Fast and Fourier ICPC Team Notebook

Contents

1 C++

1.1 C++ template

```
#include <bits/stdc++.h>
using namespace std;

//IMPRESINDIBLES PARA ICPC
#define form(i, s, e) for(int i = s; i < e; i++)
#define icin(x)  \
  int x;         \
  cin >> x;
#define llcin(x) \
  long long x;   \
  cin >> x;
#define scin(x)  \
string x;        \
cin >> x;
#define endl '\n'
#define S second
#define F first
#define pb push_back
#define sz(x) x.size()
#define all(x) x.begin(),x.end()

typedef long long ll;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef pair<int,int> pii;

const ll INF =  1e9+7;//tambien es primo
const double PI = acos(-1);
//UTILES
#define DBG(x) cerr << #x << '=' << (x) << endl
#define coutDouble cout << fixed << setprecision(17)
#define numtobin(n) bitset<32>(n).to_string()
#define bintoint(bin_str) stoi(bin_str, nullptr, 2)  //
   bin_str should be a STRING
#define LSOne(S) ((S) & -(S))

typedef double db;
typedef vector<string> vs;
```

```cpp
typedef vector<ll> vll;
typedef vector<vll> vvll;
typedef pair<int,bool> pib;
typedef pair<ll,ll> pll;
typedef vector<pii> vpii;
typedef vector<pib> vpib;
typedef vector<pll> vpll;


int main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  cout.tie(0);

  icin(nn0)
  while (nn0--) {

  }

  return 0;
}
```

1.2 Opcion

```cpp
// En caso de que no sirva #include <bits/stdc++.h>
#include <algorithm>
#include <iostream>
#include <iterator>
#include <sstream>
#include <fstream>
#include <cassert>
#include <climits>
#include <cstdlib>
#include <cstring>
#include <string>
#include <cstdio>
#include <vector>
#include <cmath>
#include <queue>
#include <deque>
#include <stack>
#include <list>
#include <map>
#include <set>
#include <bitset>
#include <iomanip>
#include <unordered_map>

////
#include <tuple>
#include <random>
#include <chrono>
```

1.3 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)) // CHECKEAR BIT-N
T = mask&(-mask); // LSO
__builtin_ffs(mask); // INDICE DEL LSO
//  iterar sobre los subconjuntos del conjunto S
for(int subset= S; subset; subset= (subset-1) & S)
for (int subset=0;subset=subset-S&S;)    // Increasing
    order
```

1.4 Random

```cpp
// 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);}
```

1.5 Custom Hash

```cpp
struct custom_hash {
    static ll splitmix64(ll x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(ll x) const {
        static const ll FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count()
            ;
        return splitmix64(x + FIXED_RANDOM);
    }
};
unordered_map<ll,int, custom_hash> mapa;
```

1.6 Other

```cpp
#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();

int main() {
  // Ouput a specific number of digits past the decimal
     point,
  // in this case 5
  // #include <iomanip>
  cout << setfill(' ') << setw(3) << 2 << endl;

  cout.setf(ios::fixed); cout << setprecision(5);
  cout << 100.0/7.0 << endl;
  cout.unsetf(ios::fixed);

  // Output the decimal point and trailing zeros
  cout.setf(ios::showpoint); cout << 100.0 << endl; cout.
     unsetf(ios::showpoint);

  // Output a + before positive values
  cout.setf(ios::showpos); cout << 100 << " " << -100 <<
     endl; cout.unsetf(ios::showpos);

  // Output numerical values in hexadecimal
  cout << hex << 100 << " " << 1000 << " " << 10000 <<
     dec << endl;
}
```

2 Strings

2.1 Z's Algorithm

```cpp
// O(|s|)
vi z_function(string &s){
  int n = s.size();
  vi z(n);
  int x = 0, y = 0;
  for(int i = 1; i < n; ++i) {
    z[i] = max(0, min(z[i-x], y-i+1));
    while (i+z[i] < n && s[z[i]] == s[i+z[i]])
      x = i, y = i+z[i], z[i]++;
  }
  return z;
}
```

2.2 KMP

```cpp
vi get_phi(string &s) { // O(|s|)
  int j = 0, n = sz(s); vi pi(n);
  for1(i,n-1){
    while (j > 0 && s[i] != s[j]) j = pi[j-1];
    j += (s[i] == s[j]);
    pi[i] = j;
  }
  return pi;
}
void kmp(string &t, string &p){ // O(|t| + |p|)
  vi phi = get_phi(p);
  int matches = 0;
  for(int i = 0, j = 0; i < sz(t); ++i ) {
    while(j > 0 && t[i] != p[j] ) j = phi[j-1];
    if(t[i] == p[j]) ++j;
    if(j == sz(p)) {
      matches++;
      j = phi[j-1];
    }
  }
}
/// Automaton
/// Complexity O(n*C) where C is the size of the alphabet
int aut[nax][26];
void kmp_aut(string &p) {
  int n = sz(p);
  vi phi = get_phi(p);
  forn(i, n+1) {
    forn(c, 26) {
      if (i==n || (i>0 && 'a'+c!= p[i])) aut[i][c] = aut[
          phi[i-1]][c];
      else  aut[i][c] = i + ('a'+c == p[i]);
    }
  }
}

/// Automaton
int wh[nax+2][MAXC];    //wh[i][j] = a donde vuelvo si
    estoy en i y pongo una j
void build(string &s){
        int lps=0;
```

```
        wh[0][s[0]-'a'] = 1;
        fore(i,1,sz(s)){
                fore(j,0,MAXC-1) wh[i][j]=wh[lps][j];
                if(i<sz(s)){
                        wh[i][s[i]-'a'] = i+1;
                        lps = wh[lps][s[i]-'a'];
                }
        }
}
```

2.3 Hashing

```
/// 1000234999, 1000567999, 1000111997, 1000777121,
    999727999, 1070777777
const int MOD[] = { 1001864327, 1001265673 }, N = 3e5;
const ii BASE(257, 367), ZERO(0, 0), ONE(1, 1);
inline int add(int a, int b, int mod) { return a+b >= mod
     ? a+b-mod : a+b; }
inline int sbt(int a, int b, int mod) { return a-b < 0 ?
    a-b+mod : a-b; }
inline int mul(int a, int b, int mod) { return ll(a) * b
    % mod;}
inline ll operator ! (const ii a) { return (ll(a.fi) <<
    32) | a.se; }
inline ii operator + (const ii& a, const ii& b) {
  return {add(a.fi, b.fi, MOD[0]), add(a.se, b.se, MOD
      [1])};}
inline ii operator - (const ii& a, const ii& b) {
  return {sbt(a.fi, b.fi, MOD[0]), sbt(a.se, b.se, MOD
      [1])};}
inline ii operator * (const ii& a, const ii& b) {
  return {mul(a.fi, b.fi, MOD[0]), mul(a.se, b.se, MOD
      [1])};}

ii base[N]{ONE};
void prepare() { for1(i, N-1) base[i] = base[i-1] * BASE;
     }
template <class type>
struct hashing {    /// HACELEEE PREPAREEEE!!!
  vii ha;        // ha[i] = t[i]*p0 + t[i+1]*p1 + t[i+2]*
      p2 + ..
  hashing(type &t): ha(sz(t)+1, ZERO){
    for(int i = sz(t) - 1; i >= 0; --i) ha[i] = ha[i+1] *
         BASE + ii{t[i], t[i]};
  }
  ii query(int l, int r){ return ha[l] - ha[r+1] * base[r
      -l+1]; } //[l,r]
};
```

2.4 Manacher Algorithm

```
// f = 1 para pares, 0 impar
//a a a a a a
//1 2 3 3 2 1   f = 0 impar
//0 1 2 3 2 1   f = 1 par
void manacher(string &s, int f, vi &d){
  int l=0, r=-1, n=sz(s);
  d.assign(n,0);
  forn(i, n){
    int k=(i>r? (1-f) : min(d[l+r-i+ f], r-i+f)) + f;
    while(i+k-f<n && i-k>=0 && s[i+k-f]==s[i-k]) ++k;
    d[i] = k - f; --k;
    if(i+k-f > r) l=i-k, r=i+k-f;
  }
//  forn(i,n) d[i] = (d[i]-1+f)*2 + 1-f;
}
```

2.5 Minimum Expression

```
int minExp(string &t) {
  int i = 0, j = 1, k = 0, n = sz(t), x, y;
  while (i < n && j < n && k < n) {
    x = i+k;
    y = j+k;
    if (x >= n) x -= n;
    if (y >= n) y -= n;
    if (t[x] == t[y]) ++k;
    else if (t[x] > t[y]) {
      i = j+1 > i+k+1 ? j+1 : i+k+1;
      swap(i, j);
      k = 0;
    } else {
      j = i+1 > j+k+1 ? i+1 : j+k+1;
      k = 0;
    }
  }
  return i;
}
```

2.6 Trie

```
const static int N = 2e6, alpha = 26, B = 30; // MAX:
    abecedario, bits
int to[N][alpha], cnt[N], sz;
inline int conv(char ch){ return ch - 'a'; }  // CAMBIAR
string to_bin(int num, int bits){  // B: Max(bits),  bits
    : size
  return bitset<B>(num).to_string().substr(B - bits);}
// AGREGAR LO QUE HAYA QUE RESETEAR !!!!
void init(){
  forn(i, sz+1) cnt[i] = 0, memset(to[i], 0, sizeof to[i
      ]);
```

```
    sz = 0;
  }
  void add(const string &s){
    int u = 0;
    for(char ch: s){
      int c = conv(ch);
      if(!to[u][c]) to[u][c] = ++sz;
      u = to[u][c];
    }
    cnt[u]++;
  }
```

2.7 Suffix Array

```
struct SuffixArray { // test line 11
        vi sa, lcp;
        SuffixArray(string& s, int lim=256){
                int n = sz(s) + 1, k = 0, a, b;
                s.pb('$');
                vi x(all(s)), y(n), ws(max(n, lim)), rank(n);
                sa = lcp = y, iota(all(sa), 0);
                for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
                        p = j;
//      iota(all(y), n - j);
//                      forn(i,n) if (sa[i] >= j) y[p++] = sa[i] - j;
                        forn(i,n) y[i] = (sa[i] - j >= 0 ? 0 : n) + sa[i]-j; // this replace the two lines
                        // before hopefully xd
                        fill(all(ws), 0);
                        forn(i,n) ws[x[i]]++;
                        for1(i,lim-1) ws[i] += ws[i - 1];
                        for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
                        swap(x, y), p = 1, x[sa[0]] = 0;
                        for1(i,n-1) a = sa[i - 1], b = sa[i], x[b] =
                                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
                }
                for1(i,n-1) rank[sa[i]] = i;
                for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k) // lcp(i): lcp suffix i-1,i
                        for (k && k--, j = sa[rank[i] - 1];
                                        s[i + k] == s[j + k]; k++);
        }
};
```

2.8 Aho-Corasick

```
const static int N = 1e5+1, alpha = 26;
int sz, to[N][alpha], fail[N], end_w[N], cnt_w[N], fail_out[N];
inline int conv(char ch) { return  ch-'a'; }
struct aho_corasick{
  int words=0;
  aho_corasick(vector<string>& str){
    forn(i, sz+1) fail[i] = end_w[i] = cnt_w[i] = fail_out[i] = 0;
    forn(i, sz+1) memset(to[i], 0, sizeof to[i]);
    sz = 0;
    for(string& s: str) add(s);
    build();
  }
  void add(string &s) {
    int v = 0;
    for(char ch : s) {
      int c = conv(ch);
      if(!to[v][c]) to[v][c] = ++sz;
      v = to[v][c];
    }
    ++cnt_w[v];
    end_w[v] = ++words;
  }
  void build() {
    queue<int> q{{0}};
    while(sz(q)) {
      int u = q.front();  q.pop();
      forn(i, alpha) {
        int v = to[u][i];
        if(!v) to[u][i] = to[ fail[u] ][i];
        else q.push(v);
        if(!u || !v) continue;
        fail[v] = to[ fail[u] ][i];
        fail_out[v] = end_w[ fail[v] ] ? fail[v] : fail_out[ fail[v] ];
        cnt_w[v] += cnt_w[ fail[v] ];
      }
    }
  }
  int match(string &s){
    int v = 0, mat = 0;
    for(char ch: s) {
      v = to[v][conv(ch)];
      mat += cnt_w[v];
    }
    return mat;
  }
};
```

2.9 Suffix Automaton

```cpp
struct node {
  int len, link;
  map<char, int> to;  // if TLE --> change to array<int,
      27> to;
  bool terminal;
};

const int N = 4e5+1; // el doble del MAXN
node st[N];
int sz, last, occ[N], cnt[N];
bool seen[N];

struct suf_aut{
  suf_aut(string& s){
    forn(i, sz) st[i] = node();
    sz = 1;
    st[0].len = st[0].terminal = last = 0;
    st[0].link = -1;
    for(char c: s) extend(c);
  }
  void extend(char c) {
    int v = sz++, p = last;
    st[v].len = st[p].len + 1;
    while(p != -1 && !st[p].to[c]) st[p].to[c] = v, p =
        st[p].link;
    if(p == -1) st[v].link = 0;
    else{
      int q = st[p].to[c];
      if(st[p].len + 1 == st[q].len) st[v].link = q;
      else{
        int w = sz++;
        st[w].len = st[p].len + 1;
        st[w].to = st[q].to;
        st[w].link = st[q].link;
        while(p != -1 && st[p].to[c] == q) st[p].to[c] =
            w, p = st[p].link;
        st[q].link = st[v].link = w;
      }
    }
    cnt[last = v] = 1;
  }
  int dfs_occ(int v){
    if(occ[v]) return occ[v];
    occ[v] = st[v].terminal;
    for(auto &[_, u] : st[v].to) occ[v] += dfs_occ(u);
    return occ[v];
  }
  void calc_cnt(){
    vi ord(sz - 1);  iota(all(ord), 1);
    sort(all(ord), [&](int i, int j){ return st[i].len >
        st[j].len; });
    for(int v: ord) cnt[st[v].link] += cnt[v];  // Add
        cnt to link
  }
  string LCS(string &t){
    int v = 0, l = 0;
    ii mx{0, -1};
    forn(i, sz(t)){
      while(v && !st[v].to.count(t[i])) v = st[v].link, l
          = st[v].len;
      if(st[v].to.count(t[i])) v = st[v].to[t[i]], ++l;
      mx = max(mx, {l, i}); // LCS ending at position i
    }
    return t.substr(mx.se - mx.fi + 1, mx.fi);
  }
  int cyclic_match(string& t){
    int n = sz(t), v = 0, l = 0, ans = 0;
    t += t;
    forn(i, sz(t)){
      while(v && !st[v].to.count(t[i])) v = st[v].link, l
          = st[v].len;
      if(st[v].to.count(t[i])) v = st[v].to[t[i]], ++l;
      if(i >= n){
        if(v && st[st[v].link].len >= n) v = st[v].link,
            l = st[v].len;
        if(!seen[v] && l >= n) seen[v] = 1, ans += cnt[v
            ]; // Match
      }
    }
    return ans;
  }
};
```

2.10 Palindromic Tree

```cpp
struct palindromic_tree{
  static const int SIGMA = 26;
  struct node{
    int link, len, p, to[SIGMA];
    node(int len, int link=0,int p=0):
        len(len),link(link),p(p){
      memset(to,0,sizeof(to));
    }
  };
  int last;
  vector<node> st;
  palindromic_tree():last(0){fore(i,-1,0)st.pb(node(i));}

  void add(int i, const string &s){
    int c = s[i]-'a';
    int p = last;
    while(s[i-st[p].len-1]!=s[i]) p=st[p].link;

    if(st[p].to[c]){
      last = st[p].to[c];
    }else{
      int q=st[p].link;
      while(s[i-st[q].len-1]!=s[i]) q=st[q].link;
```

```
        q=max(1,st[q].to[c]);
        last = st[p].to[c] = sz(st);
        st.pb(node(st[p].len+2,q,p));
      }
    }
  };
```

2.11 Suffix Tree

```
const int N=1000000,    // maximum possible number of
    nodes in suffix tree
    INF=1000000000; // infinity constant
string a;       // input string for which the suffix tree
     is being built
int t[N][26],   // array of transitions (state, letter)
    l[N],   // left...
    r[N],   // ...and right boundaries of the substring
        of a which correspond to incoming edge
    p[N],   // parent of the node
    s[N],   // suffix link
    tv,     // the node of the current suffix (if we're
        mid-edge, the lower node of the edge)
    tp,     // position in the string which corresponds
        to the position on the edge (between l[tv] and r[
        tv], inclusive)
    ts,     // the number of nodes
    la;     // the current character in the string

void ukkadd(int c) { // add character s to the tree
    suff:;      // we'll return here after each
        transition to the suffix (and will add character
        again)
    if (r[tv]<tp) { // check whether we're still within
        the boundaries of the current edge
        // if we're not, find the next edge. If it doesn'
            t exist, create a leaf and add it to the tree
        if (t[tv][c]==-1) {t[tv][c]=ts;l[ts]=la;p[ts++]=
            tv;tv=s[tv];tp=r[tv]+1;goto suff;}
        tv=t[tv][c];tp=l[tv];
    } // otherwise just proceed to the next edge
    if (tp==-1 || c==a[tp]-'a')
        tp++; // if the letter on the edge equal c, go
            down that edge
    else {
        // otherwise split the edge in two with middle in
             node ts
        l[ts]=l[tv];r[ts]=tp-1;p[ts]=p[tv];t[ts][a[tp]-'a
            ']=tv;
        // add leaf ts+1. It corresponds to transition
            through c.
        t[ts][c]=ts+1;l[ts+1]=la;p[ts+1]=ts;
        // update info for the current node - remember to
             mark ts as parent of tv
        l[tv]=tp;p[tv]=ts;t[p[ts]][a[l[ts]]-'a']=ts;ts
            +=2;
        // prepare for descent
        // tp will mark where are we in the current
            suffix
        tv=s[p[ts-2]];tp=l[ts-2];
        // while the current suffix is not over, descend
        while (tp<=r[ts-2]) {tv=t[tv][a[tp]-'a'];tp+=r[tv
            ]-l[tv]+1;}
        // if we're in a node, add a suffix link to it,
            otherwise add the link to ts
        // (we'll create ts on next iteration).
        if (tp==r[ts-2]+1) s[ts-2]=tv; else s[ts-2]=ts;
        // add tp to the new edge and return to add
            letter to suffix
        tp=r[tv]-(tp-r[ts-2])+2;goto suff;
    }
}

void build() {
    ts=2;
    tv=0;
    tp=0;
    fill(r,r+N,(int)a.size()-1);
    // initialize data for the root of the tree
    s[0]=1;
    l[0]=-1;
    r[0]=-1;
    l[1]=-1;
    r[1]=-1;
    memset (t, -1, sizeof t);
    fill(t[1],t[1]+26,0);
    // add the text to the tree, letter by letter
    for (la=0; la<(int)a.size(); ++la)
        ukkadd (a[la]-'a');
}
```

3 Graph algorithms

3.1 Articulation Points and Bridges

