UPgraded ICPC Notebook

Contents

1 C++

1.1 C++ Template

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#pragma GCC optimize("Ofast")
#pragma GCC optimize ("unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,
    mmx,avx,tune=native")
#define ll long long
#define pb push_back
#define ld long double
#define nl '\n'
#define fast cin.tie(0), cout.tie(0), ios_base::
    sync_with_stdio(false)
#define fore(i,a,b) for(ll i=a;i<b;++i)
#define rofe(i,a,b) for(ll i=a-1;i>=b;--i)
#define ALL(u) u.begin(),u.end()
#define vi vector <ll>
#define vvi vector<vi>
#define sz(a) ((ll)a.size())
#define lsb(x) ((x)&(-x))
#define lsbpos(x) __builtin_ffs(x)
#define PI acos(-1.0)
#define pii pair<ll,ll>
#define fst first
#define snd second
#define eb emplace_back
#define ppb pop_back
#define i128 __int128_t

using namespace __gnu_pbds;
using namespace std;

typedef tree<pair<int, int>, null_type, less<pair<int,
    int>>, rb_tree_tag, tree_order_statistics_node_update>
     ordered_multiset;
typedef tree<int,null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> ordered_set;

int main(){
    fast;
}
```

1.2 Bits Manipulation

```cpp
mask |= (1<<n) // prender bit-N
mask ^= (1<<n) // flippear bit-N
mask &= ~(1<<n) // apagar bit-N
if(mask&(1<<n)) // checkar bit-n
T = ((mask)&(-mask)) // LSO
__builtin_ffs(mask); // indice del LSO (indexado en 1)
//  Iterate over the subsets of S.
for(int subset= S; subset; subset= (subset-1) & S)
    for (int subset=0;subset=subset-S&S;) // Increasing
        order
```

1.3 Random

```
// Declare random number generator
mt19937_64 rng(0); // 64 bit, seed = 0
mt19937 rng(chrono::steady_clock::now().time_since_epoch
    ().count()); // 32 bit

// Use it to shuffle a vector
shuffle(all(vec), rng);

// Create int/real uniform dist. of type T in range [l, r
    ]
uniform_int_distribution<T> / uniform_real_distribution<T
    > dis(l, r);
dis(rng); // generate a random number in [l, r]

int rd(int l, int r) { return uniform_int_distribution<
    int>(l, r)(rng);}

srand(time(0)); //include this in main.
```

1.4 Other

```
#pragma GCC optimize("O3")
/*(UNCOMMENT WHEN HAVING LOTS OF RECURSIONS)\
#pragma comment(linker, "/stack:200000000")
(UNCOMMENT WHEN NEEDED) */
#pragma GCC optimize("Ofast,unroll-loops,no-stack-
    protector,fast-math")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4,popcnt,abm,
    mmx,avx,tune=native")

// Custom comparator for set/map
struct comp {
  bool operator()(const double& a, const double& b) const
        {
     return a+EPS<b;}
};
set<double,comp> w; // or map<double,int,comp>

// double inf
const double DINF=numeric_limits<double>::infinity();
```

2 Strings

2.1 Aho Corasick

```
const int MAXN = 1e6+10;

map<char, int> to[MAXN]; // if TLE change this to normal
    array.
string a[MAXN];
int lnk[MAXN],sz=1;
ll que[MAXN],endlink[MAXN];
vi leaf[MAXN],ans[MAXN];
void add_str(string s, int id) {
    int v = 0;
    for(char c: s) {
        if(!to[v].count(c)) to[v][c] = sz++;
        v = to[v][c];
    }
    leaf[v].pb(id); //Node in the automata where a word
        ends.
}
void push_links() {
    queue<int> q({0});
    while(!q.empty()) {
        int v = q.front(); q.pop();
        for(auto [c,u]: to[v])
        {
            int j = lnk[v];
            while(j && !to[j].count(c)) j=lnk[j];
            if(to[j].count(c) && to[j][c]!=u) lnk[u] = to
                [j][c];
            endlink[u] = leaf[lnk[u]].size()>0?lnk[u]:
                endlink[lnk[u]];
            q.push(u);
        }
    }
}
void walk(string s){ //KMP with multiple target patterns
    int v=0;
    fore(i,0,sz(s)) {
        char c=s[i];
        while(v && !to[v].count(c)) v=lnk[v];
        if(to[v].count(c)) v=to[v][c];
        for(int u=v;u;u=endlink[u]) for(int x: leaf[u]){
            ans[x].pb(i); //pushing the index of the main
                string where a pattern ends.
        }
    }
}

void doit(){
    string s;
    cin>>s; //main string.
    fore(i,0,n){
        cin>>a[i];
        add_str(a[i],i); //add target strings.
    }
    push_links();
    walk(s);
}
```

2.2 Hashing

```cpp
const ll maxn = 1e6 + 5;
const ll M = 1e9+9; // prime modulo.
const ll B = 131; // prime number bigger than the
    alphabet.
string s,t;
ll pws[maxn],h[maxn],tams,tamt,hasht=0,ans=0;

ll conv(char c){
    return (c-'a'+1);
}

bool sameHash(int l1, int len1, int l2, int len2){
    int r1 = l1 + len1;
    int r2 = l2 + len2;
    ll h1 = (h[r1]-h[l1]*pws[len1]%M + M)%M;
    ll h2 = (h[r2]-h[l2]*pws[len2]%M + M)%M;
    return h1 == h2;
}

void precalc(){
    tams = sz(s), tamt = sz(t);
    pws[0] = 1;
    fore(i,1,maxn) pws[i] = (pws[i-1]*B)%M;

    // Main hash.
    h[0] = conv(s[0]);
    fore(i,0,tams) h[i+1] = ((h[i]*B)+conv(s[i]))%M;

    // Target hash.
    fore(i,0,tamt) hasht = ((hasht*B)+conv(t[i]))%M;
}

void doit(){
    cin>>s; //main text.
    cin>>t; //pattern.
    precalc();

    //For all substrings, check if hashings of substring
        's' and 't' are equal:
    fore(i,tamt,tams+1){
        ll cur_hash = (h[i]-h[i-tamt]*pws[tamt]%M + M)%M;
        if (cur_hash == hasht) ans++;
    }
}
```

2.3 KMP

```cpp
vi kmp(string s){
    vi vs(sz(s));
    fore(i,1,sz(s)){
        int j = vs[i-1];
        while(j!=0 && s[i] != s[j]){
            j = vs[j-1];
        }
        if(s[i] == s[j]) j++;
        vs[i] = j;
    }
    return vs;
}

void doit(){
    string s,t,p;
    cin>>s; // main text.
    cin>>t; // target.
    p = t, p += "#", p += s;;
    vi res = kmp(p);
    ll ans = 0;
    for (auto au : res){
        if (au == sz(t)) ans++;
    }
}
```

2.4 Lyndon Factorization

```cpp
// A Lyndon word is a non-empty string that is strictly
    smaller than any of its non-trivial suffixes.
vector<string> duval(string const& s){
    int i = 0;
    vector<string> factorization;
    while (i < sz(s)) {
        int j = i + 1, k = i;
        while (j < sz(s) && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else k++;
            j++;
        }
        while (i <= k) {
            factorization.pb(s.substr(i, j - k));
            i += j - k;
        }
    }
    return factorization;
}

string duvalMinShift(string s){ // finds the minimum
    cyclic shift of a string.
    s += s;
    int i = 0, ans = 0;
    while (i < sz(s) / 2) {
        ans = i;
        int j = i + 1, k = i;
        while (j < sz(s) && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else k++;
            j++;
        }
        while (i <= k) i += j - k;
    }
    return s.substr(ans, sz(s) / 2);
}
```

2.5 Manacher

```
const ll MAXN = 1e6+5;
int d1[MAXN];//d1[i] = max odd palindrome centered on i
int d2[MAXN];//d2[i] = max even palindrome centered on i

//s  aabbaacaabbaa
//d1 1111117111111
//d2 0103010010301
void manacher(string& s){
  int l=0,r=-1,n=sz(s);
  fore(i,0,n){
    int k=i>r?1:min(d1[l+r-i],r-i);
    while(i+k<n&&i-k>=0&&s[i+k]==s[i-k])k++;
    d1[i]=k--;
    if(i+k>r)l=i-k,r=i+k;
  }
  l=0;r=-1;
  fore(i,0,n){
    int k=i>r?0:min(d2[l+r-i+1],r-i+1);k++;
    while(i+k<=n&&i-k>=0&&s[i+k-1]==s[i-k])k++;
    d2[i]=--k;
    if(i+k-1>r)l=i-k,r=i+k-1;
  }
}
```

2.6 Suffix Array

```
void csort(vi& sa, vi& r, int k){
  int n=sa.size();
  vi f(max(255,n),0),t(n);
  fore(i,0,n)f[RB(i+k)]++;
  int sum=0;
  fore(i,0,max(255,n))f[i]=(sum+=f[i])-f[i];
  fore(i,0,n)t[f[RB(sa[i]+k)]++]=sa[i];
  sa=t;
}
vi constructSA(string& s){
  int n=s.size(),rank;
  vi sa(n),r(n),t(n);
  fore(i,0,n)sa[i]=i,r[i]=s[i];
  for(int k=1;k<n;k*=2){
    csort(sa,r,k);csort(sa,r,0);
    t[sa[0]]=rank=0;
    fore(i,1,n){
      if(r[sa[i]]!=r[sa[i-1]]||RB(sa[i]+k)!=RB(sa[i-1]+k)
         )rank++;
      t[sa[i]]=rank;
    }
    r=t;
    if(r[sa[n-1]]==n-1)break;
  }
  return sa;
}

void doit(){
  string s;
  cin>>s;
  s = "$" + s; //add an extra symbol at the beginning to
     avoid conflicts.
  vi sa = constructSA(s);
}
```

2.7 Suffix Automaton

```
const ll maxn = 1e6+5;
struct state {int len,link;map<char,int> next;}; //clear
    next!!
state st[maxn];
int sz,last;
string s;

void sa_init(){
  last=st[0].len=0;sz=1;
  st[0].link=-1;
}
void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
//Finds longest common substring in 2 substrings.
string lcs (string S, string T) {
```

```
    sa_init();
    for (int i = 0; i < sz(S); i++) sa_extend(S[i]);
    int v = 0, l = 0, best = 0, bestpos = 0;
    for (int i = 0; i < T.size(); i++) {
        while (v && !st[v].next.count(T[i])) {
            v = st[v].link ;
            l = st[v].len;
        }
        if (st[v].next.count(T[i])) {
            v = st [v].next[T[i]];
            l++;
        }
        if (l > best) {
            best = l;
            bestpos = i;
        }
    }
    return T.substr(bestpos - best + 1, best);
}

ll f(ll x, vector <ll> &dp){
    if (dp[x] >= 0) return dp[x];
    ll res = 1;
    for(auto it=st[x].next.begin(); it!=st[x].next.end();
        it++) res += f(it->second,dp);
    dp[x] = res;
    return dp[x];
}
//Finds the total length of different substrings.
ll get_tot_len_diff_substings(){
    ll tot = 0;
    for(int i = 1; i < sz; i++) {
        ll shortest = st[st[i].link].len + 1;
        ll longest = st[i].len;

        ll num_strings = longest - shortest + 1;
        ll cur = num_strings * (longest + shortest) / 2;
        tot += cur;
    }
    return tot;
}
//Finds the amount of distinct substrings.
void distinctSubstrings(){
    cin>>s;
    int n = s.size();
    vi dp(maxn+5,-1);
    sa_init();
    fore(i,0,n) sa_extend(s[i]);
    ll ans = f(0,dp)-1;
    cout<<ans<<nl;
}
void dfs(int node, ll k, vi &dp, vector <char> &path){
    if (k < 0) return;
    for(const auto &[c,signode]: st[node].next){
        if (dp[signode] <= k) k -= dp[signode];
        else{
            path.pb(c);
            dfs(signode,k-1,dp,path);
            return;
        }
    }
}
//Finds the Kth biggest substring.
void substringOrder(){
    string s;
    ll k;
    cin>>s;
    cin>>k;
    int n = s.size();
    vector <ll> dp(maxn+5,-1);
    sa_init();
    fore(i,0,n) sa_extend(s[i]);
    f(0,dp);
    vector <char> path;
    dfs(0,k-1,dp,path);
    for(auto c : path) cout<<c;
    cout<<nl;
}
```

2.8 Trie

```
const int ALPHABET_SIZE = 26;

struct TrieNode
{
    struct TrieNode *children[ALPHABET_SIZE];
    bool isEndOfWord;
};

struct TrieNode *getNode(void){
    struct TrieNode *pNode =  new TrieNode;
    pNode->isEndOfWord = false;
    fore(i,0,ALPHABET_SIZE) pNode->children[i] = NULL;
    return pNode;
}

void insert(struct TrieNode *root, string key){
    struct TrieNode *pCrawl = root;
    for (int i = 0; i < key.length(); i++){
        int index = key[i] - 'a';
        if (!pCrawl->children[index])
            pCrawl->children[index] = getNode();
        pCrawl->children[index];
    }
    pCrawl->isEndOfWord = true;
}

bool search(struct TrieNode *root, string key){
    struct TrieNode *pCrawl = root;
    fore(i,0,sz(key)){
        int index = key[i] - 'a';
```

```
        if (!pCrawl->children[index]) return false;
        pCrawl = pCrawl->children[index];
    }
    return (pCrawl->isEndOfWord);
}

void doit(){
    struct TrieNode *root = getNode();
}
```

2.9 Z-Function

```
//Z Function for strings (longest prefix from start and
    from i).
//Complexity O(n).
const ll maxn = 2e5+5;
ll z[maxn];
void z_function(string s){
    for (int i = 1, l = 0, r = 0; i < sz(s); ++i) {
        if (i <= r){
            z[i] = min(r - i + 1, z[i - l]);
        }
        while (i + z[i] < sz(s) && s[z[i]] == s[i + z[i
            ]]){
            z[i]++;
        }
        if (i + z[i] - 1 > r){
            l = i, r = i + z[i] - 1;
        }
    }
}
```

3 Graph algorithms

3.1 2-SAT

```
const ll maxn = 1e5+100;

struct Sat2 {
    vector< vector<int> > g, rg;
    vector<int> component;
    vector<bool> ans;
    int tag, n, siz;
    stack<int> st;
    Sat2(int n) : n(n), siz(2*n), g(vector< vector<int>
        >(2*n)), rg(vector< vector<int> >(2*n)) {}
    void add_edge(int u, int v) {
    g[u].push_back(v);
    rg[v].push_back(u);
  }
  int neg(int u) {
      return (n+u)%siz;
  }
  void implication(int u, int v) {
    add_edge(neg(u), v);
  }
    // AND (a&b) = add(a&a), add(b&b)
  void add(int u, int v) { // OR = true (u or v is true).
    implication(u, v);
    implication(v, u);
  }
  void diff(int u, int v) { //XOR = true (both u and v
      are different).
        add(u, v);
        add(neg(u), neg(v));
  }
  void eq(int u, int v) { //XOR = false (both u and v are
        equal).
        diff(neg(u), v);
  }
    void dfs(int u, vector< vector<int> > &g, bool first)
        {
        component[u] = tag;
        for(int i = 0; i < g[u].size(); i++) {
            int v = g[u][i];
            if(component[v] == -1)
                dfs(v, g, first);
        }
        if(first) st.push(u);
    }
    bool satisfiable() {
        tag = 0;
        ans = vector<bool>(n);
        component = vector<int>(siz, -1);
        for(int i = 0; i < siz; i++) {
            if(component[i] == -1)
                dfs(i, g, true);
        }
        component = vector<int>(siz, -1);
        tag = 0;
        while(st.size()) {
            int u = st.top(); st.pop();
            if(component[u] != -1) continue;
            ++tag;
            dfs(u, rg, false);
        }
        for(int i = 0; i < n; i++) {
            if(component[i] == component[neg(i)]) return
                false;
            ans[i] = component[i] > component[neg(i)];
        }
        return true;
    }
};

void doit(){
    ll n;
```

```
    Sat2 sat(n);
    // insert clauses ...
    sat.satisfiable(); //run 2-SAT.
}
```

3.2 Bellman Ford

```
const ll maxn = 5050;
const ll mod = 1e9+7;
const ll INF = 1e17;

struct Edge {
    ll a, b, cost;
};

vector <Edge> edges;
ll n,m,ncy[maxn];

void bford(int stnode){ //When wanting to find the
   longest path, invert the signs of the costs (+ -)
    vi d(n+1, 0LL);     //to find shortest paths from
       stnode: set to INF.
                        //to find any negative cycle in
                           the graph, set to 0.
    //d[stnode] = 0;   <-- when having a starting node (
       task: find shortest paths), uncomment this.
    vi p(n+1, -1);
    int x = -1;
    fore(i,0,n){
        x = -1;
        for (Edge e : edges)
            if (d[e.a] < INF)
                if (d[e.b] > d[e.a] + e.cost) {
                    d[e.b] = max(-INF, d[e.a] + e.cost);
                    p[e.b] = e.a; //to keep track of the
                       path, pointing to the previous
                       node.
                    if (i+1 == n) ncy[e.b] = 1, x = e.b;
                       //e.b is part of a negative cycle.
                }
    }
    if (x == -1) cout<<"No negative cycles"<<nl;
    else{
        cout<<"Negative cycle: "<<nl;
        fore(i,0,n) x = p[x];
        vi cycle;
        ll start = x;
        cycle.pb(x);
        x = p[x];
        while(start != x){
            cycle.pb(x);
            x = p[x];
        }
        cycle.pb(start);
        reverse(ALL(cycle));
        for(auto au : cycle) cout<<au<<" ";
        cout<<nl;
    }
}

void doit(){
    // insert edges first.
    bford(0); // from start node.
}
```

3.3 Binary Lifting

```
const ll maxn = 2e5+5;
const ll loga = 20;
ll n,up[maxn][loga],dep[maxn];

void binlift(){
    // Assuming we have all direct parents for each node
       (up[node][0])
    fore(i,1,loga){
        fore(j,1,n+1) up[j][i] = up[up[j][i-1]][i-1];
    }
}

ll lca(ll x, ll y){ //calculate the depths of each node
   before.
    if (dep[x] < dep[y]) swap(x,y);
    ll dif = dep[x]-dep[y];
    rofe(i,loga,0){
        if (dif & (1ll<<i)) x = up[x][i];
    }
    if (x == y) return x;
    rofe(i,loga,0){
        if (up[x][i] != up[y][i]){
            x = up[x][i];
            y = up[y][i];
        }
    }
    return up[x][0];
}
```

3.4 Centroid Decomposition

```
const ll maxn = 2e5+5;
const ll loga = 20;
vi adj[maxn+5];
ll subt[maxn][loga],dep[maxn][loga],vis[maxn],cenpar[maxn
   ];
ll n,centroids_root;

ll dfsExplore(ll anode, ll node, ll depth, ll layer, vi &
   elms){
    dep[node][layer] = depth;
    subt[node][layer] = 1;
```

```
    elms.pb(node);
    for (auto au : adj[node]){
        if (anode != au && vis[au] == 0){
            subt[node][layer] += dfsExplore(node,au,depth
                +1,layer,elms);
        }
    }
    return subt[node][layer];
}
bool check(ll node, ll layer, ll tam){
    ll sum = 1;
    for (auto &au : adj[node]){
        if (dep[au][layer] > dep[node][layer]){
            sum += subt[au][layer];
            if (subt[au][layer] > tam/2) return false;
        }
    }
    if (tam-sum <= tam/2) return true;
    return false;
}
void centroidBuild(ll centroid_parent, ll node, ll layer)
   {
    vi elms;
    ll tam = dfsExplore(0,node,1,layer,elms); // change
        anode to -1 if nodes [0,n-1]
    for(auto &elm : elms){
        if (check(elm,layer,tam)){
            vis[elm] = 1;
            // Save each node's centroid parent.
            if (centroid_parent == -1){
                centroids_root = elm;
            }
            cenpar[elm] = centroid_parent;
            for(auto &signode : adj[elm]){ //expand to
                the children.
                if (vis[signode] == 0){
                    centroidBuild(elm,signode,layer+1);
                }
            }
            break;
        }
    }
}
void doit(){
    // create adjancency list first.
    centroidBuild(-1,1,0); //nodes [1,n]
}
```

3.5 Cycle Detection

```
const ll maxn = 2e5+5;
ll n,m,color[maxn],par[maxn];
vvi cycles;
vi adj[maxn];
void dfs_cycle(int u, int p){
    if (color[u] == 2) return;
    if (color[u] == 1) {
        vi v;
        int cur = p;
        v.pb(cur);
        while (cur != u){
            cur = par[cur];
            v.pb(cur);
        }
    //reverse(ALL(v)); //uncomment if graph is directed.
        cycles.pb(v);
        return;
    }
    par[u] = p;
    color[u] = 1;
    for (int v : adj[u]) {
      if (v == par[u]) { //remove IF graph is directed.
            continue;
        }
        dfs_cycle(v, u);
    }
    color[u] = 2;
}
```

3.6 Dijkstra

