodes in the Eulerian path/cycle with src at both start and end, or empty list if no cycle/path exists. To get edge indices back, add .second to s and ret.

Time: $\mathcal{O}(V + E)$

780b64, 15 lines

```
vi eulerWalk(vector<vector<pii>>& gr, int nedges, int src=0) {
    int n = sz(gr);
    vi D(n), its(n), eu(nedges), ret, s = {src};
    D[src]++; // to allow Euler paths, not just cycles
    while (!s.empty()) {
        int x = s.back(), y, e, &it = its[x], end = sz(gr[x]);
        if (it == end){ ret.push_back(x); s.pop_back(); continue; }
        tie(y, e) = gr[x][it++];
        if (!eu[e]) {
            D[x]--, D[y]++;
            eu[e] = 1; s.push_back(y);
        }
    }
    for (int x : D) if (x < 0 || sz(ret) != nedges+1) return {};
    return {ret.rbegin(), ret.rend()};
}
```

6.5 Coloring

EdgeColoring.h

Description: Given a simple, undirected graph with max degree D , computes a $(D + 1)$ -coloring of the edges such that no neighboring edges share a color. (D -coloring is NP-hard, but can be done for bipartite graphs by repeated matchings of max-degree nodes.)

Time: $\mathcal{O}(NM)$

e210e2, 31 lines

```
vi edgeColoring(int N, vector<pii> eds) {
    vi cc(N + 1), ret(sz(eds)), fan(N), free(N), loc;
    for (pii e : eds) ++cc[e.first], ++cc[e.second];
    int u, v, ncols = *max_element(all(cc)) + 1;
    vector<vi> adj(N, vi(ncols, -1));
    for (pii e : eds) {
```

```
        tie(u, v) = e;
        fan[0] = v;
        loc.assign(ncols, 0);
        int at = u, end = u, d, c = free[u], ind = 0, i = 0;
        while (d = free[v], !loc[d] && (v = adj[u][d]) != -1)
            loc[d] = ++ind, cc[ind] = d, fan[ind] = v;
        cc[loc[d]] = c;
        for (int cd = d; at != -1; cd ^= c ^ d, at = adj[at][cd])
            swap(adj[at][cd], adj[end = at][cd ^ c ^ d]);
        while (adj[fan[i]][d] != -1) {
            int left = fan[i], right = fan[++i], e = cc[i];
            adj[u][e] = left;
            adj[left][e] = u;
            adj[right][e] = -1;
            free[right] = e;
        }
        adj[u][d] = fan[i];
        adj[fan[i]][d] = u;
        for (int y : {fan[0], u, end})
            for (int& z = free[y] = 0; adj[y][z] != -1; z++);
    }
    rep(i,0,sz(eds))
        for (tie(u, v) = eds[i]; adj[u][ret[i]] != v;) ++ret[i];
    return ret;
}
```

6.6 Heuristics

MaximalCliques.h

Description: Runs a callback for all maximal cliques in a graph (given as a symmetric bitset matrix; self-edges not allowed). Callback is given a bitset representing the maximal clique.

Time: $\mathcal{O}\left(3^{n/3}\right)$, much faster for sparse graphs

b0d5b1, 12 lines

```
typedef bitset<128> B;
template<class F>
void cliques(vector<B>& eds, F f, B P = ~B(), B X={}, B R={}) {
    if (!P.any()) { if (!X.any()) f(R); return; }
    auto q = (P | X)._Find_first();
    auto cands = P & ~eds[q];
    rep(i,0,sz(eds)) if (cands[i]) {
        R[i] = 1;
        cliques(eds, f, P & eds[i], X & eds[i], R);
        R[i] = P[i] = 0; X[i] = 1;
    }
}
```

MaximumClique.h

Description: Quickly finds a maximum clique of a graph (given as symmetric bitset matrix; self-edges not allowed). Can be used to find a maximum independent set by finding a clique of the complement graph.