```
// Complexity: V + E
// Given an undirected graph
int n, timer, tin[nax], low[nax];
vi g[nax]; // adjacency list of graph

void dfs(int u, int p) {
  tin[u] = low[u] = ++timer;
  int children=0;
  for (int v : g[u]) {
    if (v == p) continue;
    if (tin[v]) low[u] = min(low[u], tin[v]);
    else {
```

```
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] > tin[u]) // BRIDGE
        IS_BRIDGE(u, v);
      if (low[v] >= tin[u] && p!=-1) // POINT
        IS_CUTPOINT(u);
      ++children;
    }
  }
  if(p == -1 && children > 1) // POINT
    IS_CUTPOINT(u);
}
void find_articulations() {
  timer = 0;
  forn(i,n) if(!tin[i]) dfs(i,-1);
}
```

3.2 Biconnected Components

```
struct edge {
        int u, v, comp; //A que componente biconexa
            pertenece
        bool bridge; //Si la arista es un puente
};
vector<int> g[nax]; //Lista de adyacencia
vector<edge> e; //Lista de aristas
stack<int> st;
int low[nax], num[nax], cont;
int art[nax]; //Si el nodo es un punto de articulacion
//vector<vector<int>> comps; //Componentes biconexas
//vector<vector<int>> tree; //Block cut tree
//vector<int> id; //Id del nodo en el block cut tree
int nbc; //Cantidad de componentes biconexas
int N, M; //Cantidad de nodos y aristas
void add_edge(int u, int v){
        g[u].pb(sz(e)); g[v].pb(sz(e));
        e.pb({u, v, -1, false});
}
void dfs(int u, int p = -1) {
        low[u] = num[u] = cont++;
        for (int i : g[u]) {
                edge &ed = e[i];
                int v = ed.u^ed.v^u;
                if(num[v]<0){
                        st.push(i);
                        dfs(v, i);
                        if (low[v] > num[u]) ed.bridge =
                            true; //bridge
                        if (low[v] >= num[u]) {
                                art[u]++; //articulation
                                int last; //start
                                    biconnected
//                              comps.pb({});
                                do {
                                        last = st.top();
                                            st.pop();
                                        e[last].comp =
                                            nbc;
//                                      comps.back().pb(e
    [last].u);
//                                      comps.back().pb(e
    [last].v);
                                } while (last != i);
                                nbc++; //end biconnected
                        }
                        low[u] = min(low[u], low[v]);
                } else if (i != p && num[v] < num[u]) {
                        st.push(i);
                        low[u] = min(low[u], num[v]);
                }
        }
}
void build_tree() {
        tree.clear(); id.resize(N); tree.reserve(2*N);
        forn(u,N)
                if (art[u]) id[u] = sz(tree); tree.pb({})
                    ;
        for (auto &comp : comps) {
    sort(all(comp));
    comp.resize(unique(all(comp)) - comp.begin());
                int node = sz(tree);
                tree.pb({});
                for (int u : comp) {
                        if (art[u]) {
                                tree[id[u]].pb(node);
                                tree[node].pb(id[u]);
                        }else id[u] = node;
                }
        }
}
void doit() {
        cont = nbc = 0;
//      comps.clear();
        forn(i,N) {
                g[i].clear(); num[i] = -1; art[i] = 0;
        }
        forn(i,N){
    if(num[i]<0) dfs(i), --art[i];
        }
}
```

3.3 Topological Sort

```
vi g[nax], ts;
bool seen[nax];
void dfs(int u){
  seen[u] = true;
  for(int v: g[u])
    if (!seen[v])
      dfs(v);
  ts.pb(u);
}
void topo(int n){
  forn(i,n) if (!seen[i]) dfs(i);
  reverse(all(ts));
}
```

3.4 Kosaraju: Strongly connected components

```
vi g[nax], gr[nax], ts;
bool seen[nax];
int scc[nax], comp;
void dfs1(int u){
  seen[u] = 1;
  for(int v: g[u]) if(!seen[v]) dfs1(v);
  ts.pb(u);
}
void dfs2(int u){
  scc[u] = comp;
  for(int v : gr[u]) if(scc[v] == -1) dfs2(v);
}
int find_scc(int n){ //TENER CREADO EL GRAFO REVERSADO gr
  forn(i, n) if(!seen[i]) dfs1(i);
  reverse(all(ts));
  memset(scc, -1, sizeof scc);
  for(int u: ts) if(scc[u] == -1) ++comp, dfs2(u);
  return comp;
}
```

3.5 Tarjan: Strongly connected components

```
vi low, num, comp, g[nax];
int scc, timer;
stack<int> st;
void tjn(int u) {
  low[u] = num[u] = timer++; st.push(u); int v;
  for(int v: g[u]) {
    if(num[v]==-1) tjn(v);
    if(comp[v]==-1) low[u] = min(low[u], low[v]);
  }
  if(low[u]==num[u]) {
    do{ v = st.top(); st.pop(); comp[v]=scc;
    }while(u != v);
    ++scc;
  }
}
void callt(int n) {
  timer = scc= 0;
  num = low = comp = vector<int>(n,-1);
  forn(i,n) if(num[i]==-1) tjn(i);
}
```

3.6 MST Kruskal

```
struct edge{
  int u, v, w;
  edge(int u, int v, int w): u(u), v(v), w(w){}
  bool operator < (const edge &o) const{ return w < o.w;}
};
vector<edge> g;
void kruskal(int n){
  sort(all(g));  dsu uf(n); // union-find
  for(auto& [u, v, w]: g)
    if(!uf.is_same_set(u, v)) uf.union_set(u, v);
}
```

3.7 MST Prim

```
//Complexity O(E * log V)
vector<ii> g[nax];
bool seen[nax];
priority_queue<ii> pq;
void process(int u) {
  seen[u] = true;
  for (ii v: g[u])
    if (!seen[v.fi])
      pq.push(ii(-v.se, v.fi));
}
int prim(int n){
  process(0);
  int total = 0, u, w;
  while (sz(pq)){
    ii e = pq.top(); pq.pop();
    tie(w,u) = e; w*=-1;
    if (!seen[u])
      total += w, process(u);
  }
  return total;
}
```

3.8 Dijkstra

```
// O ((V+E)*log V)
vector <ii> g[nax];
int d[nax], p[nax];
void dijkstra(int s, int n){
```

```
  forn(i, n) d[i] = inf, p[i] = -1;
  d[s] = 0;
  priority_queue <ii, vector <ii>,greater<ii> > q;
  q.push({0, s});
  while(sz(q)){
    auto [dist, u] = q.top();  q.pop();
    if(dist > d[u]) continue;
    for(auto& [v, w]: g[u]){
      if (d[u] + w < d[v]){
        d[v] = d[u] + w;
        p[v] = u;
        q.push(ii(d[v], v));
      }
    }
  }
}
vi find_path(int t){
  vi path;
  int cur = t;
  while(cur != -1){
    path.pb(cur);
    cur = p[cur];
  }
  reverse(all(path));
  return path;
}
```

3.9 Bellman-Ford

```
vector<ii> g[nax];
ll dist[nax];
bool bellman_ford(int s, int n){
  forn(i, n) dist[i] = inf;
  dist[s] = 0;
  forn(_, n-1){
    forn(u, n){
      if(dist[u] == inf) continue; // Unreachable
      for(auto& [v, w] : g[u])
        if(dist[u] + w < dist[v]) dist[v] = dist[u] + w,
            pa[v] = u;
    }
  }
  int start = -1;
  forn(u, n){
    if(dist[u] == inf) continue;  // Unreachable
    for(auto& [v, w] : g[u]) if(dist[u] + w < dist[v])
        start = v;
  }
  if(start == -1) return 0;
  else{  // Si se necesita reconstruir
    forn(_, n) start = pa[start];
    vi cycle{start};
    int v = start;
    while(pa[v] != start) v = pa[v], cycle.pb(v);
    cycle.pb(start); // solo si se necesita que vuelva al
        start
    reverse(all(cycle));
    return 1;
  }
}
```

3.10 Shortest Path Faster Algorithm

```
// Complexity O(V*E) worst, O(E) on average.
vector<ii> g[N];
ll dist[N];
int pa[N], cnt[N];
bool in_q[N];
bool spfa(int s, int n){
  forn(i, n) dist[i] = (i == s ? 0 : inf);
  queue<int> q({s});  in_q[s] = 1;
  int start = -1;
  while(sz(q) && start == -1) {
    int u = q.front();  q.pop();
    in_q[u] = 0;
    for(auto& [v, w] : g[u]){
      if(dist[u] + w < dist[v]) {
        dist[v] = dist[u] + w;
        pa[v] = u;
        if(!in_q[v]) {
          q.push(v);
          in_q[v] = 1;
          ++cnt[v];
          if(cnt[v] > n){ start = v;  break; }
        }
      }
    }
  }
  if(start == -1) return 0;
  else{  // Si se necesita reconstruir
    forn(_, n) start = pa[start];
    vi cycle{start};
    int v = start;
    while(pa[v] != start) v = pa[v], cycle.pb(v);
    cycle.pb(start); // solo si se necesita que vuelva al
        start
    reverse(all(cycle));
    return 1;
  }
}
```

3.11 Floyd-Warshall

```
// Complejidad O(n^3)
int dist[nax][nax];
void floyd(){
```

```
    // Hay que saber inicializar el array d.
    forn(k,n){
      forn(u,n){
        forn(v,n){
          dist[u][v] = min(dist[u][v], dist[u][k] + dist[k
              ][v]);
        }
      }
    }
  }
```

3.12 LCA Binary Lifting

```
const int L = 24;
int timer, up[nax][L+1], n;
int in[nax], out[nax];
vi g[nax];
void dfs(int u, int p){
  in[u] = ++timer;
  up[u][0] = p;
  for1(i,L) up[u][i] = up[up[u][i-1]][i-1];
  for(int v: g[u]){
    if(v==p) continue;
    dfs(v,u);
  }
  out[u] = ++timer;
}
bool anc(int u, int v){
  return in[u]<= in[v] && out[u]>= out[v];
}
void solve(int root){
  timer = 0;
  dfs(root,root);
}
int lca(int u, int v){
  if(anc(u,v)) return u;
  if(anc(v,u)) return v;
  for(int i= L; i>=0; --i){
    if(!anc(up[u][i],v))
      u = up[u][i];
  }
  return up[u][0];
}
```

3.13 2 SAT

```
// Complexity O(V+E)
int N;
vi low, num, comp, g[nax];
vector<bool> truth;
int scc, timer;
stack<int> st;
void tjn(int u) {
  low[u] = num[u] = timer++; st.push(u); int v;
  for(int v: g[u]) {
    if(num[v]==-1) tjn(v);
    if(comp[v]==-1) low[u] = min(low[u], low[v]);
  }
  if(low[u]==num[u]) {
    do{ v = st.top(); st.pop(); comp[v]=scc;
    }while(u != v);
    ++scc;
  }
}
bool solve_2SAT() {
  int n = 2*N;
  timer = scc= 0;
  num = low = comp = vi(n,-1);
  forn(i,n)
    if(num[i]==-1) tjn(i);
  truth = vector<bool>(N, false);
  forn(i,N) {
    if (comp[i] == comp[i + N]) return false;
    truth[i] = comp[i] < comp[i + N];
  }
  return true;
}
int neg(int x){
  if(x<N) return x+N;
  else return x-N;
}
void add_edge(int x, int y){
  g[x].pb(y);
}
void add_disjuntion(int x, int y){
  add_edge(neg(x), y);
  add_edge(neg(y), x);
}
void implies(int x, int y) {
  add_edge(x,y);
  add_edge(neg(y),neg(x));
}
void make_true(int u) { add_edge(neg(u), u); }
void make_false(int u) { make_true(neg(u)); }
void make_eq(int x, int y){
  implies(x, y);
  implies(y, x);
}
void make_dif(int x, int y){
  implies(neg(x), y);
  implies(neg(y), x);
}
```

3.14 2 SAT Kosaraju y Tarjan

```
// Complexity O(V+E)
```

```cpp
// KOSARAJU
int N, scc;
vi g[2][nax], ts, comp;
vector<bool> truth;

void dfs(int u, int id) {
  if(!id) comp[u] = -2;
  else comp[u] = scc;
  for (int v : g[id][u]){
    if(!id && comp[v]==-1) dfs(v,id);
    else if(id && comp[v]==-2) dfs(v,id);
  }
  if(!id) ts.pb(u);
}

bool solve_2SAT() {
  int n = 2*N;
  comp.assign(n, -1), truth.assign(N, false);
  forn(i,n) if(comp[i]==-1) dfs(i,0);
  scc= 0;
  forn(i,n){
    int v = ts[n - i - 1];
    if (comp[v] ==-2) dfs(v,1), ++scc;
  }
  forn(i,N) {
    if (comp[i] == comp[i + N]) return false;
    truth[i] = comp[i] > comp[i + N];
  }
  return true;
}
void add_edge(int x, int y){
  g[0][x].pb(y);
  g[1][y].pb(x);
}
//////////////////////////////////
// TARJAN testeado con 2 problemas
// Complexity O(V+E)
int N;
vi low, num, comp, g[nax];
vector<bool> truth;
int scc, timer;
stack<int> st;
void tjn(int u) {
  low[u] = num[u] = timer++; st.push(u); int v;
  for(int v: g[u]) {
    if(num[v]==-1) tjn(v);
    if(comp[v]==-1) low[u] = min(low[u], low[v]);
  }
  if(low[u]==num[u]) {
    do{ v = st.top(); st.pop(); comp[v]=scc;
    }while(u != v);
    ++scc;
  }
}
bool solve_2SAT() {
  int n = 2*N;
  timer = scc= 0;
  num = low = comp = vi(n,-1);
  forn(i,n) if(num[i]==-1) tjn(i);
  truth = vector<bool>(N, false);
  forn(i,N) {
    if (comp[i] == comp[i + N]) return false;
    truth[i] = comp[i] < comp[i + N];
  }
  return true;
}
int neg(int x){
  if(x<N) return x+N;
  else return x-N;
}
void add_edge(int x, int y){
  g[x].pb(y);
}
void add_disjuntion(int x, int y){
  add_edge(neg(x), y);
  add_edge(neg(y), x);
}
void implies(int x, int y) {
  add_edge(x,y);
  add_edge(neg(y),neg(x));
}
void make_true(int u) { add_edge(neg(u), u); }
void make_false(int u) { make_true(neg(u)); }
void make_eq(int x, int y){
  implies(x, y);
  implies(y, x);
}
void make_dif(int x, int y){
  implies(neg(x), y);
  implies(neg(y), x);
}
```

3.15 Centroid Decomposition

```cpp
int cnt[nax], depth[nax], f[nax], dist[25][nax];
vi g[nax];
int dfs(int u, int dep = -1, bool flag = 0, int dis = 0,
    int p = -1) {
  cnt[u] = 1;
  if(flag) dist[dep][u] = dis;
  for (int v : g[u])
    if (!depth[v] && v != p) cnt[u] += dfs(v, dep, flag,
        dis + 1, u);
  return cnt[u];
}
int get_centroid (int u, int r, int p = -1) {
  for (int v : g[u])
    if (!depth[v] && v != p && cnt[v] > r)
      return get_centroid(v, r, u);
```

```
    return u;
  }
  int decompose(int u, int d = 1) {
    int centroid = get_centroid(u, dfs(u)>>1);
    depth[centroid] = d;
    dfs(centroid, d); /// if distances is needed
    for (int v : g[centroid])
      if (!depth[v])
        f[decompose(v, d + 1)] = centroid;
    return centroid;
  }
  int lca (int u, int v) {
    for (; u != v; u = f[u])
      if (depth[v] > depth[u])
        swap(u, v);
    return u;
  }
  int get_dist(int u, int v){
    int dep_l = depth[lca(u,v)];
    return dist[dep_l][u] + dist[dep_l][v];
  }
```

3.16 Tree Binarization

```
vi g[nax];
int son[nax], bro[nax];
void binarize(int u, int p = -1){
  bool flag = 0; int prev = 0;
  for(int v : g[u]){
    if(v == p) continue;
    if(flag) bro[prev] = v;
    else son[u] = v, flag = true;
    binarize(v, u);
    prev = v;
  }
}
```

3.17 Eulerian Path

```
int n;
int edges = 0;
int out[nax], in[nax];

