Perritos Malvados ICPC Team Notebook

Contents

1 Strings

1.1 Aho-Corasick

```cpp
struct vertex {
  map<char,int> next,go;
  int p,link;
  char pch;
  vector<int> leaf; // se puede cambiar por int, en ese
      caso int leaf y leaf(0) en constructor
  vertex(int p=-1, char pch=-1):p(p),pch(pch),link(-1){}
};
vector<vertex> t;
void aho_init(){ //do not forget!!
  t.clear();t.pb(vertex());
}
void add_string(string s, int id){
  int v=0;
  for(char c:s){
    if(!t[v].next.count(c)){
      t[v].next[c]=t.size();
      t.pb(vertex(v,c));
    }
    v=t[v].next[c];
  }
  t[v].leaf.pb(id);
}
int go(int v, char c);
int get_link(int v){
  if(t[v].link<0)
    if(!v||!t[v].p)t[v].link=0;
    else t[v].link=go(get_link(t[v].p),t[v].pch);
  return t[v].link;
}
int go(int v, char c){
  if(!t[v].go.count(c))
    if(t[v].next.count(c))t[v].go[c]=t[v].next[c];
    else t[v].go[c]=v==0?0:go(get_link(v),c);
  return t[v].go[c];
}
```

1.2 Hashing

```cpp
const int K = 2;
struct Hash {
  const ll MOD[K] = {999727999, 1070777777};
  const ll P = 1777771;
  vector<ll> h[K], p[K];
  Hash(string &s) {
    int n = s.size();
    for(int k = 0; k < K; k++) {
      h[k].resize(n + 1, 0);
      p[k].resize(n + 1, 1);
      for(int i = 1; i <= n; i++) {
        h[k][i] = (h[k][i - 1] * P + s[i - 1]) % MOD[k];
        p[k][i] = (p[k][i - 1] * P) % MOD[k];
      }
    }
  }
  vector<ll> get(int i, int j) { // hash [i, j]
    j++;
    vector<ll> r(K);
    for(int k = 0; k < K; k++) {
      r[k] = (h[k][j] - h[k][i] * p[k][j - i]) % MOD[k];
      r[k] = (r[k] + MOD[k]) % MOD[k];
    }
    return r;
  }
};
```

1.3 KMP

```cpp
// pf[i] = longest proper prefix of s[0..i] which is also
    a suffix of it
// a b a a b c a
// 0 0 1 1 2 0 1
vi prefix_function(string &s) {
  int n = s.size();
  vi pf(n);
  pf[0] = 0;
  for (int i = 1, j = 0; i < n; i++) {
    while (j && s[i] != s[j]) j = pf[j - 1];
    if (s[i] == s[j]) j++;
    pf[i] = j;
  }
  return pf;
}

// numero de ocurrencias de p en s
int kmp(string &s, string &p) {
  int n = s.size(), m = p.size(), cnt = 0;
  vector<int> pf = prefix_function(p);
  for (int i = 0, j = 0; i < n; i++) {
    while (j && s[i] != p[j]) j = pf[j - 1];
    if (s[i] == p[j]) j++;
    if (j == m) {
      cnt++;
      j = pf[j - 1];
    }
  }
  return cnt;
}
```

1.4 KMP Automata

```cpp
// nodo 0 es el nodo inicial, significa que no matcheo
   nada
// nodo s.size() es el nodo final, significa que matcheo
   todo
const int MAXN = 1e5 + 5, alpha = 26;
```

```cpp
const char L = 'A'; // ojo aqui es el elemento mas bajo del alfabeto
int go[MAXN][alpha]; // go[i][j] = a donde vuelvo si estoy en i y pongo una j

void build(string &s) {
  int lps = 0;
  go[0][s[0] - L] = 1;
  int n = s.size();
  for (int i = 1; i < n + 1; i++) {
    for (int j = 0; j < alpha; j++) {
      go[i][j] = go[lps][j];
    }
    if (i < n) {
      go[i][s[i] - L] = i + 1;
      lps = go[lps][s[i] - L];
    }
  }
}
```

1.5 Manacher

```cpp
/*
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 centrado entre [i-1,i]
Time: O(n)
*/
void manacher(string &s, int f, vi &d) {
  int l = 0, r = -1, n = s.size();
  d.assign(n, 0);
  for (int i = 0; i < n; i++) {
    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;
  }
}
```

1.6 Suffix Array

```cpp
#define vi vector<int>
#define sz(x) (int)x.size()
#define all(x) x.begin(),x.end()
#define S second
#define F first
#define rep(i,a,b) for(int i=a;i<b;i++)
using namespace std;

int st[(1<<20)][20];

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

  void initST() {
    int n = sz(lcp);
    for (int i = 0; i < n; i++) st[i][0] = lcp[i];
    for (int j = 0; j < 20; j++) {
      for (int i = 0; i + (1 << j) <= n; i++) {
        st[i][j + 1] = min(st[i][j], st[i + (1 << j)][j]);
      }
    }
  }
  int RMQ(int izq, int der) {
    int k = 31 - __builtin_clz(der - izq + 1);
    return min(st[izq][k], st[der - (1 << k) + 1][k]);
  }
  int getLCP(int izq, int der) {
    if (izq == der) return sz(s) - izq;
    izq = rank[izq], der = rank[der];
    if (izq > der) swap(izq, der);
    return RMQ(izq + 1, der);
  }
};

// compare substring A and B
bool cmp(pair<int, int> A, pair<int, int> B) {
  int Q = SA.getLCP(A.first, B.first);
  if (Q >= min(A.S - A.F + 1, B.S - B.F + 1)) {
    if (A.S - A.F == B.S - B.F) return A.F < B.F;
```

```cpp
      return A.S - A.F < B.S - B.F;
    }
    return SA.rank[A.F] < SA.rank[B.F];
  }
  //      012345  6
  //      ababba  #
  // sa   = 6 5 0 2 4 1 3
  // lcp  = 0 0 1 2 0 2 1
  // rank = 2 5 3 6 4 1    posicion del sufixx i en el sa
  // lcp[i] = lcp(sa[i],sa[i-1])
  signed main() {
    string s;
    cin >> s;
    SuffixArray sa;
    sa.s = s;
    sa.init();
    return 0;
  }
```

1.7 Suffix Autómata

```cpp
/*
Automata tiene 2*|s| nodos
sa[nodo].cnt = cantidad de matcheos de este nodo, es
   decir cantidad de substrings matcheados hasta nodo
para calcular cantidad de strings que se pueden llehar
   desde un nodo hacer una dp
dp[nodo] = sa[nodo].cnt + dp[hijo_s]
R = sa[nodo].len = substring mas grande matcheado (camino
    mas largo)
L = sa.sa[sa.sa[nodo].link].len+1; es el camino mas corto
    desde 0 hasta nodo. desde cero hasta nodo hay
   substrings de sz [L,R] cada camino matchea un
   substring distinto.

Todos los substrings hasta nodo/estado tienen la misma
   cantidad de ocurrencias en el string sa[nodo].cnt

Todos los strings correspondientes a un nodo son sufijos
   distintos del substring len.

El nodo inicial del SA es 0 y te mueves con sa[nodo].nxt[
   c] solo si sa[nodo].nxt.count(c)
*/

struct suffix_automaton {
  struct node {
    int len, link, cnt, first_pos;
    bool end; // cnt is endpos size, first_pos is minimum
        of endpos
    map<char, int> next;
  };
  vector<node> sa;
  int last;
  suffix_automaton() {}
  suffix_automaton(string &s) {
    sa.reserve(s.size() * 2);
    last = add_node();
    sa[last].len = sa[last].cnt = sa[last].first_pos = sa
       [last].end = 0;
    sa[last].link = -1;
    for (char c : s) sa_append(c);
    mark_suffixes();
    build_cnt();
  }
  int add_node() {
    sa.push_back({});
    return sa.size() - 1;
  }
  void mark_suffixes() {
    // t0 is not suffix
    for (int cur = last; cur; cur = sa[cur].link)
      sa[cur].end = 1;
  }
  // This is O(N*log(N)). Can be done O(N) by doing dfs
     and counting paths to terminal nodes. a veces no es
     necesario el cnt
  void build_cnt() {
    vector<int> order(sa.size() - 1);
    iota(order.begin(), order.end(), 1);
    sort(order.begin(), order.end(), [&](int a, int b) {
        return sa[a].len > sa[b].len; });
    for (auto &i : order) sa[sa[i].link].cnt += sa[i].cnt
       ;
    sa[0].cnt = 0; // t0 is empty string
  }
  void sa_append(char c) {
    int cur = add_node();
    sa[cur].len = sa[last].len + 1;
    sa[cur].end = 0;
    sa[cur].cnt = 1;
    sa[cur].first_pos = sa[cur].len - 1;
    int p = last;
    while (p != -1 && !sa[p].next[c]) {
      sa[p].next[c] = cur;
      p = sa[p].link;
    }
    if (p == -1) sa[cur].link = 0;
    else {
      int q = sa[p].next[c];
      if (sa[q].len == sa[p].len + 1) sa[cur].link = q;
      else {
        int clone = add_node();
        sa[clone] = sa[q];
        sa[clone].len = sa[p].len + 1;
        sa[clone].cnt = 0;
        sa[q].link = sa[cur].link = clone;
        while (p != -1 && sa[p].next[c] == q) {
          sa[p].next[c] = clone;
          p = sa[p].link;
```

```
        }
      }
    }
    last = cur;
  }
  node& operator[](int i) { return sa[i]; }

  // esto no es necesario, es chanchulla
  // cantidad de apariciones de algun shift ciclico de p
     en s (automata del string s)
  int count_cyclic_shifts(string &p) {
    int m = p.size(), ans = 0, cur = 0, len = 0;
    if (m > sa.size()) return 0;

    vector<bool> vis(sa.size(), false);
    vector<int> nodes;
    string s = p + p;
    for (int i = 0; i < s.size(); ++i) {
      while (cur != -1 && !sa[cur].next.count(s[i]))
        cur = sa[cur].link, len = sa[cur].len;
      if (cur != -1)
        cur = sa[cur].next[s[i]], ++len;
      while (cur > 0 && m < sa[sa[cur].link].len + 1)
        cur = sa[cur].link, len = sa[cur].len;
      if (i >= m - 1 && cur > 0 && len >= m && !vis[cur])
          {
        ans += sa[cur].cnt;
        vis[cur] = true;
        nodes.push_back(cur);
      }
    }
    for (int x : nodes) vis[x] = false;
    return ans;
  }
};
```