Time: Runs in about 1s for n=155 and worst case random graphs (p=.90). Runs faster for sparse graphs.

f7c0bc, 49 lines

```
typedef vector<bitset<200>> vb;
struct Maxclique {
    double limit=0.025, pk=0;
    struct Vertex { int i, d=0; };
    typedef vector<Vertex> vv;
    vb e;
    vv V;
    vector<vi> C;
    vi qmax, q, S, old;
    void init(vv& r) {
        for (auto& v : r) v.d = 0;
        for (auto& v : r) for (auto j : r) v.d += e[v.i][j.i];
        sort(all(r), [](auto a, auto b) { return a.d > b.d; });
        int mxD = r[0].d;
        rep(i,0,sz(r)) r[i].d = min(i, mxD) + 1;
```

```

}
void expand(vv& R, int lev = 1) {
    S[lev] += S[lev - 1] - old[lev];
    old[lev] = S[lev - 1];
    while (sz(R)) {
        if (sz(q) + R.back().d <= sz(qmax)) return;
        q.push_back(R.back().i);
        vv T;
        for(auto v:R) if (e[R.back().i][v.i]) T.push_back({v.i});
        if (sz(T)) {
            if (S[lev]++ / ++pk < limit) init(T);
            int j = 0, mxk = 1, mnk = max(sz(qmax) - sz(q) + 1, 1);
            C[1].clear(), C[2].clear();
            for (auto v : T) {
                int k = 1;
                auto f = [&](int i) { return e[v.i][i]; };
                while (any_of(all(C[k]), f)) k++;
                if (k > mxk) mxk = k, C[mxk + 1].clear();
                if (k < mnk) T[j++].i = v.i;
                C[k].push_back(v.i);
            }
            if (j > 0) T[j - 1].d = 0;
            rep(k,mnk,mxk + 1) for (int i : C[k])
                T[j].i = i, T[j++].d = k;
            expand(T, lev + 1);
        } else if (sz(q) > sz(qmax)) qmax = q;
        q.pop_back(), R.pop_back();
    }
}
vi maxClique() { init(V), expand(V); return qmax; }
Maxclique(vb conn) : e(conn), C(sz(e)+1), S(sz(C)), old(S) {
    rep(i,0,sz(e)) V.push_back({i});
}
};
```

MaximumIndependentSet.h

Description: To obtain a maximum independent set of a graph, find a max clique of the complement. If the graph is bipartite, see MinimumVertex-Cover.

6.7 Trees

BinaryLifting.h

Description: Calculate power of two jumps in a tree, to support fast upward jumps and LCAs. Assumes the root node points to itself.
Time: construction $\mathcal{O}(N \log N)$, queries $\mathcal{O}(\log N)$

```

vector<vi> treeJump(vi& P){
    int on = 1, d = 1;
    while(on < sz(P)) on *= 2, d++;
    vector<vi> jmp(d, P);
    rep(i,1,d) rep(j,0,sz(P))
        jmp[i][j] = jmp[i-1][jmp[i-1][j]];
    return jmp;
}

int jmp(vector<vi>& tbl, int nod, int steps){
    rep(i,0,sz(tbl))
        if(steps&(1<<i)) nod = tbl[i][nod];
    return nod;
}

int lca(vector<vi>& tbl, vi& depth, int a, int b) {
    if (depth[a] < depth[b]) swap(a, b);
    a = jmp(tbl, a, depth[a] - depth[b]);
    if (a == b) return a;
    for (int i = sz(tbl); i--;) {
        int c = tbl[i][a], d = tbl[i][b];
        if (c != d) a = c, b = d;
    }
```

```

}
return tbl[0][a];
}

LCA.h
Description: Data structure for computing lowest common ancestors in a tree (with 0 as root). C should be an adjacency list of the tree, either directed or undirected.
Time:  $\mathcal{O}(N \log N + Q)$ 
"../data-structures/RMQ.h" 0f62fb, 21 lines

struct LCA {
    int T = 0;
    vi time, path, ret;
    RMQ<int> rmq;

    LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret)) {}
    void dfs(vector<vi>& C, int v, int par) {
        time[v] = T++;
        for (int y : C[v]) if (y != par) {
            path.push_back(v), ret.push_back(time[v]);
            dfs(C, y, v);
        }
    }

    int lca(int a, int b) {
        if (a == b) return a;
        tie(a, b) = minmax(time[a], time[b]);
        return path[rmq.query(a, b)];
    }
    //dist(a,b){return depth[a] + depth[b] - 2*depth[lca(a,b)];}
};

CompressTree.h
Description: Given a rooted tree and a subset S of nodes, compute the minimal subtree that contains all the nodes by adding all (at most  $|S| - 1$ ) pairwise LCA's and compressing edges. Returns a list of (par, orig_index) representing a tree rooted at 0. The root points to itself.
Time:  $\mathcal{O}(|S| \log |S|)$ 
"LCA.h" 9775a0, 21 lines

typedef vector<pair<int, int>> vpi;
vpi compressTree(LCA& lca, const vi& subset) {
    static vi rev; rev.resize(sz(lca.time));
    vi li = subset, &T = lca.time;
    auto cmp = [&](int a, int b) { return T[a] < T[b]; };
    sort(all(li), cmp);
    int m = sz(li)-1;
    rep(i,0,m) {
        int a = li[i], b = li[i+1];
        li.push_back(lca.lca(a, b));
    }
    sort(all(li), cmp);
    li.erase(unique(all(li)), li.end());
    rep(i,0,sz(li)) rev[li[i]] = i;
    vpi ret = {pii(0, li[0])};
    rep(i,0,sz(li)-1) {
        int a = li[i], b = li[i+1];
        ret.emplace_back(rev[lca.lca(a, b)], b);
    }
    return ret;
}
```