// Directed version (uncomment commented code for
   undirected)
struct edge {
        int v;
//      list<edge>::iterator rev;
        edge(int v):v(v){}
};
list<edge> g[nax];
void add_edge(int a, int b){
  out[a]++;
  in[b]++;
        ++edges;
        g[a].push_front(edge(b));//auto ia=g[a].begin();
//      g[b].push_front(edge(a));auto ib=g[b].begin();
//      ia->rev=ib;ib->rev=ia;
}
vi p;
void go(int u){
        while(sz(g[u])){
                int v=g[u].front().v;
                //g[v].erase(g[u].front().rev);
                g[u].pop_front();
                go(v);
        }
        p.push_back(u);
}

vi get_path(int u){
        p.clear();
        go(u);
        reverse(all(p));
        return p;
}

/// for undirected uncomment and check for path existance
bool eulerian(vi &tour) { /// directed graph
  int one_in = 0, one_out = 0, start = -1;
  bool ok = true;
  for (int i = 0; i < n; i++) {
    if(out[i] && start == -1) start = i;
    if(out[i] - in[i] == 1) one_out++, start = i;
    else if(in[i] - out[i] == 1) one_in++;
    else ok &= in[i] == out[i];
  }
  ok &= one_in == one_out && one_in <= 1;
  if (ok) {
    tour = get_path(start);
    if(sz(tour) == edges + 1) return true;
  }
  return false;
}
```

4 Flows

4.1 Edmons-Karp

```
// Complexity O(V*E^2)
const ll inf = 1e18;

struct EKarp{
  vector<int> p;
  vector<vector<ll>> cap, flow;
```

```
  vector<vector<int>> g;
  int n, s, t;

  EKarp(int n_){
    n = n_; g.resize(n);
    cap = flow = vector<vector<ll>>(n,vector<ll>(n));
  }

  void addEdge(int u, int v, ll c){
    cap[u][v] = c;
    g[u].pb(v); g[v].pb(u);
  }

  ll bfs(int s, int t) {
    p.assign(n, -1);  p[s] = -2;
    queue<pair<int,ll>> q;
    q.push(pair<int,ll>(s, inf));
    while (!q.empty()) {
      int u = q.front().fi; ll f = q.front().se;
      q.pop();
      for(int v: g[u]){
        if (p[v] == -1 && cap[u][v] - flow[u][v]>0) {
          p[v] = u;
          ll df = min(f, cap[u][v]-flow[u][v]);
          if (v == t) return df;
          q.push(pair<int,ll>(v, df));
        }
      }
    }
    return 0;
  }
  ll maxFlow() {
    ll mf = 0;
    ll f;
    while (f = bfs(s,t)){
      mf += f;
      int v = t;
      while (v != s) {
        int prev = p[v];
        flow[v][prev] -= f;
        flow[prev][v] += f;
        v = prev;
      }
    }
    return mf;
  }
};
```

4.2 Dinic

```
// Corte minimo: vertices con dist[v]>=0 (del lado de src
   ) VS.  dist[v]==-1 (del lado del dst)
// Para el caso de la red de Bipartite Matching (Sean V1
   y V2 los conjuntos mas proximos a src y dst
   respectivamente):
// Reconstruir matching: para todo v1 en V1 ver las
   aristas a vertices de V2 con it->f>0, es arista del
   Matching
// Min Vertex Cover: vertices de V1 con dist[v]==-1 +
   vertices de V2 con dist[v]>0
// Max Independent Set: tomar los vertices NO tomados por
    el Min Vertex Cover
// Max Clique: construir la red de G complemento (debe
   ser bipartito!) y encontrar un Max Independent Set
// Min Edge Cover: tomar las aristas del matching + para
   todo vertices no cubierto hasta el momento, tomar
   cualquier arista de el
// Complexity O(V^2*E)
const ll inf = 1e18;
struct edge {
  int to, rev; ll cap, f{0};
  edge(int to, int rev, ll cap): to(to), rev(rev), cap(
     cap){}
};
struct Dinic{
  int n, s, t; ll max_flow = 0;
  vector<vector<edge>> g;
  vi q, dis, work;
  Dinic(int n, int s, int t): n(n), s(s), t(t), g(n), q(n
     ){}
  void addEdge(int s, int t, ll cap){
    g[s].pb(edge(t, sz(g[t]), cap));
    g[t].pb(edge(s, sz(g[s])-1, 0));
  }

  bool bfs(){
    dis.assign(n, -1), dis[s] = 0;
    int qt = 0;
    q[qt++] = s;
    forn(qh, qt){
      int u = q[qh];
      for(auto& [v, _, cap, f]: g[u])
        if(dis[v] < 0 && f < cap) dis[v] = dis[u] + 1, q[
           qt++] = v;
    }
    return dis[t] >= 0;
  }
  ll dfs(int u, ll cur){
    if(u == t) return cur;
    for(int& i = work[u]; i < sz(g[u]); ++i){
      auto& [v, rev, cap, f] = g[u][i];
      if(cap <= f) continue;
      if(dis[v] == dis[u] + 1){
        ll df = dfs(v, min(cur, cap - f));
        if(df > 0){
          f += df, g[v][rev].f -= df;
          return df;
        }
      }
    }
```

```
      return 0;
    }
    ll maxFlow(){
      ll cur_flow = 0;
      while(bfs()){
        work.assign(n, 0);
        while(ll delta = dfs(s, inf)) cur_flow += delta;
      }
      max_flow += cur_flow;
      // todos los nodos con dis[u]!=-1 vs los que tienen
         dis[v]==-1 forman el min-cut, (u,v)
      return max_flow;
    }
    vii min_cut(){
      maxFlow();
      vii cut;
      forn(u, n){
        if(dis[u] == -1) continue;
        for(auto& e: g[u]) if(dis[e.to] == -1) cut.pb({u, e
           .to});
      }
      sort(all(cut)),  cut.resize(unique(all(cut)) - cut.
         begin());
      return cut;
    }
  };
```

4.3 Push-Relabel

```
  // Complexity O(V^2 * sqrt(E)) o O(V^3)
  const ll inf = 1e17;
  struct PushRelabel{
    struct edge {
      int to, rev; ll f, cap;
      edge(int to, int rev, ll cap, ll f = 0) : to(to), rev
         (rev), f(f), cap(cap) {}
    };
    void addEdge(int s, int t, ll cap){
      g[s].pb(edge(t, sz(g[t]), cap));
      g[t].pb(edge(s, sz(g[s])-1, (ll)0));
    }

    int n, s, t;
    vi height; vector<ll> excess;
    vector<vector<edge>> g;

    PushRelabel(int n_){
      n = n_; g.resize(n);
    }
    void push(int u, edge &e){
      ll d = min(excess[u], e.cap - e.f);
      edge &rev = g[e.to][e.rev];
      e.f += d;  rev.f -= d;
      excess[u] -= d;  excess[e.to] += d;
    }
    void relabel(int u){
      ll d = inf;
      for (edge e : g[u])
        if (e.cap - e.f > 0)
          d = min(d,(ll) height[e.to]);

      if (d < inf) height[u] = d + 1;
    }
    vi find_max_height_vertices(int s, int t) {
      vi max_height;
      for (int i = 0; i < n; i++)
        if (i != s && i != t && excess[i] > 0) {
          if (!max_height.empty() && height[i] > height[
             max_height[0]])
            max_height.clear();
          if (max_height.empty() || height[i] == height[
             max_height[0]])
            max_height.push_back(i);
        }
      return max_height;
    }

    ll maxFlow(){
      height.assign(n,0); excess.assign(n,0);
      ll max_flow = 0; bool pushed;
      vi current;

      height[s] = n; excess[s] = inf;
      for(edge &e: g[s])
          push(s,e);

      while(!(current = find_max_height_vertices(s,t)).
          empty()){
        for(int v: current){
          pushed = false;
          if(excess[v]==0) continue;
          for(edge &e : g[v]){
            if(e.cap - e.f>0 && height[v]== height[e.to]+1)
               {
              pushed = true;
              push(v,e);
            }
          }
          if(!pushed){
            relabel(v);
            break;
          }
        }
      }
      for (edge e : g[t]){
        edge rev = g[e.to][e.rev];
        max_flow += rev.f;
      }
      return max_flow;
    }
```

```
};
```

4.4 Konig

```
#define sz(c) ((int)c.size())
// asume que el dinic YA ESTA tirado
// asume que nodes-1 y nodes-2 son la fuente y destino
int match[maxnodes]; // match[v]=u si u-v esta en el
    matching, -1 si v no esta matcheado
int s[maxnodes]; // numero de la bfs del koning
queue<int> kq;
// s[e]%2==1  o si e esta en V1 y s[e]==-1-> lo agarras
void konig() {//O(n)
        forn(v,nodes-2) s[v] = match[v] = -1;
        forn(v,nodes-2)
        for(edge it: g[v])
            if (it.to < nodes-2 && it.f>0){
                match[v]=it.to; match[it.to]=v;
            }
        forn(v,nodes-2)
        if (match[v]==-1){
            s[v]=0;kq.push(v);
        }
        while(!kq.empty()) {
                int e = kq.front(); kq.pop();
                if (s[e]%2==1) {
                        s[match[e]] = s[e]+1;
                        kq.push(match[e]);
                } else {
                        for(edge it: g[e])
                if (it.to < nodes-2 && s[it.to]==-1){
                    s[it->to] = s[e]+1;
                    kq.push(it->to);
                }
                }
        }
}
```

4.5 MCBM Augmenting Algorithm

```
// O (V*E)
//Sacado del Vasito
vector<int> g[MAXN]; // [0,n)->[0,m)
int n,m;
int mat[MAXM];bool vis[MAXN];
int match(int x){
        if(vis[x])return 0;
        vis[x]=true;
        for(int y:g[x])if(mat[y]<0||match(mat[y])){mat[y
            ]=x;return 1;}
        return 0;
}
vector<pair<int,int> > max_matching(){
        vector<pair<int,int> > r;
        memset(mat,-1,sizeof(mat));
        fore(i,0,n)memset(vis,false,sizeof(vis)),match(i)
            ;
        fore(i,0,m)if(mat[i]>=0)r.pb({mat[i],i});
        return r;
}
```

4.6 Hungarian Algorithm

```
const ld inf = 1e18; // To Maximize set "inf" to 0, and
    negate costs
inline bool zero(ld x){ return x == 0; } // For Integer/
    LL --> change to x == 0
struct Hungarian{
  int n; vector<vd> c;
  vi l, r, p, sn;  vd ds, u, v;
  Hungarian(int n): n(n), c(n, vd(n, inf)), l(n, -1), r(n
      , -1), p(n), sn(n), ds(n), u(n), v(n){}
  void set_cost(){ forn(i, n) forn(j, n) cin >> c[i][j];
    }
        ld assign() {
    set_cost();
                forn(i, n) u[i] = *min_element(all(c[i]))
                    ;
                forn(j, n){
      v[j] = c[0][j] - u[0];
      for1(i, n-1) v[j] = min(v[j], c[i][j] - u[i]);
    }
                int mat = 0;
                forn(i, n) forn(j, n) if(r[j] == -1 &&
                    zero(c[i][j] - u[i]  -v[j])){
      l[i] = j,  r[j] = i, ++mat; break;
    }
                for(; mat < n; ++mat){
      int s = 0, j = 0, i;
      while(l[s] != -1) ++s;
      forn(k, n) ds[k] = c[s][k] - u[s] - v[k];
      fill(all(p), -1),  fill(all(sn), 0);
      while(1){
        j = -1;
        forn(k, n) if(!sn[k] && (j == -1 || ds[k] < ds[j
            ])) j = k;
        sn[j] = 1,  i = r[j];
        if(i == -1) break;
        forn(k, n) if(!sn[k]){
          auto n_ds = ds[j] + c[i][k] - u[i] - v[k];
          if(ds[k] > n_ds) ds[k] = n_ds,  p[k] = j;
        }
      }
      forn(k, n) if(k != j && sn[k]){
        auto dif = ds[k] - ds[j];
```

```
            v[k] += dif, u[r[k]] -= dif;
          }
          u[s] += ds[j];
          while(p[j] >= 0) r[j] = r[p[j]],  l[r[j]] = j,  j =
              p[j];
          r[j] = s,  l[s] = j;
                    }
                    ld val = 0;
      forn(i, n) val += c[i][l[i]];
                    return val;
            }
    void print_assignment(){ forn(i, n) cout << i+1 << " "
        << l[i]+1 << el; }
  };
```

4.7 Min-Cost Max-Flow Algorithm

```
  const ll inf = 1e18;
  struct edge{
    int to, rev; ll cap, cos, f{0};
    edge(int to, int rev, ll cap, ll cos):to(to), rev(rev),
        cap(cap), cos(cos){}
  };
  struct MCMF{
    int n, s, t;
    vector<vector<edge>> g;
    vi p;  vll dis;
    MCMF(int n): n(n), g(n){}
    void addEdge(int s, int t, ll cap, ll cos){
      g[s].pb(edge(t, sz(g[t]), cap, cos));
      g[t].pb(edge(s, sz(g[s])-1, 0, -cos));
    }
    void spfa(int v0){
      dis.assign(n, inf); dis[v0] = 0;
      p.assign(n, -1);
      vector<bool> inq(n);
      queue<int> q({v0});
      while(sz(q)){
        int u = q.front(); q.pop();
        inq[u] = 0;
        for(auto&[v, rev, cap, cos, f] : g[u]){
          if(cap - f > 0 && dis[v] > dis[u] + cos){
            dis[v] = dis[u] + cos, p[v] = rev;
            if(!inq[v]) inq[v] = 1, q.push(v);
          }
        }
      }
    }
    ll min_cos_flow(ll K){
      ll flow = 0, cost = 0;
      while(flow < K){
        spfa(s);
        if(dis[t] == inf) break;
        ll f = K - flow;
        int cur = t; // Find flow
        while(cur != s){
          int u = g[cur][p[cur]].to, rev = g[cur][p[cur]].
              rev;
          f = min(f, g[u][rev].cap - g[u][rev].f);
          cur = u;
        }
        flow += f,  cost += f * dis[t],  cur = t;     //
            Apply flow
        while(cur != s){
          int u = g[cur][p[cur]].to, rev = g[cur][p[cur]].
              rev;
          g[u][rev].f += f,  g[cur][p[cur]].f -= f;
          cur = u;
        }
      }
      if(flow < K) assert(0);
      return cost;
    }
  };
```

4.8 Min-Cost Max-Flow Algorithm 2

```
  typedef ll tf;
  typedef ll tc;
  const tf INFFLOW=1e9;
  const tc INFCOST=1e9;
  struct MCF{
          int n;
          vector<tc> prio, pot; vector<tf> curflow; vector<
              int> prevedge,prevnode;
          priority_queue<pair<tc, int>, vector<pair<tc, int
              >>, greater<pair<tc, int>>> q;
          struct edge{int to, rev; tf f, cap; tc cost;};
          vector<vector<edge>> g;
          MCF(int n):n(n),prio(n),curflow(n),prevedge(n),
              prevnode(n),pot(n),g(n){}
          void add_edge(int s, int t, tf cap, tc cost) {
                  g[s].pb((edge){t,sz(g[t]),0,cap,cost});
                  g[t].pb((edge){s,sz(g[s])-1,0,0,-cost});
          }
          pair<tf,tc> get_flow(int s, int t) {
                  tf flow=0; tc flowcost=0;
                  while(1){
                          q.push({0, s});
                          fill(all(prio),INFCOST);
                          prio[s]=0; curflow[s]=INFFLOW;
                          tc d; int u;
                          while(sz(q)){
                                  tie(d,u)=q.top(); q.pop()
                                      ;
                                  if(d!=prio[u]) continue;
                                  forn(i,sz(g[u])) {
                                          edge &e=g[u][i];
```

```
                                        int v=e.to;
                                        if(e.cap<=e.f)
                                            continue;
                                        tc nprio=prio[u]+
                                            e.cost+pot[u]-
                                            pot[v];
                                        if(prio[v]>nprio)
                                             {
                                                prio[v]=
                                                    nprio;
                                                q.push({
                                                    nprio,
                                                     v});
                                                prevnode[
                                                    v]=u;
                                                    prevedge
                                                    [v]=i;
                                                curflow[v
                                                    ]=min(
                                                    curflow
                                                    [u], e
                                                    .cap-e
                                                    .f);
                                        }
                                }
                        }
                        if(prio[t]==INFCOST) break;
                        forn(i,n) pot[i]+=prio[i];
                        tf df=min(curflow[t], INFFLOW-
                            flow);
                        flow+=df;
                        for(int v=t; v!=s; v=prevnode[v])
                             {
                                edge &e=g[prevnode[v]][
                                    prevedge[v]];
                                e.f+=df; g[v][e.rev].f-=
                                    df;
                                flowcost+=df*e.cost;
                        }
                }
                return {flow,flowcost};
        }
};
```

4.9 Blossom