1.8 Z Algorithm

```
// TIme: O(|s|)
// Maximum length of a substring that begins at position
   i and is a prefix of the string.
// a b a a b c a
// 0 0 1 2 0 0 1
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(0ll, 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 Graph algorithms

2.1 2-SAT

```
/*
indexado en 0
Time complexity: O(N)
Se puede usar desde index 0 en los nodos y la
   inicializacion tampoco es estricta e.g. sat2 S(n+5)

Notas.- En problemas de direccionar aristas e.g. grado
   salida = grado entrada
*/
struct sat2 {
  int n;
  vector<vector<vector<int>>> g;
  vector<int> tag;
  vector<bool> seen, value;
  stack<int> st;
  sat2(int n) : n(n), g(2, vector<vector<int>>(2*n)), tag
     (2*n), seen(2*n), value(2*n) { }
  int neg(int x) { return 2*n-x-1; }
  void add_or(int u, int v) { implication(neg(u), v); }
  void make_true(int u) { add_edge(neg(u), u); }
  void make_false(int u) { make_true(neg(u)); }
  void eq(int u, int v) {
    implication(u, v);
    implication(v, u);
  }
  void diff(int u, int v) { eq(u, neg(v)); }
  void implication(int u, int v) {
    add_edge(u, v);
    add_edge(neg(v), neg(u));
  }
  void add_edge(int u, int v) {
    g[0][u].push_back(v);
    g[1][v].push_back(u);
  }
  void dfs(int id, int u, int t = 0) {
    seen[u] = true;
    for(auto& v : g[id][u])
      if(!seen[v])
        dfs(id, v, t);
    if(id == 0) st.push(u);
    else tag[u] = t;
  }
  void kosaraju() {
    for(int u = 0; u < n; u++) {
      if(!seen[u]) dfs(0, u);
```

```
      if(!seen[neg(u)]) dfs(0, neg(u));
    }
    fill(seen.begin(), seen.end(), false);
    int t = 0;
    while(!st.empty()) {
      int u = st.top(); st.pop();
      if(!seen[u]) dfs(1, u, t++);
    }
  }
  bool satisfiable() {
    kosaraju();
    for(int i = 0; i < n; i++) {
      if(tag[i] == tag[neg(i)]) return false;
      value[i] = tag[i] > tag[neg(i)];
    }
    return true;
  }
};
```

2.2 Centroid Descomposition

```
/*
La altura del Centroid Tree es log(N).
El camino entre cualquier par de nodos (A,B) pasa por un
   centroide ancestro de ambos (LCA en el Centroid Tree).
Para problemas donde se hace update(nodo) y query(nodo).
   Minimizando algo por ejemplo, entonces solo actualizas
    los log(N) ancestros de nodo.
y para query(nodo) preguntas por cada ancestro de nodo,
   de esta forma revisas todos los caminos entre (nodo,
   algun otro nodo)

Time Complexity: O(N log(N))
*/

const int tam = 200005;
vi G[tam];
int del[tam], sz[tam];
int n;

void init(int nodo, int ant) {
  sz[nodo] = 1;
  for (auto it : G[nodo]) {
    if (it == ant || del[it]) continue;
    init(it, nodo);
    sz[nodo] += sz[it];
  }
}

int centroid(int nodo, int ant, int desired) {
  for (auto it : G[nodo]) {
    if (it == ant || del[it]) continue;
    if (sz[it] * 2 >= desired) return centroid(it, nodo,
        desired);
  }
  return nodo;
}

int get_centroid(int nodo) {
  init(nodo, -1);
  int desired = sz[nodo];
  return centroid(nodo, -1, desired);
}

void DC(int nodo) {
  int c = get_centroid(nodo);
  del[c] = 1;
  // aqui haces pre/calculo ?
  // update dfs(nodo)
  for (auto it : G[c]) {
    if (del[it]) continue;
    // sigues con calculo, a veces si tienes que contar
        para cada nodo caminos que pasan sobre el
    // y no solamente cantidad de caminos puedes hacer
    // delete dfs(it)
    // contar (it)
    // update dfs(it)
  }
  // * reinicias tus arreglos *
  for (auto it : G[c]) {
    if (del[it]) continue;
    DC(it, c);
  }
}
```

2.3 Dinitz

```
/*
O(V^2 * E) en la practica corre mas rapido
O(E * sqrt(V)) para unit capacity

Nota.- cuando te pide algo como que la suma no sean
   primos, se modela como grafo bipartito de (pares,
   impares)

Maximiza la ganancia seleccionando maquinas, que tienen
   un costo, para completar proyectos que requieren
   ciertas maquinas y generan ganancias.
Las maquinas pueden compartirse entre diferentes
   proyectos.
Sol.- Si no se corta la arista entre S y un proyecto, el
   proyecto se completa y genera ganancia.
Si se corta la arista entre una maquina y T, entonces se
   compra la maquina.
Entonces si no se corta la arista entre S y un proyecto,
   significa que se compran todas las maquinas requeridas
    para ese proyecto.

res=sumaProyectos-minCut
```

```cpp
*/
struct edge {
  int v, cap, inv, flow;
};

struct Dinic {
  int n, s, t;
  vector<int> lvl;
  vector<vector<edge>> g;

  Dinic(int n) : n(n), lvl(n), g(n) {}

  void add_edge(int u, int v, int c) {
    g[u].push_back({v, c, (int)g[v].size(), 0});
    g[v].push_back({u, 0, (int)g[u].size() - 1, 0});
  }

  bool bfs() {
    fill(lvl.begin(), lvl.end(), -1);
    queue<int> q;
    lvl[s] = 0;
    q.push(s);
    while (!q.empty()) {
      int u = q.front(); q.pop();
      for (auto& e : g[u]) {
        if (e.cap > 0 && lvl[e.v] == -1) {
          lvl[e.v] = lvl[u] + 1;
          q.push(e.v);
        }
      }
    }
    return lvl[t] != -1;
  }

  int dfs(int u, int nf) {
    if (u == t) return nf;
    int res = 0;
    for (auto& e : g[u]) {
      if (e.cap > 0 && lvl[e.v] == lvl[u] + 1) {
        int tf = dfs(e.v, min(nf, e.cap));
        if (tf > 0) {
          e.cap -= tf;
          g[e.v][e.inv].cap += tf;
          e.flow += tf;
          g[e.v][e.inv].flow -= tf;
          res += tf;
          nf -= tf;
          if (nf == 0) break;
        }
      }
    }
    return res > 0 ? res : (lvl[u] = -1, 0);
  }

  int max_flow(int so, int si) {
    s = so; t = si;
    int res = 0;
    while (bfs()) res += dfs(s, INT_MAX);
    return res;
  }

  vector<pair<int, int>> min_cut() {
    vector<bool> vis(n, false);
    queue<int> q;
    q.push(s);
    vis[s] = true;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      for (auto& e : g[u]) {
        if (e.cap > 0 && !vis[e.v]) {
          vis[e.v] = true;
          q.push(e.v);
        }
      }
    }

    vector<pair<int, int>> res;
    for (int u = 0; u < n; u++) {
      if (vis[u]) {
        for (auto& e : g[u]) {
          if (!vis[e.v] && e.flow > 0) {
            // res.push_back({u, e.v}); corto entre (u, e
                .v)
            // puedes anhadir un indice al struct edge
                para saber el indice de arista e.ind
          }
        }
      }
    }
    return res;
  }
  // minimun vertex cover (nodos que toquen todas aristas
     ) = matchign maximo
  vi minimum_vertex_cover(vi &left_nodes, vi &right_nodes
     ) {
    vector<bool> vis(n, false);
    queue<int> q;
    q.push(s);
    vis[s] = true;
    while (!q.empty()) {
      int u = q.front(); q.pop();
      for (auto& e : g[u]) {
        if (e.cap > 0 && !vis[e.v]) {
          vis[e.v] = true;
          q.push(e.v);
        }
      }
    }

    vi res;
    for (int u : left_nodes) if (!vis[u]) res.push_back(u
        ); // Left side vertices not reachable
```

```
        for (int u : right_nodes) if (vis[u]) res.push_back(u
            ); // Right side vertices reachable
        return res;
    }
};
```

2.4 Euler Walk

```
/*
La entrada es un vector (dest, index global de la arista)
    en dirigidos
para grafos no dirigidos las aristas de ida y vuelta
   tienen el mismo index global.
Retorna un vector de nodos en el Eulerian path/cycle
con src como nodo inicial. Si no hay solucion, retorna un
    vector vacio.

