Guarapo Enjoyers ICPC Team Notebook

Contents

1 C++

1.1 C++ template

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

#define endl '\n'
#define fi first
#define se second
#define pb push_back
#define umap unordered_map
#define all(v) v.begin(),v.end()
#define allr(v) v.rbegin(),v.rend()

typedef long long int ll;
typedef long double ld;

const int INF = 1e9;
const int MAX = 1e6+7;
const double PI = acos(-1);
const double EPS = 1e-9;

int dx[] = {1, 0, -1, 0};
int dy[] = {0, 1, 0, -1};

int main(){
  ios::sync_with_stdio(0);
```

```
  cin.tie(0); cout.tie(0);

  return 0;
}
```

2 Strings

2.1 Trie

```
const int K = 26;

struct Vertex {
  int next[K];
  bool output = false;
  Vertex() { fill(begin(next), end(next), -1);}
};

vector<Vertex> trie(1);

void add_string(string const& s) {
  int v = 0;
  for(char ch : s) {
    int c = ch - 'a';
    if (trie[v].next[c] == -1) {
      trie[v].next[c] = trie.size();
      trie.emplace_back();
    }
    v = trie[v].next[c];
  }
  trie[v].output = true;
}
```

2.2 Z's function

```
/*
    Dado un string S de longitud n retorna un arreglo Z
       de longitud n,
    donde el i-esimo termino del arreglo es la longitud
       del substring mas
    largo que empieza en la i-esima posicion de S que
       ademas es igual a
    un prefijo de S.

    ejemplo:
    S: j o s e l i j o s e j
    Z: 0 0 0 0 0 0 4 0 0 0 1

    tiempo: O(n)
*/

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

2.3 Manachers

```
/*
    Dado un string S de longitud n retorna dos arreglos
       de longitud n;
    donde el i-esimo elemento del primer arreglo es el
       numero de palindromos
    con centro par centrados en el i-esimo par (el par: i
       , i-1), y donde el
    el i-esimo elemento del segundo arreglo es el numero
       de palindromos
    con centro impar centrados en el i-esimo elemento.
    NOTA: los caracteres solos no son tomados como
       palindromos

    tiempo: O(n)
*/

    vector<vector<int>> p(2,vector<int>(n,0));
    for(int z=0,l=0,r=0;z<2;z++,l=0,r=0)
        for(int i=0;i<n;i++)
        {
            if(i<r) p[z][i]=min(r-i+!z,p[z][l+r-i+!z]);
            int L=i-p[z][i], R=i+p[z][i]-!z;
            while(L-1>=0 && R+1<n && s[L-1]==s[R+1]) p[z
               ][i]++,L--,R++;
            if(R>r) l=L,r=R;
        }
```

2.4 KMP

```
/*
    Dado un string S de longitud n retorna un arreglo PI
       de longitud n,
    donde el i-esimo termino es el la longitud del
       prefijo mas largo
```

```
    que tambien es sufijo del substring de S formado
       hasta el i-esimo
    termino desde el inicio de S.

    ejemplo:
    S: r a r a m e r a r a n t e
   PI: 0 0 1 2 0 0 1 2 3 4 0 0 0

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

2.5 Hashing

```
long long compute_hash(string const& s) {
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow)
            % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