```
/// Complexity: O(|E||V|^2)
/// Tested: https://tinyurl.com/oe5rnpk
/// Max matching undirected graph
struct network {
  struct struct_edge { int v; struct_edge * n; };
  typedef struct_edge* edge;
  int n;
  struct_edge pool[MAXE]; ///2*n*n;
  edge top;
  vector<edge> adj;
  queue<int> q;
  vector<int> f, base, inq, inb, inp, match;
  vector<vector<int>> ed;
  network(int n) : n(n), match(n, -1), adj(n), top(pool),
      f(n), base(n),
                   inq(n), inb(n), inp(n), ed(n, vector<
                       int>(n)) {}
  void add_edge(int u, int v) {
    if(ed[u][v]) return;
    ed[u][v] = 1;
    top->v = v, top->n = adj[u], adj[u] = top++;
    top->v = u, top->n = adj[v], adj[v] = top++;
  }
  int get_lca(int root, int u, int v) {
    fill(inp.begin(), inp.end(), 0);
    while(1) {
      inp[u = base[u]] = 1;
      if(u == root) break;
      u = f[ match[u] ];
    }
    while(1) {
      if(inp[v = base[v]]) return v;
      else v = f[ match[v] ];
    }
  }
  void mark(int lca, int u) {
    while(base[u] != lca) {
      int v = match[u];
      inb[ base[u ]] = 1;
      inb[ base[v] ] = 1;
      u = f[v];
      if(base[u] != lca) f[u] = v;
    }
  }
  void blossom_contraction(int s, int u, int v) {
    int lca = get_lca(s, u, v);
    fill(all(inb), 0);
    mark(lca, u); mark(lca, v);
    if(base[u] != lca) f[u] = v;
    if(base[v] != lca) f[v] = u;
    forn(u,n){
      if(inb[base[u]]) {
        base[u] = lca;
        if(!inq[u]) {
            inq[u] = 1;
            q.push(u);
        }
      }
    }
  }
  int bfs(int s) {
    fill(all(inq), 0);
    fill(all(f), -1);
```

```
    for(int i = 0; i < n; i++) base[i] = i;
    q = queue<int>();
    q.push(s);
    inq[s] = 1;
   while(sz(q)) {
      int u = q.front(); q.pop();
      for(edge e = adj[u]; e; e = e->n) {
        int v = e->v;
        if(base[u] != base[v] && match[u] != v) {
          if((v == s) || (match[v] != -1 && f[match[v]]
              != -1))
            blossom_contraction(s, u, v);
          else if(f[v] == -1) {
            f[v] = u;
            if(match[v] == -1) return v;
            else if(!inq[match[v]]) {
              inq[match[v]] = 1;
              q.push(match[v]);
            }
          }
        }
      }
    }
    return -1;
  }
  int doit(int u) {
    if(u == -1) return 0;
    int v = f[u];
    doit(match[v]);
    match[v] = u; match[u] = v;
    return u != -1;
  }
  /// (i < net.match[i]) => means match
  int maximum_matching() {
    int ans = 0;
    forn(u,n)
      ans += (match[u] == -1) && doit(bfs(u));
    return ans;
  }
};
```

5 Data Structures

5.1 Disjoint Set Union

```
struct dsu{
  vi p, r; int comp;
  dsu(int n): p(n), r(n, 1), comp(n){iota(all(p), 0);}
  int find_set(int i){return p[i] == i ? i : p[i] =
      find_set(p[i]);}
  bool is_same_set(int i, int j){return find_set(i) ==
      find_set(j);}
  void union_set(int i, int j){
    if((i = find_set(i)) == (j = find_set(j))) return;
    if(r[i] > r[j]) swap(i, j);
    r[j] += r[i];  r[i] = 0;
    p[i] = j;  --comp;
  }
};
```

5.2 SQRT Decomposition

```
// Complexity:      1. Preprocessing O(n)
// 2. Update O(1)   3. Query O(n/sqrt(n) + sqrt(n))
struct sqrt_decomp{
  int n, len;  vi a, b;
  sqrt_decomp(){}
  sqrt_decomp(vi& arr): n(sz(arr)), len(sqrt(n) + 1), a(
      arr), b(len){
    forn(i, n) b[i / len] += a[i];
  }
  void update(int pos, int val){
    b[pos / len] += val - a[pos];  // Block update
    a[pos] = val;                  // Point update
  }
  int query(int l, int r){
    int sum = 0,  b_l = l / len,  b_r = r / len;
    if(b_l == b_r) fore(i,l,r) sum += a[i]; // L, R in
        same block
    else{
      fore(i, l, len*(b_l+1) - 1) sum += a[i];  // Left
          Tail (Points)
      fore(i, len*b_r, r) sum += a[i];    // Right Tail (
          Points)
      fore(i, b_l+1, b_r-1) sum += b[i];  // Block query
    }
    return sum;
  }
};
```

5.3 Fenwick Tree

```
struct fwtree{ // 0-indexed
  int n;  vi bit;
  fwtree(int n): n(n), bit(n+1){}
  int rsq(int r){ // [0, r]
    int sum = 0;
    for(++r; r; r -= r & -r) sum += bit[r];
    return sum;
  }
  int rsq(int l, int r){return rsq(r) - (l==0 ? 0 : rsq(l
      -1));}
  void upd(int r, int v){
    for(++r; r <= n; r += r & -r) bit[r] += v;
```

```
    }
  };
```

5.4 Fenwick Tree 2D

```
struct fwtree{ // 0-indexed
  int n, m;  vector<vll> bit;
  fwtree(){}
  fwtree(int n, int m): n(n), m(m), bit(n+1, vll(m+1, 0))
     {}
  ll sum(int x, int y) { // [0, x], [0, y]
    ll v = 0;
    for(int i = x+1; i; i -= i & -i)
      for(int j = y+1; j; j -= j & -j) v += bit[i][j];
    return v;
  }
  void add(int x, int y, ll dt) {
    for(int i = x+1; i <= n; i += i & -i)
      for(int j = y+1; j <= m; j += j & -j) bit[i][j] +=
         dt;
  }
};
```

5.5 Segment Tree

```
struct stree{
  int neutro = 1e9, n, l, r, pos, val;  vi t;
  stree(int n): n(n), t(n << 2){}
  stree(const vi& a): n(sz(a)), t(n<<2){ build(1, 0, n-1,
      a); }
  inline int oper(int a, int b){ return a < b ? a : b; }
  void build(int v, int tl, int tr, const vi& a){// solo
     para el 2. constructor
    if(tl == tr){ t[v] = a[tl];  return; }
    int tm = (tl + tr) >> 1;
    build(v << 1, tl, tm, a), build((v << 1) | 1, tm+1,
       tr, a);
    t[v] = oper(t[v << 1], t[(v << 1) | 1]);
  }
  int query(int v, int tl, int tr){
    if(tl > r || tr < l) return neutro;  // estoy fuera
    if(l <= tl && tr <= r) return t[v];
    int tm = (tl + tr) >> 1;
    return oper(query(v << 1, tl, tm), query((v << 1) |
       1, tm+1, tr));
  }
  void upd(int v, int tl, int tr){
    if(tl == tr){ t[v] = val;  return; }
    int tm = (tl + tr) >> 1;
    if(pos <= tm) upd(v << 1, tl, tm);
    else upd((v << 1) | 1, tm+1, tr);
    t[v] = oper(t[v << 1], t[(v << 1) | 1]);
  }
  void upd(int idx, int num){ pos = idx, val = num, upd
     (1, 0, n-1);}
  int query(int ql, int qr){ l = ql, r = qr;  return
     query(1, 0, n-1);}
};
```

5.6 ST Lazy Propagation

```
const int N = 1e5 + 10;
int t[N << 2], lazy[N << 2];
struct stree{
  int n, l, r, val, neutro = 0;
  stree(int n): n(n){ forn(i, n << 2) t[i] = lazy[i] = 0;
     }
  stree(vector<int> &a){ n = sz(a); forn(i, n << 2) t[i]
     = lazy[i] = 0;
    build(1, 0, n-1, a);
  }
  inline int oper(int a, int b){ return a > b ? a : b; }
  inline void push(int v){
    if(lazy[v]){
      t[v << 1] += lazy[v];  lazy[v << 1] += lazy[v];
      t[(v << 1) | 1] += lazy[v];  lazy[(v << 1) | 1] +=
         lazy[v];
      lazy[v] = 0;
    }
  }
  void build(int v, int tl, int tr, vi& a){
    if(tl == tr){
      t[v] = a[tl]; return;
    }
    int tm = (tl + tr) >> 1;
    build(v << 1, tl, tm, a), build((v << 1) | 1, tm+1,
       tr, a);
    t[v] = oper(t[v << 1], t[(v << 1) | 1]);
  }
  void upd(int v, int tl, int tr){
    if(tl > r || tr < l) return;
    if(l <= tl && tr <= r){
      t[v] += val;  lazy[v] += val;
      return ;
    }
    push(v);  int tm = (tl + tr) >> 1;
    upd(v << 1, tl, tm);  upd((v << 1) | 1, tm+1, tr);
    t[v] = oper(t[v << 1], t[(v << 1) | 1]);
  }
  int query(int v, int tl, int tr){
    if(tl > r || tr < l) return neutro;
    if(l <= tl && tr <= r) return t[v];
    push(v);  int tm = (tl + tr) >> 1;
```

```
    return oper(query(v << 1, tl, tm), query((v << 1) |
        1, tm + 1, tr));
  }
  void update(int ql, int qr, int qval){
    l = ql, r = qr, val = qval, upd(1, 0, n-1); }
  int query(int ql, int qr){ l = ql, r = qr;  return
      query(1, 0, n-1); }
};
```

5.7 Persistent ST

```
const int len = 1e7, neutro = 1e9;
struct node{ int mn, l, r; };
struct stree{
  vi rts{0};  vector<node> t;
  int n, idx{0}, l, r, pos, val;
  inline int oper(int a, int b){ return a < b ? a : b; }
  stree(const vi &a): n(sz(a)), t(len){ build(0, n-1, a);
      }
  int build(int tl, int tr, const vi &a){
    int v = idx++;
    if(tl == tr){ t[v].mn = a[tl]; return v; }
    int tm = (tl + tr) >> 1;
    t[v].l = build(tl, tm, a),  t[v].r = build(tm + 1, tr
        , a);
    t[v].mn = oper(t[t[v].l].mn, t[t[v].r].mn);
    return v;
  }
  int que(int v, int tl, int tr){
    if(tl > r || tr < l) return neutro;
    if(l <= tl && tr <= r) return t[v].mn;
    int tm = (tl + tr) >> 1;
    return oper(que(t[v].l, tl, tm), que(t[v].r, tm + 1,
        tr));
  }
  int upd(int prv, int tl, int tr){
    int v = idx++;
    t[v] = t[prv];
    if(tl == tr){  t[v].mn = val;  return v; }
    int tm = (tl + tr) >> 1;
    if(pos <= tm) t[v].l = upd(t[v].l, tl, tm);
    else t[v].r = upd(t[v].r, tm + 1, tr);
    t[v].mn = oper(t[t[v].l].mn, t[t[v].r].mn);
    return v;
  }
  int query(int v, int cl, int cr){ l = cl, r = cr;
      return que(v, 0, n-1); }
  void upd(int i, int x){ pos = i, val = x, rts.pb(upd(
      rts.back(), 0, n-1)); }
};
```

5.8 Segtree 2D

```
const int N = 2500 + 1;
ll st[2*N][2*N];
struct stree{
  int n, m, neutro = 0;
  stree(int n, int m): n(n), m(m){ forn(i, 2*n) forn(j,
      2*m) st[i][j] = neutro; }
  stree(vector<vi> a): n(sz(a)), m(n ? sz(a[0]) : 0){
      build(a); }
  inline ll op(ll a, ll b){ return a+b; }
  void build(vector<vi>& a){
    forn(i, n) forn(j, m) st[i+n][j+m] = a[i][j];
    forn(i, n) fored(j, 1, m-1) st[i+n][j] = op(st[i+n][j
        <<1], st[i+n][j<<1|1]);
    fored(i, 1, n-1) forn(j, 2*m) st[i][j] = op(st[i<<1][
        j], st[i<<1|1][j]);
  }
  void upd(int x, int y, ll 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]);
  }
  ll query(int x0, int x1, int y0, int y1){ // [x0, x1],
      [y0, y1]
    ll r = neutro;
    for(int i0 = x0+n, i1 = x1+n+1; i0 < i1; i0 >>= 1, i1
        >>= 1){
      int t[4], q=0;
      if(i0&1) t[q++] = i0++;
      if(i1&1) t[q++] = --i1;
      forn(k, q) for(int j0 = y0+m, j1 = y1+m+1; 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;
  }
};
```

5.9 Segtree iterativo

```
const int N = 1e5;  // limit for array size
int t[2 * N];
struct stree{
  int n, neutro = 1e9;
  stree(int n): n(n){ forn(i, 2*n) t[i] = neutro; }
  stree(vi a): n(sz(a)){ build(a); }
  inline int op(int a, int b){ return min(a, b); }
  void build(vi& a) {
    forn(i, n) t[n + i] = a[i];
```

```
    fored(i, 1, n-1) t[i] = op(t[i<<1], t[i<<1|1]);
  }
  int query(int l, int r) {  // [l, r]
    int vl = neutro, vr = neutro;
    for(l += n, r += n+1; l < r; l >>= 1, r >>= 1) {
      if(l&1) vl = op(vl, t[l++]);
      if(r&1) vr = op(t[--r], vr);
    }
    return op(vl, vr);
  }
  void upd(int p, int val) {  // set val at position p (0 - idx)
    for (t[p += n] = val; p > 1; p >>= 1) t[p>>1] = op(t[p], t[p^1]);
  }
};
```

5.10 RMQ

```
const int N = 1e5 + 10, K = 20;  //K  has to satisfy K> log nax + 1
ll st[N][K];
struct RMQ{
  ll neutro = inf;
  inline ll oper(ll a, ll b){ return a < b ? a : b; }
  RMQ(vi& a){
    forn(i, sz(a)) st[i][0] = a[i];
    for1(j, K-1)
      forn(i, sz(a) - (1 << j) + 1)
        st[i][j] = oper(st[i][j-1], st[i + (1 << (j-1))][j-1]);
  }
  ll query(int l, int r){
    if(l > r) return neutro;
    int j = 31 - __builtin_clz(r-l+1);
    return oper(st[l][j], st[r - (1 << j) + 1][j]);
  }
};
```

5.11 Sack

```
// Time Complexity O(N*log(N))
int timer;
int cnt[nax], big[nax], fr[nax], to[nax], who[nax];
vector<int> g[nax];
int pre(int u, int p){
  int sz = 1, tmp;
  who[timer] = u;
  fr[u] = timer++;
  ii best = {-1, -1};
  for(int v: g[u]){
    if(v==p) continue;
    tmp = pre(v,u);
    sz+=tmp;
    best = max(best, {tmp, v});
  }
  big[u] = best.se;
  to[u] = timer-1;
  return sz;
}
void add(int u, int x) { /// x == 1 add, x == -1 delete
  cnt[u] += x;
}
void dfs(int u, int p, bool keep = true){
  for(int v: g[u])
    if(v!=p && v!=big[u])
      dfs(v,u, 0);
  if(big[u]!=-1) dfs(big[u], u);
  /// add all small
  for(int v: g[u])
    if(v!=p && v!=big[u])
      for(int i = fr[v]; i<= to[v]; ++i)
        add(who[i],1);
  add(u,1);
  /// Answer queries
  if(!keep)
    for(int i = fr[u]; i<= to[u]; ++i)
      add(who[i],-1);
}
void solve(int root){
  timer = 0;
  pre(root, root);
  dfs(root, root);
}
```

5.12 Heavy Light Decomposition

```
vector<int> g[nax];
int len[nax], dep[nax], in[nax], out[nax], head[nax], par[nax], idx;
void dfs_sz( int u, int d ) {
  dep[u] = d;
  int &sz = len[u]; sz = 1;
  for( auto &v : g[u] ) {
    if( v == par[u] ) continue;
    par[v] = u; dfs_sz(v, d+1);
    sz += len[v];
    if(len[ g[u][0] ] < len[v]) swap(g[u][0], v);
  }
  return ;
}
void dfs_hld( int u) {
  in[u] = idx++;
  arr[in[u]] = val[u]; /// to initialize the segment tree
  for( auto& v : g[u] ) {
```

```
      if( v == par[u] ) continue;
      head[v] = (v == g[u][0] ? head[u] : v);
      dfs_hld(v);
    }
    out[u] = idx-1;
  }
  void upd_hld( int u, int val ) {
    upd_DS(in[u], val);
  }
  int query_hld( int u, int v ) {
    int val = neutro;
    while( head[u] != head[v] ) {
      if( dep[ head[u] ] < dep[ head[v] ] ) swap(u, v);
      val = val + query_DS(in[ head[u] ], in[u]);
      u = par[ head[u] ];
    }
    if( dep[u] > dep[v] ) swap(u, v);
    val = val+query_DS(in[u], in[v]);
    return val;
  /// when updates are on edges use: (line 36)
  ///   if (dep[u] == dep[v]) return val;
  ///   val = val+query_DS(in[u] + 1, in[v]);
  }
  void build(int root) {
    idx = 0; /// DS index [0, n)
    par[root] = head[root] = root;
    dfs_sz(root, 0);
    dfs_hld(root);
    /// initialize DS
  }
```

5.13 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) {}
};
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);
}
pitem kth(pitem t, int k){
        if(!t)return 0;
        if(k==cnt(t->l))return t;
        return k<cnt(t->l)?kth(t->l,k):kth(t->r,k-cnt(t->
            l)-1);
}
pair<int,int> lb(pitem t, int key){ // position and value
    of lower_bound
        if(!t)return {0,1<<30}; // (special value)
        if(key>t->key){
                auto w=lb(t->r,key);w.fst+=cnt(t->l)+1;
                    return w;
        }
        auto w=lb(t->l,key);
        if(w.fst==cnt(t->l))w.snd=t->key;
        return w;
}
```