HLD.h

Description: Decomposes a tree into vertex disjoint heavy paths and light edges such that the path from any leaf to the root contains at most $\log(n)$ light edges. Code does additive modifications and max queries, but can support commutative segtree modifications/queries on paths and subtrees. Takes as input the full adjacency list. VALS_EDGES being true means that values are stored in the edges, as opposed to the nodes. All values initialized to the segtree default. Root must be 0.

Time: $\mathcal{O}((\log N)^2)$
"../data-structures/LazySegmentTree.h" 03139d, 46 lines

```

template <bool VALS_EDGES> struct HLD {
    int N, tim = 0;
    vector<vi> adj;
    vi par, siz, rt, pos;
    Node *tree;
    HLD(vector<vi> adj_)
        : N(sz(adj_)), adj(adj_), par(N, -1), siz(N, 1),
          rt(N),pos(N),tree(new Node(0, N)){ dfsSz(0); dfsHld(0); }
    void dfsSz(int v) {
        if (par[v] != -1) adj[v].erase(find(all(adj[v]), par[v]));
        for (int& u : adj[v]) {
            par[u] = v;
            dfsSz(u);
            siz[v] += siz[u];
            if (siz[u] > siz[adj[v][0]]) swap(u, adj[v][0]);
        }
    }
    void dfsHld(int v) {
        pos[v] = tim++;
        for (int u : adj[v]) {
            rt[u] = (u == adj[v][0] ? rt[v] : u);
            dfsHld(u);
        }
    }
    template <class B> void process(int u, int v, B op) {
        for (; rt[u] != rt[v]; v = par[rt[v]]) {
            if (pos[rt[u]] > pos[rt[v]]) swap(u, v);
            op(pos[rt[v]], pos[v] + 1);
        }
        if (pos[u] > pos[v]) swap(u, v);
        op(pos[u] + VALS_EDGES, pos[v] + 1);
    }
    void modifyPath(int u, int v, int val) {
        process(u, v, [&](int l, int r) { tree->add(l, r, val); });
    }
    int queryPath(int u, int v) { // Modify depending on problem
        int res = -1e9;
        process(u, v, [&](int l, int r) {
            res = max(res, tree->query(l, r));
        });
        return res;
    }
    int querySubtree(int v) { // modifySubtree is similar
        return tree->query(pos[v] + VALS_EDGES, pos[v] + siz[v]);
    }
};
```

LinkCutTree.h

Description: Represents a forest of unrooted trees. You can add and remove edges (as long as the result is still a forest), and check whether two nodes are in the same tree.

