Contents

1 Formulas.

1.1 Arithmetic Function.

$$\sigma_k(n) = \sum_{d|n} d^k = \prod_{i=1}^{\omega(n)} \frac{{p_i}^{(a_i+1)k} - 1}{p_i^k - 1}$$

$$J_k(n) = n^k \prod_{p|n} (1 - \frac{1}{p^k})$$

$J_k(n)$ is the number of k-tuples of positive integers all less than or equal to n that form a coprime $(k+1)$-tuple together with n.

$$\sum_{\delta|n} J_k(\delta) = n^k$$

$$\sum_{\delta|n} \delta^s J_r(\delta) J_s\left(\frac{n}{\delta}\right) = J_{r+s}(n)$$

$$\sum_{\delta|n} \varphi(\delta) d(\frac{n}{\delta}) = \sigma(n), \sum_{\delta|n} |\mu(\delta)| = 2^{\omega(n)}$$

$$\sum_{\delta|n} 2^{\omega(\delta)} = d(n^2), \sum_{\delta|n} d(\delta^2) = d^2(n)$$

$$\sum_{\delta|n} d(\frac{n}{\delta}) 2^{\omega(\delta)} = d^2(n), \sum_{\delta|n} \frac{\mu(\delta)}{\delta} = \frac{\varphi(n)}{n}$$

$$\sum_{\delta|n} \frac{\mu(\delta)}{\varphi(\delta)} = d(n), \sum_{\delta|n} \frac{\mu^2(\delta)}{\varphi(\delta)} = \frac{n}{\varphi(n)}$$

$$n|\varphi(a^n - 1)$$

$$\sum_{\substack{1\le k\le n \\ \gcd(k,n)=1}} f(\gcd(k-1,n)) = \varphi(n) \sum_{d|n} \frac{(u*f)(d)}{\varphi(d)}$$

$$\varphi\big(lcm(m,n)\big)\varphi(\gcd(m,n)) = \varphi(m)\varphi(n)$$

$$\sum_{\delta|n} d^3(\delta) = (\sum_{\delta|n} d(\delta))^2$$

$$d(uv) = \sum_{\delta|\gcd(u,v))} \mu(\delta) d\left(\frac{u}{\delta}\right) d(\frac{v}{\delta})$$

$$\sigma_k(u)\sigma_k(v) = \sum_{\delta|\gcd(u,v))} \delta^k \sigma_k(\frac{uv}{\delta^2})$$

$$\mu(n) = \sum_{k=1}^{n} [\gcd(k,n) = 1] \cos 2\pi \frac{k}{n}$$

$$\varphi(n) = \sum_{k=1}^{n} [\gcd(k,n) = 1] = \sum_{k=1}^{n} \gcd(k,n) \cos 2\pi \frac{k}{n}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n} (f*g)(k) \\ \sum_{k=1}^{n} S(\left\lfloor\frac{n}{k}\right\rfloor) = \sum_{i=1}^{n} f(i) \sum_{j=1}^{\left\lfloor\frac{n}{i}\right\rfloor} (g*1)(j) \end{cases}$$

$$\begin{cases} S(n) = \sum_{k=1}^{n} (f \cdot g)(k), g\ completely\ multiplicative \\ \sum_{k=1}^{n} S(\left\lfloor\frac{n}{k}\right\rfloor) g(k) = \sum_{k=1}^{n} (f*1)(k) g(k) \end{cases}$$

1.2 Binomial Coefficients.

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

$$\sum_{k\le n} \binom{r+k}{k} = \binom{r+n+1}{n}$$

$$\sum_{k=0}^{n} \binom{k}{m} = \binom{n+1}{m+1}$$

$$\sqrt{1+z} = 1 + \sum_{k=1}^{\infty} \frac{(-1)^{k-1}}{k \times 2^{2k-1}} \binom{2k-2}{k-1} z^k$$

$$\sum_{k=0}^{r} \binom{r-k}{m} \binom{s+k}{n} = \binom{r+s+1}{m+n+1}$$

$$C_{n,m} = \binom{n+m}{m} - \binom{n+m}{m-1}, n \ge m$$

$$\binom{n}{k} \equiv [n\&k = k] \pmod 2$$

1.3 Lucas's theorem.

For non-negative integers *m* and *n* and a prime *p*, the following congruence relation holds:

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod p$$

where

$$m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0$$

and

$$n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$$

are the base p expansions of m and n respectively. This uses the convention that $\binom{m}{n} = 0$ if $m < n$.