5.14 Implicit Treap

```
// example that supports range reverse and addition
    updates, and range sum query
// (commented parts are specific to this  problem)
typedef struct item* pitem;
struct item {
        int pr,cnt,val;
//      int sum; // (paramters for range query)
//      bool rev;int add; // (parameters for lazy prop)
        pitem l,r;
        item(int val): pr(rand()),cnt(1),val(val),l(0),r
            (0)/*,sum(val),rev(0),add(0)*/ {}
};
```

```
void push(pitem it){
	if(it){
		/*if(it->rev){
			swap(it->l,it->r);
			if(it->l)it->l->rev^=true;
			if(it->r)it->r->rev^=true;
			it->rev=false;
		}
		it->val+=it->add;it->sum+=it->cnt*it->add
		    ;
		if(it->l)it->l->add+=it->add;
		if(it->r)it->r->add+=it->add;
		it->add=0;*/
	}
}
int cnt(pitem t){return t?t->cnt:0;}
// int sum(pitem t){return t?push(t),t->sum:0;}
void upd_cnt(pitem t){
	if(t){
		t->cnt=cnt(t->l)+cnt(t->r)+1;
		// t->sum=t->val+sum(t->l)+sum(t->r);
	}
}
void merge(pitem& t, pitem l, pitem r){
	push(l);push(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 split(pitem t, pitem& l, pitem& r, int sz){ // sz:
    desired size of l
	if(!t){l=r=0;return;}
	push(t);
	if(sz<=cnt(t->l))split(t->l,l,t->l,sz),r=t;
	else split(t->r,t->r,r,sz-1-cnt(t->l)),l=t;
	upd_cnt(t);
}
void output(pitem t){ // useful for debugging
	if(!t)return;
	push(t);
	output(t->l);printf(" %d",t->val);output(t->r);
}
// use merge and split for range updates and queries
```

5.15 Implicit Treap Father

```
// node father is useful to keep track of the chain of
    each node
// alternative: splay tree
// IMPORTANT: add pointer f in struct item
void merge(pitem& t, pitem l, pitem r){
	push(l);push(r);
	if(!l||!r)t=l?l:r;
	else if(l->pr>r->pr)merge(l->r,l->r,r),l->r->f=t=
	    l;
	else merge(r->l,l,r->l),r->l->f=t=r;
	upd_cnt(t);
}
void split(pitem t, pitem& l, pitem& r, int sz){
	if(!t){l=r=0;return;}
	push(t);
	if(sz<=cnt(t->l)){
		split(t->l,l,t->l,sz);r=t;
		if(l)l->f=0;
		if(t->l)t->l->f=t;
	}
	else {
		split(t->r,t->r,r,sz-1-cnt(t->l));l=t;
		if(r)r->f=0;
		if(t->r)t->r->f=t;
	}
	upd_cnt(t);
}
void push_all(pitem t){
	if(t->f)push_all(t->f);
	push(t);
}
pitem root(pitem t, int& pos){ // get root and position
    for node t
	push_all(t);
	pos=cnt(t->l);
	while(t->f){
		pitem f=t->f;
		if(t==f->r)pos+=cnt(f->l)+1;
		t=f;
	}
	return t;
}
```

5.16 Ordered Set

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
// -------- CONSTRUCTOR -------- //
// 1. Para ordenar por MAX cambiar less<int> por greater<
    int>
// 2. Para multiset cambiar less<int> por less_equal<int>
//      Para borrar siendo multiset:
//      int idx = st.order_of_key(value);
//      st.erase(st.find_by_order(idx));
// -------- METHODS --------- //
st.find_by_order(k) // returns pointer to the k-th
    smallest element
```

```
st.order_of_key(x)  // returns how many elements are
    smaller than x
st.find_by_order(k) == st.end() // true, if element does
    not exist
```

5.17 Mo's Algorithm

```
/// Complexity: O(|N+Q|*sqrt(|N|)*|ADD/DEL|)
/// Requires add(), delete() and get_ans()
struct query {
  int l, r, idx;
};
int S; // s = sqrt(n)
bool cmp (query a, query b) {
  int x = a.l/S;
  if (x != b.l/S) return x < b.l/S;
  return (x&1 ? a.r < b.r : a.r > b.r);
}
void solve(){
  S = sqrt(n); // n = size of array
  sort(all(q), cmp);
  int l = 0, r = -1;
  forn(i, sz(q)){
    while (r < q[i].r) add(++r);
    while (l > q[i].l) add(--l);
    while (r > q[i].r) del(r--);
    while (l < q[i].l) del(l++);
    ans[q[i].idx] = get_ans();
  }
}
```

5.18 Dynamic Connectivity

```
struct dsu {
        vi p, r, c;  int comp;
        dsu(int n): p(n), r(n, 1), comp(n){iota(all(p),
            0);}
        int find_set(int i){return i == p[i] ? i :
            find_set(p[i]);}
        void union_set(int i, int j){
                if((i = find_set(i)) == (j = find_set(j))
                    ) return;
                if(r[i] > r[j]) swap(i, j);
    r[j] += r[i];  c.pb(i);
                p[i] = j;  --comp;
        }
        void rollback(int snap){
                while(sz(c) > snap){
                        int x = c.back(); c.pop_back();
                        r[p[x]] -= r[x];  p[x] = x;  ++
                            comp;
                }
        }
};
enum {ADD, DEL, QUERY};
struct Query {int type, u, v;};
struct DynCon {
        vector<Query> q;  dsu uf;
        vi mt;  map<ii, int> prv;
        DynCon(int n): uf(n){}
        void add(int i, int j){
                if(i > j) swap(i, j);
                q.pb({ADD, i, j}); mt.pb(-1);
    prv[{i, j}] = sz(q)-1;
        }
        void remove(int i, int j){
                if(i > j) swap(i, j);
                q.pb({DEL, i, j});
                int pr = prv[{i, j}];
                mt[pr] = sz(q)-1;  mt.pb(pr);
        }
        void query(){ q.pb({QUERY, -1, -1});  mt.pb(-1);}
        void process(){ // answers all queries in order
                if(!sz(q)) return;
                forn(i, sz(q)) if(q[i].type == ADD && mt[
                    i] < 0) mt[i] = sz(q);
                go(0, sz(q));
        }
        void go(int s, int e){
                if(s+1 == e){
                        if(q[s].type == QUERY) cout << uf
                            .comp << el;
                        return;
                }
                int k = sz(uf.c), m = (s+e)/2;
                fored(i, m, e-1) if(mt[i] >= 0 && mt[i] <
                     s) uf.union_set(q[i].u, q[i].v);
                go(s, m);  uf.rollback(k);
    fored(i, s, m-1) if(mt[i] >= e) uf.union_set(q[i].u,
        q[i].v);
                go(m, e);  uf.rollback(k);
        }
};
```

5.19 Link Cut Tree

```
struct Node { // Splay tree. Root's pp contains tree's
    parent.
  Node *p = 0, *pp = 0, *c[2] = {0, 0};
  bool flip = 0;
  Node(){}
  void fix() { forn(i, 2) if(c[i]) c[i]->p = this; }
  inline int up() { return p ? p->c[1] == this : -1; }
  void push() {
    if (!flip) return;
```

```
      flip = 0,  swap(c[0], c[1]);
      forn(i, 2) if(c[i]) c[i]->flip ^= 1;
    }
  void rot(int i, int b) {
    int h = i ^ b;
    Node *x = c[i],  *y = b == 2 ? x : x->c[h],  *z = b ?
        y : x;
    if ((y->p = p)) p->c[up()] = y;
    c[i] = z->c[i^1];
    if(b < 2) x->c[h] = y->c[h^1],  z->c[h^1] = b ? x :
        this;
    y->c[i^1] = b ? this : x;
    fix(),  x->fix(),  y->fix();
    if(p) p->fix();
    swap(pp, y->pp);
  }
  void splay() { // Splay *this up to the root. Finishes
      without flip set.
    for(push(); p; ) {
      if(p->p) p->p->push();
      p->push(),  push();
      int c1 = up(),  c2 = p->up();
      if(c2 == -1) p->rot(c1, 2);
      else p->p->rot(c2, c1 != c2);
    }
  }
  Node* first() { return push(), c[0] ? c[0]->first() : (
      splay(), this); }
}; // Return the MIN of the subtree rooted at this,
    splayed to the top.

struct LinkCut {
  vector<Node> node;
  LinkCut(int n) : node(n) {}
  Node* get(Node* u) { /// Move u to root aux tree.
    u->splay();
    while (Node* pp = u->pp) {
      pp->splay(),  u->pp = 0;
      if(pp->c[1]) pp->c[1]->p = 0,  pp->c[1]->pp = pp;
      pp->c[1] = u,  pp->fix(),  u = pp;
    }
    return u;  // Return the root of the root aux tree.
  }
  bool connected(int u, int v) { // are u, v in the same
      tree?
    return get(&node[u])->first() == get(&node[v])->first
        ();
  }
  void link(int u, int v) { // add an edge (u, v)
    assert(!connected(u, v)),  makeRoot(&node[u]),  node[
        u].pp = &node[v];
  }
  void cut(int u, int v) { // remove an edge (u, v)
    Node *x = &node[u], *top = &node[v];
    makeRoot(top),  x->splay(),  assert(top == (x->pp ? :
          x->c[0]));
    if(x->pp) x->pp = 0;
    else x->c[0] = top->p = 0,  x->fix();
  }
  void makeRoot(Node* u) { /// Move u to root of
      represented tree.
    get(u),  u->splay();
    if(u->c[0]) {
      u->c[0]->p = 0;
      u->c[0]->flip ^= 1;
      u->c[0]->pp = u;
      u->c[0] = 0,  u->fix();
    }
  }
};
```

6 Math

6.1 Sieve of Eratosthenes

```
// O(n)
// pr contains prime numbers
// lp[i] == i if i is prime
// else lp[i] is minimum prime factor of i
const int nax = 1e7;
int lp[nax+1];
vector<int> pr; // It can be sped up if change for an
    array

void sieve(){
  fore(i,2,nax-1){
    if (lp[i] == 0) {
      lp[i] = i; pr.pb(i);
    }
    for (int j=0, mult= i*pr[j]; j<sz(pr) && pr[j]<=lp[i]
        && mult<nax; ++j, mult= i*pr[j])
      lp[mult] = pr[j];
  }
}
```

6.2 Count primes

```
int count_primes(int n) {
  const int S = 10000;
  vector<int> primes;
  int nsqrt = sqrt(n);
  vector<char> is_prime(nsqrt + 1, true);
  fore(i,2,nsqrt){
    if (is_prime[i]) {
      primes.pb(i);
      for (int j = i * i; j <= nsqrt; j += i)
```

```
        is_prime[j] = false;
    }
  }

  int result = 0;
  vector<char> block(S);
  for (int k = 0; k * S <= n; k++) {
    fill(all(block), true);
    int start = k * S;
    for (int p : primes) {
      int start_idx = (start + p - 1) / p;
      int j = max(start_idx, p) * p - start;
      for (; j < S; j += p)
        block[j] = false;
    }
    if (k == 0)
      block[0] = block[1] = false;
    for (int i = 0; i < S && start + i <= n; i++) {
      if (block[i])
        result++;
    }
  }
  return result;
}
```

6.3 Segmented Sieve

```
// 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);
  vector<ll> primes;
  for (ll i = 2; i <= lim; ++i) {
    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;
}
```

6.4 Polynomial Multiplication

```
int ans[grado1+grado2+1];

forn(c,grado1+grado2+1) ans[c] = 0;
forn(pos,grado1+1){
  forn(ter,grado2+1)
    ans[pos + ter] += pol1[pos] * pol2[ter];
}
```

6.5 Fast Fourier Transform

```
typedef double ld;
const ld PI = acos(-1.0L);
const ld one = 1;

typedef complex<ld> C;
typedef vector<ld> vd;

void fft(vector<C>& a) {
        int n = sz(a), L = 31 - __builtin_clz(n);
        static vector<complex<ld>> R(2, 1);
        static vector<C> rt(2, 1);  // (^ 10% faster if
           double)
        for (static int k = 2; k < n; k *= 2) {
                R.resize(n); rt.resize(n);
                auto x = polar(one, PI / k);
                fore(i,k,2*k-1) rt[i] = R[i] = i&1 ? R[i
                   /2] * x : R[i/2];
        }
        vi rev(n);
        forn(i,n) rev[i] = (rev[i / 2] | (i & 1) << L) /
           2;
        forn(i,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
        for (int k = 1; k < n; k *= 2)
                for (int i = 0; i < n; i += 2 * k) forn(j
                   ,k) {
                        // C z = rt[j+k] * a[i+j+k]; //
                           (25% faster if hand-rolled)
                           /// include-line
                        auto x = (ld *)&rt[j+k], y = (ld
                           *)&a[i+j+k];         ///
                           exclude-line
                        C z(x[0]*y[0] - x[1]*y[1], x[0]*y
                           [1] + x[1]*y[0]);          //
                           / exclude-line
                        a[i + j + k] = a[i + j] - z;
                        a[i + j] += z;
                }
}

typedef vector<ll> vl;

vl conv(const vl& a, const vl& b) {
        if (a.empty() || b.empty()) return {};
```

```
        vl res(sz(a) + sz(b) - 1);
        int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
        vector<C> in(n), out(n);
        copy(all(a), begin(in));
        forn(i,sz(b)) in[i].imag(b[i]);
        fft(in);
        for (C& x : in) x *= x;
        forn(i,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
        fft(out);
        forn(i,sz(res)) res[i] = floor(imag(out[i]) / (4 * n) +0.5);
        return res;
}

vl convMod(const vl &a, const vl &b, const int &M) {
        if (a.empty() || b.empty()) return {};
        vl res(sz(a) + sz(b) - 1);
        int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
        vector<C> L(n), R(n), outs(n), outl(n);
        forn(i,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
        forn(i,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
        fft(L), fft(R);
        forn(i,n) {
                int j = -i & (n - 1);
                outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
                outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
        }
        fft(outl), fft(outs);
        forn(i,sz(res)) {
                ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
                ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
                res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
        }
        return res;
}
```

6.6 FHT

```
ll c1[MAXN+9],c2[MAXN+9];  // MAXN must be power of 2 !!
void fht(ll* p, int n, bool inv){
        for(int l=1;2*l<=n;l*=2)for(int i=0;i<n;i+=2*l)
            forn(j,l){
                ll u=p[i+j],v=p[i+l+j];
                if(!inv)p[i+j]=u+v,p[i+l+j]=u-v; // XOR
                else p[i+j]=(u+v)/2,p[i+l+j]=(u-v)/2;
                //if(!inv)p[i+j]=v,p[i+l+j]=u+v; // AND
                //else p[i+j]=-u+v,p[i+l+j]=u;
                //if(!inv)p[i+j]=u+v,p[i+l+j]=u; // OR
                //else p[i+j]=v,p[i+l+j]=u-v;
            }
}
// like polynomial multiplication, but XORing exponents
// instead of adding them (also ANDing, ORing)
vector<ll> multiply(vector<ll>& p1, vector<ll>& p2){
        int n=1<<(32-__builtin_clz(max(sz(p1),sz(p2))-1));
        forn(i,n)c1[i]=0,c2[i]=0;
        forn(i,sz(p1))c1[i]=p1[i];
        forn(i,sz(p2))c2[i]=p2[i];
        fht(c1,n,false);fht(c2,n,false);
        forn(i,n)c1[i]*=c2[i];
        fht(c1,n,true);
        return vector<ll>(c1,c1+n);
}
```

6.7 Fibonacci Matrix

```
pll fib_log(ll n, ll mod){
  if (n == 0) return {0, 1};
  auto [a, b] = fib_log(n >> 1, mod);
  ll c = a * (2*b - a + mod) % mod;
  ll d = ((a*a % mod) + (b*b % mod)) % mod;
  if (n & 1) return {d, (c + d) % mod};
  else return {c, d};
}
```

6.8 Matrix Exponentiation

```
struct matrix{ // define N
  int r, c, m[N][N];
  matrix(int r, int c):r(r),c(c){
    memset(m, 0, sizeof m);
  }
  matrix operator *(const matrix &b){
    matrix c = matrix(this->r, b.c);
    forn(i,this->r){
      forn(k,b.r){
        if(!m[i][k]) continue;
        forn(j,b.c){
          c.m[i][j] += m[i][k]*b.m[k][j];
        }
      }
    }
    return c;
  }
};
matrix pow(matrix &b, ll e){
  matrix c = matrix(b.r, b.c);
```

```
    forn(i,b.r) c.m[i][i] = 1;
    while(e){
      if(e&1LL) c = c*b;
      b = b*b , e/=2;
    }
    return c;
  }
```

6.9 Binary Exponentiation

```
int binpow(int b, int e) {
        int ans = 1;
        for (; e; b = 1LL*b*b%mod, e /= 2)
                if (e&1) ans = 1LL*ans*b%mod;
        return ans;
}
```

6.10 Euler's Totient Function

```
int phi(int n) { // O(sqrt(n))
  if(n==1) return 0;
  int ans = n;
  for (int i = 2; 1ll*i*i <= n; i++) {
    if(n % i == 0) {
      while(n % i == 0) n /= i;
      ans -= ans / i;
    }
  }
  if(n > 1) ans -= ans / n;
  return ans;
}
//////////////////
vi phi_(int n) { // O(n loglogn)
  vi phi(n + 1);
  phi[0] = 0;
  for1(i,n) phi[i] = i;
  fore(i,2,n){
    if(phi[i] != i) continue;
    for (int j = i; j <= n; j += i)
      phi[j] -= phi[j] / i;
  }
}

////////// with linear sieve when i is not a prime number
if (lp[i] == lp[i / lp[i]])
  phi[i] = phi[i / lp[i]] * lp[i];
else
  phi[i] = phi[i / lp[i]] * (lp[i] - 1);
```

6.11 Extended Euclidean (Diophantic)