Para obtener indices de aristas, anhadir .second a s y
   ret o usar mapa.
Para ver si existe respuesta, ver si ret.size() == nedges
    + 1
Para ver si existe camino euleriano con (start, end)
   tambien ver si ans.back() == end

Un grafo dirigido tiene un camino euleriano si:

Tiene exactamente un vertice con outDegree - inDegree = 1
Tiene exactamente un vertice con inDegree - outDegree = 1
Todos los demas vertices tienen inDegree = outDegree
El recorrido empieza en el vertice con outDegree -
   inDegree = 1
Correr desde este nodo y no necesito verficar lo demas (
   si no hay tal nodo correr desde uno con grado de
   salida > 0)

Nota.- Volverlo global D,its,eu si corres varias veces (
   para cada componente conexa)
Time complexity: O(V + E)
*/
vi eulerWalk(vector<vector<pii>> &gr, int nedges, int src
    = 1) {
  int n = gr.size();
  vi D(n), its(n), eu(nedges), ret, s = {src}; // cambiar
      eu a mapa<int,bool> si las aristas no son [0,nedges
     ]
  D[src]++; // para permitir Euler Paths, no solo ciclos
  while (!s.empty()) {
    int x = s.back(), y, e, &it = its[x], end = gr[x].
        size();
    if (it == end) {
      ret.pb(x);
      s.pop_back();
      continue;
    }
    tie(y, e) = gr[x][it++];
    if (!eu[e]) {
      D[x]--, D[y]++;
      s.pb(y);
      eu[e] = 1;
    }
  }
  for (int x : D) if (x < 0 || ret.size() != nedges + 1)
    return {};
  return {ret.rbegin(), ret.rend()};
}
```

2.5 HLD

```
// El camino entre dos nodos pasa por maximo log n
   aristas livianas
// los ids (en el arreglo) de los nodos de un subarbol
   son contiguos entonces puedes hacer updates a todo el
   subarbol [id[nodo],id[nodo]+sz[nodo]-1]
// en dp que solo importa el estado de atras se puede
   hacer dp[lvl][estado] para ahorrar memoria y primero
   me muevo por los livianos
// y luego por el pesado sin cambiar el lvl pq ya no
   importa

// heavy light decomposition
const int tam=200005;
int v[tam];
int bigchild[tam],padre[tam],depth[tam];
int sz[tam],id[tam],tp[tam];
int T[4*tam];
vi G[tam];
int n;
int query(int lo, int hi) {
  int ra = 0, rb = 0;
  for (lo += n, hi += n + 1; lo < hi; lo /= 2, hi /= 2) {
    if (lo & 1) ra = max(ra, T[lo++]);
    if (hi & 1) rb = max(rb, T[--hi]);
  }
  return max(ra, rb);
}
void update(int idx, int val) {
  T[idx += n] = val;
  for (idx /= 2; idx; idx /= 2) T[idx] = max(T[2 * idx],
     T[2 * idx + 1]);
}
void dfs_size(int nodo, int ant){
  sz[nodo]=1;
  int big=-1;
  int who=-1;
  padre[nodo]=ant;
  for(auto it : G[nodo]){
    if(it==ant)continue;
    depth[it]=depth[nodo]+1;
```

```
      dfs_size(it,nodo);
      sz[nodo]+=sz[it];
      if(sz[it]>big){
        big=sz[it];
        who=it;
      }
    }
    bigchild[nodo]=who;
  }
  int num=0;
  void dfs_hld(int nodo, int ant, int top){
    id[nodo]=num++;
    tp[nodo]=top;
    if(bigchild[nodo]!=-1){
      dfs_hld(bigchild[nodo],nodo,top);
    }
    for(auto it : G[nodo]){
      if(it==ant || it==bigchild[nodo])continue;
      dfs_hld(it,nodo,it);
    }
  }
  int queryPath(int a, int b){
    int res=0;
    while(tp[a]!=tp[b]){
      if(depth[tp[a]]<depth[tp[b]])swap(a,b);
      res=max(res,query(id[tp[a]],id[a]));
      a=padre[tp[a]];
    }
    if(depth[a]>depth[b])swap(a,b);
    res=max(res,query(id[a],id[b]));
    return res;
  }
  int main(){
    int c,q,a,b;
    cin>>n>>q;
    for(int i=1;i<=n;i++)cin>>v[i];
    for(int i=0;i<n-1;i++){
      cin>>a>>b;
      G[a].pb(b);
      G[b].pb(a);
    }
    dfs_size(1,0);
    dfs_hld(1,0,1);
    for(int i=1;i<=n;i++){
      update(id[i],v[i]);
    }
    while(q--){
      cin>>c;
      cin>>a>>b;
      if(c==1){
        update(id[a],b);
      }else{
        printf("%d ", queryPath(a,b));
      }
    }
    return 0;
  }
```

2.6 LCA Binary Lifting

```
const int tam = 200005;
int depth[tam];
int dp[20][tam];
vi G[tam];
int n;

void dfs(int nodo, int ant, int d) {
  depth[nodo] = d;
  dp[0][nodo] = ant;
  for (auto it : G[nodo]) {
    if (it == ant) continue;
    dfs(it, nodo, d + 1);
  }
}

void initLCA() {
  memset(dp, -1, sizeof(dp));
  dfs(1, -1, 0);
  for (int i = 1; i < 20; i++) {
    for (int v = 1; v <= n; v++) {
      if (dp[i - 1][v] != -1) {
        dp[i][v] = dp[i - 1][dp[i - 1][v]];
      }
    }
  }
}

int LCA(int a, int b) {
  if (depth[a] > depth[b]) swap(a, b);
  int dif = depth[b] - depth[a];
  for (int i = 19; i >= 0; i--) {
    if (dif & (1 << i)) b = dp[i][b];
  }
  if (a == b) return a;
  for (int i = 19; i >= 0; i--) {
    if (dp[i][a] != dp[i][b]) {
      a = dp[i][a];
      b = dp[i][b];
    }
  }
  return dp[0][a];
}
```

2.7 LCA Sparse Table

```
// index-1
#define pii pair<int, int>
#define pid pair<int, double>
```

```cpp
#define endl '\n'
#define x first
#define y second
#define que priority_queue<int, vector<int>, greater<int
    >>

const int N = 5e5 + 5;
const int INF = 0x3f3f3f3f;

int n, m, R, dn, dfn[N], mi[21][N];
vector<int> e[N];

int get(int x, int y) {
  return (dfn[x] < dfn[y] ? x : y);
}

void dfs(int id, int f) {
  mi[0][dfn[id] = ++dn] = f;
  for (auto it : e[id])
    if (it != f)
      dfs(it, id);
}

int lca(int u, int v) {
  if (u == v) return u;
  if ((u = dfn[u]) > (v = dfn[v])) swap(u, v);
  int d = __lg(v - u++);
  return get(mi[d][u], mi[d][v - (1 << d) + 1]);
}

void solve() {
  cin >> n >> m;
  R = 1;
  for (int i = 2; i <= n; i++) {
    int u; cin >> u;
    u++;
    e[u].emplace_back(i);
    e[i].emplace_back(u);
  }
  dfs(R, 0);
  for (int i = 1; i <= __lg(n); i++) {
    for (int j = 1; j + (1 << i) - 1 <= n; j++) {
      mi[i][j] = get(mi[i - 1][j], mi[i - 1][j + (1 << i)
          - 1]);
    }
  }

  for (int i = 1; i <= m; i++) {
    int u, v; cin >> u >> v;
    u++, v++;
    int ans = lca(u, v) - 1;
    cout << ans << endl;
  }
}
```

2.8 Puentes

```cpp
// si es grafo con aristas multiples (a,b) , (a,b)
// entonces usar una mapa de pares y si una arista
    aparece dos veces no puede ser puente
const int tam=2e5+5;
set<pair<int, int>> st; // puente arista entre (a, b)
vi G[tam];
int arc[tam], IN[tam];
int tiempo=0;
void dfs(int nodo, int ant){
  tiempo++;
  IN[nodo] = arc[nodo] = tiempo;
  for(auto it : G[nodo]){
    if(it == ant) continue;
    if(IN[it]){
      arc[nodo] = min(arc[nodo], IN[it]);
    } else {
      dfs(it, nodo);
      arc[nodo] = min(arc[nodo], arc[it]);
      if(arc[it] > IN[nodo]){
        st.insert({nodo, it});
      }
    }
  }
}
```

2.9 Khun

```cpp
// algoritmo de khun para grafos bipartitos  O(nm)
const int tam = 100;
vi G[tam]; // pueden tener mismo indices nodos de
    distintos grupos
bool vis[tam];
int pareja[tam];// pareja de los nodos de la derecha

bool khun(int nodo) {
  if (vis[nodo]) return false;
  vis[nodo] = 1;
  for (auto it : G[nodo]) {
    if (pareja[it] == -1 || khun(pareja[it])) {
      pareja[it] = nodo;
      return true;
    }
  }
  return false;
}

int main() {
  int m, a, b;
  cin >> m;
  for (int i = 0; i < m; i++) {
    cin >> a >> b; // de izquierda a derecha
    G[a].pb(b);
  }
  memset(pareja, -1, sizeof(pareja));
```

```
  int match = 0;
  for (int i = 1; i <= n; i++) {
    memset(vis, false, sizeof(vis)); // no olvidar
    if (khun(i)) match++; // camino aumentante
  }
  return 0;
}
```

2.10 Isomorfismo Arboles