```
const ll INF = 1e18;
const ll maxn = 2e5+5;
ll n,m,d[maxn];
vector <pii> adj[maxn]; //{adjacent node,cost}
void daikra(int stnode){
    priority_queue<pii, vector<pii>, greater<pii> > pq;
    fore(i,0,n+1) d[i]=INF;
    d[stnode]=0;
    pq.push({d[stnode],stnode});
    while(!pq.empty()){
        ll curw = pq.top().fst;
        ll node = pq.top().snd;
        pq.pop();
        if (curw != d[node]) continue;
        for(auto au : adj[node]){
            int signode = au.fst;
            ll sigw = au.snd;
            if (d[signode] > d[node] + sigw){
                d[signode] = d[node] + sigw;
                pq.push({d[signode],signode});
            }
        }
    }
```

```
}
```

3.7 Euler Path Directed

```
const int maxn = 1e5+5;

ll n,m,indeg[maxn],outdeg[maxn];
vi g[maxn],path;

// Hierholzer's algorithm
// Directed graph: going from node 1, passing through all
    edges without repeating and end at node n.
void dfs(int node){
    while(!g[node].empty()){
        int signode = g[node].back();
        g[node].pop_back();
        dfs(signode);
    }
    path.pb(node);
}

void doit(){
    //Have out and in degree for each node first.
    bool flag=true;
    fore(i,2,n) if (indeg[i] != outdeg[i]) flag=false;
    if (indeg[1]+1 != outdeg[1] || indeg[n]-1 != outdeg[n
        ] || !flag){
        cout<<"IMPOSSIBLE"<<nl;
        return;
    }
    dfs(1);
    reverse(ALL(path));
    if (sz(path) != m+1 || path.back() != n) cout<<"
        IMPOSSIBLE"<<nl;
    else{
        for(auto node : path) cout<<node<<" ";
        cout<<nl;
    }
}
```

3.8 Euler Path Undirected

```
const int maxn = 1e5+5;
const int maxm = 2e5+5;

ll seen[maxm],n,m;
vi path;
vector < pii > g[maxn]; //{neighbor node, edge index}

// Hierholzer's algorithm
// Going from node 1, passing through all edges without
    repeating and come back to node 1.
void dfs(int node){
    while(!g[node].empty()){
        auto [signode, idx] = g[node].back();
        g[node].pop_back();
        if (seen[idx]) continue;
        seen[idx]=true;
        dfs(signode);
    }
    path.pb(node);
}

void doit(){
    // Create adjacency list.
    fore(i,0,n){
        if (sz(g[i])%2){
            cout<<"IMPOSSIBLE"<<nl;
            return;
        }
    }
    dfs(0);
    if (sz(path) != m+1) cout<<"IMPOSSIBLE"<<nl;
    else{
        for(auto node : path) cout<<node+1<<" ";
        cout<<nl;
    }
}
```

3.9 Floyd Warshall

```
struct Conn{ ll a,b,c; }; //{node a, node b, cost}
const ll maxn = 1e3+5;
const ll INF = 1e18;
ll n,m,q,d[maxn][maxn];
Conn adj[maxn];

void floyd_warshall(){
    fore(i,0,n+1) fore(j,0,n+1) d[i][j]=INF;
    for (int i = 1; i<=n; i++) d[i][i] = 0;
    for (auto au : adj){ //loop through the edges
        ll nd = au.a;
        ll nd2 = au.b;
        ll cost = au.c;
        d[nd][nd2] = min(d[nd][nd2], cost);
        d[nd2][nd] = min(d[nd2][nd], cost);
    }

    fore(i,1,n+1){
        fore(j,1,n+1){
            fore(k,1,n+1){
                d[j][k] = min(d[j][k], d[j][i] + d[i][k])
                    ;
            }
        }
    } //D[j][k] = shortest distance from j --> k
}
```

3.10 Heavy Light Decomposition

```
const int maxn = 2e5+50;
const int neut = 0;
const int loga = 19;
int n,qrys,label_cont;
int up[maxn][loga],subt[maxn],dep[maxn],labe[maxn],arr[
    maxn],tp[maxn],revlabe[maxn],st[maxn*4],p[loga];
vi adj[maxn];

void upd(int pos, int val, int node = 1, int ini = 1, int
     fin = n){
    if (ini == fin){
        st[node] = val;
        return;
    }
  int mid =(ini+fin)/2;
  if (pos <= mid) upd(pos,val,2*node,ini,mid);
    else upd(pos,val,2*node+1,mid+1,fin);
  st[node]=max(st[2*node],st[2*node+1]); // operation
}
int query(int l, int r, int node = 1, int ini = 1, int
    fin = n){
  if(fin < l || r < ini) return neut; // operation
      neutral
  if(l <= ini && fin <= r) return st[node];
  int mid =(ini+fin)/2;
  return max(query(l,r,2*node,ini,mid),query(l,r,2*node
      +1,mid+1,fin)); // operation
}

void init(){
    label_cont = 1;
    p[0] = 1;
    fore(i,1,loga) p[i] = (p[i-1] * 2LL);
}

int dfs_sz(int cur, int par) {
  subt[cur] = 1;
  for (int chi : adj[cur]) {
    if (chi == par) continue;
    dep[chi] = dep[cur] + 1;
    up[chi][0] = cur;
    subt[cur] += dfs_sz(chi, cur);
  }
  return subt[cur];
}

void dfs_hld(int cur, int par, int top) {
  labe[cur] = label_cont++;
  tp[cur] = top;
  upd(labe[cur], arr[cur]); //updating the STree using
      the labeling.
  int h_chi = -1, h_sz = -1;
  for (int chi : adj[cur]) {
    if (chi == par) continue;
    if (subt[chi] > h_sz) {
      h_sz = subt[chi];
      h_chi = chi;
    }
  }
  if (h_chi == -1) return;
  dfs_hld(h_chi, cur, top); //exploring the heavy edge
      first.
  for (int chi : adj[cur]) {
    if (chi == par || chi == h_chi) continue;
    dfs_hld(chi, cur, chi); //exploring the light edges.
  }
}

void binaryLift(){
    fore(i,1,loga){
        fore(j,1,n+1) up[j][i] = up[up[j][i-1]][i-1];
    }
}

ll lca(ll x, ll y){
    if (x == y) return x;
    if (dep[x] > dep[y]) swap(x,y); //'y' is deeper.
    ll dif = dep[y] - dep[x];
    rofe(i,loga,0){
        if (p[i] <= dif){
            dif -= p[i];
            y = up[y][i];
        }
    }
    if (x == y) return x;
    rofe(i,loga,0){
        if (up[x][i] != up[y][i]){
            x = up[x][i];
            y = up[y][i];
        }
    }
    return up[x][0];
}

int pathQuery(int chi, int par) {
  int ret = 0;
  while (chi != par) {
    if (tp[chi] == chi) { //querying for the top of the
        chain, no STree needed.
      ret = max(ret, arr[chi]);
      chi = up[chi][0];
    } else if (dep[tp[chi]] > dep[par]) { //queyring for
        the whole chain.
      ret = max(ret, query(labe[tp[chi]], labe[chi]));
      chi = up[tp[chi]][0];
    } else { //querying for a part of the chain
      ret = max(ret, query(labe[par] + 1, labe[chi]));
```

```
            break;
        }
    }
    return ret;
}

void doit(){ //Example querying and updating for maximum
    value.
    init();
    // 1. Read initial values for each node.
    // 2. Read and create adjacency list.
    dfs_sz(1,1);
    dfs_hld(1,1,1);
    binaryLift();
    // for updates:
    upd(labe[node],val);
    arr[node] = val;
    // for queries:
    ll lcan = lca(node,node2);
    ll q_ans = max({pathQuery(node,lcan),pathQuery(node2,
        lcan),arr[lcan]});
}
```

3.11 Kruskal

```
const int maxn = 1e5+5;

struct Edge{ ll a,b,c;}; //{Node1, Node2, Cost}.

vector <Edge> edges;

bool cmp(Edge e1, Edge e2){ return e1.c < e2.c;}

void doit(){
    // Initialize Union Find.
    // Read edges.
    ll totalw = 0; // total MST weight.
    sort(ALL(edges),cmp);
    for(auto edge : edges){
        if (bus(edge.a) != bus(edge.b)){
            join(edge.a,edge.b);
            totalw += edge.c;
        }
    }
}
```

3.12 Tarjan

```
const ll maxn = 2e5+10;
ll n,x,y,m,foundat=1;
ll low[maxn],disc[maxn],isArt[maxn],inStack[maxn];
vi adj[maxn];
vvi scc;
vector < pii > brid;

void dfs(int node, int antnode){ //first call antnode
    should be -1.
    static stack <int> stk;
    low[node] = disc[node] = foundat;
    stk.push(node);
    inStack[node] = 1;
    foundat++;
    int children = 0;
    for (auto signode : adj[node]){
        if(disc[signode] == 0){
            children++;
            dfs(signode, node);
            if (low[signode] > disc[node]){
                brid.pb({node,signode});
            }
            low[node] = min(low[node], low[signode]);
            if (antnode == -1 && children > 1) isArt[node
                ] = 1;
            if (antnode != -1 && low[signode] >= disc[
                node]) isArt[node] = 1;
        }
        //Remove some of this IF condition according to
            the desired function of Tarjan
        //When wanting to find SCC's:
        else if (inStack[signode] == 1){
            low[node] = min(low[node], disc[signode]);
        }
        //When wanting to find bridges or articulation
            points:
        /*
        else if (antnode != signode){
            low[node] = min(low[node], disc[signode]);
        } */
    }
    if (low[node] == disc[node]){ // for SCC's
        vi scctem;
        while (true){
            ll topic = stk.top();
            stk.pop();
            scctem.pb(topic);
            inStack[topic] = 0;
            if (node == topic) break;
        }
        scc.pb(scctem);
    }
}
```

3.13 Topo Sort

```
const int maxn = 2e5+5;
int n,m,indeg[maxn];
vi adj[maxn];
```

```
vi topo;
void topo_sort(){
    // 1. Create adjacency list with nodes' indegree.
    queue <int> q;
    fore(i,0,n) if (!indeg[i]) q.push(i);
    while (!q.empty()){
        int node = q.front();
        q.pop();
        topo.pb(node);
        for(auto signode : adj[node]){
            indeg[signode]--;
            if (!indeg[signode]) q.push(signode);
        }
    }
} // If sz(topo) != n, there is a cycle in the graph.
```

4 Flows

4.1 Dinic

```
struct FlowEdge {
  int v, u;
  ll cap, flow = 0;
  FlowEdge(int v, int u, ll cap) : v(v), u(u), cap(cap)
      {}
};
struct Dinic {
  const ll flow_inf = 1e18;
  vector<FlowEdge> edges;
  vvi adj;
  int n, m = 0;
  int s, t;
  vi level, ptr;
  queue<int> q;
  Dinic(int n, int s, int t) : n(n), s(s), t(t){
    adj.resize(n);
    level.resize(n);
    ptr.resize(n);
  }
  void add_edge(int v, int u, ll cap){ // v -> u
    edges.emplace_back(v, u, cap);
    edges.emplace_back(u, v, 0);
    adj[v].push_back(m);
    adj[u].push_back(m + 1);
    m += 2;
  }
  bool bfs(){
    while (!q.empty()) {
      int v = q.front();
      q.pop();
      for (int id : adj[v]) {
        if (edges[id].cap - edges[id].flow < 1) continue;
        if (level[edges[id].u] != -1) continue;
        level[edges[id].u] = level[v] + 1;
        q.push(edges[id].u);
      }
    }
    return level[t] != -1;
  }
  ll dfs(int v, ll pushed) {
    if (pushed == 0) return 0;
    if (v == t) return pushed;
    for (ll& cid = ptr[v]; cid < sz(adj[v]); cid++) {
      int id = adj[v][cid];
      int u = edges[id].u;
      if (level[v] + 1 != level[u] || edges[id].cap -
          edges[id].flow < 1) continue;
      ll tr = dfs(u, min(pushed, edges[id].cap - edges[id
          ].flow));
      if (tr == 0) continue;
      edges[id].flow += tr;
      edges[id ^ 1].flow -= tr;
      return tr;
    }
    return 0;
  }
  ll flow(){ // run the algorithm.
    ll f = 0;
    while (true) {
      fill(ALL(level), -1);
      level[s] = 0;
      q.push(s);
      if (!bfs()) break;
      fill(ALL(ptr), 0);
      while (ll pushed = dfs(s, flow_inf)) {
        f += pushed;
      }
    }
    return f;
  }
}; // initialize dinic(size,source_index,sink_index).

vector <FlowEdge> fe[maxn];
bool ok;

void dfs(int node, vi &path){
  if (node == n-1){ //when reaching the last node.
    ok=true;
    cout<<sz(path)<<nl;
    fore(i,0,sz(path)){
      cout<<path[i]+1<<" ";
    }
    cout<<nl;
    return;
  }
```

```
    for(auto &e : fe[node]){
      // Conditions of the next node that should be
         explored:
      if (!ok && e.flow > 0 && e.u < n){
        path.pb(e.u);
        e.flow--;
        dfs(e.u,path);
        path.ppb();
      }
    }
  }

  void doit(){
    // Initialize and build the flow graph.

    // To recover the paths of the flow:
    ll mf = dinic.flow();
    cout<<mf<<nl;

    for(FlowEdge e : dinic.edges){
      if (e.flow > 0){
        fe[e.v].pb(e);
      }
    }

    vi path;
    fore(i,0,mf){
      ok=false;
      dfs(source,path);
    }
  }
```

4.2 Kuhn

```
  int fn,sn;
  vector <bool> used;
  vector <int> mt;
  vvi g;

  bool kuhn(int v){
    if (used[v]) return false;
    used[v]=true;
    for(int to : g[v]){ //simple adjacency list.
      if (mt[to] == -1 || kuhn(mt[to])){
        mt[to]=v;
        return true;
      }
    }
    return false;
  }

  ll do_kuhn(){  //Complexity: O(n*m)
    mt.assign(sn,-1); //sn is the size of the second (right
       ) part size of the graph.
    ll mm = 0;
    fore(v,0,fn){ //fn is the size of the first(left) part
       size of the graph.
      used.assign(fn,false);
      if(kuhn(v)) mm++;
    }

    /* mt[i] this is the number of the vertex of the first
       part connected by an edge
    with the vertex i of the second part (or -1, if no
       matching edge comes out of it). */

    fore(i,0,sn){
      if (mt[i] != -1) cout<<"Connection: "<<mt[i] + 1<<" (
         left part) -- > "<<i + 1<<" (right part)"<<nl;
    }
    return mm; //maximum matching.
  }
```

4.3 Min-Cost Max-Flow

```
  const ll inf = 1e18+7;

  struct FlowGraph {

      struct Edge {
          ll to, flow, cap, cost;
          Edge *res;

          Edge () : to(0), flow(0), cap(0), cost(0), res(0)
               {}
          Edge (ll to, ll flow, ll cap, ll cost) : to(to),
              flow(flow), cap(cap), cost(cost), res(0) {}

          void addFlow (ll f) {
              flow += f;
              res->flow -= f;
          }
      };

    ll n;
      vector<vector<Edge*>> adj;
      vi dis, pos;

      FlowGraph (int n) : n(n), adj(n), dis(n), pos(n) {}

      void add (int u, int v, ll cap, ll cost) {
          Edge *x = new Edge(v, 0, cap, cost);
          Edge *y = new Edge(u, cap, cap, -cost);
          x->res = y;
          y->res = x;
          adj[u].pb(x);
          adj[v].pb(y);
      }

      pii mcmf(int s, int t, ll tope) {
          vector<bool> inq(n);
          vi dis(n), cap(n);
```

```
        vector<Edge*> par(n);
        ll maxFlow = 0, minCost = 0;

        while (maxFlow < tope) { //  compute MCF: maxflow
             < tope, compute MCMF maxflow < inf
            fill(ALL(dis), inf);
            fill(ALL(par), nullptr);
            fill(ALL(cap), 0);
            dis[s] = 0;
            cap[s] = inf;
            queue<int> q;
            q.push(s);

            while (sz(q)) {
                int u = q.front();
                q.pop();
                inq[u] = 0;

                for (Edge *v : adj[u]) {
                    if (v->cap > v->flow && dis[v->to] >
                        dis[u] + v->cost) {
                        dis[v->to] = dis[u] + v->cost;
                        par[v->to] = v;
                        cap[v->to] = min(cap[u], v->cap -
                            v->flow);

                        if (!inq[v->to]) {
                            q.push(v->to);
                            inq[v->to] = 1;
                        }
                    }
                }
            }

            if (!par[t]) break;

            maxFlow += cap[t];
            minCost += cap[t] * dis[t];
            for (int u = t; u != s; u = par[u]->res->to)
                par[u]->addFlow(cap[t]);
        }

        return {maxFlow, minCost};
    }
};

void doit(){
    // define src and sink.
    // edges src to node, and node to sink have cost 0.
    // to compute flow matches (e.g assignment problems),
         run dfs over the flow graph, keep the path and
        substract one unit of flow every time.
}
```

4.4 Push Relabel

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

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

  void addEdgeWithDemands(int u, int v, ll L, ll R) {
    addEdge(S, v, L);
    addEdge(u, T, L);
    addEdge(u, v, R - L);
  }

  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() {
    // to obtain the minimal flow with demands, binary
        search with this value to find the smallest one
        that provides the maxflow with demands.
    addEdge(t, s, LLONG_MAX);
    int v = sz(g); H[S] = v; ec[T] = 1;
    vi co(2*v); co[0] = v-1;
    fore(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] = 1e9;
          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)
            fore(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); }
};

void doit(){
    ll source=0, sink=1, source_prime = 2, sink_prime =
        3;
    PushRelabel flow(graph_size + 4,source,sink,
        source_prime,sink_prime);

    // Only source and sink will be used in the manual
        connections, do not use source_prime or sink_prime
}
```

5 Data Structures

5.1 Disjoint Set Union

```
struct UnionFind{
    int ran,pad,tam;
};
UnionFind uf[maxn];

int bus(int u){
  if (uf[u].pad!=u) uf[u].pad=bus(uf[u].pad);
  return uf[u].pad;
}

void unir(int u ,int v){
  u=bus(u); v=bus(v);
  if (u==v) return;
  if (uf[u].ran>uf[v].ran){
        uf[v].pad=u;
        uf[u].tam+=uf[v].tam;
    }
  else if (uf[u].ran<uf[v].ran) {
        uf[u].pad=v;
        uf[v].tam+=uf[u].tam;
    }
  else {
        uf[u].pad=v;
        uf[v].ran++;
        uf[v].tam+=uf[u].tam;
    }
  return;
}

void init(){
    fore(i,0,n){
        uf[i].pad = i;
        uf[i].ran = 0;
        uf[i].tam = 1;
    }
}
```

5.2 DSU with Rollbacks

```
struct dsu_save {
    int v, rnkv, u, rnku;

    dsu_save() {}
    dsu_save(int v, int rnkv, int u, int rnku)
        : v(v), rnkv(rnkv), u(u), rnku(rnku) {}
};

struct dsu_with_rollbacks {
    vi p, rnk;
    int comps;
    stack<dsu_save> op;

    dsu_with_rollbacks() {}
    dsu_with_rollbacks(int n) {
        p.resize(n);
        rnk.resize(n);
        fore(i,0,n){
            p[i] = i;
            rnk[i] = 0;
        }
        comps = n;
    }

    int find_set(int v){
        return (v == p[v]) ? v : find_set(p[v]);
    }
    bool unite(int v, int u){
        v = find_set(v);
        u = find_set(u);
        if (v == u) return false;
        comps--;
        if (rnk[v] > rnk[u]) swap(v, u);
        op.push(dsu_save(v, rnk[v], u, rnk[u]));
        p[v] = u;
        if (rnk[u] == rnk[v]) rnk[u]++;
        return true;
    }
    void rollback(){
        if (op.empty()) return;
        dsu_save x = op.top();
        op.pop();
        comps++;
        p[x.v] = x.v;
        rnk[x.v] = x.rnkv;
        p[x.u] = x.u;
        rnk[x.u] = x.rnku;
```

```cpp
    }
};

struct query {
    int v, u;
    bool united;
    query(int _v, int _u) : v(_v), u(_u) {}
};

struct QueryTree {
    vector<vector<query>> t;
    dsu_with_rollbacks dsu;
    int T;

    QueryTree() {}
    QueryTree(int T, int n) : T(T) {
        dsu = dsu_with_rollbacks(n);
        t.resize(4 * T + 4);
    }

    void add_to_tree(int v, int l, int r, int ul, int ur,
         query& q) {
        if (ul > ur)
            return;
        if (l == ul && r == ur) {
            t[v].pb(q);
            return;
        }
        int mid = (l+r)/2;
        add_to_tree(2 * v, l, mid, ul, min(ur, mid), q);
        add_to_tree(2 * v + 1, mid + 1, r, max(ul, mid +
            1), ur, q);
    }
    void add_query(query q, int l, int r) {
        add_to_tree(1, 0, T - 1, l, r, q);
    }
    void dfs(int v, int l, int r, vi& ans) {
        for (query& q : t[v]) {
            q.united = dsu.unite(q.v, q.u);
        }
        if (l == r)
            ans[l] = dsu.comps;
        else {
            int mid = (l + r) / 2;
            dfs(2 * v, l, mid, ans);
            dfs(2 * v + 1, mid + 1, r, ans);
        }
        for (query q : t[v]) {
            if (q.united) dsu.rollback();
        }
    }
    vi solve() {
        vi ans(T);
        dfs(1, 0, T - 1, ans);
        return ans;
    }
};
```

```cpp
void doit(){
    QueryTree qt(q+2,n+1); //Queries and nodes are 0-
        indexed.
    query edge(x,y); // Existing edge.
    // Add the living interval of an edge [l,r]. Close
        all edges.
    qt.add_query(edge,l,r);
    // Answer queries: amount of CCs at each moment i.
    // Substract -1 to the each answer.
    vi ans = qt.solve();
}
```