2.6 Suffix Array

```
void radix_sort(vector<pair<pair<int, int>, int>> &arr) {
        // with radix sort, we actually have to sort by
           the second element first
        for (int i : vector<int>{2, 1}) {
                auto key = [&](const pair<pair<int, int>,
                    int> &x) {
                    return i == 1 ? x.first.first : x
                        .first.second;
                };
                int max = 0;
                for (const auto &i : arr) { max = std::
                    max(max, key(i)); }
                vector<int> occs(max + 1);
                for (const auto &i : arr) { occs[key(i)
                    ]++; }
                vector<int> start(max + 1);
                for (int i = 1; i <= max; i++) {
                        start[i] = start[i - 1] + occs[i
                            - 1];
                }
                vector<pair<pair<int, int>, int>> new_arr
                    (arr.size());
                for (const auto &i : arr) {
                        new_arr[start[key(i)]] = i;
                        start[key(i)]++;
                }
                arr = new_arr;
        }
}

str += '$';
const int n = str.size();  // just a shorthand

vector<pair<pair<int, int>, int>> suffs(n);
for (int i = 0; i < n; i++) { suffs[i] = {{str[i], str[i
    ]}, i}; }
std::sort(suffs.begin(), suffs.end());
vector<int> equiv(n);
for (int i = 1; i < n; i++) {
    auto [c_val, cs] = suffs[i];
    auto [p_val, ps] = suffs[i - 1];
    equiv[cs] = equiv[ps] + (c_val > p_val);
}

for (int cmp_amt = 1; cmp_amt < n; cmp_amt *= 2) {
    for (auto &[val, s] : suffs) {
        // the order numbers for the left half and right
           half respectively
        val = {equiv[s], equiv[(s + cmp_amt) % n]};
    }
    // without the radix sort optimization, we would use
       'std::sort'
    radix_sort(suffs);

    // assign numbers to the newly sorted suffixes
    for (int i = 1; i < n; i++) {
        auto [c_val, cs] = suffs[i];
        auto [p_val, ps] = suffs[i - 1];
        equiv[cs] = equiv[ps] + (c_val > p_val);
    }
}

for (int i = 1; i < n; i++) {
    cout << suffs[i].second << " \n"[i == n - 1];
}
```

3 Graph algorithms

3.1 Topological sort

```
vector<int> adj[N];
vector<bool> visited[N];
stack<int> s;

void dfs_topo(int v){
  if(visited[v]) return;
  visited[v] =  true;
  for(auto u: adj[v]) dfs_topo(u);
  s.push(v);
}

void topo(){
  for(int i = 0; i < N; i++){
    if(!visited[i]) dfs_topo(i);
  }
}
```

3.2 Dijkstra

```
vector<int> dijkstra(int n, const vector<vector<pair<int,
    int>>>& g, int ni) {
    vector<int> d(n, INT_MAX);
    priority_queue<pair<int, int>> q;
    q.push(make_pair(0, ni));
    d[ni] = 0;
    while (!q.empty()) {
        int n = q.top().second;
        q.pop();
        for (int i = 0; i < g[n].size(); i++) {
            int n2 = g[n][i].first;
            int pe = g[n][i].second;
            if (d[n] + pe < d[n2]) {
                d[n2] = d[n] + pe;
                q.push(make_pair(-d[n2], n2));
            }
        }
    }
    return d;
}
```

3.3 Floyd Warshall

```
void floyd_warshall(vector<vector<int>>& dist) {
    int n = dist.size();
    for (int k = 0; k < n; ++k) {
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (dist[i][k] < INF && dist[k][j] < INF)
                    {
                    dist[i][j] = min(dist[i][j], dist[i][
                        k] + dist[k][j]);
                }
            }
        }
    }
}
```

3.4 Kosaraju: Strongly connected components

```
void dfs_topo(int v, vector<int> adj[], vector<bool> &
    visited, stack<int> &s){
  if(visited[v]) return;
  visited[v] = true;
  for(auto u: adj[v]) dfs_topo(u, adj, visited, s);
  s.push(v);
}

void dfs2(int v, vector<int> adj_rev[], vector<bool> &
    visited, vector<int> &component){
  if(visited[v]) return;
  visited[v] = true;
  component.push_back(v);
  for(auto u: adj_rev[v]) dfs2(u, adj_rev, visited,
      component);
}

void kosajaru(){
  int n;
  vector<int> adj[n], adj_rev[n];
  vector<bool> visited(n, 0);
  vector<int> component;
  stack<int> s;

  for(int k = 0; k < n; k++){
    if(!visited[k]) dfs_topo(k, adj, visited, s);
  }

  visited.assign(n, 0);

  while(!s.empty()){
    if(!visited[s.top()]){
      dfs2(s.top(), adj_rev, visited, component);
      //use the components
      component.clear();
    }
    s.pop();
  }

}
```