```
#include <bits/stdc++.h>
#define vi vector<int>
#define pb push_back
#define S second
#define F first
using namespace std;
struct Tree{
  int n;
  vi sz;
  vector<vi>G;
  vi centroids;
  vector<vi>level;
  vi prev;
  Tree(int x){
    n=x;
    sz.resize(x+1);
    G.assign(n+1,vi());
    prev.resize(n+1);
  }
  void addEdge(int a, int b){
    G[a].pb(b);G[b].pb(a);
  }
  void centroid(int nodo, int ant){
    bool ok=1;
    for(auto it : G[nodo]){
      if(it==ant)continue;
      if(sz[it]>n/2){
        ok=false;
      }
      centroid(it,nodo);
    }
    int atras=n-sz[nodo];
    if(atras>n/2)ok=false;
    if(ok)centroids.pb(nodo);
  }
  void initsz(int nodo, int ant){
    sz[nodo]=1;
    for(auto it : G[nodo]){
      if(it!=ant){
        initsz(it,nodo);
        sz[nodo]+=sz[it];
      }
    }
  }
  void initLevels(int nodo){
    level.clear();
    vi aux;aux.pb(nodo);
    int pos=0;
    level.pb(aux);
    prev[nodo]=-1;
    while(true){
      aux.clear();
      for(auto it : level[pos]){
        for(auto j : G[it]){
          //cout<<"apagare la luz  "<<j<<endl;
          if(j==prev[it])continue;
          aux.pb(j);
          prev[j]=it;
        }
      }
      if(aux.size()==0)break;
      level.pb(aux);
      pos++;
    }
  }
};
bool check(Tree A, int a, Tree B, int b){
  A.initLevels(a);B.initLevels(b);
  if(A.level.size()!=B.level.size())return false;
  int hashA[A.n+5];
  int hashB[A.n+5];//hash del subarbol rooteado en i

  vector<vi>EA,EB;//le paso los hash de todos los hijos
     de i me
                //servira para formar el hash del
                    subarbol i
  EA.resize(A.n+1);EB.resize(A.n+1);

  for(int h=A.level.size()-1;h>=0;h--){
    map<vi,int>ind;
    for(auto it : A.level[h]){
      sort(EA[it].begin(),EA[it].end());
      ind[EA[it]]=0;
    }
    for(auto it : B.level[h]){
      sort(EB[it].begin(),EB[it].end());
      ind[EB[it]]=0;
    }

    int num=0;
    for(auto it : ind){
      it.S=num;
      ind[it.F]=num;
      num++;
    }
    //paso a sus padres
    for(auto it : A.level[h]){
      hashA[it]=ind[EA[it]];

      if(h>0)EA[A.prev[it]].pb(hashA[it]);
```

```
      }
      for(auto it : B.level[h]){
        hashB[it]=ind[EB[it]];
        if(h>0)EB[B.prev[it]].pb(hashB[it]);
      }
    }
    return hashA[a]==hashB[b];
  }
  bool isomorphic(Tree A, Tree B){
    A.initsz(1,-1);B.initsz(1,-1);
    A.centroid(1,-1);B.centroid(1,-1);
    vi CA=A.centroids,CB=B.centroids;
    if(CA.size()!=CB.size())return false;
    for(int i=0;i<CB.size();i++){
      if(check(A,CA[0],B,CB[i])){
        return true;
      }
    }
    return false;
  }
  int main() {
    int t,n,a,b;
    cin>>t;
    while(t--){
      cin>>n;
      Tree A(n);
      Tree B(n);
      for(int i=1;i<n;i++){
        cin>>a>>b;
        A.addEdge(a,b);
      }
      for(int i=1;i<n;i++){
        cin>>a>>b;
        B.addEdge(a,b);
      }
      if(isomorphic(A,B)){
        cout<<"YES"<<"\n";
      }else{
        cout<<"NO"<<"\n";
      }
    }
  }
```

2.11 Small to Large

```
  const int tam=200005;
  int a[tam];
  vi G[tam];
  int f[tam];
  set<int> F[tam];
  // small to large dfs
  void _find(int nodo, int ant){
    f[nodo]=a[nodo];
    if(ant!=-1)f[nodo]^=f[ant];
    for(auto it : G[nodo]){
      if(it!=ant){
        _find(it,nodo);
      }
    }
  }
  int res=0;
  void dfs(int nodo, int ant){
    bool flag=false;
    F[nodo].insert(f[nodo]);
    for(auto it : G[nodo]){
      if(it==ant)continue;
      dfs(it,nodo);
      if(sz(F[nodo])<sz(F[it]))swap(F[nodo],F[it]);
      for(auto it2 : F[it]){
        if(F[nodo].count(a[nodo] ^ it2))flag=true;
      }
      for(auto it2 : F[it]){
        F[nodo].insert(it2);
      }
      F[it].clear();
    }
    if(flag){
      res++;
      F[nodo].clear();
    }
  }
```

2.12 Kosaraju

```
  const int tam = 200005;
  vi G[tam], G1[tam];
  vector<bool> vis;
  vi v;
  int scc = 0;

  void dfs0(int nodo) {
    vis[nodo] = 1;
    for (auto it : G[nodo]) {
      if (!vis[it]) dfs0(it);
    }
    v.pb(nodo);
  }

  void dfs1(int nodo) {
    vis[nodo] = 1;
    for (auto it : G1[nodo]) {
      if (!vis[it]) dfs1(it);
    }
  }

  int main() {
    int n, m, a, b;
```

```
  cin >> n >> m;
  for (int i = 0; i < m; i++) {
    cin >> a >> b;
    G[a].pb(b);
    G1[b].pb(a);
  }
  vis.assign(n + 1, false);
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) dfs0(i);
  }
  reverse(v.begin(), v.end());
  vis.assign(n + 1, false);
  for (int i = 0; i < n; i++) {
    if (!vis[v[i]]) {
      scc++;
      dfs1(v[i]);
    }
  }
  return 0;
}
```

2.13 Dynamic Connectivity

```
int res = 0;
const int tam = 300005;
bool responder[tam];
vector<pair<int, int> > G[4 * tam];
int P[tam];
int sz[tam];

int _find(int x) {
  if (x == P[x]) return x;
  return _find(P[x]);
}

void push(int nodo, int b, int e, int izq, int der, int A
    , int B) {
  int L = 2 * nodo + 1, R = L + 1, mid = (b + e) / 2;
  if (b > der || e < izq) return;
  if (b >= izq && e <= der) {
    G[nodo].pb({A, B});
    return;
  }
  push(L, b, mid, izq, der, A, B);
  push(R, mid + 1, e, izq, der, A, B);
}

vi respuestas;

void go(int nodo, int b, int e) {
  int L = 2 * nodo + 1, R = L + 1, mid = (b + e) / 2;
  // aqui meto cambios
  vector<int> changes; // los vertices que eran
      representantes de su componente y dejan de serlo
  vector<pair<int, int> > change2;
  for (auto it : G[nodo]) {
    int pap1 = _find(it.F), pap2 = _find(it.S);
    if (sz[pap1] < sz[pap2])
      swap(pap1, pap2);
    if (pap1 != pap2) {
      changes.push_back(pap2);
      change2.pb({pap1, sz[pap1]});
      P[pap2] = pap1;
      sz[pap1] = sz[pap1] + sz[pap2];
    }
  }
  res -= changes.size();
  if (b == e) {
    // importante que no este arriba
    if (responder[b]) respuestas.pb(res);
  } else {
    go(L, b, mid);
    go(R, mid + 1, e);
  }
  // deshacemos los cambios
  for (int x : changes)
    P[x] = x;
  for (auto it : change2)
    sz[it.F] = it.S;
  res += changes.size();
}

int main() {
  FIFO;
  int n, q, a, b;
  cin >> n >> q;
  res = n;
  for (int i = 1; i <= n; i++) {
    P[i] = i;
    sz[i] = 1; // ojo aqui xd
  }

  char c;
  map<pair<int, int>, int> abierto;
  for (int i = 1; i <= q; i++) {
    cin >> c;
    if (c == '?') {
      responder[i] = true;
      continue;
    }
    cin >> a >> b;
    if (a > b) swap(a, b);
    if (c == '+') {
      abierto[{a, b}] = i;
    } else {
      int izq = abierto[{a, b}];
      push(0, 1, q, izq, i, a, b);
      abierto.erase({a, b});
    }
  }
```

```
  for (auto it : abierto) {
    int a = it.F.F, b = it.F.S, izq = it.S;
    push(0, 1, q, izq, q, a, b);
  }
  if (q == 0) return 0;
  go(0, 1, q);
  for (auto it : respuestas) {
    cout << it << "\n";
  }
  return 0;
}
```

2.14 Bellman Ford

```
// Time: O(n*m)
// Distancia mas corta desde source a todos los nodos
const int INF=1e18;
const int tam=2505;
vi G[tam];
vector<pair<pair<int,int>,int >> E; // (a->b, w)
int n,m;
bool cicloNegativo[tam];
void bellmanFord(int source){
  vi dist(n+1,INF);
  dist[source]=0;
  for(int i=0;i<n-1;i++){
    for(auto e:E){
      int a=e.first.first;
      int b=e.first.second;
      int w=e.second;
      if(dist[a]!=INF && dist[b]>dist[a]+w){
        dist[b]=dist[a]+w;
      }
    }
  }
  for(auto e:E){
    int a=e.first.first;
    int b=e.first.second;
    int w=e.second;
    if(dist[a]!=INF && dist[b]>dist[a]+w){
      cicloNegativo[b]=true;// b esta en un ciclo
          negativo
      // marca al menos un nodo de cada ciclo negativo
    }
  }
}
```

2.15 Squeeze graphs with simple cycles

```
// solo funciona para undirected
void dfs(int nodo, int ant){
  id[nodo]=nodo;
  for(auto it : G[nodo]){
    if(it==ant)continue;
    if(id[it]){
      if(id[it]==it){
        id[nodo]=it;
      }
    }else{
      dfs(it,nodo);
      if(id[it]!=it)id[nodo]=id[it];
    }
  }
}
```

3 Data Structures

3.1 2D BIT

```
#include<bits/stdc++.h>
#define lcm(a,b) (a/__gcd(a,b))*b
#define fast ios_base::sync_with_stdio(false);cin.tie(0);
    cout.tie(0);
#define ll long long int
#define vi vector<int>
#define vll vector<ll>
#define pb push_back
#define F first
#define S second
#define mp make_pair
//"\n"
//__builtin_popcount(x)
// a+b=2*(a&b) + (a^b)
using namespace std;
const int tam=1005;
int n,q;
int T[tam][tam];
void update(int x, int y, int val){
    x++;y++;
    for(;x<tam;x+=x&-x){
        for(int l=y;l<tam;l+=l&-l)T[x][l]+=val;
    }
}
int query(int x, int y){
    x++;y++;
    int res=0;
    for(;x>0;x-=x&-x){
        for(int l=y;l>0;l-=l&-l)res+=T[x][l];
    }
    return res;
}
int main()
{
    cin>>n>>q;
    string s;
```