5.3 Fenwick Tree

```cpp
const int maxn = 1e5+5;
int arr[maxn];

struct Fen{
    // Sum of values. (1-indexed).
    void add(int x, int v){
        while (x <= maxn-5){
            arr[x] += v;
            x += lsb(x);
        }
    }
    // Getting to whole prefix [1,x]
    int get(int x){
        int freq = 0;
        while (x > 0){
            freq += arr[x];
            x -= lsb(x);
        }
        return freq;
    }
}; // To simulate add range updates [l,r,x], add +x in
   position l, and -x in position r+1
```

5.4 Merge Sort Tree

```cpp
struct Node{
    vi v;
    void order(){
        sort(ALL(v));
    }
    int get(int val_l, int val_r){ //nos interesa saber
        si al menos hay 1 elemento en el rango [val_l,
        val_r]
        return lower_bound(ALL(v),val_l)!=upper_bound(ALL
            (v),val_r);
    }
};
```

```
struct MSTree{
    vector <Node> st; int n;
    MSTree(int n): st(4*n + 5), n(n) {}
    void upd(int node, int ini, int fin, int pos, int val){
        st[node].v.pb(val);
        if (ini == fin) return;
        int mid = (ini+fin)/2;
        if (pos <= mid) upd(2*node,ini,mid,pos,val);
        else upd(2*node + 1,mid+1,fin,pos,val);
    }
    int query(int node, int ini, int fin, int l, int r, int val_l, int val_r){
        if (fin < l || r < ini) return 0;
        if (l <= ini && fin <= r) return st[node].get(val_l,val_r);
        int mid = (ini+fin)/2;
        return (query(2*node,ini,mid,l,r,val_l,val_r)|
            query(2*node + 1,mid+1,fin,l,r,val_l,val_r));
    }
    void order(){ fore(i,1,4*n + 5) st[i].order();} // after all insertions, sort all nodes.
    void upd(int pos, int val){ upd(1,1,n,pos,val);}
    int query(int l, int r, int val_l, int val_r){ return query(1,1,n,l,r,val_l,val_r);}
};
```

5.5 Monotonic Deque

```
const ll maxn = 2e5+10;
deque <int> q; //monotonic deque keeping maximums in front.

void add(int x){
    while(!q.empty() && q.back() < x) q.pop_back();
    q.pb(x);
}

void remove(int x){
    if (!q.empty() && q.front() == x) q.pop_front();
}

void clear(){
    while(!q.empty()) q.pop_back();
}

int getBest(){ return q.front(); }
```

5.6 Mo's Algorithm

```
const int maxn = 1e6+5;

struct qu{int l,r,id;};

ll n,nq,sq,res;

bool qcomp(const qu &a, const qu &b){
    if(a.l/sq!=b.l/sq) return a.l<b.l;
    return (a.l/sq)&1?a.r<b.r:a.r>b.r;
}
qu qs[maxn];
ll a[maxn],cnt[maxn],ans[maxn];

void add(int i){
    if (cnt[a[i]] == 0) res++;
    cnt[a[i]]++;
}

void remove(int i){
    cnt[a[i]]--;
    if (cnt[a[i]] == 0) res--;
}

ll get_ans(){
    return res;
}

void mos(){ // example amount of distinct elements in [l, r)
    fore(i,0,nq) qs[i].id=i;
    sq = sqrt(n)+0.5;
    sort(qs,qs+nq,qcomp); //sort the queries.
    int l=0,r=0;
    res=0;
    fore(i,0,nq){ // Must have queries like: [l,r)
        qu q=qs[i];
        while(l>q.l)add(--l);
        while(r<q.r)add(r++);
        while(l<q.l)remove(l++);
        while(r>q.r)remove(--r);
        ans[q.id]=get_ans();
    }
}
```

5.7 Mo's with Updates

```
struct qu{ll l,r,t,id;};

struct upd{ll pos,val;};

ll n,nq,sq,nu,res,len;

qu qs[maxn];
upd ups[maxn];
ll a[maxn],cnt[maxn],ans[maxn];
ll belong[maxn],lef[maxn],rig[maxn];

bool qcomp(const qu &x, const qu &y){
    if (belong[x.l] != belong[y.l]) return x.l < y.l;
```

```
    if (belong[x.r] != belong[y.r]) return x.r < y.r;
    return x.t < y.t;
}

void prepare(){
  len = pow(n,0.66666);
  sq = ceil(1.0*n/len);
  fore(i,0,sq){
    rig[i]=i*len;
    lef[i+1] = rig[i]+1;
  }
  rig[sq]=n-1;
  fore(i,1,sq+1){ //computing the belonging block of each
      position.
    fore(j,lef[i],rig[i]+1) belong[j]=i;
  }
}

void add(ll i){
    if (cnt[a[i]] == 0) res++;
    else if (cnt[a[i]] == 1) res--;
    cnt[a[i]]++;
}

void remove(ll i){
    if (cnt[a[i]] == 1) res--;
    else if (cnt[a[i]] == 2) res++;
    cnt[a[i]]--;
}

void update(ll id, ll l, ll r){
  ll pos = ups[id].pos;
  if (l <= pos && pos < r) remove(pos);
  swap(a[pos],ups[id].val);
  if (l <= pos && pos < r) add(pos);
}

ll get_ans(){
    return res;
}

void mos(){ // example amount of distinct elements in [l,
   r)
    fore(i,0,nq) qs[i].id=i;
    prepare();
    sort(qs,qs+nq,qcomp); //sort the queries.
    ll l=0,r=0;
    ll ut=0; //update time.
    res=0;
    fore(i,0,nq){ // Must have queries like: [l,r)
        qu q=qs[i];
        // move query range.
        while(l>q.l)add(--l);
        while(r<q.r)add(r++);
        while(l<q.l)remove(l++);
        while(r>q.r)remove(--r);
        // do and undo updates
        while (ut < q.t) update(ut,l,r), ut++;
        while(ut > q.t) ut--, update(ut,l,r);
        //get answer.
        ans[q.id]=get_ans();
    }
}

void doit(){
    ll ops; //amount of operations.
  cin>>n>>ops;
  fore(i,0,n) cin>>a[i]; //initial values.

  nq = nu = 0; //number of queries and updates.
  fore(i,0,ops){
    ll x,y,z;
    cin>>x>>y>>z;
    if (x == 1){ //update.
      ups[nu].pos = y;
      ups[nu].val = z;
      nu++;
    }
    else{ //query.
      z++;
      qs[nq].l = y;
      qs[nq].r = z;
      qs[nq].t = nu;
      nq++;
    }
  }
  mos();
  fore(i,0,nq) cout<<ans[i]<<nl;
}
```

5.8 Ordered Set

```
//Tested with: https://cses.fi/problemset/task/1076
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace __gnu_pbds;

typedef tree<pair<int, int>, null_type, less<pair<int,
   int>>, rb_tree_tag, tree_order_statistics_node_update>
    ordered_multiset;
typedef tree<int,null_type, less<int>, rb_tree_tag,
   tree_order_statistics_node_update> ordered_set;

ordered_multiset omst;
//find_by_order(k): finds the element that is the Kth in
   the set.
//order_of_key(k): finds the number of elements strictly
   smaller than k (or {k,pos} if multiset).

ll get_kth_element(ll k){ return (*omst.find_by_order(k))
   .fst;} // 0-indexed.
```

```
ll get_elements_less_than_k(ll k, ll pos){ return (omst.
    order_of_key({k, pos}));}

void insert(ll val, ll pos){ omst.insert({val,pos}); }

void erase(ll val, ll pos){ omst.erase(omst.find_by_order
    (omst.order_of_key({val,pos})));}
```

5.9 Segment Tree 2D

```
int n,m;
const int MAXN = 1e3;
int a[MAXN][MAXN],st[2*MAXN][2*MAXN];
const int NEUT = 0;

int op(int x, int y){ return x+y; }

void build(){
  fore(i,0,n)fore(j,0,m)st[i+n][j+m]=a[i][j];
  fore(i,0,n)for(int j=m-1;j;--j)
    st[i+n][j]=op(st[i+n][j<<1],st[i+n][j<<1|1]);
  for(int i=n-1;i;--i)fore(j,0,2*m)
    st[i][j]=op(st[i<<1][j],st[i<<1|1][j]);
}
void upd(int x, int y, int v){ //[x,y] coordinates, and
    value v.
  st[x+n][y+m]=v;
  for(int j=y+m;j>1;j>>=1)st[x+n][j>>1]=op(st[x+n][j],st[
      x+n][j^1]);
  for(int i=x+n;i>1;i>>=1)for(int j=y+m;j;j>>=1)
    st[i>>1][j]=op(st[i][j],st[i^1][j]);
}
// (x0,y0) inclusive, (x1,y1) exclusive (excluding that
    row and column).
int query(int x0, int y0, int x1, int y1){
  int r=NEUT;
  for(int i0=x0+n,i1=x1+n;i0<i1;i0>>=1,i1>>=1){
    int t[4],q=0;
    if(i0&1)t[q++]=i0++;
    if(i1&1)t[q++]=--i1;
    fore(k,0,q)for(int j0=y0+m,j1=y1+m;j0<j1;j0>>=1,j1
        >>=1){
      if(j0&1)r=op(r,st[t[k]][j0++]);
      if(j1&1)r=op(r,st[t[k]][--j1]);
    }
  }
  return r;
}
```

5.10 Segment Tree Beats

```
const ll maxn = 2e5+5;
const ll lzneut = 0;
const ll neut = 0;
const ll INF = 1e18+5;

struct Node{
    ll sum,max1,max2,maxc,min1,min2,minc,lazy;
};

ll a[maxn],N;

struct STBeats{
    vector <Node> st; int n;
    STBeats(int n): st(4*n + 5), n(n) {}

    void merge(int t) {
        // sum
        st[t].sum = st[t << 1].sum + st[t << 1 | 1].sum;

        // max
        if (st[t << 1].max1 == st[t << 1 | 1].max1) {
            st[t].max1 = st[t << 1].max1;
            st[t].max2 = max(st[t << 1].max2, st[t << 1 |
                1].max2);
            st[t].maxc = st[t << 1].maxc + st[t << 1 |
                1].maxc;
        } else {
            if (st[t << 1].max1 > st[t << 1 | 1].max1) {
                st[t].max1 = st[t << 1].max1;
                st[t].max2 = max(st[t << 1].max2, st[t <<
                    1 | 1].max1);
                st[t].maxc = st[t << 1].maxc;
            } else {
                st[t].max1 = st[t << 1 | 1].max1;
                st[t].max2 = max(st[t << 1].max1, st[t <<
                    1 | 1].max2);
                st[t].maxc = st[t << 1 | 1].maxc;
            }
        }

        // min
        if (st[t << 1].min1 == st[t << 1 | 1].min1) {
            st[t].min1 = st[t << 1].min1;
            st[t].min2 = min(st[t << 1].min2, st[t << 1 |
                1].min2);
            st[t].minc = st[t << 1].minc + st[t << 1 |
                1].minc;
        } else {
            if (st[t << 1].min1 < st[t << 1 | 1].min1) {
                st[t].min1 = st[t << 1].min1;
                st[t].min2 = min(st[t << 1].min2, st[t <<
                    1 | 1].min1);
                st[t].minc = st[t << 1].minc;
            } else {
```

```cpp
                st[t].min1 = st[t << 1 | 1].min1;
                st[t].min2 = min(st[t << 1].min1, st[t <<
                    1 | 1].min2);
                st[t].minc = st[t << 1 | 1].minc;
            }
        }
    }

    void push_add(int t, int tl, int tr, ll v) {
        if (v == 0) { return; }
        st[t].sum += (tr - tl + 1) * v;
        st[t].max1 += v;
        if (st[t].max2 != -INF) { st[t].max2 += v; }
        st[t].min1 += v;
        if (st[t].min2 != INF) { st[t].min2 += v; }
        st[t].lazy += v;
    }

    // corresponds to a chmin update
    void push_max(int t, ll v, bool l) {
        if (v >= st[t].max1) { return; }
        st[t].sum -= st[t].max1 * st[t].maxc;
        st[t].max1 = v;
        st[t].sum += st[t].max1 * st[t].maxc;
        if (l) {
            st[t].min1 = st[t].max1;
        } else {
            if (v <= st[t].min1) {
                st[t].min1 = v;
            } else if (v < st[t].min2) {
                st[t].min2 = v;
            }
        }
    }

    // corresponds to a chmax update
    void push_min(int t, ll v, bool l) {
        if (v <= st[t].min1) { return; }
        st[t].sum -= st[t].min1 * st[t].minc;
        st[t].min1 = v;
        st[t].sum += st[t].min1 * st[t].minc;
        if (l) {
            st[t].max1 = st[t].min1;
        } else {
            if (v >= st[t].max1) {
                st[t].max1 = v;
            } else if (v > st[t].max2) {
                st[t].max2 = v;
            }
        }
    }

    void pushdown(int t, int tl, int tr) {
        if (tl == tr) return;
        // sum
        int tm = (tl + tr) >> 1;
        push_add(t << 1, tl, tm, st[t].lazy);
        push_add(t << 1 | 1, tm + 1, tr, st[t].lazy);
        st[t].lazy = 0;

        // max
        push_max(t << 1, st[t].max1, tl == tm);
        push_max(t << 1 | 1, st[t].max1, tm + 1 == tr);

        // min
        push_min(t << 1, st[t].min1, tl == tm);
        push_min(t << 1 | 1, st[t].min1, tm + 1 == tr);
    }

    void build(int t = 1, int tl = 0, int tr = N - 1) {
        st[t].lazy = 0;
        if (tl == tr) {
            st[t].sum = st[t].max1 = st[t].min1 = a[tl];
            st[t].maxc = st[t].minc = 1;
            st[t].max2 = -INF;
            st[t].min2 = INF;
            return;
        }

        int tm = (tl + tr) >> 1;
        build(t << 1, tl, tm);
        build(t << 1 | 1, tm + 1, tr);
        merge(t);
    }
    //[l,r] ai += b
    void update_add(int l, int r, ll v, int t = 1, int tl
        = 0, int tr = N - 1) {
        if (r < tl || tr < l) { return; }
        if (l <= tl && tr <= r) {
            push_add(t, tl, tr, v);
            return;
        }
        pushdown(t, tl, tr);

        int tm = (tl + tr) >> 1;
        update_add(l, r, v, t << 1, tl, tm);
        update_add(l, r, v, t << 1 | 1, tm + 1, tr);
        merge(t);
    }
    //[l,r] ai = min(ai,x)
    void update_chmin(int l, int r, ll v, int t = 1, int
       tl = 0, int tr = N - 1){
        if (r < tl || tr < l || v >= st[t].max1) { return
            ; }
        if (l <= tl && tr <= r && v > st[t].max2) {
            push_max(t, v, tl == tr);
            return;
        }
        pushdown(t, tl, tr);

        int tm = (tl + tr) >> 1;
        update_chmin(l, r, v, t << 1, tl, tm);
```

```
        update_chmin(l, r, v, t << 1 | 1, tm + 1, tr);
        merge(t);
    }
    //[l,r] ai = max(ai,x)
    void update_chmax(int l, int r, ll v, int t = 1, int
        tl = 0, int tr = N - 1) {
        if (r < tl || tr < l || v <= st[t].min1) { return
            ; }
        if (l <= tl && tr <= r && v < st[t].min2) {
            push_min(t, v, tl == tr);
            return;
        }
        pushdown(t, tl, tr);

        int tm = (tl + tr) >> 1;
        update_chmax(l, r, v, t << 1, tl, tm);
        update_chmax(l, r, v, t << 1 | 1, tm + 1, tr);
        merge(t);
    }
    //print sum [l,r]
    ll query_sum(int l, int r, int t = 1, int tl = 0, int
         tr = N - 1) {
        if (r < tl || tr < l) { return 0; }
        if (l <= tl && tr <= r) { return st[t].sum; }
        pushdown(t, tl, tr);

        int tm = (tl + tr) >> 1;
        return query_sum(l, r, t << 1, tl, tm) +
            query_sum(l, r, t << 1 | 1, tm + 1, tr);
    }
};
```

5.11 Segment Tree Lazy

```
const ll maxn = 2e5+100;
const ll lzneut = 0;
const ll neut = 0;

ll a[maxn];

struct STree{ //Lazy Segment Tree with set and add
    updates with sum get query.
    vi st,lzadd,lzset; int n;
    STree(int n): st(4*n + 5,neut),lzadd(4*n + 5,0),lzset
        (4*n + 5,0), n(n){}

    void build(int node, int ini, int fin){
        if (ini == fin){
            st[node] = a[ini];
            return;
        }
        int mid = (ini+fin)/2;
        build(2*node,ini,mid);
        build(2*node + 1,mid+1,fin);
        st[node] = st[2*node] + st[2*node + 1];
    }
    void increment(int node, int ini, int fin, ll val){
        lzadd[node] += val;
        st[node] += (fin-ini+1)*val;
    }

    void assign(int node, int ini, int fin, ll val){
        lzset[node] = val;
        lzadd[node] = 0;
        st[node] = (fin-ini+1)*val;
    }

    void push(int node, int ini, int fin){
        int mid = (ini+fin)/2;
        if (lzset[node]){
            assign(2*node,ini,mid,lzset[node]);
            assign(2*node + 1,mid+1,fin,lzset[node]);
            lzset[node] = 0;
        }
        if (lzadd[node]){
            increment(2*node,ini,mid,lzadd[node]);
            increment(2*node + 1,mid+1,fin,lzadd[node]);
            lzadd[node] = 0;
        }
    }

    void setUpdate(int node, int ini, int fin, int l, int
         r, ll val){
        if (fin < l || r < ini) return;
        if (l <= ini && fin <= r){
            assign(node,ini,fin,val);
            return;
        }
        push(node,ini,fin);
        ll mid = (ini+fin)/2;
        setUpdate(2*node,ini,mid,l,r,val);
        setUpdate(2*node + 1,mid+1,fin,l,r,val);
        st[node] = st[2*node] + st[2*node + 1];
    }

    void addUpdate(int node, int ini, int fin, int l, int
         r, ll val){
        if (fin < l || r < ini) return;
        if (l <= ini && fin <= r){
            increment(node,ini,fin,val);
            return;
        }
        push(node,ini,fin);
        ll mid = (ini+fin)/2;
        addUpdate(2*node,ini,mid,l,r,val);
        addUpdate(2*node + 1,mid+1,fin,l,r,val);
        st[node] = st[2*node] + st[2*node + 1];
    }

    ll query(int node, int ini, int fin, int l, int r){
        if (fin < l || r < ini) return neut;
```

```
        if (l <= ini && fin <= r){
            return st[node];
        }
        push(node,ini,fin);
        int mid = (ini+fin)/2;
        ll lsum = query(2*node,ini,mid,l,r);
        ll rsum = query(2*node + 1,mid+1,fin,l,r);
        st[node] = st[2*node] + st[2*node + 1];
        return lsum + rsum;
    }
    void build(){ build(1,1,n);} //[1,n]
    void setUpdate(int l, int r, ll val){ setUpdate(1,1,n
       ,l,r,val);} //[l,r]
    void addUpdate(int l, int r, ll val){ addUpdate(1,1,n
       ,l,r,val);} //[l,r]
    ll query(int l, int r){ return query(1,1,n,l,r);} //[
       l,r]
};
```

5.12 Segment Tree Persistent

```
const ll maxn = 2e5+100;
ll a[maxn];
struct Vertex {
    Vertex *l, *r;
    ll sum;

    Vertex(int val) : l(nullptr), r(nullptr), sum(val) {}
    Vertex(Vertex *l, Vertex *r) : l(l), r(r), sum(0) {
        if (l) sum += l->sum;
        if (r) sum += r->sum;
    }
};
Vertex* build(int tl, int tr) {
    if (tl == tr) return new Vertex(a[tl]);
    int tm = (tl + tr) / 2;
    return new Vertex(build(tl, tm), build(tm+1, tr));
}
ll get_sum(Vertex* v, int tl, int tr, int l, int r) {
    if (l > r)
        return 0;
    if (l == tl && tr == r)
        return v->sum;
    int tm = (tl + tr) / 2;
    return get_sum(v->l, tl, tm, l, min(r, tm)) + get_sum
       (v->r, tm+1, tr, max(l, tm+1), r);
}
// creates new nodes on the path of the updated position.
Vertex* update(Vertex* v, int tl, int tr, int pos, ll
   new_val){
    if (tl == tr)
        return new Vertex(new_val);
    int tm = (tl + tr) / 2;
    if (pos <= tm)
        return new Vertex(update(v->l, tl, tm, pos,
           new_val), v->r);
    else
        return new Vertex(v->l, update(v->r, tm+1, tr,
           pos, new_val));
}
Vertex* create_copy(Vertex* v){ //creates copy of the
   whole segment tree
  return new Vertex(v->l,v->r);
}
```