3.5 Tarjan: Strongly connected components

```
vector<int> adj[MAXN];
vector<int> low, num, cmp;
stack<int> st;
int scc, timer;

void dfs(int u){
  low[u] = num[u] = timer++;
  st.push(u);
  for(auto v: adj[u]){
    if(num[v]==-1) dfs(v);
    if(cmp[v]==-1) low[u] = min(low[u], low[v]);
  }
  int v;
  if(low[u]==num[u]){
    do{
      v = st.top();
      st.pop();
      cmp[v] = scc;
    }while(u!=v);
    ++scc;
  }
}

void tarjan(int n){
  timer = scc = 0;
  num = low = cmp = vector<int>(n, -1);
  for(int i = 0; i < n; i++)
    if(num[i]==-1) dfs(i);
}
```

3.6 MST Kruskal

```
int id[tam];

void init(int n) {
    for (int i = 0; i <= n; i++) id[i] = i;
}

int dset(int a) {
    if (id[a] == a) return a;
    return id[a] = dset(id[a]);
}

void unir(int a, int b) {
    id[dset(b)] = id[dset(a)];
}

int kruskal(int n, int m, vector<pair<int, pair<int, int
    >>>& g) {
    sort(g.begin(), g.end());
    init(n);
    int ans = 0;
    for (int i = 0; i < m; i++) {
        int u = g[i].second.first;
        int v = g[i].second.second;
        int w = g[i].first;
        if (dset(u) != dset(v)) {
            ans += w;
            unir(u, v);
        }
    }
    return ans;
}
```

3.7 Centroid Decomposition

```
const int MAXN = 1e5, K = 25;

int sz[MAXN], depth[MAXN], pa[MAXN], dist[K][MAXN];
vector<int> adj[MAXN];

int dfs(int u, int dep=-1, bool flag=0, int dis = 0, int
    p=-1){
  sz[u] = 1;
  if(flag) dist[dep][u] = dis;
  for(auto v: adj[u]){
    if(!depth[v] && v != p)
      sz[u] += dfs(v, dep, flag, dis+1, u);
  }
  return sz[u];
}

int centroid(int u, int r, int p=-1){
  for(auto v: adj[u]){
    if(!depth[v] && v != p && sz[v] > r)
      return centroid(v, r, u);
  }
  return u;
}

int decompose(int u, int d = 1){
  int n = dfs(u);
  int c = centroid(u, n >> 1);
  depth[c] = d;
  dfs(c, d); // if distances is needed
  for(auto v: adj[c]){
    if(!depth[v])
      pa[decompose(v, d+1)] = c;
  }
  return c;
}

int lca(int u, int v){
  for(; u != v; u = pa[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];
}
```

4 Flows

4.1 Edmons-Karp

```
int n;
vector<int> capacity[];
vector<int> adj[];

//complexity O(V*E^2)

int bfs(int s, int t, vector<int>& parent){
  fill(parent.begin(), parent.end(), -1);
  parent[s] = -2;
  queue<pair<int,int>> q;
  q.push({s, INF});

  while(!q.empty()){
    int cur = q.front().first;
    int flow = q.front().second;
    q.pop();
    for(auto nxt: adj[cur]){
      if(parent[nxt] == -1 && capacity[cur][nxt]){
        parent[nxt] = cur;
        int new_flow = min(flow, capacity[cur][nxt]);
        if(nxt == t) return new_flow;
        q.push({nxt, new_flow});
      }
    }
  }

  return 0;
}

int maxflow(int s, int t){
  int flow = 0;
  vector<int> parent(n);
  int new_flow;

  while(new_flow = bfs(s, t, parent)){
    flow += new_flow;
    int cur = t;
    while(cur != s){
      int prev = parent[cur];
      capacity[prev][cur] -= new_flow;
      capacity[cur][prev] += new_flow;
      cur = prev;
    }
  }
  return flow;
}
```



```
}
```