```
// a*x+b*y = g
ll gcde(ll a, ll b, ll& x, ll& y) {
  x = 1, y = 0;
  ll x1 = 0, y1 = 1, a1 = a, b1 = b;
  ll q;
  while (b1) {
    q = a1 / b1;
    tie(x, x1) = make_tuple(x1, x - q * x1);
    tie(y, y1) = make_tuple(y1, y - q * y1);
    tie(a1, b1) = make_tuple(b1, a1 - q * b1);
  }
  return a1;
}

bool find_any_solution(ll a, ll b, ll c, ll &x0, ll &y0,
    ll &g) {
  g = gcde(abs(a), abs(b), x0, y0);
  if (c % g) return false;

  x0 *= c / g;
  y0 *= c / g;
  if (a < 0) x0 = -x0;
  if (b < 0) y0 = -y0;
  return true;
}
```

6.12 Inversa modular

```
// O(mod)
const int mod;
int inv[mod];
void precalc(){
  inv[1] = 1;
  fore(i,2,mod-1) inv[i] = (mod - (mod/i) * inv[mod%i] %
     mod) % mod;
}

/////////////////////////////////////
ll inverse(ll a, ll m){
  ll x, y;
  ll g = gcde(a, m, x, y);
  if (g != 1) {
    cout << "No solution!";
    return -1;
  }else{
    x = (x % m + m) % m;
    return x;
  }
}
```

6.13 Legendre's Formula

```cpp
// Complexity O(log_k (n))
// If k is prime
int fact_pow (int n, int k) {
  int x = 0;
  while(n) {
    n /= k; x += n;
  }
  return x;
}
// If k is composite k = k1^p1 * k2^p2 * ... * km^pm
// min 1..m   ai/ pi  where ai is fact_pow(n, ki)
```

6.14 Mobious

```cpp
int mu[nax], f[nax], h[nax];
void pre(){
  mu[0] = 0; mu[1] = 1;
  for(int i = 1; i<nax; ++i){
    if(mu[i]==0) continue;
    for(int j= i+i; j<nax; j+=i){
      mu[j] -= mu[i];
    }
  }

  for(int i = 1; i < nax; ++i){
    for(int j = i; j < nax; j += i){
      f[j] += h[i]*mu[j/i];
    }
  }
}
////////
void pre(){
  mu[0] = 0; mu[1] = 1;
  fore(i,2,N){
    if (lp[i] == 0) {
      lp[i] = i; mu[i] = -1;
      pr.pb(i);
    }
    for (int j=0, mult= i*pr[j]; j<sz(pr) && pr[j]<=lp[i]
        && mult<=N; ++j, mult= i*pr[j]){
      if(i%pr[j]==0) mu[mult] = 0;
      else mu[mult] = mu[i]*mu[pr[j]];
      lp[mult] = pr[j];
    }
  }
}
```

6.15 Miller Rabin Test

```cpp
ll mulmod(ll a, ll b, ll m) {
        ll r=a*b-(ll)((long double)a*b/m+.5)*m;
        return r<0?r+m:r;
}
ll binpow(ll b, ll e, ll m){
  ll r = 1;
  while(e){
    if(e&1) r = mulmod(r, b,m);
    b = mulmod(b,b,m);
    e = e/2;
  }
  return r;
}

bool is_prime(ll n, int a, ll s, ll d){
        if(n==a) return true;
        ll x=binpow(a,d,n);
        if(x==1 || x+1==n)return true;
        forn(k,s-1){
                x=mulmod(x,x,n);
                if(x==1) return false;
                if(x+1==n) return true;
        }
        return false;
}
int ar[]={2,3,5,7,11,13,17,19,23,29,31,37};
bool rabin(ll n){ // true iff n is prime
        if(n==2) return true;
        if(n<2 || n%2==0) return false;
        ll s=0,d=n-1;
        while(d%2==0)++s,d/=2;
        forn(i,12) if(!is_prime(n,ar[i],s,d)) return
            false;
  return true;
}

///////////////////////////////////////

bool isPrime(ll n) {
        if (n < 2 || n % 6 % 4 != 1) return (n | 1) == 3;
        ll A[] = {2, 325, 9375, 28178, 450775, 9780504,
            1795265022};
        ll s=0,d=n-1;
        while(d%2==0)++s,d/=2;
        for (ll a : A) {   // ^ count trailing zeroes
                ll p = binpow(a%n, d, n), i = s;
                while (p != 1 && p != n - 1 && a % n && i
                    --)
                        p = mulmod(p, p, n);
                if (p != n-1 && i != s) return 0;
        }
        return 1;
}
```

6.16 Pollard Rho

```cpp
ll rho(ll n){
```

```
        if(!(n&1))return 2;
        ll x=2,y=2,d=1;
        ll c=rand()%n+1;
        while(d==1){
                x=(mulmod(x,x,n)+c)%n;
                y=(mulmod(y,y,n)+c)%n;
                y=(mulmod(y,y,n)+c)%n;
                if(x>=y)d=__gcd(x-y,n);
                else d=__gcd(y-x,n);
        }
        return d==n?rho(n):d;
}
void fact(ll n, map<ll,int>& f){ //O (lg n)^3
        if(n==1)return;
        if(rabin(n)){f[n]++;return;}
        ll q=rho(n);fact(q,f);fact(n/q,f);
}
```

6.17 Chinese Remainder Theorem

```
pll extendedEuclid(ll a, ll b){ // a * x + b * y = __gcd(
    a,b)
        ll x,y;
        if (b==0) return {1,0};
        auto p=extendedEuclid(b,a%b);
        x=p.se;
        y=p.fi-(a/b)*x;
        if(a*x+b*y==-__gcd(a,b)) x=-x, y=-y;
        return {x,y};
}
pair<pll,pll> diophantine(ll a, ll b, ll r) {
  //a*x+b*y=r where r is multiple of __gcd(a,b);
        ll d=__gcd(a,b);
  a/=d; b/=d; r/=d;
        auto p = extendedEuclid(a,b);
        p.fi*=r; p.se*=r;
        // assert(a*p.fi+b*p.se==r);
        return {p,{-b,a}}; // solutions: p+t*ans.se
}
ll inv(ll a, ll m) {
        assert(__gcd(a,m)==1);
        ll x = diophantine(a,m,1).fi.fi;
        return ((x%m)+m)%m;
}
#define MOD(a,m) (((a)%m+m)%m)
pll sol(tuple<ll,ll,ll> c){ //requires inv, diophantine
  ll a=get<0>(c), x1=get<1>(c), m=get<2>(c), d=__gcd(a,m)
      ;
  if(d==1) return pll(MOD(x1*inv(a,m),m), m);
  else return x1%d ? pll({-1LL,-1LL}) : sol(make_tuple(a/
      d,x1/d,m/d));
}
pair<ll,ll> crt(vector< tuple<ll,ll,ll> > &cond) { //
    returns: (sol, lcm)
        ll x1=0,m1=1,x2,m2;
        for(auto &t: cond){
                tie(x2,m2)=sol(t);
                if((x1-x2)%__gcd(m1,m2))return {-1,-1};
                if(m1==m2)continue;
                ll k=diophantine(m2,-m1,x1-x2).fi.se,l=m1
                    *(m2/__gcd(m1,m2));
                x1=MOD((__int128_t)m1*k+x1,l); m1=l;
        }
        return sol(make_tuple(1,x1,m1));
} //cond[i]={ai,bi,mi} ai*xi=bi (mi); assumes lcm fits in
    ll
```

6.18 Simplex

```
#include "../c++/template.cpp"
vi X, Y;
ld Z;
int n, m;
vd b, c;  // Cantidades, costos
vector<vd> a;  // Variables, restricciones
void pivot(int x, int y){
  swap(X[y], Y[x]);
  b[x] /= a[x][y];
  forn(j, m) if(j != y) a[x][j] /= a[x][y];
  a[x][y] = 1 / a[x][y];
  forn(i, n) if(i != x && abs(a[i][y]) > eps){
    b[i] -= a[i][y] * b[x];
    forn(j, m) if(j != y) a[i][j] -= a[i][y] * a[x][j];
    a[i][y] = -a[i][y] * a[x][y];
  }
  Z += c[y] * b[x];
  forn(j, m) if(j != y) c[j] -= c[y] * a[x][j];
  c[y] = -c[y] * a[x][y];
}

pair<ld, vd> simplex(){  // maximizar Z = c * x dado ax
    <= b, x_i >= 0
  X.resize(m),  iota(all(X), 0);
  Y.resize(n),  iota(all(Y), m);
  while(1){
    int x = min_element(all(b)) - b.begin(), y = -1;
    if(b[x] > -eps) break;
    forn(j, m) if(a[x][j] < -eps){ y = j; break; }
    if(y == -1) return {-1, {}}; // no solution to Ax<=b
    pivot(x, y);
  }
  while(1){
    int x = -1, y = max_element(all(c)) - c.begin();
    if(c[y] < eps) break;
    ld mn = inf;
    forn(i, n) if(a[i][y] > eps && b[i] / a[i][y] < mn)
      mn = b[i] / a[i][y], x = i;
```

```
    if(x == -1) return {inf, {}};   // c^T x is unbounded
    pivot(x, y);
  }
  vd ans(m);
  forn(i, n) if(Y[i] < m) ans[Y[i]] = b[i];
  return {Z, ans};
}
```

6.19 Gauss Jordan

```
int gauss(vector<vector<double>> &a, vector<double> &ans)
    {
  int n = sz(a), m = sz(a[0]) - 1;
  vi where(m, -1);
  for(int col=0, row=0; col<m && row<n; ++col) {
    int sel = row;
    fore(i,row,n-1)
      if(abs(a[i][col]) > abs(a[sel][col])) sel = i;
    if(abs(a[sel][col]) < eps) continue;

    fore(i,col,m) swap (a[sel][i], a[row][i]);
    where[col] = row;

    forn(i,n){
      if (i != row) {
        double c = a[i][col] / a[row][col];
        for (int j=col; j<=m; ++j) a[i][j] -= a[row][j] *
            c;
      }
    }
    ++row;
  }

  ans.assign(m, 0);
  forn(i,m){
    if(where[i] != -1) ans[i] = a[where[i]][m] / a[where[
        i]][i];
  }
  forn(i,n){
    double sum = 0;
    forn(j,m) sum += ans[j] * a[i][j];
    if(abs(sum - a[i][m]) > eps) return 0;
  }

  forn(i,m) if(where[i] == -1) return 1e9; /// infinitas
      soluciones
  return 1;
}
```

6.20 Gauss Jordan Modular

```
const int eps = 0, mod = 1e9+7;

int gauss(vector<vi> &a, vi &ans) {
  int n = sz(a), m = sz(a[0]) - 1;
  vi where(m, -1);
  for(int col=0, row=0; col<m && row<n; ++col) {
    int sel = row;
    fore(i,row,n-1)
      if(abs(a[i][col]) > abs(a[sel][col])) sel = i;
    if(abs(a[sel][col]) <= eps) continue;

    fore(i,col,m) swap (a[sel][i], a[row][i]);
    where[col] = row;

    forn(i,n){
      if (i != row) {
        int c = 1LL*a[i][col] * inv(a[row][col])%mod;
        for (int j=col; j<=m; ++j) a[i][j] = (mod + a[i][
            j] - (1LL*a[row][j] * c)%mod)%mod;
      }
    }
    ++row;
  }

  ans.assign(m, 0);
  forn(i,m){
    if(where[i] != -1) ans[i] = 1LL*a[where[i]][m] * inv(
        a[where[i]][i])%mod;
  }
  forn(i,n){
    ll sum = 0;
    forn(j,m) sum = (sum + 1LL*ans[j] * a[i][j])%mod;
    if(abs(sum - a[i][m]) > eps) return 0;
  }

  forn(i,m) if(where[i] == -1) return 1e9; /// infinitas
      soluciones
  return 1;
}
```

6.21 Berlekamp Massey

```
// taken from https://codeforces.com/blog/entry/61306
struct ber_ma{
        vi BM(vi &x){
                vi ls,cur; int lf,ld;
                forn(i,sz(x)){
                        ll t=0;
                        forn(j,sz(cur)) t=(t+x[i-j-1]*(ll
                            )cur[j])%mod;
                        if((t-x[i])%mod==0) continue;
                        if(!sz(cur)){
                                cur.resize(i+1);
                                lf=i; ld=(t-x[i])%mod;
                                continue;
                        }
                        ll k=-(x[i]-t)*inv(ld,mod);
                        vi c(i-lf-1); c.pb(k);
```

```
                    forn(j,sz(ls)) c.pb(-ls[j]*k%mod);
                    if(sz(c)<sz(cur)) c.resize(sz(cur));
                    forn(j,sz(cur)) c[j]=(c[j]+cur[j])%mod;
                    if(i-lf+sz(ls)>=sz(cur)) ls=cur,lf=i,ld=(t-x[i])%mod;
                    cur=c;
                }
                forn(i,sz(cur)) cur[i]=(cur[i]%mod+mod)%mod;
                return cur;
        }

        int m; //length of recurrence
        //a: first terms
        //h: relation
        vector<ll> a, h, t_, s, t;
        //calculate p*q mod f
        inline vector<ll> mull(vector<ll> p, vector<ll> q){
                forn(i,2*m) t_[i]=0;
                forn(i,m) if(p[i])
                        forn(j,m)
                                t_[i+j]=(t_[i+j]+p[i]*q[j])%mod;
                for(int i=2*m-1;i>=m;--i) if(t_[i])
                        forn(j,m)
                                t_[i-j-1]=(t_[i-j-1]+t_[i]*h[j])%mod;
                forn(i,m) p[i]=t_[i];
                return p;
        }
        inline ll calc(ll k){
    if(k < sz(a)) return a[k];
                forn(i,m) s[i]=t[i]=0;
                s[0]=1;
                if(m!=1) t[1]=1;
                else t[0]=h[0];

                while(k){
                        if(k&1LL) s = mull(s,t);
                        t = mull(t,t); k/=2;
                }
                ll su=0;
                forn(i,m) su=(su+s[i]*a[i])%mod;
                return (su%mod+mod)%mod;
        }
        ber_ma(vi &x){
                vi v = BM(x); m=sz(v);
                h.resize(m), a.resize(m), s.resize(m);
                t.resize(m), t_.resize(2*m);
                forn(i,m) h[i]=v[i],a[i]=x[i];
        }
};
```


6.22 Lagrange Interpolation

```
#include "mint.cpp"
const int N = 1e6;
mint f[N], fr[N];
void initC(){
  if(f[0] == 1) return; // Already precalculated
  f[0] = 1;
  for1(i, N-1) f[i] = f[i-1] * i;
  fr[N-1] = bpow(f[N-1], mod-2);
  fored(i, 1, N-1) fr[i-1] = fr[i] * i;
}
// mint C(int n, int k) { return k<0 || k>n ? 0 : f[n] * fr[k] * fr[n-k]; }
struct LagrangePol {
  int n;
  vector<mint> y, den, l, r;
  LagrangePol(vector<mint> f): n(sz(f)), y(f), den(n), l(n), r(n){// f[i] := f(i)
    // Calcula interpol. pol P in O(n) := deg(P) = sz(v)-1
    initC();
    forn(i, n) {
      den[i] = fr[n-1-i] * fr[i];
      if((n-1-i) & 1) den[i] = -den[i];
    }
  }
  mint eval(mint x){ // Evaluate LagrangePoly P(x) in O(n)
    l[0] = r[n-1] = 1;
    for1(i, n-1) l[i] = l[i-1] * (x - i + 1);
    fored(i, 0, n-2) r[i] = r[i+1] * (x - i - 1);
    mint ans = 0;
    forn(i, n) ans += l[i] * r[i] * y[i] * den[i];
    return ans;
  }
};
```

6.23 Discrete Log

```
// Returns minimum x for which a ^ x % m = b % m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
```

```
    }
    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;
    unordered_map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }
    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

6.24 Fractions

```
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; }
};
```

6.25 Modular Int

```
typedef long long ll;
const int mod = 1e9 + 7;
template <class T>
T bpow(T b, int e) {
  T a(1);
  do{
    if(e & 1) a *= b;
    b *= b;
  }while(e >>= 1);
  return a;
}
struct mint {
  int x;
  mint(): x(0){}
  mint(ll v) : x((v % mod + mod) % mod) {} // be careful
     of negative numbers!
  // Helpers to shorten code
  #define add(a, b) a + b >= mod ? a + b - mod : a + b
  #define sub(a, b) a < b ? a + mod - b : a - b
  #define yo *this
  #define cmint const mint&

  mint &operator += (cmint o) { return x = add(x, o.x),
     yo; }
  mint &operator -= (cmint o) { return x = sub(x, o.x),
     yo; }
  mint &operator *= (cmint o) { return x = ll(x) * o.x %
     mod, yo; }
  mint &operator /= (cmint o) { return yo *= bpow(o, mod
     -2); }

  mint operator + (cmint b) const { return mint(yo) += b;
     }
  mint operator - (cmint b) const { return mint(yo) -= b;
     }
  mint operator * (cmint b) const { return mint(yo) *= b;
     }
  mint operator / (cmint b) const { return mint(yo) /= b;
     }

  mint operator - () const { return mint() - mint(yo); }
  bool operator == (cmint b) const { return x == b.x; }
  bool operator != (cmint b) const { return x != b.x; }

  friend ostream& operator << (ostream &os, cmint p) {
     return os << p.x; }
  friend istream& operator >> (istream &is, mint &p) {
     return is >> p.x; }
};
```

7 Dynamic Programming

7.1 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];
  forn(i,m+1)
    forn(j,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];
}
```

7.2 Longest common subsequence

```
const int nax = 1005;
int dp[nax][nax];
int lcs(const string &s, const string &t){
  int n = sz(s), m = sz(t);
  forn(j,m+1) dp[0][j] = 0;
  forn(i,n+1) dp[i][0] = 0;
  for1(i,n){
    for1(j,m){
      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];
}
```

7.3 Longest increasing subsequence