5.13 Segment Tree

```
const ll maxn = 2e5+10;
ll a[maxn];
struct STree{
    vi st; int n;
    STree(int n): st(4*n + 5),n(n){}

    void build(int node,int ini,int fin){
        if (ini == fin) {
            st[node] = a[ini];
            return;
        }
        int mid = (ini+fin)/2;
        build(2*node,ini,mid); //Left sohn
        build(2*node + 1,mid+1,fin); //Right sohn
        st[node] = (st[2*node] + st[2*node + 1]); //
           desired operation
        return;
    }
    ll query(int node,int ini,int fin,int l,int r){
        if (l <= ini && fin <= r) return st[node]; //
           Fully in
        if (ini > r || fin < l) return 0; //Fully out
        int mid = (ini+fin)/2;
        return (query(2*node,ini,mid,l,r) + query(2*node
           + 1,mid+1,fin,l,r));
    }
    void upd(int node,int ini,int fin,int pos, ll val){
        if (fin < pos || pos < ini) return; //Fully out
        if (ini == fin && ini == pos) {
            st[node] = val;
            return;
        }
        int mid = (ini+fin)/2;
        upd(2*node,ini,mid,pos,val);
        upd(2*node + 1,mid+1,fin,pos,val);
        st[node] = (st[2*node] + st[2*node + 1]);
```

```
            return;
        }
        void build(){ build(1,1,n);}
        void upd(int pos, ll val){ upd(1,1,n,pos,val);}
        ll query(int l, int r){ return query(1,1,n,l,r);}
};
```

5.14 Sparse Table

```
const ll maxn = 2e5+500;
const ll INF = 1e18;
const ll loga = 22;
ll n,k,sp[maxn][loga],a[maxn];

ll query(int l, int r){
  //Check in steps of powers of 2.
  int tam = (r-l+1);
  ll res = INF;
    rofe(i,loga,0){
    if (tam & (1ll<<i)){
      res = min(res,sp[l][i]);
      l += (1ll<<i);
    }
  }
  return res;
}

void build(){ // Minimums sparse table.
    fore(i,1,n+1) sp[i][0]=a[i];
    fore(i,1,loga){
        fore(j,1,n+1){
            if (j + (1ll<<i) - 1 <= n){ //
                sp[j][i] = min(sp[j][i-1], sp[j + (1ll<<(
                    i-1))][i-1]);
            }
        }
    }
}
```

5.15 Sqrt Decomposition

```
const int maxn = 5e5+10;
const int block_amount = 710; // block_amount squared
    should be > maxn.
vi b[710]; //blocks of the SQRT.
int a[maxn];
int n,bsize;

void build_blocks(){
    bsize = sqrt(n)+1;
    fore(i,0,n) b[i/bsize].pb(a[i]);

    fore(i,0,bsize+1) sort(ALL(b[i])); //sort the blocks.
}

ll query(int l, int r, ll x){ // in [l,r] get amount of
    values >= x.
    int ans=0;
    int cl = l/bsize;
    int cr = r/bsize;
    if (cl == cr){
        fore(i,l,r+1){
            if (x <= a[i]) ans++;
        }
    }
    else{
        fore(i,l,(cl+1)*bsize){ //get prefix:
            if (x <= a[i]) ans++;
        }
        fore(i,cl+1,cr){ //mid part:
            ans += (sz(b[i]) - (lower_bound(ALL(b[i]),x)
                - b[i].begin()));
        }
        fore(i,cr*bsize,r+1){ //get suffix:
            if (x <= a[i]) ans++;
        }
    }
    return ans;
}

void update(int pos, ll x){ // point update in O(bsize)
    int cb = pos/bsize;
    int idx = lower_bound(ALL(b[cb]),a[pos]) - b[cb].
        begin();
    b[cb][idx] = a[pos] = x;
    sort(ALL(b[cb]));
}
```

5.16 Treap Implicit

```
typedef struct Node *pnode;

const ll maxn = 1e6+10;

struct Node {
  Node(ll val) : val(val), weight(rand()), size(1),
     lazy_tag(0) {}
    ll val, sum;  // val -> a[i], sum = sum of all a[i]
      in subtree
  ll weight, size;
  bool rev = false;  // whether this range is reversed
  pnode l = nullptr;
    pnode r = nullptr;
  ll lazy_tag; // neutral value is 0.
};

int size(pnode node) { return node ? node->size : 0; }
```

```
ll sum(pnode node) { return node ? node->sum : 0; }

void push(pnode node) {
  if (!node) { return; }
  if (node->rev){  // need to reverse this range
    node->rev = false;
    swap(node->l, node->r);
    if (node->l) { node->l->rev ^= true; }
    if (node->r) { node->r->rev ^= true; }
  }
  if (node->lazy_tag){ // need to update the sum of this
     range.
    node->sum += node->lazy_tag * size(node);
        node->val += node->lazy_tag;
    if (node->l) { node->l->lazy_tag += node->lazy_tag;}
    if (node->r) { node->r->lazy_tag += node->lazy_tag;}
    node->lazy_tag = 0;
  }
}

void pull(pnode node) {
  if (!node) { return; }
  push(node->l), push(node->r);
  assert(!node->lazy_tag);
  node->size = size(node->l) + size(node->r) + 1;
  node->sum = sum(node->l) + sum(node->r) + node->val;
}

// merges treaps l and r into treap
void merge(pnode &node, pnode l, pnode r) {
  push(l), push(r);
  if (!l || !r) {
    node = l ? l : r;
  } else if (l->weight > r->weight) {
    merge(l->r, l->r, r), node = l;
  } else {
    merge(r->l, l, r->l), node = r;
  }
  pull(node);
}

// splits treap into l, r; l: [0, val), r: [val, )
void split(pnode node, pnode &l, pnode &r, int val) {
  if (!node) return void(l = r = nullptr);
  push(node);
  if (val > size(node->l)) {
    split(node->r, node->r, r, val - size(node->l) - 1),
       l = node;
  } else {
    split(node->l, l, node->l, val), r = node;
  }
  pull(node);
}

struct Treap {
  Node *root = nullptr;  // root of this treap

void insert(int i, int x) {
  Node *l, *r;
  split(root, l, r, i);
  auto v = new Node(x);
  merge(l, l, v);
  merge(root, l, r);
}

void del(int i) {
  Node *l, *r;
  split(root, l, r, i);
  split(r, root, r, 1);
  merge(root, l, r);
}

  void swap_intervals(int l1, int r1, int l2, int r2) {
      if (l1 > l2) {
    swap(l1, l2);
    swap(r1, r2);
  }
      assert(r1 <= l2);

      pnode a, b, c, d, e;
      split(root, a, b, l1);
      split(b, b, c, r1 - l1);
      split(c, c, d, l2 - r1);
      split(d, d, e, r2 - l2);

      merge(root, a, d);
      merge(root, root, c);
      merge(root, root, b);
      merge(root, root, e);
  }

// updates the range [l, r)
void upd(int l, int r, function<void(Node *)> f) {
  Node *a, *b, *c;
  split(root, a, b, l);
  split(b, b, c, r - l);
  if (b) { f(b); }
  // merge all the splits back into the main treap
  merge(root, a, b);
  merge(root, root, c);
}
// queries the range [l,r)
template <typename R> R query(int l, int r, function<R(
   Node *)> f) {
  Node *a, *b, *c;
  split(root, a, b, l);
  split(b, b, c, r - l);
  assert(b);
  R x = f(b);
  merge(root, a, b);
  merge(root, root, c);
  return x;
}
```

```
  void print_treap(pnode node) {
    if (!node) return;
    push(node);
    print_treap(node->l);
    cout<<node->val<<" ";
    print_treap(node->r);
  }

  void print_all() {
    print_treap(root);
    cout << nl;
  }
};

void doit() {
    int pos, val, l, r, x;
    Treap treap;
    // insert:
    treap.insert(pos, val);
    // delete:
    treap.del(pos);
    // update [l, r) reverse:
    treap.upd(l, r, [](Node *node) { node->rev ^= true;
        });
    // update [l, r) adding a value x:
    treap.upd(l, r, [x](Node *node) { node->lazy_tag += x
        ; });
    // query for the sum in [l, r)
    ll range_sum = treap.query<ll>(l, r, [](Node *node) {
        return node->sum; });
}

int main(){ srand(time(0)); }
```

5.17 Treap

```
typedef struct item *pitem;

struct item {
  int pr,key,cnt;
  pitem l,r;
  item(int key):key(key),pr(rand()),cnt(1),l(0),r(0) {}
    item(int key, int pr): key(key), pr(pr), cnt(1), l(0)
        , r(0) {}
};
int cnt(pitem t){return t?t->cnt:0;}
void upd_cnt(pitem t){if(t)t->cnt=cnt(t->l)+cnt(t->r)+1;}
void split(pitem t, int key, pitem& l, pitem& r){ // l: <
    key, r: >= key
  if(!t)l=r=0;
  else if(key<t->key)split(t->l,key,l,t->l),r=t;
  else split(t->r,key,t->r,r),l=t;
  upd_cnt(t);
}
void insert(pitem& t, pitem it){
  if(!t)t=it;
  else if(it->pr > t->pr)split(t,it->key,it->l,it->r),t=
      it;
  else insert(it->key<t->key?t->l:t->r,it);
  upd_cnt(t);
}
void merge(pitem& t, pitem l, pitem r){
  if(!l||!r)t=l?l:r;
  else if(l->pr>r->pr)merge(l->r,l->r,r),t=l;
  else merge(r->l,l,r->l),t=r;
  upd_cnt(t);
}
void erase(pitem& t, int key){
  if(t->key==key)merge(t,t->l,t->r);
  else erase(key<t->key?t->l:t->r,key);
  upd_cnt(t);
}
void unite(pitem &t, pitem l, pitem r){
  if(!l||!r){t=l?l:r;return;}
  if(l->pr<r->pr)swap(l,r);
  pitem p1,p2;split(r,l->key,p1,p2);
  unite(l->l,l->l,p1);unite(l->r,l->r,p2);
  t=l;upd_cnt(t);
}

//Explore the treap going left or right according to the
    target value.
ll explore(pitem t, ll key){
    if (!t) return 0;
    ll res = 0;
    if (t->key < key){
        res += cnt(t->l) + 1;
        res += explore(t->r,key);
    }
    else{ //t->key >= key
        res += explore(t->l,key);
    }
    return res;
}

void kthSmallest(pitem t, int sz, int &kth){
    if (!t) return;
    if (cnt(t->l) + 1 == sz){
        kth = t->key;
        return;
    }
    else if(cnt(t->l) + 1 < sz){
        kthSmallest(t->r,sz - cnt(t->l) - 1,kth);
    }
    else kthSmallest(t->l,sz,kth);
}

void kthLargest(pitem t, int sz, ll &kth){
    if (!t) return;
    if (cnt(t->r) + 1 == sz){
        kth = t->key;
```

```
            return;
        }
        else if (cnt(t->r) + 1 < sz){
            kthLargest(t->l,sz - cnt(t->r) - 1, kth);
        }
        else kthLargest(t->r,sz,kth);
}

void solveCrops(){ //Spoj prefix crops problem.
    map <ll,pitem> mp;
    ll n,q;
    cin>>n>>q;
    vector <ll> a(n);
    //Having individual treaps for each number.
    //The values are the positions in the array.
    fore(i,0,n){
        cin>>a[i];
        insert(mp[a[i]],new item(i));
    }
    while(q--){
        int pos,nw;
        cin>>pos>>nw;
        erase(mp[a[pos]],pos);
        a[pos] = nw;
        insert(mp[a[pos]], new item(pos));
        //check amount of items equal to a[pos] in [0,pos
            )
        cout<<explore(mp[a[pos]],pos)<<nl;
    }
}

void solveDogs(){ //Codeforces Dogs Show problem.
    map <ll,pitem> mp;
    ll n,k,pos=0,best=0;
    cin>>n>>k;
    vector <ll> uni,a(n+1);
    fore(i,1,n+1){
        cin>>a[i];
        ll dif = max(0LL,a[i]-i);
        uni.pb(dif);
    }
    sort(ALL(uni));
    uni.erase(unique(ALL(uni)),uni.end());

    fore(i,1,n+1){
        insert(mp[max(0LL,a[i]-i)],new item(i));
    }
    fore(i,0,uni.size()){
        ll delay = uni[i];
        best = max(best,explore(mp[delay],k-delay));
        //join 2 treaps: {root,left,right};
        if (i+1<uni.size()) unite(mp[uni[i+1]],mp[uni[i
            +1]],mp[delay]);
    }
    cout<<best<<nl;
}
```



```
int main(){ srand(time(0)); }
```

6 Math

6.1 Binary Exponentiation

```
const ll mod = 1e9+7;

ll expo(ll x, ll pw){
    ll res = 1;
    while(pw > 0){
        if(pw&1)
            res = (res*x)%mod;
        x = (x*x)%mod;
        pw >>= 1;
    }
    return res;
} // For mul-inverses: pw = mod-2
```

6.2 Binomial Coefficient

```
const ll maxn = 1e6 + 5;
const ll mod = 1e9+7;
ll f[maxn];

ll expo(ll x, ll pw){
    ll res = 1;
    while(pw > 0){
        if(pw&1)
            res = (res*x)%mod;
        x = (x*x)%mod;
        pw >>= 1;
    }
    return res;
}

ll nCk(ll a, ll b){ //precalculate factorials % mod.
    b = (f[b]*f[a-b])%mod;
    ll inverse = expo(b,mod-2);
    ll res = (f[a]*inverse)%mod;
    return res;
}
```

6.3 Bitwise AND Convolution

```
const ll mod = 998244353;

void supersetZetaTransform(vi &v){
    int n = sz(v); // n must be a power of 2.
    for(int j = 1; j<n; j <<= 1){
```

```
        fore(i,0,n){
            if (i&j) v[i^j] += v[i], v[i^j] %= mod;
        }
    }
}

void supersetMobiusTransform(vi &v){
    int n = sz(v); // n must be a power of 2.
    for(int j = 1; j<n; j <<= 1){
        fore(i,0,n){
            if (i&j) v[i^j] -= v[i], v[i^j] += mod, v[i^j
                ] %= mod;
        }
    }
}
// c_k = Total sum where (i,j), i&j = k of a_i*b_j
vi andConvolution(vi a, vi b){
    supersetZetaTransform(a);
    supersetZetaTransform(b);
    fore(i,0,sz(a)) a[i] *= b[i], a[i] %= mod;
    supersetMobiusTransform(a);
    return a;
}

void doit(){
    n = 1<<n;
    // Then read values of a and b arrays.
    // get the answer vector.
    // apply modulo when printing answers.
}
```

6.4 Bitwise OR Convolution

```
const ll mod = 998244353;

void subsetZetaTransform(vi &v){
    int n = sz(v); // n must be a power of 2.
    for(int j = 1; j<n; j <<= 1){
        fore(i,0,n){
            if (i&j) v[i] += v[i^j], v[i] %= mod;
        }
    }
}

void subsetMobiusTransform(vi &v){
    int n = sz(v); // n must be a power of 2.
    for(int j = 1; j<n; j <<= 1){
        fore(i,0,n){
            if (i&j) v[i] -= v[i^j], v[i] += mod, v[i] %=
                 mod;
        }
    }
}

// c_k = Total sum where (i,j), i|j = k of a_i*b_j
vi orConvolution(vi a, vi b){
    subsetZetaTransform(a);
    subsetZetaTransform(b);
    fore(i,0,sz(a)) a[i] *= b[i], a[i] %= mod;
    subsetMobiusTransform(a);
    return a;
}

void doit(){
    // read a and b arrays.
    // get the answer vector
    // print answers % mod.
}
```

6.5 Bitwise XOR Convolution

```
const int mod = 998244353;

int inverse(int x, int mod) {
  return x == 1 ? 1 : mod - mod / x * inverse(mod % x,
     mod) % mod;
}

void xormul(vi a, vi b, vi &c){
  int m = sz(a);
  c.resize(m);
  for (int n = m / 2; n > 0; n /= 2)
    for (int i = 0; i < m; i += 2 * n)
      for (int j = 0; j < n; j++) {
        int x = a[i + j], y = a[i + j + n];
        a[i + j] = (x + y) % mod;
        a[i + j + n] = (x - y + mod) % mod;
      }
  for (int n = m / 2; n > 0; n /= 2)
    for (int i = 0; i < m; i += 2 * n)
      for (int j = 0; j < n; j++) {
        int x = b[i + j], y = b[i + j + n];
        b[i + j] = (x + y) % mod;
        b[i + j + n] = (x - y + mod) % mod;
      }
  fore(i,0,m) c[i] = a[i] * b[i] % mod;
  for (int n = 1; n < m; n *= 2)
    for (int i = 0; i < m; i += 2 * n)
      for (int j = 0; j < n; j++) {
        int x = c[i + j], y = c[i + j + n];
        c[i + j] = (x + y) % mod;
        c[i + j + n] = (x - y + mod) % mod;
      }
  int mrev = inverse(m, mod);
    fore(i,0,m) c[i] = c[i] * mrev % mod;
}

// Given two arrays of size 2^N, find:
// c_k = Total_Sum of (i,j) where (i [XOR] j) == k, a_i*
   b_j
```

```
void doit(){
    int n;
    vi a(1<<n),b(1<<n),c; //c is answer vector of size
        (1<<n).
}
```

6.6 Catalan Numbers

```
const ll mod = 1e9+7;
const ll maxn = 1e5+5;
int catalan[maxn+5];
void init() {
    catalan[0] = catalan[1] = 1;
    for (int i=2; i<=maxn; i++) {
        catalan[i] = 0;
        for (int j=0; j < i; j++) {
            catalan[i] += (catalan[j] * catalan[i-j-1]) %
                mod;
            if (catalan[i] >= mod) {
                catalan[i] -= mod;
            }
        }
    }
}
/*
Formula to get the nth catalan: C_n = (1/(n+1))*nck(2*n,n
    )
Applications of Catalan Numbers, where Cn is:
    > Number of correct bracket sequence consisting of 'n
       ' opening and 'n' closing brackets.
    > The number of rooted full binary trees with 'n + 1'
        leaves (vertices are not numbered).
      A rooted binary tree is full if every vertex has
         either two children or no children
    > The number of ways to completely parenthesize 'n +
       1' factors.
    > The number of triangulations of a convex polygon
       with 'n + 2' sides (i.e. the number of
      partitions of polygon into disjoint triangles by
         using the diagonals).
    > The number of ways to connect the '2n' points on a
       circle to form  'n' disjoint chords.
    > The number of non-isomorphic full binary trees with
        'n' internal nodes (i.e. nodes
      having at least one son).
    > The number of monotonic lattice paths from point
       (0, 0) to point (n, n) in a square lattice
      of size 'n' x 'n', which do not pass above the main
          diagonal (i.e. connecting '(0, 0)' to
     '(n, n)')
    > Number of permutations of length 'n' that can be
       stack sorted (i.e. it can be shown that the
      rearrangement is stack sorted if and only if there
         is no such index i < j < k, such that
      a_k < a_i < a_j).
    > The number of non-crossing partitions of a set of '
       n' elements.
    > The number of ways to cover the ladder 1...n using
       'n' rectangles (The ladder consists of
      'n' columns, where i-th column has a height i).
*/
```

6.7 Euler Totient

```
vector <int> phi;

//Amount of coprime numbers (gcd(a,b) == 1) for each
    number in (1 <= i <= n).
//counting the number of integers between 1 and i, which
    are coprime to i.
void euler_totient(int n) {
    phi.resize(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;
        }
    }
}

//Amount of numbers 0 <= i < m such that gcd(a+i,m) ==
    gcd(a,m)
int phiFunc(int a, int m) {
    ll y = m/__gcd(a,m);
    ll ans = y;
    for(ll i = 2; i*i<=m; i++){
        if (y%i == 0){
            ans -= ans/i;
            while(y%i == 0) y /= i;
        }
    }
    if (y>1) ans -= ans/y;
    return ans;
}
```

6.8 Extended Euclidean

```
//a * x + b * y = gcd(a,b), where a and b are given.
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;
}
```