Time: All operations take amortized $\mathcal{O}(\log N)$.

```

struct Node { // Splay tree. Root's pp contains tree's parent.
    Node *p = 0, *pp = 0, *c[2];
    bool flip = 0;
    Node() { c[0] = c[1] = 0; fix(); }
    void fix() {
        if (c[0]) c[0]->p = this;
        if (c[1]) c[1]->p = this;
    }
```

```
// (+ update sum of subtree elements etc. if wanted)
}
void pushFlip() {
    if (!flip) return;
    flip = 0; swap(c[0], c[1]);
    if (c[0]) c[0]->flip ^= 1;
    if (c[1]) c[1]->flip ^= 1;
}
int up() { return p ? p->c[1] == this : -1; }
void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i], *y = b == 2 ? x : x->c[h], *z = b ? y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i ^ 1];
    if (b < 2) {
        x->c[h] = y->c[h ^ 1];
        y->c[h ^ 1] = x;
    }
    z->c[i ^ 1] = this;
    fix(); x->fix(); y->fix();
    if (p) p->fix();
    swap(pp, y->pp);
}
void splay() {
    for (pushFlip(); p; ) {
        if (p->p) p->p->pushFlip();
        p->pushFlip(); pushFlip();
        int c1 = up(), c2 = p->up();
        if (c2 == -1) p->rot(c1, 2);
        else p->p->rot(c2, c1 != c2);
    }
}
Node* first() {
    pushFlip();
    return c[0] ? c[0]->first() : (splay(), this);
}
};

struct LinkCut {
    vector<Node> node;
    LinkCut(int N) : node(N) {}

    void link(int u, int v) { // add an edge (u, v)
        assert(!connected(u, v));
        makeRoot(&node[u]);
        node[u].pp = &node[v];
    }
    void cut(int u, int v) { // remove an edge (u, v)
        Node *x = &node[u], *top = &node[v];
        makeRoot(top); x->splay();
        assert(top == (x->pp ? x->c[0]));
        if (x->pp) x->pp = 0;
        else {
            x->c[0] = top->p = 0;
            x->fix();
        }
    }
    bool connected(int u, int v) { // are u, v in the same tree?
        Node* nu = access(&node[u])->first();
        return nu == access(&node[v])->first();
    }
    void makeRoot(Node* u) {
        access(u);
        u->splay();
        if (u->c[0]) {
            u->c[0]->p = 0;
            u->c[0]->flip ^= 1;
            u->c[0]->pp = u;
            u->c[0] = 0;
        }
    }
};
```

```
u->fix();
}
}
Node* access(Node* u) {
    u->splay();
    while (Node* pp = u->pp) {
        pp->splay(); u->pp = 0;
        if (pp->c[1]) {
            pp->c[1]->p = 0; pp->c[1]->pp = pp; }
        pp->c[1] = u; pp->fix(); u = pp;
    }
    return u;
}
};

DirectedMST.h
Description: Finds a minimum spanning tree/arborescence of a directed
graph, given a root node. If no MST exists, returns -1.
Time: O(E log V)
"/data-structures/UnionFindRollback.h" 39e620, 60 lines

struct Edge { int a, b; ll w; };
struct Node {
    Edge key;
    Node *l, *r;
    ll delta;
    void prop() {
        key.w += delta;
        if (l) l->delta += delta;
        if (r) r->delta += delta;
        delta = 0;
    }
    Edge top() { prop(); return key; }
};
Node *merge(Node *a, Node *b) {
    if (!a || !b) return a ? b;
    a->prop(), b->prop();
    if (a->key.w > b->key.w) swap(a, b);
    swap(a->l, (a->r = merge(b, a->r)));
    return a;
}
void pop(Node*& a) { a->prop(); a = merge(a->l, a->r); }

pair<ll, vi> dmst(int n, int r, vector<Edge>& g) {
    RollbackUF uf(n);
    vector<Node*> heap(n);
    for (Edge e : g) heap[e.b] = merge(heap[e.b], new Node{e});
    ll res = 0;
    vi seen(n, -1), path(n), par(n);
    seen[r] = r;
    vector<Edge> Q(n), in(n, {-1,-1}), comp;
    deque<tuple<int, int, vector<Edge>>> cycs;
    rep(s,0,n) {
        int u = s, qi = 0, w;
        while (seen[u] < 0) {
            if (!heap[u]) return {-1,{};};
            Edge e = heap[u]->top();
            heap[u]->delta -= e.w, pop(heap[u]);
            Q[qi] = e, path[qi++] = u, seen[u] = s;
            res += e.w, u = uf.find(e.a);
            if (seen[u] == s) {
                Node* cyc = 0;
                int end = qi, time = uf.time();
                do cyc = merge(cyc, heap[w = path[--qi]]);
                while (uf.join(u, w));
                u = uf.find(u), heap[u] = cyc, seen[u] = -1;
                cycs.push_front({u, time, {Q[qi], Q[end]}});
            }
        }
    }
    rep(i,0,qi) in[uf.find(Q[i].b)] = Q[i];
};
```

```
}

for (auto& [u,t,comp] : cycs) { // restore sol (optional)
    uf.rollback(t);
    Edge inEdge = in[u];
    for (auto& e : comp) in[uf.find(e.b)] = e;
    in[uf.find(inEdge.b)] = inEdge;
}
rep(i,0,n) par[i] = in[i].a;
return {res, par};
}
```