```
    vector<string>M;
    for(int i=0;i<n;i++){
        cin>>s;
        M.pb(s);
        for(int l=0;l<n;l++){
            if(s[l]=='*'){
                update(i,l,1);
            }
        }
    }
    while(q--){
        int c,x1,x2,y1,y2;
        cin>>c;
        if(c==1){
            cin>>x1>>y1;
            x1--;y1--;
            if(M[x1][y1]=='*'){
                M[x1][y1]='.';
                update(x1,y1,-1);
            }else{
                M[x1][y1]='*';
                update(x1,y1,1);
            }
        }else{
            cin>>x1>>y1>>x2>>y2;
            x1--;y1--;x2--;y2--;
            cout<<query(x2,y2)-query(x2,y1-1)-query(x1-1,
               y2)+query(x1-1,y1-1)<<endl;
        }
    }

    return 0;
}
```

3.2 DSU Rollback

```
/*
Para sacar checkpoint int CP = st.size()
Para rollback rollback(CP)
LLamar a init(n) al inicio

Note.- index 1 de los nodos, cuidado con los indices de
   las aristas al hacer Dynamic Connectivity
dynamic connectivity se realiza sobre los indices de las
   queries simulando el paso del tiempo
y las aristas viven en ciertos rangos de tiempo (se
   simula con dfs y segment tree)

Time Complexity: O(log(n)) para find y union
*/

struct RB_DSU {
    vi P;
    vi sz;
    stack<int> st;
    int scc;

    void init(int n) {
        P.resize(n+1);
        sz.resize(n+1, 1);
        scc = n;
        for (int i = 1; i <= n; i++) P[i] = i;
    }

    int _find(int a) {
        if (P[a] == a)
            return a;
        return _find(P[a]);
    }

    void _union(int a, int b) {
        a = _find(a);
        b = _find(b);
        if (a == b) return;
        if (sz[a] > sz[b]) swap(a, b);
        P[a] = b;
        sz[b] += sz[a];
        scc--;
        st.push(a);
    }

    void rollback(int t) {
        while (st.size() > t) {
            int a = st.top();
            st.pop();
            sz[P[a]] -= sz[a];
            P[a] = a;
            scc++;
        }
    }
};
```

3.3 Implicit Segment Tree

```
// Node *T = new Node;
// query(T, 0, top, 0, top);  top = 1e9 e.g.
// update(T, 0, top, y1, y2);
struct Node {
    int valor;
    int lazy;
    Node *L, *R;
    Node() : valor(0), lazy(0), L(NULL), R(NULL) {}
    void propagate(int b, int e) {
        if (lazy == 0) return;
        lazy = 0;
        valor = (e - b + 1) - valor;
        if (b == e) return;
        if (!L) L = new Node();
        if (!R) R = new Node();
        L->lazy ^= 1;
```

```
            R->lazy ^= 1;

            // esta vaina no es necesaria solo cuando da MLE
            if (L && L->lazy == 0 && L->valor == 0) {
                delete L;
                L = NULL;
            }
            if (R && R->lazy == 0 && R->valor == 0) {
                delete R;
                R = NULL;
            }
        }
};

void update(Node *nodo, int b, int e, int izq, int der) {
    nodo->propagate(b, e);
    if (b > der || e < izq) return;
    if (b >= izq && e <= der) {
        nodo->lazy ^= 1;
        nodo->propagate(b, e);
        return;
    }
    int mid = (b + e) / 2;
    if (!nodo->L) nodo->L = new Node();
    if (!nodo->R) nodo->R = new Node();
    update(nodo->L, b, mid, izq, der);
    update(nodo->R, mid + 1, e, izq, der);
    nodo->valor = nodo->L->valor + nodo->R->valor;
}

int query(Node *nodo, int b, int e, int izq, int der) {
    if (b > der || e < izq) return 0;
    nodo->propagate(b, e);
    if (b >= izq && e <= der) return nodo->valor;
    int mid = (b + e) / 2;
    return query(nodo->L, b, mid, izq, der) + query(nodo
        ->R, mid + 1, e, izq, der);
}
```

3.4 Mos

```
// Complexity: O(|N+Q|*sqrt(|N|)*|meter/quitar|)
// Requiere meter(), quitar()

vector<pair<pair<int,int>,int> >Q;// {{izq,der},id}
int tami = 300; // o sqrt(n)+1
bool comp(pair<pair<int,int>,int> a,pair<pair<int,int>,
    int> b){
    if(a.F.F/tami!=b.F.F/tami){
        return a.F.F/tami<b.F.F/tami;
    }
    return a.F.S<b.F.S;
}
// main
sort(Q.begin(),Q.end(),comp);
int L=0,R=-1;
int respuesta=0;
for(int i=0;i<q;i++){
    int izq=Q[i].F.F;
    int der=Q[i].F.S;
    int ind=Q[i].S;
    while(L>izq)meter(--L);
    while(R<der)meter(++R);
    while(R>der)quitar(R--);
    while(L<izq)quitar(L++);
    res[ind]=respuesta;
}
```

3.5 Mos on Trees

```
// Si en el rango un nodo aparece dos veces entonces no
    se toma en cuenta (se cancela)
// Para una query en camino [u,v], IN[u]<=IN[v]
// Si LCA(u,v) = u -> Rango Query [IN[u],IN[v]]
// Si No -> Rango Query [OUT[u],IN[v]] + [IN[LCA],IN[LCA
    ]] (o sea falta considerar el LCA)

// Cuando las consultas son sobre las aristas
// Si LCA(u,v) = u -> Rango Query [IN[u]+1,IN[v]]
// Si No -> Rango Query [OUT[u],IN[v]]

const int tam = 100005;
vector<pair<int, int>> G[tam];
int dp[20][tam];// esto para LCA
int tiempo = -1;
int IN[tam];// tiempo de entrada
int OUT[tam];// tiempo de salida
int A[3*tam];// los nodos en orden del dfs
int depth[tam];
int valor[tam];// valor del nodo/arista

void dfs(int nodo, int ant, int llega, int d) {
    depth[nodo] = d+1;
    dp[0][nodo] = ant;
    valor[nodo] = llega;
    IN[nodo] = ++tiempo;
    A[IN[nodo]] = nodo;
    for (auto it : G[nodo]) {
        int v = it.first;
        int val = it.second;
        if (v == ant) continue;
        dfs(v, nodo, val, d+1);
    }
    OUT[nodo] = ++tiempo;
    A[OUT[nodo]] = nodo;
}
```

3.6 Segment Tree

```
// suma en rango
vi v;

struct ST {
  int N;
  vi T;

  void init(int n) {
    N = n;
    T.assign(4 * N, 0);
  }

  void build(int nodo, int b, int e) {
    int mid = (b + e) / 2, L = nodo * 2 + 1, R = L + 1;
    if (b == e) {
      T[nodo] = v[b];
      return;
    }
    build(L, b, mid);
    build(R, mid + 1, e);
    T[nodo] = T[L] + T[R];
  }

  void update(int nodo, int b, int e, int pos, int val) {
    int mid = (b + e) / 2, L = nodo * 2 + 1, R = L + 1;
    if (b == e) {
      T[nodo] = val;
      return;
    }
    if (pos <= mid) update(L, b, mid, pos, val);
    else update(R, mid + 1, e, pos, val);
    T[nodo] = T[L] + T[R];
  }

  int query(int nodo, int b, int e, int izq, int der) {
    int mid = (b + e) / 2, L = nodo * 2 + 1, R = L + 1;
    if (b >= izq && e <= der) return T[nodo];
    if (der <= mid) {
      return query(L, b, mid, izq, der);
    }
    if (izq > mid) {
      return query(R, mid + 1, e, izq, der);
    }
    return query(L, b, mid, izq, der) + query(R, mid + 1,
        e, izq, der);
  }
};

int main() {
  ST tree;
  tree.init(3);
  v.pb(1); v.pb(2); v.pb(3);
  tree.build(0, 0, 2);
  return 0;
}
```

3.7 Sparse Table

```
// Time complexity: O(n log n)
int ST[20][500005]; // log2(MAXN) = 20

void init(vi &a) {
  int n = a.size();
  for (int i = 0; i < n; i++) ST[0][i] = a[i];
  for (int i = 1; (1 << i) <= n; 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))
          ]);
    }
  }
}

// index-0
int query(int l, int r) {
  int len = r - l + 1;
  int pot = __lg(len);
  return min(ST[pot][l], ST[pot][r - (1 << pot) + 1]);
}
```

4 Math

4.1 Factorizacion Criba

```
const int tam=1e7+5;
int small[tam];
void criba(){
  for(int i=2;i<tam;i++){
    if(small[i]==0){
      for(int l=i;l<tam;l+=i){
        if(small[l]==0)small[l]=i;
      }
    }
  }
}
vi factorizar(int x){ // primos en orden ascendente
   2,2,5,7...
  vi ans;
  while(x>1){
    ans.pb(small[x]);
    x/=small[x];
  }
  return ans;
}
```

4.2 Linear Diophantine

```
ll div_ceil(ll a, ll b, bool ceil){
  ll ans = abs(a/b);
  bool pos = (a<0)==(b<0);
  if(a%b and pos==ceil) ans++;
  if(!pos) ans*=-1;
  return ans;
}
// |x|+|y| es minimo y x es minimo
ll gcd_ext(ll a, ll b, ll &xo, ll &yo){
  if(b==0){
    xo = 1, yo = 0;
    return a;
  }
  ll x1, y1;
  ll g = gcd_ext(b,a%b,x1,y1);
  xo = y1;
  yo = x1-(a/b)*y1;
  return g;
}
//sol return (y) for b in ax + by = c
//sol retorna minimo x + y creo
ll sol(ll a, ll b, ll c){
  ll xo, yo;
  ll g = gcd_ext(a,b,xo,yo);
  assert(c%g==0);
  a/=g,b/=g,c/=g;
  xo*=c,yo*=c;
  ll k;
  if(a>0) k = div_ceil(1-yo,a,1);
  else k = div_ceil(1-yo,a,0);
  return yo+k*a;
}

bool check(int a, int b, int c){
  if(a==0)return c%b==0;
  if(b==0)return c%a==0;
  return (abs(c)%__gcd(abs(a),abs(b)))==0;
}
```

4.3 Linear Sieve