6.9 FFT

```
typedef pair<ll,ll> ii;

const ll MAXN=1<<20; //watch out with RTEs (increase MAXN
    ):
typedef vector<ll> poly;
struct CD {
  double r,i;
  CD(double r=0, double i=0):r(r),i(i){}
  double real()const{return r;}
  void operator/=(const int c){r/=c, i/=c;}
};
CD operator*(const CD& a, const CD& b){
  return CD(a.r*b.r-a.i*b.i,a.r*b.i+a.i*b.r);}
CD operator+(const CD& a, const CD& b){return CD(a.r+b.r,
    a.i+b.i);}
CD operator-(const CD& a, const CD& b){return CD(a.r-b.r,
    a.i-b.i);}
const double pi=acos(-1.0);

CD cp1[MAXN+9],cp2[MAXN+9];
int R[MAXN+9];
void dft(CD* a, int n, bool inv){
  fore(i,0,n)if(R[i]<i)swap(a[R[i]],a[i]);
  for(int m=2;m<=n;m*=2){
    double z=2*pi/m*(inv?-1:1);
    CD wi=CD(cos(z),sin(z));
    for(int j=0;j<n;j+=m){
      CD w(1);
      for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
        CD u=a[k];CD v=a[k2]*w;a[k]=u+v;a[k2]=u-v;w=w*wi;
      }
    }
  }
  if(inv)fore(i,0,n)a[i]/=n;
}
poly multiply(poly& p1, poly& p2){
  int n=p1.size()+p2.size()+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  fore(i,0,m){R[i]=0;fore(j,0,cnt)R[i]=(R[i]<<1)|((i>>j)
      &1);}
  fore(i,0,m)cp1[i]=0,cp2[i]=0;
  fore(i,0,p1.size())cp1[i]=p1[i];
  fore(i,0,p2.size())cp2[i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  fore(i,0,m)cp1[i]=cp1[i]*cp2[i];
  dft(cp1,m,true);
  poly res;
  n-=2;
  fore(i,0,n)res.pb((ll)floor(cp1[i].real()+0.5));
  return res;
}

void getBigNumMulti(vector <ll> &c){ //Big numbers
    multiplication.
    vector <char> r;
    while(!c.empty()&&!c.back()) c.pop_back(); //quitar
        todos los 0 extras.
  if(c.empty()){
        cout<<0<<nl;
        return;
    }
  ll x=0;
    //Normalizar los coeficientes para representarlos
        como digitos.
  fore(i,0,c.size()){
    x+=c[i];
        r.pb((char)(x%10) + '0');
    x/=10;
  }
  while(x){ //carry que sobra.
    r.pb((char)(x%10) + '0');
    x/=10;
  }
  reverse(ALL(r));
    fore(i,0,r.size()) cout<<r[i];
    cout<<nl;
}

void stringMatchShift(){ //All possible scalar products
    with strings.
  string s;
  cin>>s;
  int n = s.size();
  vector <ll> a1(n,0),a2(2*n,0),b1(n,0),b2(2*n,0),c1(n,0)
      ,c2(2*n,0);
  vector <ll> ra,rb,rc;

  //Create binary polynomial for each letter.
  fore(i,0,n){
    if (s[i] == 'a') a1[i] = 1;
    else if (s[i] == 'b') b1[i] = 1;
    else c1[i] = 1;
  }
  //Make the dup for each letter to multiply:
  fore(i,0,n){
    a2[i] = a2[i+n] = a1[i];
    b2[i] = b2[i+n] = b1[i];
    c2[i] = c2[i+n] = c1[i];
  }
  //Append the rest of the zeros (Step 1):
```

```cpp
  fore(i,0,n){
    a1.pb(0), b1.pb(0), c1.pb(0);
  }
  //Reverse the arrays (Step 2):
  reverse(ALL(a1));
  reverse(ALL(b1));
  reverse(ALL(c1));
  //Multiply the polynomials:
  ra = multiply(a1,a2);
  rb = multiply(b1,b2);
  rc = multiply(c1,c2);
  int shif = 1;
  //Left shift match (Step 1, then Step 2):
  for(int i = (2*n)-2; i>=n; i--){
    cout<<"L-shift: "<<shif<<" "<<ra[i] + rb[i] + rc[i]<<nl;
    shif++;
  }
  //Right shift match (Step 2, then Step 1):
  shif = 1;
  for(int i = n-2; i>=0; i--){
    cout<<"R-shift: "<<shif<<" "<<ra[i] + rb[i] + rc[i]<<nl;
    shif++;
  }
}

//String matching with wildcards ('?')
vector<ll> string_matching(string &s, string &t) {
    int n = s.size(), m = t.size();
    vector<ll> s1(n), s2(n), s3(n);
    //assign any non zero number for non '?'s
    for(int i = 0; i < n; i++) s1[i] = s[i] == '?' ? 0 : s[i] - 'a' + 1;
    for(int i = 0; i < n; i++) s2[i] = s1[i] * s1[i];
    for(int i = 0; i < n; i++) s3[i] = s1[i] * s2[i];
    vector<ll> t1(m), t2(m), t3(m);
    for(int i = 0; i < m; i++) t1[i] = t[i] == '?' ? 0 : t[i] - 'a' + 1;
    for(int i = 0; i < m; i++) t2[i] = t1[i] * t1[i];
    for(int i = 0; i < m; i++) t3[i] = t1[i] * t2[i];
    reverse(ALL(t1));
    reverse(ALL(t2));
    reverse(ALL(t3));
    vector<ll> s1t3 = multiply(s1, t3);
    vector<ll> s2t2 = multiply(s2, t2);
    vector<ll> s3t1 = multiply(s3, t1);
    vector<ll> res(n);
    for(int i = 0; i < n; i++) res[i] = s1t3[i] - s2t2[i] * 2 + s3t1[i];
    vector<ll> oc;
    for(int i = m - 1; i < n; i++) if(res[i] == 0) oc.pb(i - m + 1);
    return oc;
}
```

6.10 Fractions

```cpp
struct frac{
    ll num, den;
    frac(){}
    frac(ll num, ll den):num(num), den(den){
        if(!num) den = 1;
        if(num > 0 && den < 0) num = -num,  den = -den;
        simplify();
    }
    void simplify(){
        ll g = __gcd(abs(num), abs(den));
        if(g) num /= g,  den /= g;
    }
    frac operator+(const frac& b){ return {num*b.den + b.num*den, den*b.den};}
    frac operator-(const frac& b){ return {num*b.den - b.num*den, den*b.den};}
    frac operator*(const frac& b){ return {num*b.num, den*b.den};}
    frac operator/(const frac& b){ return {num*b.den, den*b.num};}
    bool operator<(const frac& b)const{ return num*b.den < den*b.num; }
};
```

6.11 GCD Convolution

```cpp
const ll mod = 998244353;

vi primeEnumerate(int n){
    vi p;
    vector <bool> b(n+1,1);
    fore(i,2,n+1){
        if (b[i]) p.pb(i);
        for(int j : p){
            if (i*j>n) break;
            b[i*j]=0;
            if (i%j == 0) break;
        }
    }
    return p;
}

void multipleZetaTransform(vi &v){
    const int n = sz(v)-1;
    for(int p : primeEnumerate(n)){
        for (int i = n/p; i; i--){
            v[i] = (v[i]+v[i*p])%mod;
        }
    }
}
```

```
void multipleMobiusTransform(vi &v){
    const int n = sz(v)-1;
    for(int p : primeEnumerate(n)){
        for(int i = 1; i*p <= n; i++){
            v[i] = (v[i]-v[i*p]+mod)%mod;
        }
    }
}

// c_k = TotalSum where gcd(i,j)=k of a_i*b_j modulo mod.
vi gcdConvolution(vi a, vi b){
    multipleZetaTransform(a);
    multipleZetaTransform(b);
    fore(i,0,sz(a)) a[i] = (a[i]*b[i])%mod;
    multipleMobiusTransform(a);
    return a;
}

void doit(){
    //insert elements between [1,n].
    // answers [1,n].
}
```

6.12 LCM Convolution

```
const ll mod = 998244353;

vi primeEnumerate(int n){ //Linear sieve.
    vi p;
    vector <bool> b(n+1,1);
    fore(i,2,n+1){
        if (b[i]) p.pb(i);
        for(int j : p){
            if (i*j>n) break;
            b[i*j]=0;
            if (i%j == 0) break;
        }
    }
    return p;
}

void divisorZetaTransform(vi &v){
    const int n = sz(v)-1;
    for(int p : primeEnumerate(n)){
        for (int i = 1; i*p <= n; i++){
            v[i*p] = (v[i*p]+v[i])%mod;
        }
    }
}

void divisorMobiusTransform(vi &v){
    const int n = sz(v)-1;
    for(int p : primeEnumerate(n)){
        for(int i = n/p; i; i--){
            v[i*p] = (v[i*p]-v[i]+mod)%mod;
        }
    }
}

// c_k = TotalSum where lcm(i,j)=k of a_i*b_j modulo mod.
vi lcmConvolution(vi a, vi b){
    divisorZetaTransform(a);
    divisorZetaTransform(b);
    fore(i,0,sz(a)) a[i] = (a[i]*b[i])%mod;
    divisorMobiusTransform(a);
    return a;
}

void doit(){
    //insert elements between [1,n].
    // answers [1,n].
}
```

6.13 Matrix Exponentiation Kth Term

```
const ll mod = 1e9+7;
ll tc,n,m,k;

vvi mul(vvi a, vvi b) {
    vvi c(sz(a), vi(sz(b[0])));
    for (int i = 0; i < sz(a); i++)
        for (int j = 0; j < sz(b); j++)
            for ( int k = 0; k < sz(a); k++)
                (c[i][j] += a[i][k]*b[k][j]%mod)%=mod;
    return c;
}

vvi exp( vvi x, ll y) {
    vvi r(sz(x), vi(sz(x)));
    bool flag = false;
    while (y>0){
        if (y&1) {
            if (!flag) r = x, flag = true;
            else r = mul(r,x);
        }
        y=y>>1;
        x = mul(x,x);
    }
    return r;
}

void doit(){
    // define base cases and solve directly.
    // example f(0)=1, f(1)=2, f(2)=3.
    // Function: F(n) = 3*F(n-1) + 2*F(n-2) + F(n-3) + 3.
    mat[0] = {3,2,1,3};
    mat[1] = {1,0,0,0};
    mat[2] = {0,1,0,0};
    mat[3] = {0,0,0,1}; //to keep the +3 constant.
    vi iniv = {3,2,1,1}; //Initial vector.
```

```
    mat = exp(mat,k-2); //substract (dims-2) to k.
    ll ans = 0;
    fore(i,0,4){
        ll aux = (mat[0][i]*iniv[i])%mod;
        ans = (ans + aux)%mod;
    }
}
```

6.14 Matrix Exponentiation

```
const ll mod = 1e9+7;
ll tc,n,m,k;

vvi mul(vvi a, vvi b) {
    vvi c(sz(a), vi(sz(b[0])));
    for (int i = 0; i < sz(a); i++)
        for (int j = 0; j < sz(b); j++)
            for ( int k = 0; k < sz(a); k++)
                (c[i][j] += a[i][k]*b[k][j]%mod)%=mod; //
                    for amount of paths.
                //c[i][j] = min(c[i][j], a[i][k] + b[k][j
                    ]); //for shortest path.
    return c;
}

vvi exp( vvi x, int y) { // matrix and desired power.
    vvi r(sz(x), vi(sz(x),0ll)); //0ll: amount of paths.
        INF: shortest path
    for ( int i = 0; i < sz(x); i++) r[i][i] = 1; //1ll:
        amount of paths. 0ll: shortest path.
    while (y>0){
        if (y&1) {
            r = mul(r,x);
        }
        y=y>>1;
        x = mul(x,x);
    }
    return r;
}

void doit(){
    // build adjacency (or costs) matrix of size(n*n).
    // after exponentiating mat[i][j] denotes the path
        from i to j.
}
```

6.15 Miller Rabin

```
ll mulmod(ll a, ll b, ll m){
   ll x = 0,y = a % m;
   while (b > 0) {
      if (b % 2 == 1) {
         x = (x + y) % m;
      }
      y = (y * 2) % m;
      b /= 2;
   }
   return x % m;
}

ll modulo(ll base, ll e, ll m) {
   ll x = 1;
   ll y = base;
   while (e > 0) {
      if (e % 2 == 1)
         x = (x * y) % m;
         y = (y * y) % m;
         e = e / 2;
   }
   return x % m;
}

bool Miller(ll p, int iteration) { //number and amount of
     iterations.
   if (p < 2) {
      return false;
   }
   if (p != 2 && p % 2==0) {
      return false;
   }
   ll s = p - 1;
   while (s % 2 == 0) {
      s /= 2;
   }
   for (int i = 0; i < iteration; i++) {
      ll a = rand() % (p - 1) + 1, temp = s;
      ll mod = modulo(a, temp, p);
      while (temp != p - 1 && mod != 1 && mod != p - 1) {
         mod = mulmod(mod, mod, p);
         temp *= 2;
      }
      if (mod != p - 1 && temp % 2 == 0) {
         return false;
      }
   }
   return true;
}
```

6.16 Mobius Function

```
const ll maxn = 1e7+1;
ll mobius[maxn],sum[maxn];
/* Mobius function: mu(n)
    mu(n) = 1, if n = 1.
    mu(n) = 0, if n has a squared prime factor.
    mu(n) = (-1)^k, if n is a product of k distinct prime
          factors.
*/
```

```
void computeMobius(){
    mobius[1] = -1;
    for (int i = 1; i < maxn; i++) {
        if (mobius[i]) {
            mobius[i] = -mobius[i];
            for (int j = 2 * i; j < maxn; j += i) {
                mobius[j] += mobius[i]; }
        }
    }
}
```

6.17 Modular Int

```
struct mint {
  const static int M = 998244353;
  ll v = 0;
  mint() {}
  mint(ll v) { this->v = (v % M + M) % M; }
  mint operator+(const mint &o) const { return v + o.v; }
  mint &operator+=(const mint &o) {
    v = (v + o.v) % M;
    return *this;
  }
  mint operator*(const mint &o) const { return v * o.v; }
  mint operator-(const mint &o) const { return v - o.v; }
  mint &operator-=(const mint &o) {
    mint t = v - o.v;
    v = t.v;
    return *this;
  }
  mint operator^(int y) const {
    mint r = 1, x = v;
    for (y <<= 1; y >>= 1; x = x * x)
      if (y & 1) r = r * x;
    return r;
  }
  mint inv() const {
    assert(v);
    return *this ^ M - 2;
  }
  friend istream &operator>>(istream &s, mint &v) {
    return s >> v.v;
    return s;
  }
  friend ostream &operator<<(ostream &s, const mint &v) {
      return s << v.v; }
  mint operator/(mint o) { return *this * o.inv(); }
};
```

6.18 NTT

```
// MAXN must be power of 2 !!
// MOD-1 needs to be a multiple of MAXN !!
// big mod and primitive root for NTT:
const int MOD=998244353,RT=3,MAXN=1<<21;
const int loga = 17;
typedef vector<ll> poly;

int mulmod(ll a, ll b){return a*b%MOD;}
int addmod(int a, int b){int r=a+b;if(r>=MOD)r-=MOD;
    return r;}
int submod(int a, int b){int r=a-b;if(r<0)r+=MOD;return r
    ;}
int pm(ll a, ll e){
  int r=1;
  while(e){
    if(e&1)r=mulmod(r,a);
    e>>=1;a=mulmod(a,a);
  }
  return r;
}
int inv(int a){return pm(a,MOD-2);}

struct CD {
  int x;
  CD(int x):x(x){}
  CD(){}
  int get()const{return x;}
};
CD operator*(const CD& a, const CD& b){return CD(mulmod(a
    .x,b.x));}
CD operator+(const CD& a, const CD& b){return CD(addmod(a
    .x,b.x));}
CD operator-(const CD& a, const CD& b){return CD(submod(a
    .x,b.x));}
vector<int> rts(MAXN+9,-1);
CD root(int n, bool inv){
  int r=rts[n]<0?rts[n]=pm(RT,(MOD-1)/n):rts[n];
  return CD(inv?pm(r,MOD-2):r);
}
CD cp1[MAXN+9],cp2[MAXN+9];
int R[MAXN+9];
void dft(CD* a, int n, bool inv){
  fore(i,0,n)if(R[i]<i)swap(a[R[i]],a[i]);
  for(int m=2;m<=n;m*=2){
    CD wi=root(m,inv);
    for(int j=0;j<n;j+=m){
      CD w(1);
      for(int k=j,k2=j+m/2;k2<j+m;k++,k2++){
        CD u=a[k];CD v=a[k2]*w;a[k]=u+v;a[k2]=u-v;w=w*wi;
      }
    }
  }
  if(inv){
    CD z(pm(n,MOD-2));
    fore(i,0,n)a[i]=a[i]*z;
  }
}
```

```
poly multiply(poly& p1, poly& p2){
  int n=p1.size()+p2.size()+1;
  int m=1,cnt=0;
  while(m<=n)m+=m,cnt++;
  fore(i,0,m){R[i]=0;fore(j,0,cnt)R[i]=(R[i]<<1)|((i>>j)
     &1);}
  fore(i,0,m)cp1[i]=0,cp2[i]=0;
  fore(i,0,p1.size())cp1[i]=p1[i];
  fore(i,0,p2.size())cp2[i]=p2[i];
  dft(cp1,m,false);dft(cp2,m,false);
  fore(i,0,m)cp1[i]=cp1[i]*cp2[i];
  dft(cp1,m,true);
  poly res;
  n-=2;
  fore(i,0,n)res.pb(cp1[i].x);
  return res;
}
```

6.19 Pascal Triangle

```
const ll maxn = 1005;
const ll mod = 1e9+7;
ll c[maxn][maxn];

void pascal(){
    c[0][0] = 1;
    fore(i,1,maxn){
        c[i][0]=c[i][i]=1;
        fore(j,1,i) c[i][j]=(c[i-1][j-1]+c[i-1][j])%mod;
    }
}
```

6.20 Sieve Linear

```
const ll maxn = 1e6+5;
ll lp[maxn];
vi primes;

void sieve_linear(){
    fore(i,2,maxn){
        if (!lp[i]){
            lp[i]=i;
            primes.pb(i);
        }
        for (int j=0; j < sz(primes) && pr[j]<=lp[i] && i
           *pr[j]<maxn; j++) {
            lp[i * pr[j]] = pr[j];
        }
    }
}
```

6.21 Sieve

```
const ll maxn = 1e6+5;
bool c[maxn];

void sieve(){
    c[1]=true;
    fore(i,1,maxn){
        if (!c[i]){
            for (int j = 2; i*j<maxn; j++){
                criba[i*j] = true;
            }
        }
    }
}
```

6.22 Sieve Segmented

```
// Complexity O((R-L+1)*log(log(R)) + sqrt(R)*log(log(R))
   )
// R-L+1 roughly 1e7  R-- 1e12
vector<bool> segmentedSieve(ll L, ll R) {
  // generate all primes up to sqrt(R)
  ll lim = sqrt(R);
  vector<bool> mark(lim + 1, false);
  vi primes;
  fore(i,2,lim+1) {
    if (!mark[i]) {
      primes.emplace_back(i);
      for (ll j = i * i; j <= lim; j += i)
        mark[j] = true;
    }
  }
  vector<bool> isPrime(R - L + 1, true);
  for (ll i : primes)
    for (ll j = max(i * i, (L + i - 1) / i * i); j <= R;
        j += i)
      isPrime[j - L] = false;
  if (L == 1)
    isPrime[0] = false;
  return isPrime;
}
```

7 Dynamic Programming

7.1 Convex Hull Trick

```
const ll maxn = 1e5+5;

struct CHT { //For Optimizing DPs that can be modeled as
   y = mx + b.
```

```
//This code is made to find the minimums. Maximums
    can also be found.
//Use When slopes are in decreasing order for
    minimums: m1 > m2 > ... > mk
//Use when slopes are in increasing order for maximums:
    m1 < m2 < ... < mk
struct Line {
    ll slope, yIntercept;

    Line(ll slope, ll yIntercept) : slope(slope),
        yIntercept(yIntercept) {}

    ll val(ll x) {
        return slope * x + yIntercept;
    }

    ll intersect(Line y) {
        return (y.yIntercept - yIntercept + slope - y
            .slope - 1) / (slope - y.slope);
    }
};

deque<pair<Line, ll>> dq;

void insert(ll slope, ll yIntercept) {
    Line newLine(slope, yIntercept);

    //Pop lines until all lines become useful.
        Popping the lines that become irrelevant.
// For minimums >=
// For maximums <=
    while (!dq.empty() && dq.back().second >= dq.back
        ().first.intersect(newLine)) dq.pop_back();


    if (dq.empty()) {
        dq.emplace_back(newLine, 0);
        return;
    }

    dq.emplace_back(newLine, dq.back().first.
        intersect(newLine));
}

ll query(ll x) { //When x values are given in
    ascending order: x1 < x2 < .. xk.
    //Just need to use a deque, no need to use Binary
        Search.
    while (sz(dq) > 1) {
        if (dq[1].second <= x) dq.pop_front();
        else break;
    }

    return dq[0].first.val(x);
}

ll query2(ll x) { //Use Binary Search when x values
    are given without a specific order.
    auto qry = *lower_bound(dq.rbegin(), dq.rend(),
                        make_pair(Line(0, 0), x),
                        [&](const pair<Line, ll>
                            &a, const pair<Line,
                            ll> &b) {
                            return a.second > b.
                                second;
                        });

        return qry.first.val(x);
    }
};

ll dp[maxn],a[maxn],b[maxn];

void doit(){
    CHT cht;
    cht.insert(b[1],dp[1]); //slope and yIntercept.
    dp[2] = cht.query(a[2]);
    //...
}
```