```
// Complejidad n log n
int lis(const vi &a) {
  int n = a.size();
  vi d(n+1, inf);
  d[0] = -inf;

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

  int ans = 0;
  for (int i = 0; i <= n; i++) {
    if (d[i] < inf) ans = i;
  }
  return ans;
}
```

7.4 Trick to merge intervals

```
// Option 1
for(int len= 0; len<n; ++len){
    for(int l= 0; l<n-len; ++l){
        int r= l+len;
        dp[l][r]= max(dp[l+1][r], dp[l][r-1]);
    }
}
// Option 2
for(int l= n-1; l>=0; --l){
    for(int r= l; r<n ; ++r){
        dp[l][r]= max(dp[l+1][r], dp[l][r-1]);
    }
}
```

7.5 Trick Sets DP

```
// Complexity O(N*2^N)
const int N;
int dp[1<<N][N+1];
int F[1<<N];
int A[1<<N];
// ith bit is ON S(mask, i) = S(mask, i-1)
// ith bit is OFF S(mask,i) = S(mask, i-1) + S(mask^(1<<i
    ), i-1)
//iterative version
forn(mask,(1<<N)){
        dp[mask][0] = A[mask];  //handle base case
            separately (leaf states)
        forn(i,N){
                if(mask & (1<<i))
                        dp[mask][i+1] = dp[mask][i] + dp[
                            mask^(1<<i)][i];
                else
                        dp[mask][i+1] = dp[mask][i];
        }
        F[mask] = dp[mask][N];
}
//memory optimized, super easy to code.
forn(i,(1<<N)) F[i] = A[i];
forn(i,N)
  forn(mask,(1<<N)){
    if(mask & (1<<i)) F[mask] += F[mask^(1<<i)];
  }
```

7.6 Divide and Conquer

```
const ll inf = 1e18;
const int nax = 1e3+20, kax = 20;

ll C[nax][nax], dp[kax][nax];
int n;

void compute(int k, int l, int r, int optl, int optr){
  if(l>r) return ;
  int mid= (l+r)/2, opt;
  pll best= {inf,-1};
  for(int i= max(mid,optl); i<= optr ; ++i ){
    best = min(best, {dp[k-1][i+1] + C[mid][i] ,i}  );
  }
  tie(dp[k][mid], opt) = best;
  compute(k,l, mid-1, optl, opt);
  compute(k,mid+1, r, opt, optr);
}

inside main(){
  fore(k,1,K) // definir el caso base k = 0.
    compute(k,0,n-1,0,n-1);
}
```

7.7 Knuth's Optimization

```
const int nax = 1e3+20;
const ll inf = LONG_LONG_MAX;
ll dp[nax][nax];
int k[nax][nax];
int C[nax][nax];  // puede depender de k

int main(){
  for(int len=2; len<n; ++len){
    for(int l=0; l< n-len; ++l){
      int r= l+len;
      ll &ans= dp[l][r];
      if(len== 2){
        k[l][r]= l+1;
        ans= C[l][r];
        continue;
      }
      ans= inf;
      for(int i= k[l][r-1]; i<= k[l+1][r]; ++i ){
        if(ans> dp[l][i]+ dp[i][r]){
          ans= dp[l][i] + dp[i][r];
          k[l][r]= i;
        }
      }
      ans+= C[l][r];
    }
  }
  cout<< dp[0][n-1]<<el;
}
```

7.8 Convex Hull Trick

```
struct line {
  ll m, b;
  ll eval(ll x) { return m * x + b; }
  ld inter(line &l) { return (ld) (b - l.b) / (l.m - m);
    }
};

struct cht {
  vector<line> lines;
  vector<ld> inter;
  int n;
  inline bool ok(line &a, line &b, line &c) {
    return a.inter(c) > a.inter(b);
  }
  void add(line &l) { /// m1 < m2 < m3 ...
    n = sz(lines);
    if(n && lines.back().m == l.m && lines.back().b >= l.
        b) return;
    if(n == 1 && lines.back().m == l.m && lines.back().b
        < l.b) lines.pop_back(), n--;
    while(n >= 2 && !ok(lines[n-2], lines[n-1], l)) {
      --n;
      lines.pop_back(); inter.pop_back();
    }
    lines.pb(l); n++;
    if(n >= 2) inter.pb(lines[n-2].inter(lines[n-1]));
  }
  ll get_max(ld x) {
    if(sz(lines) == 0) return LLONG_MIN;
    if(sz(lines) == 1) return lines[0].eval(x);
    int pos = lower_bound(all(inter), x) - inter.begin();
    return lines[pos].eval(x);
  }
};
```

7.9 CH Trick Dynamic

```
typedef ll T;
const T is_query = -(1LL << 62);
struct line {
        T m, b;
        mutable multiset<line>::iterator it, end;
        const line *succ(multiset<line>::iterator it)
            const {
                return (++it == end ? nullptr : &*it);
        }
        bool operator < (const line &l) const {
                if(l.b != is_query) return m < l.m;
                auto s = succ(it);
                if(!s) return 0;
                return b - s->b < ld(s->m - m) * l.m;
        }
```

```
};
struct CHT : public multiset<line> {
        iterator nex(iterator y){ return ++y; }
        iterator pre(iterator y){ return --y; }
        bool bad(iterator y) {
                auto z = nex(y);
                if(y == begin()) {
                        if(z == end()) return 0;
                        return y->m == z->m && y->b <= z->b;
                }
                auto x = pre(y);
                if(z == end()) return y->m == x->m && y->b == x->b;
                return ld(x->b - y->b)*(z->m - y->m) >= ld(y->b - z->b)*(y->m - x->m);
        }
        void add(T m, T b) {
                auto y = insert(line{m, b});
                y->it = y,  y->end = end();
                if(bad(y)){ erase(y);  return; }
                while(nex(y) != end() && bad(nex(y))) erase(nex(y));
                while(y != begin() && bad(pre(y))) erase(pre(y));
        }
        T eval(T x) { /// max
                line l = *lower_bound(line{x, is_query});
                return l.m*x + l.b;
        }
};
```

8 Geometry

8.1 Point

```
struct pt{
        ld x, y;
        pt(){}
        pt(ld x, ld y): x(x), y(y){}
        pt(ld ang): x(cos(ang)), y(sin(ang)){}  // Polar unit point: ang(randians)
  // ------- BASIC OPERATORS -------  //
  pt operator+(pt p){ return pt(x+p.x, y+p.y); }
        pt operator-(pt p){ return pt(x-p.x, y-p.y); }
        pt operator*(ld t){ return pt(x*t, y*t); }
        pt operator/(ld t){ return pt(x/t, y/t); }
        ld operator*(pt p){ return x*p.x + y*p.y; }
        ld operator%(pt p){ return x*p.y - y*p.x; }
  // ------- COMPARISON OPERATORS -------  //
  bool operator==(pt p){ return abs(x - p.x) <= eps && abs(y - p.y) <= eps; }
        bool operator<(pt p)const{ // for sort, convex hull/set/map
                return x < p.x - eps || (abs(x - p.x) <= eps && y < p.y - eps); }
        bool operator!=(pt p){ return !operator==(p); }
  // --------------- NORMS ---------------  //
        ld norm2(){ return *this**this; }
        ld norm(){ return sqrt(norm2()); }
        pt unit(){ return *this/norm(); }
        // ------------ SIDE, LEFT------------ //
  ld side(pt p, pt q){ return (q-p) % (*this-p); }// C is : >0 L, ==0 on AB, <0 R
        bool left(pt p, pt q){ // Left of directed line PQ? (eps == 0 if integer)
                return side(p, q) > eps; } // (change to >= -eps to accept collinear)
  // --------------- ANGLES --------------- //
  ld angle(){ return atan2(y, x); } // Angle from origin, in [-pi, pi]
  ld min_angle(pt p){ return acos(*this*p / (norm()*p.norm())); } // In [0, pi]
  ld angle(pt a, pt b, bool CW){ // Angle< AB(*this) > in direction CW
    ld ma = (a - b).min_angle(*this - b);
    return side(a, b) * (CW ? -1 : 1) <= 0 ? ma : 2*pi - ma; }
  bool in_angle(pt a, pt b, pt c, bool CW=1){ // Is pt inside infinite angle ABC
    return angle(a, b, CW) <= c.angle(a, b, CW); } // From AB to AC in CW direction
  // --------------- ROTATIONS --------------- //
        pt rot(pt p){ return pt(*this % p,*this * p); }// use ccw90(1,0), cw90(-1,0)
        pt rot(ld ang){ return rot(pt(sin(ang), cos(ang))); } // CCW, ang (radians)
        pt rot_around(ld ang, pt p){ return p + (*this - p).rot(ang); }
  pt perp(){ return rot(pt(1, 0)); }
  // --------------- SEGMENTS --------------- //
  bool in_disk(pt p, pt q){ return (p - *this) * (q - *this) <= 0; }
  bool on_segment(pt p, pt q){ return side(p, q) == 0 && in_disk(p, q); }
};
int sgn(ld x){
  if(x < 0) return -1;
  return x == 0 ? 0 : 1;
}
void segment_intersection(pt a, pt b, pt c, pt d, vector<pt>& out){ // AB y CD
  ld sa = a.side(c, d), sb = b.side(c, d);
  ld sc = c.side(a, b), sd = d.side(a, b);        // proper cut
  if(sgn(sa)*sgn(sb) < 0 && sgn(sc)*sgn(sd) < 0) out.pb((a*sb - b*sa) / (sb-sa));
```

```
    for(pt p : {c, d}) if(p.on_segment(a, b)) out.pb(p);
    for(pt p : {a, b}) if(p.on_segment(c, d)) out.pb(p);
  }
```

8.2 Line

```
// Add point.cpp Basic operators
struct line{
  pt v;  ld c;
  line(){}
  line(pt p, pt q): v(q - p), c(v % p){}
  line(pt v, ld c): v(v), c(c){}
  line(ld a, ld b, ld c): v({b, -a}), c(c){}
  bool operator<(line l){ return v % l.v > 0; }
  bool operator/(line l){ return v % l.v == 0; } // abs()
      <= eps
  pt operator^(line l){ // LINE - LINE Intersection
    if(*this / l) return pt(inf, inf);  // PARALLEL
    return (l.v*c - v*l.c) / (v % l.v);
  }
  ld side(pt p){ return v % p - c; }
  bool has(pt p){ return v % p == c; }
  pt proj(pt p) { return p - v.perp() * side(p) / v.norm2
      (); }
  pt refl(pt p) { return proj(p) * 2 - p; }
  bool cmp_proj(pt p, pt q){ return v * p < v * q; }
  ld dist(pt p){ return abs(side(p)) / v.norm(); }
  ld dist2(pt p){ return side(p) * side(p) / double(v.
      norm2()); }

  bool operator==(line l){ return *this / l && c == l.c;
      }
        ld angle(line l){ return v.min_angle(l.v); }   //
            /angle bet. 2 lines
  line perp_at(pt p) { return {p, p + v.perp() }; }
  line translate(pt t) { return {v, c + (v % t)}; }
  line shift_left(ld dist){ return {v, c + dist * v.norm
      ()}; }
};
```

8.3 Convex Hull

```
struct pt{
  ld x, y;
  pt(){}
  pt(ld x, ld y): x(x), y(y){}
  bool operator<(pt p)const{ // for sort, convex hull/set
      /map
              return x < p.x - eps || (abs(x - p.x) <=
                  eps && y < p.y - eps); }
  pt operator-(pt p){return pt(x - p.x, y - p.y);}
  ld operator%(pt p){return x * p.y - y * p.x;}
  ld side(pt p, pt q){return (q - p) % (*this - p);}
};
// CCW order, excludes collinear points
// Change .side(r[sz(r)-2], p[i]) > 0 to include
    collinear
vector<pt> chull(vector<pt>& p){
  if(sz(p) < 3) return p;
  vector<pt> r;
  sort(all(p));  // first x, then y
  forn(i, sz(p)){ // lower hull
    while(sz(r) > 1 && r.back().side(r[sz(r)-2], p[i]) >=
        0) r.pop_back();
    r.pb(p[i]);
  }
  r.pop_back();
  int k = sz(r);
  fored(i, 0, sz(p)-1){ // upper hull
    while(sz(r) > k+1 && r.back().side(r[sz(r)-2], p[i])
        >= 0) r.pop_back();
    r.pb(p[i]);
  }
  r.pop_back();
  return r;
}
```

8.4 Polygon

```
#include "line.cpp"
#include "circle.cpp"
#include "convex_hull.cpp"
int sgn(double x){return x<-eps?-1:x>eps;}
struct poly {
        int n, normal = -1; vector<pt> p;
        poly(){}
        poly(const vector<pt>& p): p(p), n(sz(p)){}

        double area(){
                double r=0.;
                forn(i, n) r += p[i] % p[(i+1)%n];
                return abs(r)/2; // negative if CW,
                    positive if CCW
        }

        bool isConvex() {
    bool pos=false, neg=false;
    forn(i, n) {
      int s = p[(i+2)%n].side(p[i], p[(i+1)%n]);
      pos |= s > 0;
      neg |= s < 0;
    }
    return !(pos && neg);
  }

        pt centroid(){ // (barycenter)
```

```
            pt r(0,0); double t=0;          ///REVISAR
            forn(i,n){
                r = r+(p[i]+p[(i+1)%n])*(p[i]%p[(i+1)%n]);
                t += p[i]%p[(i+1)%n];
            }
            return r/t/3;
        }

        bool has(pt q){ /// O(n)
            forn(i, n) if(q.on_segment(p[i], p[(i+1) % n])) return true;
            int cnt = 0;
            forn(i, n){
                int j = (i+1)%n;
                int k = sgn((q - p[j]) % (p[i] - p[j]));
                int u = sgn(p[i].y - q.y), v = sgn(p[j].y - q.y);
                if(k > 0 && u < 0 && v >= 0) ++cnt;
                if(k < 0 && v < 0 && u >= 0) --cnt;
            }
            return cnt!=0;
        }
// ------------ HAS_LOG ------------ //
        void remove_col(){ // helper
  vector<pt> s;
  forn(i, n) if(!p[i].on_segment(p[(i-1+n) % n], p[(i+1) % n])) s.pb(p[i]);
  p.swap(s);  n = sz(p);
}
void normalize(){  // helper
  remove_col();
  if(p[2].left(p[0], p[1])) reverse(all(p));
  int pi = min_element(all(p)) - p.begin();
  vector<pt> s(n);
  forn(i, n) s[i] = p[(pi+i) % n];
  p.swap(s);  n = sz(p);
}
bool has_log(pt q){ /// O(log(n)) only CONVEX.
  if(normal == -1) normal = 1, normalize();
  if(q.left(p[0], p[1]) || q.left(p[n-1], p[0])) return false;
  int l = 1, r = n-1;    // returns true if point on boundary
  while(l+1 < r){           // (change sign of EPS in left
    int m = (l+r) / 2;        //  to return false in such case)
    if(!q.left(p[0], p[m])) l = m;
    else r = m;
  }
  return !q.left(p[l], p[l+1]);
}
// ------------ FARTHEST------------ //
        pt farthest(pt v){ /// O(log(n)) only CONVEX
            if(n < 10){
                int k=0;
                for1(i,n-1) if(v*(p[i]-p[k]) > eps) k=i;
                return p[k];
            }
            if(n == sz(p)) p.pb(p[0]);
            pt a = p[1]-p[0];
            int s=0, e=n, ua=v*a>eps;
            if(!ua && v*(p[n-1]-p[0]) <= eps) return p[0];
            while(1){
                int m = (s+e)/2; pt c=p[m+1]-p[m];
                int uc = v*c> eps;
                if(!uc && v*(p[m-1]-p[m]) <= eps) return p[m];
                if(ua && (!uc||v*(p[s]-p[m]) > eps))e=m;
                else if(ua || uc || v*(p[s]-p[m]) >= -eps) s=m, a=c, ua=uc;
                else e=m;
                assert(e>s+1);
            }
        }

        poly cut(line l){    // cut CONVEX polygon by line l
            vector<pt> q;  // returns part at left of l.pq
            forn(i, n) {
                int d0 = sgn(l.side(p[i])), d1 = sgn(l.side(p[(i+1) % n]));
                if(d0 >= 0) q.pb(p[i]);
                line m(p[i], p[(i+1) % n]);
                if(d0*d1 < 0 && !(l / m)) q.pb(l ^ m);
            }
            return poly(q);
        }
        ld intercircle(circle c){ /// area of intersection with circle
            ld r = 0.;
            forn(i,n){
                int j = (i+1)%n; ld w = c.intertriangle(p[i], p[j]);
                if((p[j]-c.o)%(p[i]-c.o) > 0) r+=w;
                else r-=w;
            }
            return abs(r);
        }
```

```
		ld callipers(){ // square distance: pair of most distant points
			ld r=0;	// prereq: convex, ccw, NO COLLINEAR POINTS
			for(int i=0,j=n<2?0:1; i<j; ++i){
				for(;;j=(j+1)%n){
					r = max(r,(p[i]-p[j]).norm2());
					if((p[(i+1)%n]-p[i])%(p[(j+1)%n]-p[j]) <= eps) break;
				}
			}
			return r;
		}
};

// / max_dist between 2 points (pa, pb) of 2 Convex polygons (a, b)
ld rotating_callipers(vector<pt>& a, vector<pt>& b){
  pair<ll, int> start = {-1, -1};
  if(sz(a) == 1) swap(a, b);
  forn(i, sz(a)) start = max(start, {(b[0] - a[i]).norm2(), i});
  if(sz(b) == 1) return start.fi;

  ld r = 0;
  for(int i = 0, j = start.se; i<sz(b); ++i){
    for(;; j = (j+1) % sz(a)){
      r = max(r, (b[i] - a[j]).norm2());
      if((b[(i+1) % sz(b)] - b[i]) % (a[(j+1) % sz(a)] - a[j]) <= eps) break;
    }
  }
  return r;
}
```

8.5 Circle