6.8 Math

6.8.1 Number of Spanning Trees

Create an $N \times N$ matrix mat , and for each edge $a \rightarrow b \in G$, do $\text{mat}[a][b]--$, $\text{mat}[b][b]++$ (and $\text{mat}[b][a]--$, $\text{mat}[a][a]++$ if G is undirected). Remove the i th row and column and take the determinant; this yields the number of directed spanning trees rooted at i (if G is undirected, remove any row/column).

6.8.2 Erdős–Gallai theorem

A simple graph with node degrees $d_1 \geq \dots \geq d_n$ exists iff $d_1 + \dots + d_n$ is even and for every $k = 1 \dots n$,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

Strings (7)

KMP.h

Description: $\text{pi}[x]$ computes the length of the longest prefix of s that ends at x , other than $s[0\dots x]$ itself (abacaba -> 0010123). Can be used to find all occurrences of a string.

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

```
vi pi(const string& s) {
    vi p(sz(s));
    rep(i,1,sz(s)) {
        int g = p[i-1];
        while (g && s[i] != s[g]) g = p[g-1];
        p[i] = g + (s[i] == s[g]);
    }
    return p;
}
```

```
vi match(const string& s, const string& pat) {
    vi p = pi(pat + '\0' + s), res;
    rep(i, sz(p)-sz(s), sz(p))
        if (p[i] == sz(pat)) res.push_back(i - 2 * sz(pat));
    return res;
}
```

Zfunc.h

Description: $z[i]$ computes the length of the longest common prefix of $s[i:]$ and s , except $z[0] = 0$. (abacaba -> 0010301)

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

```
vi Z(const string& S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,1,sz(S)) {
```

```
z[i] = i >= r ? 0 : min(r - i, z[i - 1]);
while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
    z[i]++;
if (i + z[i] > r)
    l = i, r = i + z[i];
}
return z;
```

Manacher.h

Description: For each position in a string, computes $p[0][i]$ = half length of longest even palindrome around pos i , $p[1][i]$ = longest odd (half rounded down).

Time: $\mathcal{O}(N)$

```
array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi, 2> p = {vi(n+1), vi(n)};
    rep(z, 0, 2) for (int i=0, l=0, r=0; i < n; i++) {
        int t = r-i+!z;
        if (i<r) p[z][i] = min(t, p[z][l+t]);
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=1 && R+1<n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }
    return p;
}
```

MinRotation.h

Description: Finds the lexicographically smallest rotation of a string.

Usage: rotate(v.begin(), v.begin()+minRotation(v), v.end());

Time: $\mathcal{O}(N)$

```
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b, 0, N) rep(k, 0, N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
        if (s[a+k] > s[b+k]) {a = b; break;}
    }
    return a;
}
```

SuffixArray.h

Description: Builds suffix array for a string. $sa[i]$ is the starting index of the suffix which is i 'th in the sorted suffix array. The returned vector is of size $n+1$, and $sa[0] = n$. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: $lcp[i] = lcp(sa[i], sa[i-1])$, $lcp[0] = 0$. The input string must not contain any zero bytes.