7.2 Edit Distance

```
// O(m*n) donde cada uno es el tamano de cada string
int editDist(string &s1, string &s2){
    int m = sz(s1), n = sz(s2);
    int dp[m+1][n+1];
    fore(i,0,m+1)
        fore(j,0,n+1){
            if (i==0) dp[i][j] = j;
            else if (j==0) dp[i][j] = i;
            else if (s1[i-1] == s2[j-1]) dp[i][j] = dp[i-1][j
                -1];
            else dp[i][j] = 1 + min({dp[i][j-1],  // Insert
                                    dp[i-1][j],  // Remove
                                    dp[i-1][j-1]}); //
                                        Replace
        }
    return dp[m][n];
}
```

7.3 Knapsack 01 Optimization

```
/*  0/1 Knapsack optimization where sum of all items is ~
    N
    There will be at most sqrt(N) different items.
    Array 'cnt' represents the count of a specific item.
*/

const ll maxn = 1e5+50;
const ll maxnsq = 400;

ll n,m,cnt[maxn],dp[maxnsq][maxn];
vi c;
```

```cpp
void calculateDp(){ //DP in O(N*sqrt(N))
    dp[0][0]=0;
    fore(i,1,n+5) dp[0][i] = -1;
    fore(i,1,sz(c)){ // c is the array of unique items.
        fore(j,1,n+1){
            if(dp[i-1][j] >= 0)
                dp[i][j] = 0;
            else if(j-c[i] >= 0 && dp[i][j - c[i]] >= 0
                and dp[i][j - c[i]] < cnt[c[i]])
                dp[i][j] = dp[i][j - c[i]] + 1;
            else
                dp[i][j] = -1;
        }
    }
}
```

7.4 Longest Common Subsequence

```cpp
const int maxn = 1005;
int dp[maxn][maxn];
int lcs(const string &s, const string &t){
    int n = sz(s), m = sz(t);
    fore(j,0,m+1) dp[0][j] = 0;
    fore(i,0,n+1) dp[i][0] = 0;
    fore(i,1,n+1){
        fore(j,1,m+1){
        dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        if (s[i-1] == t[j-1]){
            dp[i][j] = max(dp[i][j], dp[i-1][j-1] + 1);
        }
        }
    }
    return dp[n][m];
}
```

7.5 Longest Increasing Subsequence

```cpp
// Longest increasing subsequence O(nlogn)
const ll INF = 1e18;
int lis(const vi &a) {
    int n = sz(a);
    vi d(n+1, INF);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        int j = upper_bound(ALL(d), a[i]) - d.begin();
        if (d[j-1] < a[i] && a[i] < d[j]) d[j] = a[i];
    }

    int ans = 0;
    fore(i,0,n+1) if (d[i]<INF) ans = i;
    return ans;
    }
```

7.6 Sum Over Subsets

```cpp
const ll maxbit = 20;
const ll maxn = 1<<20;
ll dp[maxn][maxbit+1]; //{mask,last bit}
ll n,sos[maxn],a[maxn];

void sum_over_subsets(){
    fore(mask,0,maxn){
    dp[mask][0] = a[mask];
    fore(x,0,maxbit){
      dp[mask][x+1] = dp[mask][x];
      if (mask & (1<<x)){
        dp[mask][x+1] += dp[mask - (1ll<<x)][x];
      }
    }
    sos[mask] = dp[mask][maxbit];
  }
}
```

8 Geometry

8.1 Template 2D

```cpp
const ld EPS = 1e-6;
const ll INF = 1e18;

struct Point {
    ld x, y; //cambiar tipo de dato de acuerdo al
       problema
    void read(){ cin>>x>>y; }
    Point operator +(const Point& b) const { return Point
       {x+b.x, y+b.y}; } //suma de puntos
    Point operator -(const Point& b) const { return Point
       {x-b.x, y-b.y}; } //resta de puntos
    ll operator *(const Point& b) const { return (ll) x *
        b.y - (ll) y * b.x; }
    Point operator *(const ld k) const { return Point{x*k
       ,y*k}; }
    bool operator <(const Point& b) const { return x == b
       .x ? y < b.y : x < b.x; }
    void operator +=(const Point& b) { x += b.x; y += b.y
       ; }
    void operator -=(const Point &b) { x -= b.x; y -= b.y
       ; }
    void operator *=(const int k) { x *= k; y *= k; }
    bool operator ==(const Point &b){
        if (b.x == (*this).x && b.y == (*this).y) return
            true;
```

```
        return false;
    }
    ld magnitude() const { return sqrt((x*x) + (y*y)); }
        //longitud hipotenusa
    ld dot (const Point &b){ return (x * b.x) + (y * b.y)
        ; } //producto punto.
    // Si es el producto punto es positivo, el angulo
        entre los vectores es menor a 90 grados, igual a 0
         los vectores son perpendiculares y si es negativo
         el angulo es obtuso.
    ld dist (const Point & b){ return (*this - b).
        magnitude(); } //distancia entre 2 puntos.
    ll cross(const Point& b, const Point& c) const { //
        Producto cruz
        ll cruz = (b - *this) * (c - *this);
        if (cruz < 0) return -1; //Clockwise (right)
        if (cruz > 0) return +1; //Counter-clockwise (
            left)
        return 0; //Collinear.
    }
    ld rawCross(const Point &a, const Point &b) const {
    return (a - *this) * (b - *this);
  }
    bool onSegment(Point p, Point r){ //checa si un punto
         esta en el segmento entre dos puntos (delimitado
        como si fuera un rectangulo)
        if ((*this).x <= max(p.x, r.x) && (*this).x >=
            min(p.x, r.x) && (*this).y <= max(p.y, r.y) &&
             (*this).y >= min(p.y, r.y)) return true;

        return false;
    }
    ld angleBetweenVectors(const Point &b){ //this: (b-a)
        , Point b: (c-a).
        ld ang = acos((*this).dot(b)/((*this).magnitude()
             * b.magnitude()));
        ang = (ang * 180.0) / PI;
        return ang; //return angle in degrees.
    }
};

struct LineToPoint{ //calcula la distancia entre un punto
     y una recta.
    Point p1,p2;
    ld dist(Point refPoint){
        return abs((refPoint - p1) * (refPoint - p2)) /
            p1.dist(p2);
    }
};

ld degreesToRadians(ld degrees) {
    return degrees * PI / 180.0;
}

ld radiansToDegrees(ld radians){
    return radians * (180.0 / PI);
}

signed main(){}
```

8.2 Template 3D

```
struct point3{
  ld x,y,z;
  void read(){
    cin>>x>>y>>z;
  }
  point3(): x(0), y(0), z(0) {}
  point3(ld x, ld y, ld z): x(x), y(y), z(z) {}
  point3 operator - (const point3 &b) const { return
      point3(x - b.x, y - b.y, z - b.z);}
  bool operator == (const point3 &b) const {return x == b
      .x && y == b.y && z == b.z;}
  ld dot(const point3 &b) const { return x*b.x + y*b.y +
      z*b.z;}
  point3 cross(const point3 &b) const { return {(y*b.z) -
       (z*b.y), (z*b.x) - (x*b.z), (x*b.y) - (y*b.x)};}
};

struct plane{
  point3 n; ld d;
  plane(): n(0,0,0), d(0) {}
  plane(point3 n, ld d): n(n), d(d) {}
  plane(point3 p1, point3 p2, point3 p3): plane((p2-p1).
      cross(p3-p1), p1.dot((p2-p1).cross(p3-p1))) {} //
      Initialize by giving 3 points in the plane.
  ld side(const point3 &p) const { return ((*this).n).dot
      (p) - (*this).d;}
    /*
    If side(p) > 0: The point p is on the positive side
        of the plane (in the direction of the normal).
    If side(p) < 0: The point p is on the negative side
        of the plane.
    If side(p) == 0: The point p lies on the plane.
    */
};
```

8.3 Formulas

```
// Volume of a sphere.
ld volumeSphere(ld rad){
    return (4.0/3.0)*PI*rad*rad*rad;
}

// Volume of a sphere cap.
ld volumeCap(ld h, ld rad){
    return PI*h*h*(rad-(h/3.0));
}
```

```
// Area of a triangle given vertices A, B, and C
ld areaTriangle(Point A, Point B, Point C) {
    return fabs((A.x * (B.y - C.y) + B.x * (C.y - A.y) +
        C.x * (A.y - B.y)) / 2.0);
}

// Area of a circle
ld areaCircle(ld radius) {
    return PI * radius * radius;
}

// Area of a trapezoid given bases and height
ld areaTrapezoid(ld base1, ld base2, ld height) {
    return 0.5 * (base1 + base2) * height;
}

// Volume of a cone
ld volumeCone(ld radius, ld height) {
    return (PI * radius * radius * height) / 3.0;
}

// Volume of a cylinder
ld volumeCylinder(ld radius, ld height) {
    return PI * radius * radius * height;
}

// Volume of a rectangular prism
ld volumeRectPrism(ld length, ld width, ld height) {
    return length * width * height;
}

// Volume of a pyramid with a rectangular base
ld volumePyramid(ld length, ld width, ld height) {
    return (length * width * height) / 3.0;
}

// Area of a parallelogram given two vectors (base and
   height)
ld areaParallelogram(Point base, Point heightVec) {
    return fabs(base * heightVec);
}

// Perimeter of a polygon given vertices (assuming
   vertices are in order)
ld perimeterPolygon(vector<Point> &vertices) {
    ld perimeter = 0.0;
    fore(i, 0, sz(vertices)) {
        perimeter += vertices[i].dist(vertices[(i + 1) %
            sz(vertices)]);
    }
    return perimeter;
}

// Volume of a prism with base area and height
ld volumePrism(ld baseArea, ld height) {
    return baseArea * height;
}

// Surface area of a sphere
ld surfaceAreaSphere(ld radius) {
    return 4 * PI * radius * radius;
}

// Surface area of a cylinder
ld surfaceAreaCylinder(ld radius, ld height) {
    return 2 * PI * radius * (radius + height);
}
```

8.4 Angular Sweep

```
struct Point{
    ll x,y,nume,denom,idx,typ,quad;
};

ll quadrantLocation(Point p){ //4 quadrants in 2D space.
    if (p.x == 0 || p.y == 0){
        if (p.x == 0 && p.y == 0) return 0; //origin.
        else if (p.y == 0){
            if (p.x > 0) return 1;
            else return 3;
        }
        else{
            if (p.y > 0) return 2;
            else return 4;
        }
    }
    else{
        if (p.x > 0 && p.y > 0) return 1;
        else if (p.x < 0 && p.y > 0) return 2;
        else if (p.x < 0 && p.y < 0) return 3;
        else return 4;
    }
}

ll n;
vector <Point> a,q1,q2,q3,q4;
vi active;

void init(){
    a.clear(), a.resize(0);
    q1.clear(), q1.resize(0);
    q2.clear(), q2.resize(0);
    q3.clear(), q3.resize(0);
    q4.clear(), q4.resize(0);
    active.clear(), active.resize(n+1);
}

bool cmp(Point p1, Point p2){
    ll f1 = p1.nume*p2.denom;
    ll f2 = p1.denom*p2.nume;
    if (f1 == f2){
        return p1.typ < p2.typ;
    }
```

```
    return f1 < f2;
}

void toqVector(Point &p){
    if (p.quad == 1) q1.pb(p);
    else if (p.quad == 2) q2.pb(p);
    else if (p.quad == 3) q3.pb(p);
    else q4.pb(p);
}

ld findXIntercept(Point p1, Point p2) {
    // Calculate the slope
    ld m = (ld)(p2.y - p1.y) / (ld)(p2.x - p1.x);

    // Calculate the y-intercept (b) using one of the
       points
    ld b = p1.y - m * p1.x;

    // Calculate the x-coordinate where y = 0
    ld xIntercept = -b / m;

    return xIntercept;
}

void angularSort(){
    sort(ALL(q1),cmp);
    sort(ALL(q2),cmp);
    sort(ALL(q3),cmp);
    sort(ALL(q4),cmp);
    for(auto elm : q1) a.pb(elm);
    for(auto elm : q2) a.pb(elm);
    for(auto elm : q3) a.pb(elm);
    for(auto elm : q4) a.pb(elm);
}

void angularSweep(){ // go through the max amount of
   points from the origin [0,0]
    ll pos = 0;
    ll cnt = 0;
    ll cur = 0;
    ll ans = 0;
    while(cnt < 2*n){ //pass through all points twice.
        if (a[pos].typ == 1){ //activate
            cur++;
            active[a[pos].idx] = 1;
            ans = max(ans,cur);
        }
        else{ //deactivate
            if (active[a[pos].idx] == 1){
                cur--;
                active[a[pos].idx] = 0;
                cnt++;
            }
        }
        pos = (pos+1)%(2*n);
    }
    cout<<ans<<nl;
}
```

```
void lineSettings(){
    cin>>n;
    fore(i,0,n){
        Point p1,p2;
        cin>>p1.x>>p1.y>>p2.x>>p2.y;
        p1.idx = p2.idx = i;
        ll g1 = __gcd(abs(p1.x),abs(p1.y));
        ll g2 = __gcd(abs(p2.x),abs(p2.y));
        p1.nume = p1.y/g1;
        p1.denom = p1.x/g1;
        p2.nume = p2.y/g2;
        p2.denom = p2.x/g2;
        p1.quad = quadrantLocation(p1);
        p2.quad = quadrantLocation(p2);
        if (p1.quad > p2.quad) swap(p1,p2); //p1 estara
           en un cuadrante mas chico siempre.

        if (p1.quad == 1 && p2.quad == 3){
            ld xInt = findXIntercept(p1,p2);
            if (xInt > 0.0){
                p2.typ = 1;
                p1.typ = 2;
            }
            else{
                p1.typ = 1;
                p2.typ = 2;
            }
        }
        else if (p1.quad == 1 && p2.quad == 4){
            p1.typ = 2;
            p2.typ = 1;
        }
        else if (p1.quad == 2 && p2.quad == 4){
            ld xInt = findXIntercept(p1,p2);
            if (xInt > 0.0){
                p2.typ = 1;
                p1.typ = 2;
            }
            else{
                p1.typ = 1;
                p2.typ = 2;
            }
        }
        else{
            if (p1.quad == p2.quad){
                vector <Point> aux;
                aux.pb(p1);
                aux.pb(p2);
                sort(ALL(aux),cmp);
                p1 = aux[0];
                p2 = aux[1];
                p1.typ = 1;
                p2.typ = 2;
            }
            else if (p1.quad < p2.quad){
                p1.typ = 1;
```

```
                p2.typ = 2;
            }
        }
        toqVector(p1);
        toqVector(p2);
    }
}

void doit(){
    lineSettings();
    angularSort();
    angularSweep();
}
```

8.5 Check Parallelism

```
bool checkParallelism(Point p1, Point p2, Point p3, Point
    p4){ //(p1 -- p2) es una linea (p3 -- p4) es la otra
    linea.
    Point pr1 = {p2.x-p1.x,p2.y-p1.y};
    Point pr2 = {p4.x-p3.x,p4.y-p3.y};
    double cp = (pr1*pr2);
    return abs(cp) < EPS; //son paralelas si su producto
        cruz = 0.
}
```

8.6 Check Perpendicularity

```
bool checkPerpendicular(Point p1, Point p2, Point p3,
    Point p4){ //(p1 -- p2) es una linea (p3 -- p4) es la
    otra linea.
    Point pr1 = {p2.x-p1.x,p2.y-p1.y};
    Point pr2 = {p4.x-p3.x,p4.y-p3.y};
    double dotp = pr1.dot(pr2);
    return abs(dotp) < EPS; // son perpendiculares si su
        producto punto = 0;
}
```

8.7 Circle-Line Intersection

```
vector <Point> circleLineIntersection(double a, double b,
    double c, double r){
    //Dados los coeficientes de la ecuacion de la recta y
        el radio del circulo con centro en el origen
    vector <Point> pts;
    double x0 = -a*c/(a*a+b*b), y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS){} // 0 points.
    else if (abs (c*c - r*r*(a*a+b*b)) < EPS) pts.pb({x0,
        y0}); // 1 point.
    else{ // 2 points.
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax, ay, bx, by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;
        pts.pb({ax,ay});
        pts.pb({bx,by});
    }
    return pts;
}
```

8.8 Closest Pair

```
ll closestPair(vector <pii> pts){
    //Calcula el par de puntos en 2D mas cercanos entre
        si, retorna su distancia euclidiana.
  int n = sz(pts);
    sort(ALL(pts));
    set<pii> s;

  ll ans = INF;
    int pos = 0;
    fore(i,0,n){
        ll d = ceil(sqrt(ans));
        while (pts[i].first - pts[pos].first >= d) {
            s.erase({pts[pos].second, pts[pos].first});
            pos++;
        }

        auto it1 = s.lower_bound({pts[i].second - d, pts[
            i].first});
        auto it2 = s.upper_bound({pts[i].second + d, pts[
            i].first});

        for (auto it = it1; it != it2; it++) {
            ll dx = pts[i].first - it->second;
            ll dy = pts[i].second - it->first;
      if (ans > 1LL * dx * dx + 1LL * dy * dy){
        ans = 1LL * dx * dx + 1LL * dy * dy;
      }
        }
        s.insert({pts[i].second, pts[i].first});
    }
  return ans;
}
```

8.9 Convex Hull

```
vector <Point> calculateHull(vector <Point> &p, int n){
    //Calculo del Convex Hull
    if (n <= 2) return p;
```

```
    vector<Point> hull;
    int tam = 0;
    sort(ALL(p));
    fore(t,0,2){
        fore(i,0,n){
            while(sz(hull)-tam >= 2){
                Point p1 = hull[sz(hull)-2];
                Point p2 = hull[sz(hull)-1];
                //Producto cruz: P1 ---> P2 ---> P3
                //agregar (<=) si tambien se quieren
                    incluir los puntos colineales, sino
                    solo (<)
                if(p1.cross(p2, p[i]) <= 0) break;
                hull.pop_back();
            }
            hull.pb(p[i]);
        }
        hull.pop_back();
        tam = sz(hull);
        reverse(ALL(p));
    }
    return hull;
}
```

8.10 Equation of Line

```
// Dados 2 puntos de una recta, devuelve los coeficientes
    de Ax + By + C = 0
vector <ld> equation_of_line(Point p1, Point p2){
    ld a = p1.y-p2.y;
    ld b = p2.x-p1.x;
    ld c = -(a*p1.x) - (b*p1.y);
    return {a,b,c};
}
```

8.11 Half Plane Intersection

```
const ld eps = 1e-9, inf = 1e9;

struct Halfplane {
    Point p, pq;
    ld angle;

  // IMPORTANT: Consider the left part of a vector as the
      inside part of the halfplane.
    Halfplane(const Point& a, const Point& b) : p(a), pq(
        b - a) {
        angle = atan2l(pq.y, pq.x);
    }

    bool out(const Point& r) {
    Point ot = r-p;
    return (pq*ot) < -eps;
    }

    bool operator < (const Halfplane& e) const {
        return angle < e.angle;
    }

    friend Point inter(const Halfplane& s, const
        Halfplane& t) {
    Point ot = t.p - s.p;
        ld alpha = (ot*t.pq) / (s.pq*t.pq);
        return s.p + (s.pq * alpha);
    }
};

vector<Point> hp_intersect(vector<Halfplane>& H) {
    Point box[4] = {  // Bounding box in CCW order
        Point(inf, inf),
        Point(-inf, inf),
        Point(-inf, -inf),
        Point(inf, -inf)
    };

    for(int i = 0; i<4; i++) { // Add bounding box half-
        planes.
        Halfplane aux(box[i], box[(i+1) % 4]);
        H.pb(aux);
    }

    // Sort by angle and start algorithm
    sort(ALL(H));
    deque<Halfplane> dq;
    int len = 0;
    for(int i = 0; i < sz(H); i++) {

        // Remove from the back of the deque while last
            half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[len-1], dq[
            len-2]))) {
            dq.pop_back();
            --len;
        }

        // Remove from the front of the deque while first
            half-plane is redundant
        while (len > 1 && H[i].out(inter(dq[0], dq[1])))
            {
            dq.pop_front();
            --len;
        }

        // Special case check: Parallel half-planes
        if (len > 0 && fabsl((H[i].pq*dq[len-1].pq)) <
            eps) {
            // Opposite parallel half-planes that ended
                up checked against each other.
            if (H[i].pq.dot(dq[len-1].pq) < 0.0) return
                vector<Point>();
```

```cpp
            if (H[i].out(dq[len-1].p)){ // Same direction half-plane: keep only the leftmost half-plane.
                dq.pop_back();
                --len;
            }
            else continue;
        }

        // Add new half-plane
        dq.push_back(H[i]);
        ++len;
    }

    // Final cleanup: Check half-planes at the front against the back and vice-versa
    while (len > 2 && dq[0].out(inter(dq[len-1], dq[len-2]))) {
        dq.pop_back();
        --len;
    }

    while (len > 2 && dq[len-1].out(inter(dq[0], dq[1]))) {
        dq.pop_front();
        --len;
    }

    // Report empty intersection if necessary
    if (len < 3) return vector<Point>();

    // Reconstruct the convex polygon from the remaining half-planes.
    vector<Point> pts;
    for(int i = 0; i < sz(dq); i++){
        int j = (i + 1) % sz(dq);
        pts.pb(inter(dq[i], dq[j]));
    }
    return pts;
}
```