5 Data Structures

5.1 Disjoin Set Union

```
int padre[150005], fam[150005];

void init()
{
    for (int i = 0; i < 150005; i ++)
    {
        padre[i] = i;
        fam[i] = 1;
    }
}

int Find(int a)
{
    if (padre[a] == a)
    {
        return a;
    }
    return padre[a] = Find(padre[a]);
}

void unir(int a, int b)
{
    int A = Find(a);
    int B = Find(b);
    if (A != B)
    {
        padre[A] = padre[B];
        fam[B] += fam[A];
        fam[A] = 0;
    }
}
```

5.2 Fenwick Tree

```
int BIT[1000], a[1000], n;
void update(int x, int delta) //pos, val
{
      for(; x <= n; x += x&-x)
        BIT[x] += delta;
}
int query(int x) //Range Q: query(r) - query(l-1)
{
     int sum = 0;
     for(; x > 0; x -= x&-x)
        sum += BIT[x];
     return sum;
}
```

5.3 Segment Tree

```
const int N = 1e5;
int t[4 * N];

int merge(int nodeL, int nodeR){
  return nodeL + nodeR;
}

void build(int a[], int v, int tl, int tr){
  if(tl == tr) t[v] = a[tl];
  else{
    int tm = (tl + tr)/2;
    build(a, v*2, tl , tm);
    build(a, v*2+1, tm+1, tr);
    t[v] = merge(t[v*2], t[v*2+1]);
  }
}

int query(int v, int tl, int tr, int l, int r){
  if(l > r) return 0;
  if(l == tl && r == tr) return t[v];
  int tm = (tl + tr)/2;
  int nodeL = query(v*2, tl, tm, l, min(r, tm) );
  int nodeR = query(v*2+1, tm+1, tr, max(l, tm+1), r);
  return merge(nodeL, nodeR);
}

void update(int v, int tl, int tr, int pos, int new_val){
  if(tl == tr) t[v] = new_val;
  else{
    int tm = (tl + tr)/2;
    if(pos <= tm) update(v*2, tl, tm, pos, new_val);
    else update(v*2+1, tm+1, tr, pos, new_val);
    t[v] = merge(t[v*2], t[v*2+1]);
  }
}
```

5.4 Segment Tree Lazy

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

#define endl '\n'
#define PI acos(-1)
#define ll long long int
#define ld long double

const int INF = 1e9+7;

const int N = 1e5 + 7;

int t[4*N];
int lazy[4*N];

void build(int a[], int v, int tl, int tr){
  if(tl == tr) t[v] = a[tl];
  else{
    int tm = (tl+tr)/2;
    build(a, v*2, tl, tm);
    build(a, v*2+1, tm+1, tr);
    t[v] = merge(t[v*2], t[v*2]+1);
  }
}

void push(int v){
  t[v*2] += lazy[v];
  lazy[v*2] += lazy[v];
  t[v*2+1] += lazy[v];
  lazy[v*2+1] += lazy[v];
  lazy[v] = 0;
}

void update(int v, int tl, int tr, int l, int r, int add)
  {
  if(l > r) return;
  if(l == tl && r == tr){
    t[v] += add;
    lazy[v] += add;
  }else{
    push(v);
    int tm = (tl+tr)/2;
    update(v*2, tl, tm, l, min(r, tm), add);
    update(v*2+1, tm+1, tr, max(l, tm+1), r, add);
    t[v] = merge(t[v*2], t[v*2+1]);
  }
}

int query(int v, int tl, int tr, int l, int r){
  if(l > r) return -INF;
  if(l == tl && tr == r) return t[v];
  push(v);
  int tm = (tl+tr)/2;
  int nl = query(v*2, tl, tm, l, min(r, tm));
  int nr = query(v*2, tm+1, tr, max(tm+1, l), r);
  return merge(nl, nr);
}