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

```
struct SuffixArray {
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)+1), y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i, 0, n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i, 0, n) ws[x[i]]++;
            rep(i, 1, lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i, 1, n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        rep(i, 1, n) rank[sa[i]] = i;
    }
}
```

```
for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
    for (k && k--, j = sa[rank[i] - 1];
         s[i + k] == s[j + k]; k++);
}
```

SuffixTree.h

Description: Ukkonen's algorithm for online suffix tree construction. Each node contains indices $[l, r]$ into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining $[l, r]$ substrings. The root is 0 (has $l = -1, r = 0$), non-existent children are -1. To get a complete tree, append a dummy symbol – otherwise it may contain an incomplete path (still useful for substring matching, though).

Time: $\mathcal{O}(26N)$

```
struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*mazlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;

    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
                p[m++]=v; v=s[v]; q=r[v]; goto suff; }
            v=t[v][c]; q=l[v];
        }
        if (q==-1 || c==toi(a[q])) q++; else {
            l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
            p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
            l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;
            v=s[p[m]]; q=l[m];
            while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]; }
            if (q==r[m]) s[m]=v; else s[m]=m+2;
            q=r[v]-(q-r[m]); m+=2; goto suff;
        }
    }

    SuffixTree(string a) : a(a) {
        fill(r, r+N, sz(a));
        memset(s, 0, sizeof s);
        memset(t, -1, sizeof t);
        fill(t[1], t[1]+ALPHA, 0);
        s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
        rep(i, 0, sz(a)) ukkadd(i, toi(a[i]));
    }
}
```

```
// example: find longest common substring (uses ALPHA = 28)
pii best;
int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
    rep(c, 0, ALPHA) if (t[node][c] != -1)
        mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
        best = max(best, {len, r[node] - len});
    return mask;
}

static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
}
```

Hashing.h

Description: Self-explanatory methods for string hashing.

2d2a67, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
typedef uint64_t ull;
struct H {
    ull x; H(ull x=0) : x(x) {}
    H operator+(H o) { return x + o.x + (x + o.x < x); }
    H operator-(H o) { return *this + ~o.x; }
    H operator*(H o) { auto m = (__uint128_t)x * o.x;
        return H((ull)m) + (ull)(m >> 64); }
    ull get() const { return x + !x; }
    bool operator==(H o) const { return get() == o.get(); }
    bool operator<(H o) const { return get() < o.get(); }
};
static const H C = (1ll)1e11+3; // (order ~ 3e9; random also ok)
```

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

```
vector<H> getHashes(string& str, int length) {
    if (sz(str) < length) return {};
    H h = 0, pw = 1;
    rep(i, 0, length)
        h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h};
    rep(i, length, sz(str)) {
        ret.push_back(h = h * C + str[i] - pw * str[i-length]);
    }
    return ret;
}
```

```
H hashString(string& s){H h{}; for(char c:s) h=h*C+c;return h;}
```

AhoCorasick.h

Description: Aho-Corasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(–, word) finds all words (up to $N\sqrt{N}$ many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries.

Time: construction takes $\mathcal{O}(26N)$, where N = sum of length of patterns. find(x) is $\mathcal{O}(N)$, where N = length of x. findAll is $\mathcal{O}(NM)$.

f35677, 66 lines

```
struct AhoCorasick {
    enum {alpha = 26, first = 'A'}; // change this!
    struct Node {
        // (nmatches is optional)
        int back, next[alpha], start = -1, end = -1, nmatches = 0;
        Node(int v) { memset(next, v, sizeof(next)); }
    };
    vector<Node> N;
    vi backp;
```

```

void insert(string& s, int j) {
    assert(!s.empty());
    int n = 0;
    for (char c : s) {
        int& m = N[n].next[c - first];
        if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
        else n = m;
    }
    if (N[n].end == -1) N[n].start = j;
    backp.push_back(N[n].end);
    N[n].end = j;
    N[n].nmatches++;
}