8.12 Line Intersection

```cpp
bool doIntersect(Point p1, Point q1, Point p2, Point q2){
    //Checa si 2 lineas se intersectan o no.
    int o1 = p1.cross(q1, p2);
    int o2 = p1.cross(q1, q2);
    int o3 = p2.cross(q2, p1);
    int o4 = p2.cross(q2, q1);

    if (o1 != o2 && o3 != o4) return true;
    if (o1 == 0 && p2.onSegment(p1, q1)) return true;
    if (o2 == 0 && q2.onSegment(p1, q1)) return true;
    if (o3 == 0 && p1.onSegment(p2, q2)) return true;
    if (o4 == 0 && q1.onSegment(p2, q2)) return true;
    return false;
}
```

8.13 Planar Graph

```cpp
// Sort the points counterclockwise around a reference point
bool sort_ccw(const Point& p, const Point& a, const Point& b) {
    return atan2(a.y - p.y, a.x - p.x) < atan2(b.y - p.y, b.x - p.x);
}

// Find a face of the graph
vector<Point> find_face(map<Point, vector<Point>>& neighbors, const Point& u, const Point& v) {
    vector<Point> face;
    Point current = v, previous = u;

    face.pb(previous);
    while (true) {
        face.pb(current);
        vector<Point>& current_neighbors = neighbors[current];
        auto index = find(ALL(current_neighbors), previous) - current_neighbors.begin();
        int next_index = (index + 1) % sz(current_neighbors);
        Point next_vertex = current_neighbors[next_index];
        if (next_vertex.x == u.x && next_vertex.y == u.y) break;
        previous = current;
        current = next_vertex;
    }
    face.pb(u);
    return face;
}

// Find the outer edge of the graph
pair<Point, Point> find_outer_edge(map<Point, vector<Point>>& mp) {
    auto leftmost = min_element(ALL(mp), [](const pair<Point, vector<Point>>& a, const pair<Point, vector<Point>>& b) {
        return tie(a.first.x, a.first.y) < tie(b.first.x, b.first.y);
    })->first;

    vector<Point>& N_leftmost = mp[leftmost];
    sort(ALL(N_leftmost), [&leftmost](const Point& a, const Point& b) {
        return sort_ccw(leftmost, a, b);
    });
```

```cpp
    Point u = N_leftmost[0];
    return {leftmost, u};
}

void doit(){
    int n; // n points.
    map<Point, vector<Point>> mp; //adjacency list.
    set<pair<Point, Point>> seen; //seen edges.

    for (int i = 0; i < n; ++i) {
        ll x1, y1, x2, y2;
        cin >> x1 >> y1 >> x2 >> y2;
        Point p1 = {x1,y1}, p2 = {x2,y2};
        mp[p1].pb(p2);
        mp[p2].pb(p1);
    }

    for (auto& p : mp) {
        //Sort each adjacency list in counter-clockwise
           order.
        sort(ALL(p.second), [&p](const Point& a, const
           Point& b) {
            return sort_ccw(p.first, a, b);
        });
    }

    auto [p, q] = find_outer_edge(mp);
    vector<Point> outer = find_face(mp, p, q);

    fore(i,0,sz(outer)-1) seen.insert({outer[i], outer[(i
       +1)%sz(outer)]});

    for (const auto& p : mp) { // find inner faces of the
         planar graph:
        for (const auto& q : p.second) {
            if (seen.count({p.first, q})) continue;
            seen.insert({p.first, q});
            vector<Point> face = find_face(mp, p.first, q
               );
            fore(i,0,sz(face)-1) seen.insert({face[i],
               face[(i+1)%sz(face)]});
        }
    }
}
```

8.14 Point Inside Polygon Linear

```cpp
// Checa si un punto dado esta DENTRO, FUERA o en
   FRONTERA con un poligono
string checkPointInsidePolygon(vector <Point> P, Point
   point, int n){
    P[0] = point;
    ll count = 0;
    if (n < 3) return "OUTSIDE";
    fore(i,1,n+1){
        int j = (i == n ? 1 : i+1);
        if(P[i].x <= P[0].x && P[0].x < P[j].x && P[0].
           cross(P[i], P[j]) < 0) count++;
        else if(P[j].x <= P[0].x && P[0].x < P[i].x && P
           [0].cross(P[j], P[i]) < 0)  count++;

        if ((min(P[i].x,P[j].x) <= point.x && point.x <=
           max(P[i].x,P[j].x)) && (min(P[i].y,P[j].y) <=
           point.y && point.y <= max(P[i].y,P[j].y)) &&
           point.cross(P[i],P[j]) == 0){
            return "BOUNDARY";
        }
    }
    if (count%2 == 1) return "INSIDE";
    return "OUTSIDE";
}
```

8.15 Point Inside Polygon Optimized

```cpp
int sgn(ll val) { return val > 0 ? 1 : (val == 0 ? 0 :
   -1); }

bool pointInTriangle(Point a, Point b, Point c, Point
   point){
    ll s1 = abs(a.rawCross(b, c));
    ll s2 = abs(point.rawCross(a, b)) + abs(point.
       rawCross(b, c)) + abs(point.rawCross(c, a));
    return s1 == s2;
}

//Precalculation for queries to know if a point lies
   inside of a convex polygon.
void prepareConvexPolygon(int &n, vector<Point> &points,
   vector<Point> &seq, Point &translation){ //seq and
   translation are empty here.
    n = points.size();
    int pos = 0;
    for (int i = 1; i < n; i++) {
        if (points[i] < points[pos])
            pos = i;
    }
    rotate(points.begin(), points.begin() + pos, points.
       end());

    n--;
    seq.resize(n);
    for (int i = 0; i < n; i++)
        seq[i] = points[i + 1] - points[0];
    translation = points[0];
}

//Know if a point lies inside of a convex polygon in O(
   logN)
bool pointInConvexPolygon(Point point, int &n, vector<
   Point> &seq, Point &translation) {
```

```
    point = point - translation;
    if (seq[0]*point != 0 && sgn(seq[0]*point) != sgn(seq
        [0]*seq[n-1])) return false;
    if (seq[n - 1]*point != 0 && sgn(seq[n - 1]*point) !=
         sgn(seq[n - 1]*seq[0])) return false;
  if (seq[0]*point == 0) return seq[0].dot(seq[0]) >=
     point.dot(point);
  int l = 0, r = n - 1;
    while (r - l > 1) {
        int mid = (l + r) / 2;
        int pos = mid;
        if (seq[pos]*point >= 0) l = mid;
    else r = mid;
    }
    int pos = l;
    return pointInTriangle(seq[pos], seq[pos + 1], Point
        {0,0}, point);
}

void doit(){
    int n;
    vector <Point> poly; //with input.
    vector <Point> seq; //empty.
    Point translation;
    prepareConvexPolygon(n,poly,seq,translation);
    // then call pointInConvexPolygon() for queries.
}
```

8.16 Polygon Area

```
ld getPolygonArea(vector <Point> poly){ //Calculo de area
     de poligono
    ll ans = 0;
    poly.pb(poly.front());
    fore(i,1,sz(poly)) ans += (poly[i-1]*poly[i]);
    return abs(ans)/2.0;
}
```

9 Miscellaneous

9.1 Coordinate Compression

```
vi a;
map <ll,ll> mp;
int pos = 0;
sort(ALL(a));
st.erase(unique(ALL(a)),a.end());
for (auto au : a){
  mp[au] = pos;
  pos++;
}
```

9.2 Isomorphism Rooted

```
const ll maxn = 2e5+100;
map <vector <ll>, ll> mp;
ll idx=1;

int dfs(int anode, int node, vector < vector <ll> > &adj)
    {
    vector <ll> v;
    for(auto au : adj[node]){
        if (anode != au) v.pb(dfs(node,au,adj));
    }
    sort(ALL(v));
    if (!mp.count(v)) mp[v] = idx, idx++;
    return mp[v];
}

void doit(){
    ll tree1 = dfs(1,1,adj);
    ll tree2 = dfs(1,1,adj2);
    cout<<(tree1 == tree2 ? "Same" : "Diff")<<nl;
}
```

9.3 Isomorphism Unrooted

```
vi center(int n, vvi &adj) {
    int deg[n+1] = {0};
    virtual v;
    for (int i = 1; i <= n; i++) {
        deg[i] = sz(adj[i]);
        if (deg[i] == 1)
            v.pb(i), deg[i]--;
    }
    int m = sz(v);
    while(m < n) {
        vi vv;
        for (auto i: v) {
            for (auto j: adj[i]) {
                deg[j]--;
                if (deg[j] == 1)
                    vv.pb(j);
            }
        }
        m += sz(vv);
        v = vv;
    }
    return v;
}
map<vi, ll> mp;
int idx = 0;

int dfs(int s, int p, vvi &adj) {
    vi v;
```

```
        for (auto i: adj[s]) {
            if (i != p)
                v.pb(dfs(i, s, adj));
        }
        sort(ALL(v));
        if (!mp.count(v)) mp[v] = idx++;
        return mp[v];
    }

    void doit(){
        // build adjacency lists (1-indexed nodes).
        vi v1 = center(n,adj), v2 = center(n,adj2);
        bool flag = false;
        int s1 = dfs(v1[0], -1, adj);
            for(auto s : v2){
            int s2 = dfs(s, -1, adj2);
            if (s1 == s2){
                flag=true;
                break;
            }
        }
        cout<<(flag ? "YES" : "NO")<<nl;
    }
```

9.4 Max Subarray Sum

```
    const ll maxn = 2e5+100;
    ll a[maxn];

    struct Node{
        ll max_sum,sumL,sumR,sum;
        Node operator+(Node b) {
            return {max(max(max_sum, b.max_sum), sumR + b.
                sumL),
                        max(sumL, sum + b.sumL), max(b.sumR,
                            sumR + b.sum),
                        sum + b.sum};
        }
    };

    struct STree{
        vector <Node> st; int n;
        STree(int n): st(4*n + 5), n(n){}

        void build(int node, int ini, int fin){
            if (ini == fin){
                st[node] = {max(0ll, a[ini]), max(0ll, a[ini
                    ]), max(0ll, a[ini]), a[ini]};
                return;
            }
            int mid = (ini+fin)/2;
            build(2*node,ini,mid);
            build(2*node + 1,mid+1,fin);
            st[node] = st[2*node] + st[2*node + 1];
        }

        void update(int node, int ini, int fin, int pos, ll
            val){
            if (fin < pos || pos < ini) return;
            if (ini == fin && ini == pos){
                st[node] = {max(0ll, val), max(0ll, val), max
                    (0ll, val), val};
                return;
            }
            ll mid = (ini+fin)/2;
            update(2*node,ini,mid,pos,val);
            update(2*node + 1,mid+1,fin,pos,val);
            st[node] = st[2*node] + st[2*node + 1];
        }

        void build(){ build(1,1,n);}
        void update(int pos, ll val){ update(1,1,n,pos,val);}
    };

    void doit(){
        // read values and build ST.
        // queries: st.st[1].max_sum
    }
```

9.5 Parallel Binary Search

```
    bool changed=true;
    while(changed){
        changed=false;
        // HERE CLEAR THE DATA STRUCTURE BEING USED!!
        fore(i,1,n+1){
            if (l[i] != r[i]) tocheck[(l[i] + r[i])/2].pb(i);
        }
        fore(i,1,q+1){
            apply(qs[i].l,qs[i].r); //Apply the i-th query on
                the DS.
            while(sz(tocheck[i])){
                changed=true;
                int id = tocheck[i].back();
                tocheck[i].pop_back();

                // Move l[id] and r[id] accordingly.
                if (check()) l[id] = i+1;
                else r[id]=i;
            }
        }
    }
```

9.6 Small to Large

```
    const ll maxn = 2e5+5;
    ll n,res[maxn];
```

```
vi adj[maxn];
set <ll> colors[maxn];

void dfs(int anode, int node){ //amount of distinct
    elements in a subtree.
    for(auto au : adj[node]){
        if (anode != au){
            dfs(node,au);
            //current node's set should always be the
                larger one.
            if (sz(colors[node]) < sz(colors[au])){
                swap(colors[node],colors[au]);
            }
            for(auto elm : colors[au]) colors[node].
                insert(elm);
        }
    }
    res[node] = sz(colors[node]);
}
```

9.7 Ternary Search 2D

```
const ld INF = 1e4+100;
const ld eps = 1e-5;

struct Point{ld x,y;};

ld costf(Point p){return 0;}

ld get_y(ld x){ // looking for minimums.
    ld l = -INF;
    ld r = INF;
    while(r - l > eps){
        ld m1 = l + (r - l) / 3;
        ld m2 = r - (r - l) / 3;
        Point p1,p2;
        p1 = {x,m1}, p2 = {x,m2};
        ld f1 = costf(p1)*(-1);
        ld f2 = costf(p2)*(-1);
        if (f1 < f2) l = m1;
        else r = m2;
    }
    return costf({x,{(l+r)/2}});
}

ld get_xy(){ // looking for minimums.
    ld l = -INF;
    ld r = INF;
    while(r - l > eps){
        ld m1 = l + (r - l) / 3;
        ld m2 = r - (r - l) / 3;
        ld f1= get_y(m1)*(-1);
        ld f2 = get_y(m2)*(-1);
        if (f1 < f2) l = m1;
        else r = m2;
    }
    return get_y((l+r)/2);
}

void doit(){
    cout<<fixed<<setprecision(10)<<get_xy()<<nl;
}
```

9.8 Ternary Search

```
double f(double x){ return x; }

double ternary_search(double l, double r) { //use long
    doubles (ld) for more precision.
    double eps = 1e-9;
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
        double f2 = f(m2);
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);
} //to find the minimum of a function, invert the sign
    (-1) of the result of f(x)
```

10 Theory

DP Optimization Theory

Name	Original Recurrence	Sufficient Condition	From	To
CH 1	$dp[i] = min_{j<i}\{dp[j] + b[j] * a[i]\}$	$b[j] \geq b[j+1]$ Optionally $a[i] \leq a[i+1]$	$O(n^2)$	$O(n)$
CH 2	$dp[i][j] = min_{k<j}\{dp[i-1][k] + b[k] * a[j]\}$	$b[k] \geq b[k+1]$ Optionally $a[j] \leq a[j+1]$	$O(kn^2)$	$O(kn)$
D&Q	$dp[i][j] = min_{k<j}\{dp[i-1][k] + C[k][j]\}$	$A[i][j] \leq A[i][j+1]$	$O(kn^2)$	$O(kn \log n)$
Knuth	$dp[i][j] = min_{i<k<j}\{dp[i][k] + dp[k][j]\} + C[i][j]$	$A[i, j-1] \leq A[i, j] \leq A[i+1, j]$	$O(n^3)$	$O(n^2)$

Notes:

- $A[i][j]$ - the smallest k that gives the optimal answer, for example in $dp[i][j] = dp[i-1][k] + C[k][j]$
- $C[i][j]$ - some given cost function
- We can generalize a bit in the following way $dp[i] = \min_{j<i}\{F[j] + b[j] * a[i]\}$, where $F[j]$ is computed from $dp[j]$ in constant time

Combinatorics

Sums

$\sum_{k=0}^{n} k = n(n+1)/2$	$\binom{n}{k} = \frac{n!}{(n-k)!k!}$
$\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$	$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$	$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$
$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$	$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$
$\sum_{k=0}^{n} k^4 = (6n^5 + 15n^4 + 10n^3 - n)/30$	$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$
$\sum_{k=0}^{n} k^5 = (2n^6 + 6n^5 + 5n^4 - n^2)/12$	$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$
$\sum_{k=0}^{n} x^k = (x^{n+1} - 1)/(x-1)$	$12! \approx 2^{28.8}$
$\sum_{k=0}^{n} kx^k = (x - (n+1)x^{n+1} + nx^{n+2})/(x-1)^2$	$20! \approx 2^{61.1}$
$1 + x + x^2 + \cdots = 1/(1-x)$	

- Hockey-stick identity
  $\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$
- Number of ways to color n-objects with r-colors if all colors must be used at least once
  $\sum_{k=0}^{r} \binom{r}{k}(-1)^{r-k}k^n$ o $\sum_{k=0}^{r} \binom{r}{r-k}(-1)^k(r-k)^n$

Binomial coefficients

Number of ways to pick a multiset of size k from n elements: $\binom{n+k-1}{k}$
Number of n-tuples of non-negative integers with sum s: $\binom{s+n-1}{n-1}$, at most s: $\binom{s+n}{n}$
Number of n-tuples of positive integers with sum s: $\binom{s-1}{n-1}$
Number of lattice paths from $(0,0)$ to (a,b), restricted to east and north steps: $\binom{a+b}{a}$

Multinomial theorem

. $(a_1 + \cdots + a_k)^n = \sum \binom{n}{n_1, \ldots, n_k} a_1^{n_1} \ldots a_k^{n_k}$, where $n_i \geq 0$ and $\sum n_i = n$.

$$\binom{n}{n_1, \ldots, n_k} = M(n_1, \ldots, n_k) = \frac{n!}{n_1! \ldots n_k!}$$

$$M(a, \ldots, b, c, \ldots) = M(a + \cdots + b, c, \ldots)M(a, \ldots, b)$$

Catalan numbers

.

- $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$ con $n \geq 0$, $C_0 = 1$ y $C_{n+1} = \frac{2(2n+1)}{n+2}C_n$
  $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$
- 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670

- C_n is the number of: properly nested sequences of n pairs of parentheses; rooted ordered binary trees with $n+1$ leaves; triangulations of a convex $(n+2)$-gon.

Derangements

. Number of permutations of $n = 0, 1, 2, \ldots$ elements without fixed points is $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$ Recurrence: $D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$. Corollary: number of permutations with exactly k fixed points is $\binom{n}{k} D_{n-k}$.

Stirling numbers of 1^{st} kind

. $s_{n,k}$ is $(-1)^{n-k}$ times the number of permutations of n elements with exactly k permutation cycles. $|s_{n,k}| = |s_{n-1,k-1}| + (n-1)|s_{n-1,k}|$. $\sum_{k=0}^{n} s_{n,k}\, x^k = x^{\underline{n}}$

Stirling numbers of 2^{nd} kind

. $S_{n,k}$ is the number of ways to partition a set of n elements into exactly k non-empty subsets. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = 1$. $x^n = \sum_{k=0}^{n} S_{n,k}\, x^{\underline{k}}$

Bell numbers

. B_n is the number of partitions of n elements. $B_0, \ldots = 1, 1, 2, 5, 15, 52, 203, \ldots$ $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k = \sum_{k=1}^{n} S_{n,k}$. Bell triangle: $B_r = a_{r,1} = a_{r-1,r-1}$, $a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

Bernoulli numbers

. $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1} \sum_{k=0}^{n} \binom{n+1}{k} B_k m^{n+1-k}$.
$\sum_{j=0}^{m} \binom{m+1}{j} B_j = 0$. $B_0 = 1$, $B_1 = -\frac{1}{2}$. $B_n = 0$, for all odd $n \neq 1$.

Eulerian numbers

. $E(n,k)$ is the number of permutations with exactly k descents ($i : \pi_i < \pi_{i+1}$) / ascents ($\pi_i > \pi_{i+1}$) / excedances ($\pi_i > i$) / $k+1$ weak excedances ($\pi_i \geq i$).
Formula: $E(n,k) = (k+1)E(n-1,k) + (n-k)E(n-1,k-1)$. $x^n = \sum_{k=0}^{n-1} E(n,k) \binom{x+k}{n}$.

Burnside's lemma

. The number of orbits under group G's action on set X:
$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X_g|$, where $X_g = \{x \in X : g(x) = x\}$. ("Average number of fixed points.")
Let $w(x)$ be weight of x's orbit. Sum of all orbits' weights: $\sum_{o \in X/G} w(o) = \frac{1}{|G|} \sum_{g \in G} \sum_{x \in X_g} w(x)$.

Number Theory

Linear diophantine equation

. $ax + by = c$. Let $d = \gcd(a,b)$. A solution exists iff $d|c$. If (x_0, y_0) is any solution, then all solutions are given by $(x,y) = (x_0 + \frac{b}{d}t, y_0 - \frac{a}{d}t)$, $t \in \mathbb{Z}$. To find some solution (x_0, y_0), use extended GCD to solve $ax_0 + by_0 = d = \gcd(a,b)$, and multiply its solutions by $\frac{c}{d}$.

Linear diophantine equation in n variables: $a_1x_1 + \cdots + a_nx_n = c$ has solutions iff $\gcd(a_1, \ldots, a_n)|c$. To find some solution, let $b = \gcd(a_2, \ldots, a_n)$, solve $a_1x_1 + by = c$, and iterate with $a_2x_2 + \cdots = y$.