```
// Time : O(n)
// lp[i] menor primo que divide a i
const int tam=10000000;
vector<int>primes;
vector<int>lp(tam+1);
void linear_sieve(){
  for(int i=2;i<tam;i++){
    if(lp[i]==0){
      lp[i]=i;
      primes.pb(i);
    }
    for(int l=0;i * primes[l]<tam;l++){
      lp[i*primes[l]]=primes[l];
      if(i%primes[l]==0)break;
    }
  }
}
```

4.4 Moebius

```
// 0 si es divisible por algun cuadrado
// 1 si esta libre de cuadrados y tiene un numero par de
    factores primos
// -1 libre de cuadrados y tiene un numero impar de
    factores primos
// si quiero contar solo los que tiene gcd 1
const int tam=200005;
int mou[tam];
int check(int num){
  if(num==1)return 1;
  int cant=0;
  for(int i=2;i*i<=num;i++){
    if(num%i==0){
      cant++;
      num/=i;
      if(num%i==0){
        return 0;
      }
    }
  }
  if(num>1)cant++;
  if(cant%2){
    return -1;
  }
  return 1;
}
void init(){
  for(int i=1;i<tam;i++){
    mou[i]=check(i);
  }
}
// si quiero que el gcd del arreglo sea 1 tcs tengo que
    restarle todos que sean multiplos de 2,3,..... y le
    sumo 6 ... pq se repiten
// FACILITO
// si m es el mayor numero de mi arreglo tcs res[m]=cal(m
    ), res[m-1]=cal(m-1)-sumatoria(cal(multiplos(m-1)))
// cal(m)=crear la respuesta con multiplos de m
```

4.5 Pascal

```
ll pascal[5005][5005];
pascal[0][0]=1;
for(int i=1;i<=5000;i++){
  for(int j=0;j<=i;j++){
```

```
      if(j==0 || j==i)pascal[i][j]=1;
      else
      pascal[i][j]=(pascal[i-1][j-1]+pascal[i-1][j])%MOD;
    }
  }
  int n,k;
  cout<<pascal[n][k]<<endl;
```

4.6 Coeficiente Binomial

```
const int MOD=998244353;
ll fact[5005];
void init(){
   fact[0]=1;
   for(int i=1;i<=5000;i++){
      fact[i]=(fact[i-1]*i)%MOD;
   }
}
ll Pou(int a, int n){
   if(n==0)return 1;
   if(n%2==0){
      ll A=Pou(a,n/2);
      return (A*A)%MOD;
   }else{
      ll A=Pou(a,n/2);
      A=(A*A)%MOD;
      return (A*a)%MOD;
   }
}
ll nck(int n, int k){
   if(n<k)return 0;
   ll res=(fact[n]*Pou((fact[k]*fact[n-k])%MOD,MOD-2))%
      MOD;
   return res;
}
```

4.7 Euler Phi

```
/*
phi(n)= cantidad de numeros menores a n que son coprimos
    con n
phi(5) = 4, phi(6)=2

Time complexity: < O(N (log N))
*/

const int tam=1e5+5;
int phi[tam];
void init(){
  for(int i=1;i<tam;i++){
    phi[i]=i;
  }
  for(int i=2;i<tam;i++){
    if(phi[i]==i){
      for(int j=i;j<tam;j+=i){
        phi[j]-=phi[j]/i;
      }
    }
  }
}
```

4.8 Discrete Logarithm

```
// 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;
}
```

4.9 Miller Rabin

```
ll mul (ll a, ll b, ll mod) {
  ll ret = 0;
  for(a %= mod, b %= mod; b != 0;
    b >>= 1, a <<= 1, a = a >= mod ? a - mod : a) {
    if (b & 1) {
      ret += a;
      if (ret >= mod) ret -= mod;
    }
```

```
  }
  return ret;
}
ll fpow (ll a, ll b, ll mod) {
  ll ans = 1;
  for (; b; b >>= 1, a = mul(a, a, mod))
    if (b & 1)
      ans = mul(ans, a, mod);
  return ans;
}
bool witness (ll a, ll s, ll d, ll n) {
  ll x = fpow(a, d, n);
  if (x == 1 || x == n - 1) return false;
  for (int i = 0; i < s - 1; i++) {
    x = mul(x, x, n);
    if (x == 1) return true;
    if (x == n - 1) return false;
  }
  return true;
}
ll test[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 0};
bool is_prime (ll n) {
  if (n < 2) return false;
  if (n == 2) return true;
  if (n % 2 == 0) return false;
  ll d = n - 1, s = 0;
  while (d % 2 == 0) ++s, d /= 2;
  for (int i = 0; test[i] && test[i] < n; ++i)
    if (witness(test[i], s, d, n))
      return false;
  return true;
}
```

4.10 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;
}
```

5 Geometry

5.1 Closest Pair of Points

```
/*
Retorna indices (index 0) de los puntos mas cercanos.
Tiempo: O(n log n)
*/
long long dist2(pair<int, int> a, pair<int, int> b) {
  return 1LL * (a.F - b.F) * (a.F - b.F) + 1LL * (a.S - b
      .S) * (a.S - b.S);
}

pair<int, int> closest_pair(vector<pair<int, int>> a) {
  int n = a.size();
  assert(n >= 2);
  vector<pair<pair<int, int>, int>> p(n);
  for (int i = 0; i < n; i++) p[i] = {a[i], i};
  sort(p.begin(), p.end());
  int l = 0, r = 2;
  long long ans = dist2(p[0].F, p[1].F);
  pair<int, int> ret = {p[0].S, p[1].S};
  while (r < n) {
    while (l < r && 1LL * (p[r].F.F - p[l].F.F) * (p[r].F
        .F - p[l].F.F) >= ans) l++;
    for (int i = l; i < r; i++) {
      long long nw = dist2(p[i].F, p[r].F);
      if (nw < ans) {
        ans = nw;
        ret = {p[i].S, p[r].S};
      }
    }
    r++;
  }
  return ret;
}
```

5.2 General

```
struct Point {
  ll x, y;
  // Operadores para comparacion
  bool operator == (Point b) { return x == b.x && y == b.
    y; }
  bool operator != (Point b) { return !(*this == b); }
```

```cpp
  bool operator < (const Point &o) const { return y < o.y
      || (y == o.y && x < o.x); }
  bool operator > (const Point &o) const { return y > o.y
      || (y == o.y && x > o.x); }
};
// Operadores aritmeticos
Point operator +(const Point &A, const Point &B) { return
    {A.x + B.x, A.y + B.y}; }
Point operator -(const Point &A, const Point &B) { return
    {A.x - B.x, A.y - B.y}; }
Point operator *(const Point &A, const ll &K) { return {A
    .x * K, A.y * K}; }
ll dot(const Point &A, const Point &B) { return A.x * B.x
    + A.y * B.y; }
ll cross(const Point &A, const Point &B) { return A.x * B
    .y - A.y * B.x; }
ll turn(const Point &A, const Point &B, const Point &C) {
    return cross(B - A, C - A); }
ll dist2(const Point &A, const Point &B) { return dot(A -
    B, A - B); }
// Calcula la envoltura convexa de un conjunto de puntos
vector<Point> convex_hull(vector<Point> p) {
  if (p.size() <= 1) return p; // Envoltura convexa
      trivial para <= 2 puntos

  sort(p.begin(), p.end());
  vector<Point> ch;
  ch.reserve(p.size() + 1);

  for (int i = 0; i < 2; i++) {
    int start = ch.size();
    for (auto &a : p) {
      // Si se necesitan puntos colineales, usa < y
          primero elimina puntos repetidos en p
      while (ch.size() >= start + 2 && turn(ch[ch.size()
          - 2], ch.back(), a) <= 0)
        ch.pop_back();
      ch.push_back(a);
    }
    ch.pop_back();
    reverse(p.begin(), p.end());
  }

  if (ch.size() == 2 && ch[0] == ch[1]) ch.pop_back();

  return ch;
}

// Calcula el angulo entre dos vectores en grados
double angle(Point a, Point b) {
  double aV = sqrt(dot(a, a));
  double bV = sqrt(dot(b, b));
  double resRad = acos(dot(a, b) / (aV * bV)); //
      Radianes
  double resDeg = resRad * 180.0 / acos(-1);
  return resDeg;
}

// Calcula la altura desde un punto a una linea formada
    por dos puntos
double calcular_H(Point a, Point b, Point P) {
  // (a, b) * H = area del paralelogramo
  double area = abs(cross(P - a, b - a));
  double disAB = sqrt(dist2(a, b));
  double H = area / disAB;
  return H;
}

// Verifica la interseccion entre dos segmentos de linea
bool check_segment_intersection(Point a1, Point a2, Point
     b1, Point b2) {
  // Caso paralelo
  if (cross(a2 - a1, b2 - b1) == 0) {
    if (turn(a1, a2, b1) == 0) {
      for (int i = 0; i < 2; i++) {
        if (max(a1.x, a2.x) < min(b1.x, b2.x) || max(a1.y
            , a2.y) < min(b1.y, b2.y)) {
          return false;
        }
        swap(a1, b1);
        swap(a2, b2);
      }
      return true;
    }
    return false;
  }

  // Caso no paralelo
  for (int i = 0; i < 2; i++) {
    ll s1 = turn(a1, a2, b1);
    ll s2 = turn(a1, a2, b2);
    if ((s1 < 0 && s2 < 0) || (s1 > 0 && s2 > 0)) {
      return false;
    }
    swap(a1, b1);
    swap(a2, b2);
  }

  return true;
}

// Calcula el area doble de un poligono (la suma de areas
     de triangulos)
int doble_area(vector<Point> &p) {
  int n = p.size();
  int res = 0;
  for (int i = 0; i < n; i++) {
    res += cross(p[i], p[(i + 1) % n]);
  }
  return abs(res);
}
```