AhoCorasick(vector<string>& pat) : N(1, -1) {
    rep(i, 0, sz(pat)) insert(pat[i], i);
    N[0].back = sz(N);
    N.emplace_back(0);

    queue<int> q;
    for (q.push(0); !q.empty(); q.pop()) {
        int n = q.front(), prev = N[n].back;
        rep(i, 0, alpha) {
            int &ed = N[n].next[i], y = N[prev].next[i];
            if (ed == -1) ed = y;
            else {
                N[ed].back = y;
                (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
                    = N[y].end;
                N[ed].nmatches += N[y].nmatches;
                q.push(ed);
            }
        }
    }
}

vi find(string word) {
    int n = 0;
    vi res; // ll count = 0;
    for (char c : word) {
        n = N[n].next[c - first];
        res.push_back(N[n].end);
        // count += N[n].nmatches;
    }
    return res;
}

vector<vi> findAll(vector<string>& pat, string word) {
    vi r = find(word);
    vector<vi> res(sz(word));
    rep(i, 0, sz(word)) {
        int ind = r[i];
        while (ind != -1) {
            res[i - sz(pat[ind]) + 1].push_back(ind);
            ind = backp[ind];
        }
    }
    return res;
}
};

```


Techniques (A)

techniques.txt	159 lines
Recursion	
Divide and conquer	
Finding interesting points in N log N	
Algorithm analysis	
Master theorem	
Amortized time complexity	
Greedy algorithm	
Scheduling	
Max contiguous subvector sum	
Invariants	
Huffman encoding	
Graph theory	
Dynamic graphs (extra book-keeping)	
Breadth first search	
Depth first search	
* Normal trees / DFS trees	
Dijkstra's algorithm	
MST: Prim's algorithm	
Bellman-Ford	
Konig's theorem and vertex cover	
Min-cost max flow	
Lovasz toggle	
Matrix tree theorem	
Maximal matching, general graphs	
Hopcroft-Karp	
Hall's marriage theorem	
Graphical sequences	
Floyd-Warshall	
Euler cycles	
Flow networks	
* Augmenting paths	
* Edmonds-Karp	
Bipartite matching	
Min. path cover	
Topological sorting	
Strongly connected components	
2-SAT	
Cut vertices, cut-edges and biconnected components	
Edge coloring	
* Trees	
Vertex coloring	
* Bipartite graphs (=> trees)	
* 3^n (special case of set cover)	
Diameter and centroid	
K'th shortest path	
Shortest cycle	
Dynamic programming	
Knapsack	
Coin change	
Longest common subsequence	
Longest increasing subsequence	
Number of paths in a dag	
Shortest path in a dag	
Dynprog over intervals	
Dynprog over subsets	
Dynprog over probabilities	
Dynprog over trees	
3^n set cover	
Divide and conquer	
Knuth optimization	
Convex hull optimizations	
RMQ (sparse table a.k.a 2^k-jumps)	
Bitonic cycle	
Log partitioning (loop over most restricted)	
Combinatorics	

Computation of binomial coefficients
Pigeon-hole principle
Inclusion/exclusion
Catalan number
Pick's theorem
Number theory
Integer parts
Divisibility
Euclidean algorithm
Modular arithmetic
* Modular multiplication
* 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
Unimodality and convex functions
Binary search on derivative
Numerical methods
Numeric integration
Newton's method
Root-finding with binary/ternary search
Golden section search
Matrices
Gaussian elimination
Exponentiation by squaring
Sorting
Radix sort
Geometry
Coordinates and vectors
* Cross product
* Scalar product
Convex hull
Polygon cut
Closest pair
Coordinate-compression
Quadtrees
KD-trees
All segment-segment intersection
Sweeping
Discretization (convert to events and sweep)
Angle sweeping
Line sweeping
Discrete second derivatives
Strings
Longest common substring
Palindrome subsequences

Knuth-Morris-Pratt
Tries
Rolling polynomial hashes
Suffix array
Suffix tree
Aho-Corasick
Manacher's algorithm
Letter position lists
Combinatorial search
Meet in the middle
Brute-force with pruning
Best-first (A*)
Bidirectional search
Iterative deepening DFS / A*
Data structures
LCA (2^k-jumps in trees in general)
Pull/push-technique on trees
Heavy-light decomposition
Centroid decomposition
Lazy propagation
Self-balancing trees
Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
Monotone queues / monotone stacks / sliding queues
Sliding queue using 2 stacks
Persistent segment tree