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


}
```

5.5 Segment Tree 2D

```
const int N = 1000;
int t[4*N][4*N];
```

```
int a[N][N];

int merge(int nodeL, int nodeR){
  return nodeL + nodeR;
}

void build_y(int vx, int lx, int rx, int vy, int ly, int
    ry){
  if(ly == ry){
    if(lx == rx) t[vx][vy] = a[lx][ly];
    else t[vx][vy]= merge(t[vx*2][vy], t[vx*2+1][vy]);
  }else{
    int my = (ly + ry)/2;
    build_y(vx, lx, rx, vy*2, ly, my);
    build_y(vx, lx, rx, vy*2+1, my+1, ry);
    t[vx][vy] = merge(t[vx][vy*2], t[vx][vy*2+1]);
  }
}

void build_x(int vx, int lx, int rx){
        if(lx != rx){
                int mx = (lx + rx)/2;
                build_x(vx*2, lx, mx);
                build_x(vx*2+1, mx+1, rx);
        }
        build_y(vx, lx, rx, 1, 0, N-1);
}

int query_y(int vx, int vy, int tly, int try_, int ly,
    int ry){
        if(ly > ry) return 0;
        if(ly == tly && try_ == ry) return t[vx][vy];
        int tmy = (tly, try_)/2;
        int nodeL_y = query_y(vx, vy*2, tly, tmy, ly, min
            (ry, tmy));
        int nodeR_y = query_y(vx, vy*2+1, tmy+1, try_,
            max(ly, tmy+1), ry);
        return merge(nodeL_y, nodeR_y);
}

int query_x(int vx, int tlx, int trx, int lx, int rx, int
     ly, int ry){
        if(lx > rx) return 0;
        if(lx == tlx && trx == rx) return query_y(vx, 1,
            0, N-1, ly, ry);
        int tmx = (tlx + trx)/2;
        int nodeL_x = query_x(vx*2, tlx, tmx, lx, min(rx,
             tmx), ly, ry);
        int nodeR_x = query_x(vx*2+1, tmx+1, trx, max(lx,
             tmx+1), rx, ly, ry);
        return merge(nodeL_x, nodeR_x);
}

void update_y(int vx, int lx, int rx, int vy, int ly, int
     ry, int x, int y, int new_val){
        if(ly == ry){
                if(lx == rx) t[vx][vy] = new_val;
                else t[vx][vy] = merge(t[vx*2][vy], t[vx
                    *2+1][vy]);
        }else{
                int my = (ly + ry)/2;
                if(y <= my) update_y(vx, lx ,rx, vy*2, ly
                    , my, x , y, new_val);
                else update_y(vx, lx, rx, vy*2+1, my+1,
                    ry, x, y, new_val);
                t[vx][vy] = merge(t[vx][vy*2], t[vx][vy
                    *2+1]);
        }
}

void update_x(int vx, int lx, int rx, int x, int y, int
    new_val){
        if(lx != rx){
                int mx = (lx + rx)/2;
                if(x <= mx) update_x(vx*2, lx, mx, x, y,
                    new_val);
                else update_x(vx*2+1, mx+1, rx, x, y,
                    new_val);
        }
        update_y(vx, lx, rx, 1, 0, N-1, x, y, new_val);
}
```

5.6 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;

//Nota:
//1. Para un set decreciente usar greater<int>
//2. Para un multiset cambiar less<int> por less_equal<
    int>
//  2.1 Para borrar un elemento
//  int idx = st.order_of_key(value);
//  st.erase(st.find_by_order(idx));

st.find_by_order(k); // return pointer to the k-th
st.order_of_key(x); // return how many elements are
    smaller than x
```