```
// Add point.cpp and line.cpp Basic operators
struct circle {
		pt o; ld r;
		circle(pt o, ld r):o(o),r(r){}
		bool has(pt p){ return (o-p).norm() <= r+eps;}
		vector<pt> operator^(circle c){ // ccw
			vector<pt> s;
			ld d = (o - c.o).norm();
			if(d > r + c.r + eps || d + min(r, c.r) + eps < max(r, c.r)) return s;
			ld x = (d*d - c.r*c.r + r*r)/(2*d);
			ld y = sqrt(r*r - x*x);
			pt v = (c.o - o) / d;
			s.pb(o + v*x - v.rot(pt(1, 0))*y);
			if(y > eps) s.pb(o + v*x + v.rot(pt(1, 0))*y);
			return s;
		}
		vector<pt> operator^(line l){
			vector<pt> s;
			pt p = l.proj(o);
			ld d = (p-o).norm();
			if(d - eps > r) return s;
			if(abs(d-r) <= eps){ s.pb(p); return s; }
			d=sqrt(r*r - d*d);
			s.pb(p + l.v.unit() * d);
			s.pb(p - l.v.unit() * d);
			return s;
		}
		vector<pt> tang(pt p){
			ld d = sqrt((p-o).norm2()-r*r);
			return *this^circle(p,d);
		}
		bool in(circle c){ return (o-c.o).norm() + r <= c.r + eps; } // non strict
		ld intertriangle(pt a, pt b){ // area of intersection with oab
			if(abs((o-a) % (o-b)) <= eps) return 0.;
			vector<pt> q = {a}, w = *this ^ line(a, b);
			if(sz(w) == 2) for(auto p: w) if((a-p) * (b-p)<-eps) q.pb(p);
			q.pb(b);
			if(sz(q) == 4 && (q[0] - q[1]) * (q[2] - q[1]) > eps) swap(q[1], q[2]);
			ld s = 0;
			forn(i, sz(q)-1){
				if(!has(q[i]) || !has(q[i+1])) s += r*r * (q[i] - o).min_angle(q[i+1] - o) / 2;
				else s += abs((q[i] - o) % (q[i+1] - o) / 2);
			}
			return s;
		}
};

vector<ld> intercircles(vector<circle> c){
		vector<ld> r(sz(c) + 1); // r[k]: area covered by at least k circles
		forn(i, sz(c)){			// O(n^2 log n) (high constant)
			int k = 1;  pt O = c[i].o;
			vector<pair<pt, int>> p = {
				{c[i].o + pt(1,0) * c[i].r, 0},
				{c[i].o - pt(1,0) * c[i].r, 0}};
			forn(j, sz(c)) if(j != i){
				bool b0 = c[i].in(c[j]), b1 = c[j].in(c[i]);
				if(b0 && (!b1 || i < j)) ++k;
				else if(!b0 && !b1){
```

```
                        auto v = c[i] ^ c[j];
                        if(sz(v) == 2){
                                p.pb({v[0], 1});
                                    p.pb({v[1],
                                   -1});
                                if(cmp(v[1] - O,
                                    v[0] - O)) ++k
                                    ;
                        }
                    }
            }   // FOR "cmp" see "radial_order.cpp"
            sort(all(p), [&](auto& a, auto& b){
                return cmp(a.fi - O, b.fi - O); });
            forn(j, sz(p)){
                    pt p0 = p[j ? j-1 : sz(p)-1].fi,
                       p1 = p[j].fi;
                    ld a = (p0 - c[i].o).min_angle(p1
                        - c[i].o);
                    r[k] += (p0.x - p1.x)*(p0.y + p1.
                       y)/2 + c[i].r*c[i].r*(a - sin(
                       a))/2;
                    k += p[j].se;
            }
        }
        return r;
}
```

8.6 Radial Order

```
typedef double ld;
struct pt{
  ld x, y;
  pt(){}
  pt(ld x, ld y): x(x), y(y){}
  pt operator-(pt p){ return pt(x-p.x, y-p.y); }
  ld operator%(pt p){ return x*p.y - y*p.x; }
  int cuad(){
    if(x > 0 && y >= 0) return 0;
    if(x <= 0 && y > 0) return 1;
    if(x < 0 && y <= 0) return 2;
    if(x >= 0 && y < 0) return 3;
    return -1; // x == 0 && y == 0
  }
};
bool cmp(pt p1, pt p2){ // Around Origin(0, 0):  -->
    sort(all(pts), cmp);
  int c1 = p1.cuad(), c2 = p2.cuad();
  return c1 == c2 ? p1.y*p2.x < p1.x*p2.y : c1 < c2;
} // Around const pt O(x, y):
//  -->  sort(all(pts), [&](pt& pi, pt& pj){ return cmp(
    pi - O, pj - O); });
```

8.7 Halfplane

```
typedef double ld;
const ld eps = 1e-7, inf = 1e12;
struct pt {  // for 3D add z coordinate
  ld x, y;
  pt(){}
  pt(ld x, ld y): x(x), y(y){}
  pt operator-(pt p){return pt(x - p.x, y - p.y);}
  pt operator*(ld t){return pt(x * t, y * t);}
  pt operator/(ld t){return pt(x / t, y / t);}
  ld operator%(pt p){return x * p.y - y * p.x;}
  ld operator*(pt p){return x * p.x + y * p.y;}
};

struct halfplane {
  pt p, v;  ld c, angle;
  halfplane(){}
  halfplane(pt p, pt q): p(p), v(q - p), c(v % p), angle(
     atan2(v.y, v.x)){}
  bool operator<(halfplane b)const{ return angle < b.
     angle; }
  bool operator/(halfplane l){ return abs(v % l.v) <= eps
     ; } // 2D
  pt operator^(halfplane l){ return *this / l ? pt(inf,
     inf) : (l.v*c - v*l.c) / (v % l.v);}
  bool out(pt q){ return v % q < c; } // try < c-eps
};

vector<pt> intersect(vector<halfplane> b){
  vector<pt> bx = {{inf, inf}, {-inf, inf}, {-inf, -inf},
      {inf, -inf}};
  forn(i, 4) b.pb(halfplane(bx[i], bx[(i+1) % 4]));
  sort(all(b));
  int n = sz(b), q = 1, h = 0;
  vector<halfplane> c(sz(b) + 10);
  forn(i, n){
    while(q < h && b[i].out(c[h] ^ c[h-1])) --h;
    while(q < h && b[i].out(c[q] ^ c[q+1])) ++q;
    c[++h] = b[i];
    if(q < h && abs(c[h].v % c[h-1].v) < eps){
      if(c[h].v * c[h-1].v <= 0) return {};
      h--;
      if(b[i].out(c[h].p)) c[h] = b[i];
    }
  }
  while(q < h-1 && c[q].out(c[h] ^ c[h-1])) --h;
  while(q < h-1 && c[h].out(c[q] ^ c[q+1])) ++q;
  if(h - q <= 1) return {};
  c[h+1] = c[q];
  vector<pt> s;
  fore(i, q, h) s.pb(c[i] ^ c[i+1]);
  return s;
}
```

8.8 KD Tree

```
struct pt{
        ld x, y;
        pt(){}
        pt(ld x, ld y): x(x), y(y){}
  pt operator+(pt p){ return pt(x+p.x, y+p.y); }
        pt operator-(pt p){ return pt(x-p.x, y-p.y); }
        ld operator*(pt p){ return x*p.x + y*p.y; }
        ld norm2(){ return *this * *this; }
        bool operator<(pt p)const{ // for sort, convex
            hull/set/map
                return x < p.x - eps || (abs(x - p.x) <=
                    eps && y < p.y - eps); }
};
inline bool onx(pt a, pt b){ return a.x < b.x; }
inline bool ony(pt a, pt b){ return a.y < b.y; }
// Given a set of N points, answer queries of nearest
    point in O(log(N))
struct Node {
        pt pp;
        ll x0 = inf, x1 = -inf, y0 = inf, y1 = -inf;
        Node *fir = 0, *sec = 0;
        inline ll distance(pt p){
                ll x = min(max(x0, p.x), x1);
                ll y = min(max(y0, p.y), y1);
                return (pt(x, y) - p).norm2();
        }
        Node(vector<pt>&& vp): pp(vp[0]){
                for(pt& p: vp){
                        x0 = min(x0, p.x), x1 = max(x1, p
                            .x);
                        y0 = min(y0, p.y), y1 = max(y1, p
                            .y);
                }
                if(sz(vp) > 1){
                        sort(all(vp), x1 - x0 >= y1 - y0
                            ? onx : ony);
                        int m = sz(vp) / 2;
                        fir = new Node({vp.begin(), vp.
                            begin() + m});
                        sec = new Node({vp.begin() + m,
                            vp.end()});
                }
        }
};
struct KDTree {
        Node* root;
        KDTree(const vector<pt>& vp):root(new Node({all(
            vp)})) {}
        pair<ll, pt> search(pt p, Node *node){
                if(!node->fir){ // To avoid query point
                    as answer:
                        // ADD:  if(p == node -> pp)
                            return {INF, pt()};
                        return {(p - node->pp).norm2(),
                            node->pp};
                }
                Node *f = node->fir, *s = node->sec;
                ll bf = f->distance(p), bs = s->distance(
                    p);
                if(bf > bs) swap(bf, bs), swap(f, s);
                auto best = search(p,f);
                if(bs < best.fi) best = min(best, search(
                    p, s));
                return best;
        }
        pair<ll, pt> nearest(pt p){ return search(p, root
            ); }
};
```

8.9 Minkowski Sum

```
struct pt{
  ld x, y;
  pt(){}
  pt(ld x, ld y): x(x), y(y){}
  pt operator+(pt p){return pt(x + p.x, y + p.y);}
  pt operator-(pt p){return pt(x - p.x, y - p.y);}
  ld operator%(pt p){return x * p.y - y * p.x;}
  ld side(pt p, pt q){return (q - p) % (*this - p);}
};

struct mink_sum{
  vector<pt> p, q, pol;
  mink_sum(){}
  mink_sum(vector<pt>& p1, vector<pt>& p2, bool inter =
      1): p(p1), q(p2){
    if(inter) for(auto& [x, y] : q) x = -x, y = -y;
    pol.reserve(sz(p) + sz(q));
    reorder(p),  reorder(q);
    forn(i, 2) p.pb(p[i]), q.pb(q[i]);
          int i = 0, j = 0;
    while(i+2 < sz(p) || j+2 < sz(q)){
      pol.pb(p[i] + q[j]);
      auto cro = (p[i+1] - p[i]) % (q[j+1] - q[j]);
      i += cro >= -eps;
      j += cro <= eps;
    }
  }
  void reorder(vector<pt> &p){
    if(p[2].side(p[0], p[1]) < 0) reverse(all(p));
    int pos = 0;
    forn(i, sz(p)) if(ii{p[i].y, p[i].x} < ii{p[pos].y, p
        [pos].x}) pos = i;
    rotate(p.begin(), p.begin() + pos, p.end());
  }
  bool has(pt p){
    int cnt = 0;
```

```
    forn(i, sz(pol)) cnt += p.side(pol[i], pol[(i+1) % sz
        (pol)]) >= 0;
    return cnt == sz(pol);
  }
  bool intersect(pt shift = pt(0, 0)){ return has(shift);
      }
};  // Do polygons p1 and p2+shift intersect?
```

9 Miscellaneous

9.1 Counting Sort

```
// it suppose that every element is non-negative
// in other case just translate to the right the elements
void counting_sort(vi &a){
  int n = sz(a);
  int maximo = *max_element(all(a));
  vector<int> cnt(maximo+1);
  forn(i,n) ++cnt[a[i]];
  for(int i = 0, j = 0; i <= maximo; ++i)
    while(cnt[i]--) a[j++] = i;
}
```

9.2 Expression Parsing

```
bool delim(char c) {
  return c == ' ';
}
bool is_op(char c) {
  return c == '+' || c == '-' || c == '*' || c == '/';
}
bool is_unary(char c) {
  return c == '+' || c=='-';
}
int priority (char op) {
  if (op < 0) return 3; // unary operator
  if (op == '+' || op == '-') return 1;
  if (op == '*' || op == '/') return 2;
  return -1;
}
void process_op(stack<int>& st, char op) {
  if (op < 0) {
    int l = st.top(); st.pop();
    switch (-op) {
      case '+': st.push(l); break;
      case '-': st.push(-l); break;
    }
  } else {
    int r = st.top(); st.pop();
    int l = st.top(); st.pop();
    switch (op) {
      case '+': st.push(l + r); break;
      case '-': st.push(l - r); break;
      case '*': st.push(l * r); break;
      case '/': st.push(l / r); break;
    }
  }
}

int evaluate(string& s) {
  stack<int> st;
  stack<char> op;
  bool may_be_unary = true;
  forn(i,sz(s)) {
    if (delim(s[i]))
      continue;

    if (s[i] == '(') {
      op.push('(');
      may_be_unary = true;
    } else if (s[i] == ')') {
      while (op.top() != '(') {
        process_op(st, op.top());
        op.pop();
      }
      op.pop();
      may_be_unary = false;
    } else if (is_op(s[i])) {
      char cur_op = s[i];
      if (may_be_unary && is_unary(cur_op))
        cur_op = -cur_op;
      while (sz(op) && (
          (cur_op >= 0 && priority(op.top()) >= priority(
             cur_op)) ||
              (cur_op < 0 && priority(op.top()) >
                 priority(cur_op))
              )) {
        process_op(st, op.top());
        op.pop();
      }
      op.push(cur_op);
      may_be_unary = true;
    } else {
      int number = 0;
      while (i < sz(s) && isalnum(s[i]))
        number = number * 10 + s[i++] - '0';
      --i;
      st.push(number);
      may_be_unary = false;
    }
  }

  while(sz(op)){
    process_op(st, op.top());
    op.pop();
  }
```

```
    return st.top();
  }
```

9.3 Ternary Search

```
  double ternary_search(double l, double r) {
    while (r - l > eps) {
      double m1 = l + (r - l) / 3;
      double m2 = r - (r - l) / 3;
      double f1 = f(m1), f2 = f(m2);
      if (f1 < f2) l = m1;
      else r = m2;
    }
    return f(l);                    //return the maximum of
        f(x) in [l, r]
  }
```

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

$\sum_{k=0}^{n} k = n(n+1)/2$

$\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$

$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$

$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$

$\sum_{k=0}^{n} k^4 = (6n^5 + 15n^4 + 10n^3 - n)/30$

$\sum_{k=0}^{n} k^5 = (2n^6 + 6n^5 + 5n^4 - n^2)/12$

$\sum_{k=0}^{n} x^k = (x^{n+1} - 1)/(x-1)$

$\sum_{k=0}^{n} kx^k = (x - (n+1)x^{n+1} + nx^{n+2})/(x-1)^2$

$1 + x + x^2 + \cdots = 1/(1-x)$

$\binom{n}{k} = \frac{n!}{(n-k)!k!}$

$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$

$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$

$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$

$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$

$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$

$12! \approx 2^{28.8}$

$20! \approx 2^{61.1}$

- 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 $n = 2^rs+1$ with odd s, and a random integer $1 < a < n$.
If $a^s \equiv 1 \pmod{n}$ or $a^{2^js} \equiv -1 \pmod{n}$ 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 $\sum_{d|n}\mu(d)f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n}\mu(d)^2f(d) =$

$\prod_{p|n}(1+f(p))$.
$\sum_{d|n}\mu(d) = e(n) = [n == 1]$.
$S_f(n) = \prod_{p=1}(1+f(p_i)+f(p_i^2)+...+f(p_i^{e_i}))$, 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 $z = 0, 1, \dots, n-1$, 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 $z! = \Gamma(z+1) = \sqrt{2\pi}\ z^{z+1/2}\ e^{-z}(1 + \frac{1}{12z} + \frac{1}{288z^2} - \frac{139}{51840z^3} + \ldots)$

Taylor series. $f(x) = f(a) + \frac{x-a}{1!}f'(a) + \frac{(x-a)^2}{2!}f^{(2)}(a) + \cdots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \ldots$.

$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$

$\ln x = 2(a + \frac{a^3}{3} + \frac{a^5}{5} + \ldots)$, where $a = \frac{x-1}{x+1}$. $\ln x^2 = 2\ln x$.

$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$, $\arctan x = \arctan c + \arctan \frac{x-c}{1+xc}$ (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

$p \to q \equiv \neg p \lor q$

$p \to q \equiv \neg q \to \neg p$

$p \lor q \equiv \neg p \to q$

$p \land q \equiv \neg(p \to \neg q)$

$\neg(p \to q) \equiv p \land \neg q$

$(p \to q) \land (p \to r) \equiv p \to (q \land r)$

$(p \to q) \lor (p \to r) \equiv p \to (q \lor r)$

$(p \to r) \land (q \to r) \equiv (p \land q) \to r$

$(p \to r) \lor (q \to r) \equiv (p \lor q) \to r$

$(p \land q) \lor (r \land s) \equiv (p \lor r) \land (p \lor s) \land (q \lor r) \land (q \lor s)$

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 $k = k_1^{p_1} * k_2^{p_2} \dots k_m^{p_m}$
  $x = min_{1 \leq j \leq m} \{\frac{a_j}{p_j}\}$ where a_j is Legendre's formula for k_j
- Divisor Formulas of $n!$ Find all prime numbers $\leq n$ $\{p_1, \dots, p_m\}$ 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.