5.3 Polygon

```
// ver si punto esta en un ccw poligono convexo, O(log n)
enum {OUT, ON, IN};
int E0 = 0;
int point_in_convex_polygon( const vector < Point > &pol,
    const Point &p ) {
  int low = 1, high = pol.size() - 1;
  while( high - low > 1 ) {
    int mid = ( low + high ) / 2;
    if( turn( pol[0], pol[mid], p ) >= -E0 ) low = mid;
    else high = mid;
  }
  if( turn( pol[0], pol[low], p ) < -E0 ) return OUT;
  if( turn( pol[low], pol[high], p ) < -E0 ) return OUT;
  if( turn( pol[high], pol[0], p ) < -E0 ) return OUT;

  if( low == 1 && turn( pol[0], pol[low], p ) <= E0 )
    return ON;
  if( turn( pol[low], pol[high], p ) <= E0 ) return ON;
  if( high == (int) pol.size() -1 && turn( pol[high], pol
    [0], p ) <= E0 ) return ON;
  return IN;
}

// punto en poligono cualquiera
bool pointlineintersect(Point P1, Point P2, Point P3) {
  if (cross(P2 - P1, P3 - P1) != 0) return false;
  return (min(P2.x, P3.x) <= P1.x && P1.x <= max(P2.x, P3
    .x))
    && (min(P2.y, P3.y) <= P1.y && P1.y <= max(P2.y, P3.y
      ));
}

int point_in_polygon(vector<Point> &pol, Point P) {
  int cnt = 0;
  bool boundary = false;
  int N = pol.size();
  for (int i = 0; i < N; i++) {
    int j = (i + 1) % N;
    if (pointlineintersect(P, pol[i], pol[j])) {
      boundary = true;
      break;
    }
    if (pol[i].y <= P.y && P.y < pol[j].y && cross(pol[j]
      - pol[i], P - pol[i]) < 0) cnt++;
    else if (pol[j].y <= P.y && P.y < pol[i].y && cross(
      pol[i] - pol[j], P - pol[j]) < 0) cnt++;
  }
  if (boundary) return ON;
  return (cnt & 1) ? IN : OUT;
}
```

5.4 Sort Counter Clockwise

```
bool up(Point a) {
  return a.y > 0 || (a.y == 0 && a.x >= 0);
}
bool cmp(Point a, Point b) {
  if (up(a) != up(b)) return up(a) > up(b);
  return cross(a, b) > 0;
}

// this starts from the half line x<=0, y=0
int group(Point a) {
  if (a.y < 0) return -1;
  if (a.y == 0 && a.x >= 0) return 0;
  return 1;
}
bool cmp(Point a, Point b) {
  if (group(a) == group(b)) return cross(a, b) > 0;
  return group(a) < group(b);
}
```

6 Other

6.1 DP DC

```
const int tam=8005;
const ll INF=1e17;
ll locura[tam];
ll pref[tam];
ll dp[805][tam];
ll riesgo(int l, int r){
  if(l>r)return 0;
  return (pref[r]-pref[l-1])*(r-l+1);
}
// solve dp retorna k
ll solvedp(int g,int pos, int izq, int der){
  dp[g][pos]=INF;
  int k;
  for(int i=izq;i<=der;i++){
    ll curr=dp[g-1][i]+riesgo(i+1,pos);
    if(curr<dp[g][pos]){
      dp[g][pos]=curr;
      k=i;
    }
  }
  return k;
}
void solve(int g,int l, int r, int izq, int der){
  if(l>r)return;
  if(l==r){
    solvedp(g,l,izq,der);
    return;
  }
  int mid=(l+r)/2;
```

```
  int k=solvedp(g,mid,izq,der);
  solve(g,mid+1,r,k,der);
  solve(g,l,mid-1,izq,k);
}
int main(){
  // puedo aplicar D&C pq la transicion es dp[G][i]=dp[G
      -1][algo] + C(G,i)
  // la funcion no es decreciente nunca respecto a k
  // algo de G,i <= algo de G,i+1
  int L,G,x;
  cin>>L>>G;
  if(G>L)G=L;
  for(int i=1;i<=L;i++){
    cin>>locura[i];
    pref[i]=pref[i-1]+locura[i];
  }
  for(int i=1;i<=L;i++){
    dp[1][i]=riesgo(1,i);// caso base cuando solo tomo un
         guardia
  }
  for(int i=2;i<=G;i++){
    solve(i,1,L,1,L);
  }
  cout<<dp[G][L]<<endl;

  return 0;
}
// https://www.hackerrank.com/contests/ioi-2014-practice-
   contest-2/challenges/guardians-lunatics-ioi14/problem
```

6.2 DP DC Amortizado

```
const int tam=100005;
ll a[tam];
ll cnt[tam];
const ll INF=1e16;
ll dp[25][tam];//G y pos
ll TOT=0;
int L=1,R;
void add(int x){TOT+=cnt[x]++;}
void del(int x){TOT-=--cnt[x];}
ll query(int l,int r){
  while(L>l) add(a[--L]);
  while(R<r) add(a[++R]);
  while(L<l) del(a[L++]);
  while(R>r) del(a[R--]);
  return TOT;
}
int solvedp(int g,int pos, int izq, int der){
  int k=0;
  dp[g][pos]=INF;
  for(int i=izq;i<=min(der,pos-1);i++){
    ll curr=dp[g-1][i]+query(i+1,pos);
    if(curr<dp[g][pos]){
      dp[g][pos]=curr;
      k=i;
    }
  }
  return k;
}
void solve(int g,int l, int r, int izq, int der){
  if(l>r)return;
  int mid=(l+r)/2;
  int k=solvedp(g,mid,izq,der);
  solve(g,l,mid-1,izq,k);
  solve(g,mid+1,r,k,der);
}
int main(){
  fast
  fast
  ll n,k;
  cin>>n>>k;
  ll acum=0;
  for(int i=1;i<=n;i++){
    cin>>a[i];
    acum+=cnt[a[i]];cnt[a[i]]++;
    dp[1][i]=acum;
  }
  memset(cnt,0,sizeof(cnt));
  for(int i=2;i<=k;i++){
    solve(i,1,n,1,n);
  }
  cout<<dp[k][n]<<endl;
  return 0;
}
```

6.3 Index Compression

```
for (int i = 1; i <= n; i++) {
  cin >> valor[i];
  ind.pb(valor[i]);
}
sort(ind.begin(), ind.end());
for (int i = 1; i <= n; i++) {
  valor[i] = lower_bound(ind.begin(), ind.end(), valor[i
      ]) - ind.begin() + 1;
}
```

6.4 Knapsack Optimization

```
bitset<100001> posi;
posi[0] = 1;
for (int t : comps) posi |= posi << t;
for (int i = 1; i <= n; ++i) cout << posi[i];

// cuando suma maxima es tam = 2e5
```

```
// entonces la cantidad de numeros diferentes es sqrt(2e5
    )
// lo que hago es dejar como maximo 2 repeticiones en
    cada valor
// entonces cada dos i's le paso uno a 2*i y me queda
    solo sqrt(n) numeros
// ya que cada i solo aparece maximo 2 veces

for(int i=1;i<tam;i++){
  if(cant[i]>=3){
    int mv=cant[i]/2;
    if(cant[i]%2==0)mv--;
    cant[i]-=mv*2;
    cant[2*i]+=mv;
  }
}

bitset<tam> dp;
dp[0]=1;
for(int i=1;i<tam;i++){// importante empezar en 1
  for(int l=0;l<cant[i];l++){
    dp|=dp<<i;
  }
}
```

6.5 Kth Permutation

```
int _find(ll &k, ll n){
  if(n==1) return 0;
  n--;
  int ind;
  ll n2 = n;
  while(k >= n2 && n > 1) {
    n2 *= (n-1);
    n--;
  }
  ind = k / n2;
  k %= n2;
  return ind;
}

vi kthPermutation(ll n, ll k){
  vi Ans;
  set<int> st;
  for (int i = 1; i <= n; i++) st.insert(i);
  auto it = st.begin();
  k--;
  for(int i = 0; i < n; i++) {
    int index = _find(k, n-i);
    advance(it, index);
    Ans.pb(*it);
    st.erase(it);
    it = st.begin();
  }
  return Ans;
}

int main(){
  vi res;
  res = kthPermutation(4, 2);
  for(int i = 0; i < res.size(); i++) cout << res[i] << "
    ";
  return 0;
}

// https://codeforces.com/contest/1443/problem/E
```

6.6 LIS

```
int LIS(vi &a){
  vi v;
  for(int i = 0; i < a.size(); i++){
    auto it = lower_bound(v.begin(), v.end(), a[i]); //
        cambiar a upper_bound para LNDS
    if(it == v.end()){
      v.pb(a[i]); // v.size() es LIS que termina en a[i]
    } else {
      *it = a[i]; // it-v.begin()+1 es LIS que termina en
          a[i]
    }
  }
  return v.size();
}

// retornar los indices del LIS (index-0)
vi LIS2(vi v) {
  int n = v.size();
  vi dp; dp.pb(-1e9);
  vi curr(n);
  for (int i = 0; i < n; i++) {
    int izq = 0, der = dp.size() - 1;
    int pos = dp.size(); // Posicion por defecto es al
        final
    while (izq <= der) {
      int mid = (izq + der) / 2;
      if (dp[mid] >= v[i]) { // LNDS if(dp[mid] <= v[i])
        pos = mid; // LNDS pos = mid + 1;
        der = mid - 1; // LNDS izq = mid + 1;
      } else {
        izq = mid + 1; // LNDS der = mid - 1;
      }
    }
    curr[i] = pos;
    if (pos == dp.size()) {
      dp.push_back(v[i]);
    } else {
      dp[pos] = v[i];
    }
  }
```

```
  vi ans;
  int x = dp.size() - 1;
  for (int i = n - 1; i >= 0; i--) {
    if (curr[i] == x) {
      ans.pb(i);
      x--;
    }
  }
  reverse(ans.begin(), ans.end());
  return ans;
}
```