5.7 RMQ

```
const int MAXN = 1e5 + 10, K = 25; // K has to satisfy K
    > log MAXN + 1
ll st[K+1][MAXN];

//[0, .., N-1];
ll query(int l, int r){
```

```
  int ln = r-l+1;
  int k = 0;
  while((1 << (k+1)) <= ln) k++;
  return min(st[k][l], st[k][r -(1<<k)+1]);
}

void sparse_table(){
  vector<ll> a;
  int N = a.size();

  for(int i = 0; i < N; i++) st[0][i] = a[i];

  for(int i = 1; i <= K; i++){
    for(int j = 0; j + (1 << i) <= N; j++){
      st[i][j] = min(st[i-1][j], st[i-1][j + (1 << (i-1))
          ]);
    }
  }
}
```

6 Math

6.1 Sieve of Eratosthenes

```
int tam = 1000000;
vector < bool > criba(1000000, true);

void init()
{
    criba[0] = false;
    criba[1] = false;
    for (int i = 2; i <= sqrt(tam); i ++)
    {
        if (!criba[i]) continue;
        for (int j = 2; i*j <= tam; j ++)
        {
            criba[i*j] = false;
        }
    }
    return;
}
```

6.2 Extended Euclidean (Diophantine)

```
int gcde(int a, int b, int &x, int &y){
  x = 1, y = 0;
  int x1 = 0, y1 = 1;
  int a1 = a, b1 = b;
  while(b1){
    int 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_sol(int a, int b, int c, int& x0, int &y0,
    int& 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;
}
```

6.3 Inversa Modular

```
void inv_to_m(int m){
  vector<int> invs(m);
  invs[1] = 1;
  for(int i = 2; i < m; i++)
    invs[i] = m - (m/i) * invs[m%i] % m;
}

void inv(int a, int b){
  int x, y, m;
  int g = gcde(a, b, x, y);

  if(g!=1) cout << "No solution";
  else x = (x%m+m)%m;
  cout << x;
}
```

6.4 Euler's Totient Function

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

//complexity O(n log log n)
void phi_1_to_n(int n ){
  vector<int> phi(n+1);
  for(int i = 0; i <= n; i++)
```

```
    phi[i] = i;

  for(int i = 2; i <= n; i++){
    if(phi[i] == i){
      for(int j = 1; j <= n; j += i)
        phi[j] -= phi[j] / i;
    }
  }
}
```

6.5 FFT

```
#define cd complex<double>
const double PI = acos(-1);

void fft(vector<cd>& a, bool invert){
  int n = a.size();

  for(int i = 1, j = 0; i < n; i++){
    int bit = n >> 1;
    for(; j & bit; bit >>= 1)
      j ^= bit;
    j ^= bit;
    if(i < j) swap(a[i], a[j]);

  }

  for(int len = 2; len <= n; len <<= 1){
    double ang = 2 * PI / len * (invert? -1: 1);
    cd wlen(cos(ang), sin(ang));
    for(int i = 0; i < n; i += len){
      cd w(1);
      for(int j = 0; j < len/2; j++){
        cd u = a[i+j], v = a[i+j+len/2] * w;
        a[i+j] = u+v;
        a[i+j+len/2] = u-v;
        w *= len;
      }
    }
  }

  if(invert)
    for(cd& x: a) x/=n;
}
```

6.6 Multiply

```
vector<int> multiply(vector<int> a, vector<int> b){
  vector<cd> fa(a.begin(), a.end());
  vector<cd> fb(b.begin(), b.end());

  int n = 1;
  while(n < a.size() + b.size()) n << 1;
  fa.resize(n);

  fb.resize(n);

  fft(fa, false);
  fft(fb, false);
  for(int i = 0; i < n; i++)
    fa[i] += fb[i];
  fft(fa, true);

  vector<int> result(n);
  for(int i = 0; i < n; i++)
    result[i] = round(fa[i].real());