Extended GCD

```
// Finds g = gcd(a,b) and x, y such that ax+by=g.
// Bounds: |x|<=b+1, |y|<=a+1.
void gcdext(int &g, int &x, int &y, int a, int b)
{ if (b == 0) { g = a; x = 1; y = 0; }
  else        { gcdext(g, y, x, b, a % b); y = y - (a / b) * x; } }
```

Multiplicative inverse of a modulo m: x in $ax + my = 1$, or $a^{\phi(m)-1} \pmod{m}$.

Chinese Remainder Theorem

. System $x \equiv a_i \pmod{m_i}$ for $i = 1, \ldots, n$, with pairwise relatively-prime m_i has a unique solution modulo $M = m_1m_2 \ldots m_n$: $x = a_1b_1\frac{M}{m_1} + \cdots + a_nb_n\frac{M}{m_n} \pmod{M}$, where b_i is modular inverse of $\frac{M}{m_i}$ modulo m_i.

System $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$ has solutions iff $a \equiv b \pmod{g}$, where $g = \gcd(m,n)$. The solution is unique modulo $L = \frac{mn}{g}$, and equals: $x \equiv a + T(b-a)m/g \equiv b + S(a-b)n/g \pmod{L}$, where S and T are integer solutions of $mT + nS = \gcd(m,n)$.

Prime-counting function

. $\pi(n) = |\{p \leq n : p \text{ is prime}\}|$. $n/\ln(n) < \pi(n) < 1.3n/\ln(n)$. $\pi(1000) = 168$, $\pi(10^6) = 78498$, $\pi(10^9) = 50\ 847\ 534$. n-th prime $\approx n \ln n$.

Miller-Rabin's primality test

. Given $n = 2^r s + 1$ with odd s, and a random integer $1 < a < n$.
If $a^s \equiv 1 \pmod{n}$ or $a^{2^j s} \equiv -1 \pmod{n}$ for some $0 \leq j \leq r-1$, then n is a probable prime. With bases 2, 7 and 61, the test indentifies all composites below 2^{32}. Probability of failure for a random a is at most 1/4.

Pollard-ρ

. Choose random x_1, and let $x_{i+1} = x_i^2 - 1 \pmod{n}$. Test $\gcd(n, x_{2^k+i} - x_{2^k})$ as possible n's factors for $k = 0, 1, \ldots$ Expected time to find a factor: $O(\sqrt{m})$, where m is smallest prime power in n's factorization. That's $O(n^{1/4})$ if you check $n = p^k$ as a special case before factorization.

Fermat primes

. A Fermat prime is a prime of form $2^{2^n} + 1$. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form $2^n + 1$ is prime only if it is a Fermat prime.

Fermat's Theorem

. Let m be a prime and x and m coprimes, then:

- $x^{m-1} \equiv 1 \mod m$
- $x^k \mod m = x^{k \mod (m-1)} \mod m$
- $x^{\phi(m)} \equiv 1 \mod m$

Perfect numbers

. $n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

Carmichael numbers

. A positive composite n is a Carmichael number ($a^{n-1} \equiv 1 \pmod{n}$ for all a: $\gcd(a, n) = 1$), iff n is square-free, and for all prime divisors p of n, $p - 1$ divides $n - 1$.

Number/sum of divisors

. $\tau(p_1^{a_1} \ldots p_k^{a_k}) = \prod_{j=1}^k (a_j + 1)$. $\sigma(p_1^{a_1} \ldots p_k^{a_k}) = \prod_{j=1}^k \frac{p_j^{a_j+1} - 1}{p_j - 1}$.

Product of divisors

. $\mu(n) = n^{\frac{\tau(n)}{2}}$

- if p is a prime, then: $\mu(p^k) = p^{\frac{k(k+1)}{2}}$
- if a and b are coprimes, then: $\mu(ab) = \mu(a)^{\tau(b)} \mu(b)^{\tau(a)}$

Euler's phi function

. $\phi(n) = |\{m \in \mathbb{N}, m \leq n, \gcd(m, n) = 1\}|$.

- $\phi(mn) = \frac{\phi(m)\phi(n)\gcd(m,n)}{\phi(\gcd(m,n))}$.
- $\phi(p) = p - 1$ si p es primo
- $\phi(p^a) = p^a(1 - \frac{1}{p}) = p^{a-1}(p - 1)$
- $\phi(n) = n(1 - \frac{1}{p_1})(1 - \frac{1}{p_2})...(1 - \frac{1}{p_k})$ donde p_i es primo y divide a n

Euler's theorem

. $a^{\phi(n)} \equiv 1 \pmod{n}$, if $\gcd(a, n) = 1$.
Wilson's theorem

. p is prime iff $(p-1)! \equiv -1 \pmod{p}$.

Mobius function

. $\mu(1) = 1$. $\mu(n) = 0$, if n is not squarefree. $\mu(n) = (-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. If for all $n \in N$, $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d) F(\frac{n}{d})$, and vice versa. $\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d}$. $\sum_{d|n} \mu(d) = 1$.
If f is multiplicative, then $\sum_{d|n} \mu(d) f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n}(1 + f(p))$.
$\sum_{d|n} \mu(d) = e(n) = [n == 1]$.
$S_f(n) = \prod_{p=1}(1 + f(p_i) + f(p_i^2) + ... + f(p_i^{e_i}))$, p - primes(n).

Legendre symbol

. If p is an odd prime, $a \in \mathbb{Z}$, then $\left(\frac{a}{p}\right)$ equals 0, if $p|a$; 1 if a is a quadratic residue modulo p; and -1 otherwise. Euler's criterion: $\left(\frac{a}{p}\right) = a^{\left(\frac{p-1}{2}\right)} \pmod p$.

Jacobi symbol

. If $n = p_1^{a_1} \cdots p_k^{a_k}$ is odd, then $\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{k_i}$.

Primitive roots

. If the order of g modulo m (min $n > 0$: $g^n \equiv 1 \pmod m$) is $\phi(m)$, then g is called a primitive root. If Z_m has a primitive root, then it has $\phi(\phi(m))$ distinct primitive roots. Z_m has a primitive root iff m is one of 2, 4, p^k, $2p^k$, where p is an odd prime. If Z_m has a primitive root g, then for all a coprime to m, there exists unique integer $i = \text{ind}_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod m$. $\text{ind}_g(a)$ has logarithm-like properties: $\text{ind}(1) = 0$, $\text{ind}(ab) = \text{ind}(a) + \text{ind}(b)$.

If p is prime and a is not divisible by p, then congruence $x^n \equiv a \pmod p$ has $\gcd(n, p-1)$ solutions if $a^{(p-1)/\gcd(n,p-1)} \equiv 1 \pmod p$, and no solutions otherwise. (Proof sketch: let g be a primitive root, and $g^i \equiv a \pmod p$, $g^u \equiv x \pmod p$. $x^n \equiv a \pmod p$ iff $g^{nu} \equiv g^i \pmod p$ iff $nu \equiv i \pmod p$.)

Discrete logarithm problem

. Find x from $a^x \equiv b \pmod m$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n = \lceil \sqrt{m} \rceil$, and $x = ny - z$. Equation becomes $a^{ny} \equiv ba^z \pmod m$. Precompute all values that the RHS can take for $z = 0, 1, \ldots, n-1$, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

Pythagorean triples

. Integer solutions of $x^2 + y^2 = z^2$ All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n, \gcd(m, n) = 1$ and $m \not\equiv n \pmod 2$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

- Given an arbitrary pair of integers m and n with $m > n > 0$:
  $a = m^2 - n^2$, $b = 2mn$, $c = m^2 + n^2$
- The triple generated by Euclid's formula is primitive if and only if m and n are coprime and not both odd.
- To generate all Pythagorean triples uniquely:
  $a = k(m^2 - n^2)$, $b = k(2mn)$, $c = k(m^2 + n^2)$
- If m and n are two odd integer such that $m > n$, then:
  $a = mn$, $b = \frac{m^2-n^2}{2}$, $c = \frac{m^2+n^2}{2}$
- If $n = 1$ or 2 there are no solutions. Otherwise
  n is even: $((\frac{n^2}{4} - 1)^2 + n^2 = (\frac{n^2}{4} + 1)^2)$
  n is odd: $((\frac{n^2-1}{2})^2 + n^2 = (\frac{n^2+1}{2})^2)$

Postage stamps/McNuggets problem

. Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax + by$ $(x, y \geq 0)$, and the largest is $(a-1)(b-1) - 1 = ab - a - b$.

Fermat's two-squares theorem

. Odd prime p can be represented as a sum of two squares iff $p \equiv 1 \pmod 4$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k + 3$ occurs an even number of times in n's factorization.

RSA

. Let p and q be random distinct large primes, $n = pq$. Choose a small odd integer e, relatively prime to $\phi(n) = (p-1)(q-1)$, and let $d = e^{-1} \pmod{\phi(n)}$. Pairs (e, n) and (d, n) are the public and secret keys, respectively. Encryption is done by raising a message $M \in Z_n$ to the power e or d, modulo n.

String Algorithms

Burrows-Wheeler inverse transform

. Let $B[1..n]$ be the input (last column of sorted matrix of string's rotations.) Get the first column, $A[1..n]$, by sorting B. For each k-th occurence of a character c at index i in A, let $next[i]$ be the index of corresponding k-th occurence of c in B. The r-th fow of the matrix is $A[r]$, $A[next[r]]$, $A[next[next[r]]]$, ...

Huffman's algorithm

. Start with a forest, consisting of isolated vertices. Repeatedly merge two trees with the lowest weights.

Graph Theory

Euler's theorem

. For any planar graph, $V - E + F = 1 + C$, where V is the number of graph's vertices, E is the number of edges, F is the number of faces in graph's planar drawing, and C is the number of connected components. Corollary: $V - E + F = 2$ for a 3D polyhedron.

Vertex covers and independent sets

. Let M, C, I be a max matching, a min vertex cover, and a max independent set. Then $|M| \leq |C| = N - |I|$, with equality for bipartite graphs. Complement of an MVC is always a MIS, and vice versa. Given a bipartite graph with partitions (A, B), build a network: connect source to A, and B to sink with edges of capacities, equal to the corresponding nodes' weights, or 1 in the unweighted case. Set capacities of the original graph's edges to the infinity. Let (S, T) be a minimum s-t cut. Then a maximum(-weighted) independent set is $I = (A \cap S) \cup (B \cap T)$, and a minimum(-weighted) vertex cover is $C = (A \cap T) \cup (B \cap S)$.

Matrix-tree theorem

. Let matrix $T = [t_{ij}]$, where t_{ij} is the number of multiedges between i and j, for $i \neq j$, and $t_{ii} = -\deg_i$. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T.

Euler tours

. Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected. Construction:

```
doit(u):
  for each edge e = (u, v) in E, do: erase e, doit(v)
  prepend u to the list of vertices in the tour
```

Stable marriages problem

. While there is a free man m: let w be the most-preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

Stoer-Wagner's min-cut algorithm

. Start from a set A containing an arbitrary vertex. While $A \neq V$, add to A the most tightly connected vertex ($z \notin A$ such that $\sum_{x \in A} w(x, z)$ is maximized.) Store cut-of-the-phase (the cut between the last added vertex and rest of the graph), and merge the two vertices added last. Repeat until the graph is contracted to a single vertex. Minimum cut is one of the cuts-of-the-phase.

Tarjan's offline LCA algorithm

. (Based on DFS and union-find structure.)

```
DFS(x):
  ancestor[Find(x)] = x
  for all children y of x:
     DFS(y); Union(x, y); ancestor[Find(x)] = x
  seen[x] = true
  for all queries {x, y}:
    if seen[y] then output "LCA(x, y) is ancestor[Find(y)]"
```

Strongly-connected components

. Kosaraju's algorithm.
1. Let G^T be a transpose G (graph with reversed edges.)
1. Call DFS(G^T) to compute finishing times $f[u]$ for each vertex u.
3. For each vertex u, in the order of decreasing $f[u]$, perform DFS(G, u).
4. Each tree in the 3rd step's DFS forest is a separate SCC.

2-SAT

. Build an implication graph with 2 vertices for each variable – for the variable and its inverse; for each clause $x \vee y$ add edges $(\overline{x}, y)$ and $(\overline{y}, x)$. The formula is satisfiable iff x and $\overline{x}$ are in distinct SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from sinks to sources (i.e. Kosaraju's last step), assigning 'true' to all variables of the current SCC (if it hasn't been previously assigned 'false'), and 'false' to all inverses.

Randomized algorithm for non-bipartite matching

. Let G be a simple undirected graph with even $|V(G)|$. Build a matrix A, which for each edge $(u,v) \in E(G)$ has $A_{i,j} = x_{i,j}$, $A_{j,i} = -x_{i,j}$, and is zero elsewhere. Tutte's theorem: G has a perfect matching iff $\det G$ (a multivariate polynomial) is identically zero. Testing the latter can be done by computing the determinant for a few random values of $x_{i,j}$'s over some field. (e.g. Z_p for a sufficiently large prime p)

Prufer code of a tree

. Label vertices with integers 1 to n. Repeatedly remove the leaf with the smallest label, and output its only neighbor's label, until only one edge remains. The sequence has length $n-2$. Two isomorphic trees have the same sequence, and every sequence of integers from 1 and n corresponds to a tree. Corollary: the number of labelled trees with n vertices is n^{n-2}.

Erdos-Gallai theorem

. A sequence of integers $\{d_1, d_2, \dots, d_n\}$, with $n-1 \geq d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$ is a degree sequence of some undirected simple graph iff $\sum d_i$ is even and $d_1 + \cdots + d_k \leq k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i)$ for all $k = 1, 2, \dots, n-1$.

Games

Grundy numbers

. For a two-player, normal-play (last to move wins) game on a graph (V,E): $G(x) = \text{mex}(\{G(y) : (x,y) \in E\})$, where $\text{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

Sums of games

.

- *Player chooses a game and makes a move in it.* Grundy number of a position is xor of grundy numbers of positions in summed games.
- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them.* A position is losing iff each game is in a losing position.
- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.
- *Player must move in all games, and loses if can't move in some game.* A position is losing if any of the games is in a losing position.

Misère Nim

. A position with pile sizes $a_1, a_2, \dots, a_n \geq 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*.

Bit tricks

Clearing the lowest 1 bit: `x & (x - 1)`, all trailing 1's: `x & (x + 1)`
Setting the lowest 0 bit: `x | (x + 1)`
Enumerating subsets of a bitmask m:
`x=0; do { ...; x=(x+1+~m)&m; } while (x!=0);`
`__builtin_ctz/__builtin_clz` returns the number of trailing/leading zero bits.
`__builtin_popcount(unsigned x)` counts 1-bits (slower than table lookups).
For 64-bit unsigned integer type, use the suffix '`ll`', i.e. `__builtin_popcountll`.
XOR

Let's say F(L,R) is XOR of subarray from L to R.
Here we use the property that F(L,R)=F(1,R) XOR F(1,L-1)

Math

Stirling's approximation

$z! = \Gamma(z+1) = \sqrt{2\pi}\ z^{z+1/2}\ e^{-z}(1 + \frac{1}{12z} + \frac{1}{288z^2} - \frac{139}{51840z^3} + \dots)$

Taylor series

. $f(x) = f(a) + \frac{x-a}{1!} f'(a) + \frac{(x-a)^2}{2!} f^{(2)}(a) + \cdots + \frac{(x-a)^n}{n!} f^{(n)}(a) + \dots$.
$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$

$\ln x = 2(a + \frac{a^3}{3} + \frac{a^5}{5} + \dots)$, where $a = \frac{x-1}{x+1}$. $\ln x^2 = 2 \ln x$.
$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$, $\arctan x = \arctan c + \arctan \frac{x-c}{1+xc}$ (e.g c=.2)
$\pi = 4 \arctan 1$, $\pi = 6 \arcsin \frac{1}{2}$

Fibonacci Period

Si p es primo , $\pi(p^k) = p^{k-1}\pi(p)$
$\pi(2) = 3$ $\pi(5) = 20$
Si n y m son coprimos $\pi(n * m) = lcm(\pi(n), \pi(m))$

List of Primes

1e5	3	19	43	49	57	69	103	109	129	151	153
1e6	33	37	39	81	99	117	121	133	171	183	
1e7	19	79	103	121	139	141	169	189	223	229	
1e8	7	39	49	73	81	123	127	183	213		

2-SAT

Rules
$p \to q \equiv \neg p \vee q$
$p \to q \equiv \neg q \to \neg p$
$p \vee q \equiv \neg p \to q$
$p \wedge q \equiv \neg(p \to \neg q)$
$\neg(p \to q) \equiv p \wedge \neg q$
$(p \to q) \wedge (p \to r) \equiv p \to (q \wedge r)$
$(p \to q) \vee (p \to r) \equiv p \to (q \vee r)$
$(p \to r) \wedge (q \to r) \equiv (p \wedge q) \to r$
$(p \to r) \vee (q \to r) \equiv (p \vee q) \to r$
$(p \wedge q) \vee (r \wedge s) \equiv (p \vee r) \wedge (p \vee s) \wedge (q \vee r) \wedge (q \vee s)$

Summations

- $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=1}^{n} i^3 = (\frac{n(n+1)}{2})^2$
- $\sum_{i=1}^{n} i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$
- $\sum_{i=1}^{n} i^5 = \frac{(n(n+1))^2(2n^2+2n-1)}{12}$
- $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ para $x \neq 1$

Compound Interest

- N is the initial population, it grows at a rate of R. So, after X years the popularion will be $N \times (1+R)^X$

Great circle distance or geographical distance

- d = great distance, ϕ = latitude, λ = longitude, Δ = difference (all the values in radians)
- σ = central angle, angle form for the two vector
- $d = r * \sigma$, $\sigma = 2 * \arcsin(\sqrt{\sin^2(\frac{\Delta\phi}{2}) + \cos(\phi_1)\cos(\phi_2)\sin^2(\frac{\Delta\lambda}{2})})$

Theorems

- There is always a prime between numbers n^2 and $(n+1)^2$, where n is any positive integer
- There is an infinite number of pairs of the from $\{p, p+2\}$ where both p and $p+2$ are primes.
- Every even integer greater than 2 can be expressed as the sum of two primes.
- Every integer greater than 2 can be written as the sum of three primes.
- $a^d \equiv a^{d \mod \phi(n)} \mod n$
  if $a \in \mathbb{Z}^{n_*}$ or $a \notin \mathbb{Z}^{n_*}$ and $d \mod \phi(n) \neq 0$
- $a^d \equiv a^{\phi(n)} \mod n$
  if $a \notin \mathbb{Z}^{n_*}$ and $d \mod \phi(n) = 0$
- thus, for all a, n and d (with $d \geq \log_2(n)$)
  $a^d \equiv a^{\phi(n)+d \mod \phi(n)} \mod n$

Law of sines and cosines

- a, b, c: lenghts, A, B, C: opposite angles, d: circumcircle
- $\frac{a}{sin(A)} = \frac{b}{sin(B)} = \frac{c}{sin(C)} = d$
- $c^2 = a^2 + b^2 - 2ab\cos(C)$

Heron's Formula

- $s = \frac{a+b+c}{2}$
- $Area = \sqrt{s(s-a)(s-b)(s-c)}$
- a, b, c there are the lenghts of the sides

Legendre's Formula

Largest power of k, x, such that $n!$ is divisible by k^x

- If k is prime, $x = \frac{n}{k} + \frac{n}{k^2} + \frac{n}{k^3} + \ldots$
- If k is composite $k = k_1^{p_1} * k_2^{p_2} \ldots k_m^{p_m}$
  $x = min_{1 \le j \le m}\{\frac{a_j}{p_j}\}$ where a_j is Legendre's formula for k_j
- Divisor Formulas of $n!$ Find all prime numbers $\le n$ $\{p_1, \ldots, p_m\}$ Let's define e_j as Legendre's formula for p_j
- Number of divisors of $n!$ The answer is $\prod_{j=1}^{m}(e_j + 1)$
- Sum of divisors of $n!$ The answer is $\prod_{j=1}^{m} \frac{p_j^{e_j+1}-1}{e_j-1}$

Max Flow with Demands

Max Flow with Lower bounds of flow for each edge

- feasible flow in a network with both upper and lower capacity constraints, no source or sink: capacities are changed to upper bound — lower bound. Add a new source and a sink. let M[v] = (sum of lower bounds of ingoing edges to v) — (sum of lower bounds of outgoing edges from v). For all v, if M[v]¿0 then add edge (S,v) with capacity M, otherwise add (v,T) with capacity -M. If all outgoing edges from S are full, then a feasible flow exists, it is the flow plus the original lower bounds. maximum flow in a network with both upper and lower capacity constraints, with source s and sink t: add edge (t,s) with capacity infinity. Binary search for the lower bound, check whether a feasible exists for a network WITHOUT source or sink (B).

Pick's Theorem

- $A = i + \frac{b}{2} - 1$
- A : area of the polygon.
- i : number of interior integer points.
- b : number of integer points on the boundary.