1.4 Fibonacci Numbers.

$$F(z) = \frac{z}{1 - z - z^2}$$

$$f_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}, \phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

$$\sum_{k=1}^{n} f_k = f_{n+2} - 1$$

$$\sum_{k=1}^{n} {f_k}^2 = f_n f_{n+1}$$

$$\sum_{k=0}^{n} f_k f_{n-k} = \frac{1}{5}(n-1) f_n + \frac{2}{5} n f_{n-1}$$

$${f_n}^2 + (-1)^n = f_{n+1} f_{n-1}$$

$$f_{n+k} = f_n f_{k+1} + f_{n-1} f_k$$

$$f_{2n+1} = {f_n}^2 + {f_{n+1}}^2$$

$$(-1)^k f_{n-k} = f_n f_{k-1} - f_{n-1} f_k$$

Modulo $f_n, f_{mn+r} \equiv \begin{cases} f_r, & m \bmod 4 = 0; \\ (-1)^{r+1} f_{n-r}, & m \bmod 4 = 1; \\ (-1)^n f_r, & m \bmod 4 = 2; \\ (-1)^{r+1+n} f_{n-r}, & m \bmod 4 = 3; \end{cases}$

1.5 Stirling Cycle Numbers.

Arrangements of an n elements set into k cycles.

$$\begin{bmatrix} n+1 \\ k \end{bmatrix} = n \begin{bmatrix} n \\ k \end{bmatrix} + \begin{bmatrix} n \\ k-1 \end{bmatrix}, \quad \begin{bmatrix} n+1 \\ 2 \end{bmatrix} = n!\, H_n$$

$$x^{\underline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} (-1)^{n-k} x^k, \qquad x^{\overline{n}} = \sum_k \begin{bmatrix} n \\ k \end{bmatrix} x^k$$

1.6 Stirling Subset Numbers.

Partitions of an n elements set into k non-empty sets.

$$\begin{Bmatrix} n+1 \\ k \end{Bmatrix} = k \begin{Bmatrix} n \\ k \end{Bmatrix} + \begin{Bmatrix} n \\ k-1 \end{Bmatrix}$$

$$x^n = \sum_k \begin{Bmatrix} n \\ k \end{Bmatrix} x^{\underline{k}} = \sum_k \begin{Bmatrix} n \\ k \end{Bmatrix} (-1)^{n-k} x^{\overline{k}}$$

$$m! \begin{Bmatrix} n \\ m \end{Bmatrix} = \sum_k \binom{m}{k} k^n (-1)^{m-k}$$

$$\begin{Bmatrix} n+1 \\ m+1 \end{Bmatrix} = \sum_k \binom{n}{k} \begin{Bmatrix} k \\ m \end{Bmatrix} = \sum_{k=0}^{n} \begin{Bmatrix} k \\ m \end{Bmatrix} (m+1)^{n-k}$$

1.7 Eulerian Numbers.

Permutations $\{1,2,\ldots,n\}$ with k ascents.

$$\left\langle {n \atop k} \right\rangle = (k+1) \left\langle {n-1 \atop k} \right\rangle + (n-k) \left\langle {n-1 \atop k-1} \right\rangle$$

$$x^n = \sum_k \left\langle {n \atop k} \right\rangle \binom{x+k}{n}$$

$$\left\langle {n \atop m} \right\rangle = \sum_{k=0}^{m} \binom{n+1}{k} (m+1-k)^n (-1)^k$$

1.8 Harmonic Numbers.

$$\sum_{k=1}^{n} H_k = (n+1) H_n - n$$

$$\sum_{k=1}^{n} k H_k = \frac{n(n+1)}{2} H_n - \frac{n(n-1)}{4}$$

$$\sum_{k=1}^{n} \binom{k}{m} H_k = \binom{n+1}{m+1} \left(H_{n+1} - \frac{1}{m+1}\right)$$