  //for multiplying two long numbers
  /*
  int carry = 0;
  for(int i = 0; i < n; i++){
    result[i] +=  carry;
    carry = result[i]/10;
    result[i] %= 10;
  }
  */

  return result;
}
```

6.7 Miller Rabin Test

```
using u128 = __uint128_t;

unsigned long long int binpower(unsigned long long int
    base, unsigned long long int e, unsigned long long int
     mod) {
    unsigned long long int result = 1;
    base %= mod;
    while (e) {
        if (e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(unsigned long long int n, unsigned
    long long int a, unsigned long long int d, int s) {
    unsigned long long int x = binpower(a, d, n);
    if (x == 1 || x == n - 1)
        return false;
    for (int r = 1; r < s; r++) {
        x = (u128)x * x % n;
        if (x == n - 1)
            return false;
    }
    return true;
};
```

```
bool MillerRabin(unsigned long long int n) {
    if (n < 2)
        return false;

    int r = 0;
    unsigned long long int d = n - 1;
    while ((d & 1) == 0) {
        d >>= 1;
        r++;
    }

    for (int a : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,
        37}) {
        if (n == a)
            return true;
        if (check_composite(n, a, d, r))
            return false;
    }
    return true;
}
```

6.8 Binary Exponentiation

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

7 Dynamic Programming

7.1 Longest increasing subsequence

```
int lis(vector<int> a){
  int n = a.size();
  vector<int> d(n+1, INF);
  d[0] = -INF;

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

  int ans = 0;
  for(int l = 1; l <= n; l++){
    if(d[l] < INF)
      ans = l;
  }
  return ans;
}
```

7.2 Max 2D Range Sum

```
int max_2d_range_sum(vector<vector<int>>& A) {
    int n = A.size();
    vector<vector<int>> prefixSum = A;

    for (int i = 0; i < n; ++i) {
        for (int j = 1; j < n; ++j) {
            prefixSum[i][j] += prefixSum[i][j - 1];
        }
    }

    int maxSubRect = -127 * 100 * 100; //Minimum possible
        value

    for (int l = 0; l < n; ++l) {
        for (int r = l; r < n; ++r) {
            int subRect = 0;
            // Kadane's by row
            for (int row = 0; row < n; ++row) {
                if (l > 0) subRect += prefixSum[row][r] -
                    prefixSum[row][l - 1];
                else subRect += prefixSum[row][r];
                if (subRect < 0) subRect = 0;
                maxSubRect = max(maxSubRect, subRect);
            }
        }
    }
    return maxSubRect;
}
```

8 Geometry

8.1 Point

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

#define endl '\n'
#define PI acos(-1)
#define ll long long int
#define ld long double

const int INF = 1e9+7;

const ld EPS = 1e-9;
```

```
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

  // Aritmetic operations
  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); }

  // Dot product and cross product
  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) {return !operator==(p); }
  bool operator<(pt p) const{
    return x < p.x - EPS || (abs(x - p.x) <= EPS && y < p
        .y - EPS);
  }

  // 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); }
  bool left(pt p, pt q) {return side(p, q) > EPS;}//>= -
      EPS For collinear

  // Angles
  // Angle from origin
  ld angle() {return atan2(y, x); } //[-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){ // BA-BT
    ld ma = (a-b).min_angle(*this -b);
    return side(a, b) * (CW? -1: 1) <= 0 ? ma : 2*PI - ma
        ;
  } // Angle < AB(*this) > in direction CW
  bool in_angle(pt a, pt b, pt c, bool CW = 1) {
    return angle(a, b, CW) <= c.angle(a, b, CW);
  } // Is pt inside infinite angle ABC

  // Rotations
  // use ccw90(1, 0), cw90(-1, 0)
  pt rot(pt r) {return pt(*this %r, *this * r); }
  // CCW, ang(radians)
  pt rot(ld a) {return rot(pt(sin(a), cos(a))); }
  pt rot_around(ld a, pt p) { return p + (*this - p).rot(
      a);}

  //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);
  if(sgn(sa)*sgn(sb) < 0 && sgn(sc)*sgn(sd) < 0)
    out.push_back((a*sb - b*sa) / (sb-sa));

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

8.2 Convex Hull

