Chiang Mai University

# PramorKangDamCPP

Suthana Kwaonueng, Theerada Siri, Nitikhon Chantatham

2024-09-04

# Contest (1)

## template.cpp

86 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;
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 F0R(i, a) for (int i=0; i<(a); i++)
#define FORd(i,a,b) for (int i = (b)-1; i >= a; i--)
#define F0Rd(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();
}
```

## .vimrc

21 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
```

## .bashrc

1 lines

```
export PATH=$PATH:~/scripts/
```

## build.sh

1 lines

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

## stress.sh

22 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:** $\mathcal{O}(\log N)$

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:** $\mathcal{O}\left(\log N^3\right)$

<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:** $\mathcal{O}(\log N)$

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
    }
}

ll sum(int i){//culmulative sum to ith data
    ll 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 $\mathcal{O}(N \log N)$ 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=1,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:** $\mathcal{O}(\log N)$

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
ll 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> 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, 1));
  }
  if(Z(h2)) ans.pop_back();
  return ans;
}
```

## KMP.h

**Description:** pattern searching
**Time:** $\mathcal{O}(N+M)$

4dfee5, 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 subbarray 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:** $\mathcal{O}(\log N)$

62aeee, 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 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 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:** $\mathcal{O}\left(\sqrt{N}\right)$

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:** $\mathcal{O}\left(\sqrt{N}\right)$

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][k] += a[i][j] * other.a[j][k];
                }
            }
        }
        return product;
    }
};
//for calculating nth recurrence of function
//entry Matrix[i][j] is problability/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

```
//ceil2 work for bot positive and nagative a(maybe. never test
     it)
//ceil(a/b)
int ceil2(int a,int b){
  int res=a/b;
  if(b*res!=a)res+=(a>0)&(b>0);
  return res;
}
```

# 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=c

081502, 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 determinant

046106, 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:** nCk

4bb9fe, 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 manul

3373be, 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^(c-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 compression.

5d985a, 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 strongy connected components in G
// output: adj_cond -- adjacency list of G^SCC (by root
     vertices)
void strongy_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 (auto 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.

f63792, 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 Bipartile 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:** $\mathcal{O}\left(E * V^2\right)$ 9b8492, 38 lines

```
//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:** $\mathcal{O}\left(abs(left) * M\right)$ 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. 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;
}
```

# 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 $x = -b/2a$.

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

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. $a_n = (d_1 n + d_2) r^n$.

## 4.2 Trigonometry

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

$$\tan(v + w) = \frac{\tan v + \tan w}{1 - \tan v \tan w}$$

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

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

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

## 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: $\frac{a + b}{a - b} = \frac{\tan \frac{\alpha + \beta}{2}}{\tan \frac{\alpha - \beta}{2}}$

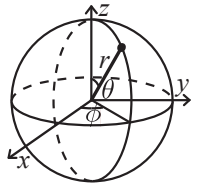
### 4.3.2 Quadrilaterals

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

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

### 4.3.3 Spherical coordinates

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




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

## 4.4 Derivatives/Integrals

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\tan x = 1+\tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\mathrm{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

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

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

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

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

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

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

## 4.6 Series

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

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

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

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

## 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) $\mu = \mathbb{E}(X) = \sum_x xp_X(x)$ 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 $\sigma$ 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 $\mathrm{Bin}(n,p)$, $n = 1,2,\ldots,$ $0 \leq p \leq 1$.

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

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

$\mathrm{Bin}(n,p)$ is approximately $\mathrm{Po}(np)$ 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 $\mathrm{Fs}(p)$, $0 \leq p \leq 1$.

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

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

**Poisson distribution**

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

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

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

### 4.7.2 Continuous distributions

**Uniform distribution**

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

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

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

**Exponential distribution**

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

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

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

**Normal distribution**

Most real random values with mean $\mu$ and variance $\sigma^2$ 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

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