6.7 Pragmas

```
#include <iostream>
#include <chrono>
#include <thread>

int main()
{
    using namespace std::chrono_literals;

    std::this_thread::sleep_for(-9999999999999ms);
}

//------------------

#include <iostream>
using namespace std;
#pragma GCC target ("avx2")
#pragma GCC optimization ("O3")
#pragma GCC optimization ("unroll-loops")
#pragma GCC target("popcnt")

#pragma GCC target("avx,avx2,sse3,ssse3,sse4.1,sse4.2,
   tune=native")
#pragma GCC optimize(3)
#pragma GCC optimize("O3")
#pragma GCC optimize("inline")
#pragma GCC optimize("-fgcse")
#pragma GCC optimize("-fgcse-lm")
#pragma GCC optimize("-fipa-sra")
#pragma GCC optimize("-ftree-pre")
#pragma GCC optimize("-ftree-vrp")
#pragma GCC optimize("-fpeephole2")
#pragma GCC optimize("-fsched-spec")
#pragma GCC optimize("-falign-jumps")
#pragma GCC optimize("-falign-loops")
#pragma GCC optimize("-falign-labels")
#pragma GCC optimize("-fdevirtualize")
#pragma GCC optimize("-fcaller-saves")
#pragma GCC optimize("-fcrossjumping")
#pragma GCC optimize("-fthread-jumps")
#pragma GCC optimize("-freorder-blocks")
#pragma GCC optimize("-fschedule-insns")
#pragma GCC optimize("inline-functions")
#pragma GCC optimize("-ftree-tail-merge")
#pragma GCC optimize("-fschedule-insns2")
#pragma GCC optimize("-fstrict-aliasing")
#pragma GCC optimize("-falign-functions")
#pragma GCC optimize("-fcse-follow-jumps")
#pragma GCC optimize("-fsched-interblock")
#pragma GCC optimize("-fpartial-inlining")
#pragma GCC optimize("no-stack-protector")
#pragma GCC optimize("-freorder-functions")
#pragma GCC optimize("-findirect-inlining")
#pragma GCC optimize("-fhoist-adjacent-loads")
#pragma GCC optimize("-frerun-cse-after-loop")
#pragma GCC optimize("inline-small-functions")
#pragma GCC optimize("-finline-small-functions")
#pragma GCC optimize("-ftree-switch-conversion")
#pragma GCC optimize("-foptimize-sibling-calls")
#pragma GCC optimize("-fexpensive-optimizations")
#pragma GCC optimize("inline-functions-called-once")
#pragma GCC optimize("-fdelete-null-pointer-checks")
```

6.8 Puntos con el mismo slope

```
int dx=X-x;
int dy=Y-y;
int g=__gcd(abs(dx),abs(dy));
pair<ii,int> meto={{x,y},h};
if(dx<0 || (dx==0 && dy<0)){
        dx=-dx;
        dy=-dy;
}
dx/=g;
dy/=g;
M[{dx,dy}].pb(meto);


int dx=X-x;
int dy=Y-y;
int g=__gcd(abs(dx),abs(dy));
pair<ii,int> meto={{x,y},h};
dx/=g;
dy/=g;
M[{dx,dy}].pb(meto);
```

6.9 Simulated Annealing

```
#include <bits/stdc++.h>
#define int long long
#define pb push_back
#define F first
```

```
#define S second
#define vi vector<int>
#define ll long double
#define FIFO ios_base::sync_with_stdio(0);cin.tie(0);cout.tie(0)
using namespace std;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());

struct Point{
    ll x,y;
    bool operator == (Point b) { return x == b.x && y == b.y; }
    bool operator != (Point b) { return !(*this == b); }
    bool operator < (const Point &o) const { return y < o.y || (y == o.y && x < o.x); }
    bool operator > (const Point &o) const { return y > o.y || (y == o.y && x > o.x); }
};
Point operator +(const Point &A, const Point &B) { return {A.x+B.x, A.y+B.y};}
Point operator -(const Point &A, const Point &B) { return {A.x-B.x, A.y-B.y};}
Point operator *(const Point &A, const ll &K) { return {A.x*K, A.y*K};}
ll dot(const Point &A, const Point &B) { return A.x*B.x + A.y*B.y; }
ll cross(const Point &A, const Point &B) { return A.x*B.y - A.y*B.x; }
ll turn(const Point &A, const Point &B, const Point &C) { return cross(B-A,C-A); }
ll dist2(const Point &A, const Point &B) { return dot(A-B,A-B); }

int n;
vector<Point> P;
vi peso;

double calcularMagnitud(const Point &p) {
    return sqrt(p.x * p.x + p.y * p.y);
}

ll costo(Point x) {
    ll res=0;
    for(int i=0;i<n;i++){
        res+=sqrt((P[i].x-x.x)*(P[i].x-x.x)+(P[i].y-x.y)*(P[i].y-x.y))*peso[i];
    }
    return res;
}

signed main()
{
    FIFO;
    cin>>n;
    for(int i=0;i<n;i++){
        ll x,y,w;
        cin>>x>>y>>w;
        P.pb({x,y});
        peso.pb(w);
    }
    cout<<fixed<<setprecision(3);

    ll tempereatura=1000;
    Point res;
    res.x = uniform_real_distribution<double>(-1000,1000)(rng);
    res.y = uniform_real_distribution<double>(-1000,1000)(rng);
    ll respuesta=costo(res);
    // cout<<costo({-23028.0752575625, 23636.2571347542})<<endl;
    // return 0;
    vector<ll> T;
    while(clock() / (double) CLOCKS_PER_SEC <= 0.975749){
        Point nuevo = res;
        ll randi = uniform_real_distribution<double>(-1000,1000)(rng);
        nuevo.x += randi * tempereatura;
        randi = uniform_real_distribution<double>(-1000,1000)(rng);
        nuevo.y += randi * tempereatura;
        // cout<<nuevo.x<<" "<<nuevo.y<<" "<<costo(nuevo)<<"\n";
        ll nuevo_costo = costo(nuevo);
        if(nuevo_costo < respuesta){
            res = nuevo;
            respuesta = nuevo_costo;
        }
        else{
                        double DELTA = abs(respuesta - nuevo_costo);
            double prob = exp(DELTA / tempereatura);
            if(prob < uniform_real_distribution<double>(0,1)(rng)){
                                res = nuevo;
                                respuesta = nuevo_costo;
                        }
        }
        T.pb(tempereatura);
        tempereatura*=0.9999;
    }

    cout<<res.x<<" "<<res.y<<"\n";

    return 0;
}

// https://www.luogu.com.cn/problem/P1337
```

6.10 SOS DP

```
// iterative version
for (int mask = 0; mask < (1 << N); ++mask) {
  dp[mask][-1] = A[mask];  // handle base case separately
       (leaf states)
  for (int i = 0; i < N; ++i) {
    if (mask & (1 << i))
      dp[mask][i] = dp[mask][i - 1] + dp[mask ^ (1 << i)
          ][i - 1];
    else
      dp[mask][i] = dp[mask][i - 1];
  }
  F[mask] = dp[mask][N - 1];
}

// memory optimized, super easy to code.
for (int i = 0; i < (1 << N); ++i)
  F[i] = A[i];
for (int i = 0; i < N; ++i)
  for (int mask = 0; mask < (1 << N); ++mask) {
    if (mask & (1 << i))
      F[mask] += F[mask ^ (1 << i)];
  }
```

6.11 Submascaras

```
for(int mask=0;mask<=16;mask++){
   for(int submask=mask;submask>0;submask=(submask-1)&
       mask){

   }
}
```

6.12 Ternary Search

```
double ternary_search(double l, double r) {
  double eps = 1e-9; // set the error limit here
  while (r - l > eps) {
    double m1 = l + (r - l) / 3;
    double m2 = r - (r - l) / 3;
    double f1 = f(m1); // evaluates the function at m1
    double f2 = f(m2); // evaluates the function at m2
    if (f1 < f2)
      l = m1;
    else
      r = m2;
  }
  return f(l); // return the maximum of f(x) in [l, r]
}
```

7 Theory

DP Optimization Theory

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

Notes:

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

Combinatorics

Sums

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

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

Binomial coefficients

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

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

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

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

Catalan numbers.

- $C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!n!}$ con $n \geq 0$, $C_0 = 1$ y $C_{n+1} = \frac{2(2n+1)}{n+2}C_n$
  $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$
- 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670
- C_n is the number of: properly nested sequences of n pairs of parentheses; rooted ordered binary trees with $n+1$ leaves; triangulations of a convex $(n+2)$-gon.

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

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

Stirling numbers of 2^{nd} kind. $S_{n,k}$ is the number of ways to partition a set of n elements into exactly k non-empty subsets. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = 1$. $x^n = \sum_{k=0}^{n} S_{n,k}\, x^{\underline{k}}$

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

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

Eulerian numbers. $E(n,k)$ is the number of permutations with exactly k descents $(i : \pi_i < \pi_{i+1})$ / ascents $(\pi_i > \pi_{i+1})$ / excedances $(\pi_i > i)$ / $k+1$ weak

excedances ($\pi_i \geq i$).
Formula: $E(n,k) = (k+1)E(n-1,k) + (n-k)E(n-1,k-1)$. $x^n = \sum_{k=0}^{n-1} E(n,k)\binom{x+k}{n}$.

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

Number Theory

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

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

Extended GCD

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

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

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

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

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

Miller-Rabin's primality test. Given $n = 2^rs + 1$ 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 $z = 0, 1, \ldots, n-1$, 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 $x \lor y$ 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 $n-1 \geq d_1 \geq d_2 \geq \dots \geq d_n \geq 0$ is a degree sequence of some undirected simple graph iff $\sum d_i$ is even and $d_1 + \dots + d_k \leq k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i)$ 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 $a_1 \oplus a_2 \oplus \dots \oplus a_n = 0$ (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`.
`a + b = 2 * (a \& b) + (a \oplus b)`

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, $\phi =$ latitude, $\lambda =$ longitude, $\Delta =$ difference (all the values in radians)
- $\sigma =$ 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 $\prod_{j=1}^{m} \frac{p_j^{e_j+1}-1}{e_j-1}$

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.