```
int orientation(pt a, pt b, pt c){
  ld v = a%b + b%c + c%a;
  if(v < -EPS) return -1; //CW
  if(v > EPS) return +1; //CCW
  return 0;
}

bool cw(pt a, pt b, pt c, bool coll){
  int o = orientation(a, b, c);
  return o < 0 || (coll && o == 0);
}

bool collinear(pt a, pt b, pt c){
  return orientation(a, b, c) == 0;
}

void convex_hull(vector<pt>& a, bool coll = false){
  pt p0 = *min_element(a.begin(), a.end());
  sort(a.begin(), a.end(), [&p0](const pt& a, const pt& b
      ){
    int o = orientation(p0, a, b);
    if (o == 0)
      return (p0 - a).norm2() < (p0 - b).norm2();
    return o < 0;
  });
```

```
if(coll){
  int i = a.size() -1;
  while(i >= 0 && collinear(p0, a[i], a.back())) i--;
  reverse(a.begin()+i+1, a.end());
}

vector<pt> st;
for(int i = 0; i < a.size(); i++){
  while(st.size() > 1 && ! cw(st[st.size()-2], st.back
      (), a[i], coll))
        st.pop_back();
      st.push_back(a[i]);
    }

    a = st;
  }
```

9 Miscellaneous

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: $\left(\frac{a}{p}\right) = a^{(\frac{p-1}{2})} \pmod p$.

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 $i = \mathrm{ind}_g(a)$ modulo $\phi(m)$, such that $g^i \equiv a \pmod m$. $\mathrm{ind}_g(a)$ has logarithm-like properties: $\mathrm{ind}(1) = 0$, $\mathrm{ind}(ab) = \mathrm{ind}(a) + \mathrm{ind}(b)$.

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

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

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

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

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

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

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

String Algorithms

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

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

Graph Theory

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

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

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

Euler tours. Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists

iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected. Construction:

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

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

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

Tarjan's offline LCA algorithm. (Based on DFS and union-find structure.)

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

Strongly-connected components. Kosaraju's algorithm.

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

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

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

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

Erdos-Gallai theorem. A sequence of integers $\{d_1, d_2, \ldots, d_n\}$, 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 $k = 1, 2, \ldots, n-1$.

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

Bit tricks

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

Math

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

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

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

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

$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots$, $\arctan x = \arctan c + \arctan \frac{x-c}{1+xc}$ (e.g c=.2)

$\pi = 4\arctan 1$, $\pi = 6\arcsin \frac{1}{2}$

Fibonacci Period Si p es primo , $\pi(p^k) = p^{k-1}\pi(p)$

$\pi(2) = 3$ $\pi(5) = 20$

Si n y m son coprimos $\pi(n*m) = lcm(\pi(n), \pi(m))$

List of Primes

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

2-SAT Rules

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

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

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

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

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

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

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

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

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

$(p \wedge q) \vee (r \wedge s) \equiv (p \vee r) \wedge (p \vee s) \wedge (q \vee r) \wedge (q \vee s)$

Summations

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

Compound Interest

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

Great circle distance or geographical distance

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

Theorems

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

Law of sines and cosines

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

Heron's Formula

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

Legendre's Formula Largest power of k, x, such that $n!$ is divisible by k^x

- If k is prime, $x = \frac{n}{k} + \frac{n}{k^2} + \frac{n}{k^3} + \dots$

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