1.9 Pentagonal Number Theorem.

$$\prod_{n-1}^{\infty} (1 - x^n) = \sum_{n=-\infty}^{\infty} (-1)^k x^{k(3k-1)/2}$$

$$p(n) = p(n-1) + p(n-2) - p(n-5) - p(n-7) + \cdots$$

$$f(n,k) = p(n) - p(n-k) - p(n-2k) + p(n-5k) + p(n-7k) - \cdots$$

1.10 Bell Numbers.

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$$

$$B_{p^m+n} \equiv m B_n + B_{n+1} \quad (mod\ p)$$

1.11 Bernoulli Numbers.

$$B_n = 1 - \sum_{k=0}^{n-1} \binom{n}{k} \frac{B_k}{n-k+1}$$

$$G(x) = \sum_{k=0}^{\infty} \frac{B_k}{k!} x^k = \frac{1}{\sum_{k=0}^{\infty} \frac{x^k}{(k+1)!}}$$

$$S_m(n) = \frac{1}{m+1} \sum_{k=0}^{m} \binom{m+1}{k} B_k n^{m-k+1}$$

1.12 Tetrahedron Volumen.

$$V = \frac{\sqrt{4u^2v^2w^2 - \sum_{cyc} u^2(v^2 + w^2 - U^2)^2 + \prod_{cyc}(v^2 + w^2 - U^2)}}{12}$$

1.13 BEST Thoerem.

Counting the number of different Eulerian circuits in directed graphs.

$$ec(G) = t_w(G) \prod_{v \in V} (\deg(v) - 1)!$$

When calculating $t_w(G)$ for directed multigraphs, the entry $q_{i,j}$ for distinct i and j equals $-m$, where m is the number of edges from i to j, and the entry $q_{i,i}$ equals the indegree of i minus the number of loops at i. It is a property of Eulerian graphs that $t_v(G) = t_w(G)$ for every two vertices u and w in a connected Eulerian graph G.

1.14 Cycles.

Let the number of n-permutations whose cycle lengths all belong to the set S be denoted

by $g_s(n)$. Then

$$\sum_{n=0}^{\infty} g_s(n) \frac{x^n}{n!} = exp\left(\sum_{n \in S} \frac{x^n}{n}\right)$$

1.15 Derangements.

$$D(n) = (n-1)\big(D(n-1) + D(n-2)\big) = nD(n-1) + (-1)^n$$

1.16 Involutions.

An involution is a permutation with maximum cycle length 2, and it is its own inverse.

$$a(n) = a(n-1) + (n-1)a(n-2)$$

$$a(0) = a(1) = 1$$

$$1, 2, 4, 10, 26, 76, 232, 764, 2620, 9496, 35696, 140152$$

1.17 Factorial, Stirling's Aproximations.

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

2 Graph Theory

2.1 Erdos-Gallai theorem.

Erdos-Gallai theroem. 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 $d_1 + d_2 + \cdots + d_k \leq k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i)$ for all $k = 1,2,\ldots,n-1$.

```
bool is_graphic(vector<int> d){
  int n = d.size();
  sort(d.rbegin(), d.rend());
  vector<int> s(n + 1);
  for (int i = 0; i < n; ++i)
    s[i + 1] = s[i] + d[i];
  if (s[n] % 2)
    return false;
  for (int k = 1; k <= n; ++k){
    int p = lower_bound(d.begin() + k, d.end(), k,
                        greater<int>()) - d.begin();
    if (s[k] > k * (p - 1) + s[n] - s[p])
      return false;
  }
  return true;
}
```

2.2 Euler's theorem.

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 F is the number of connected components.
Corollary: $V - E + F = 2$ for a 3D polyhedron.

2.3 Kirchhoff's Matrix-tree theorem, Cayley's formula.

Kirchhoff'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 1equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T.

Kirchhoff's theorem for directed multigraphs can be modified to count the number of oriented spanning trees in directed multigraphs. The matrix T is constructed as follows:

- The entry $t_{ij} = -m$ for all $i \neq j$, where m is the number of edges from i to j;

- The entry $t_{ii} = indeg_i - loops_i$, where $loops_i$ is the number of loops at i.

The number of oriented spanning trees rooted at a vertex i is the determinant of the matrix gotten by removing the i-th row and column of T.

Cayley's formula. Counts the number of distinct labeled trees of a complete graph Kn. n^{n-2}

2.4 Generating ($\mathbf{k}$-vertex-connected, $\mathbf{l}$-edge-connected, $\mathbf{d}$-minimum-degre) graph.

Sea G un grafo, k el menor numero de vertices a quitar para hacer a G desconexo, l el menor numero de arcos a quitar para hacer a G desconexo, y d el minimo degree de todos los vertices en G. Para todo grafo se cumple que $k \leq l \leq d$.

Para construir un grafo dados los valores de k, l y d:

Primero se verifica que se cumpla $k \leq l \leq d$, luego se ponen $2(d+1)$ vertices y luego se hacen con los primeros $(d+1)$ vertices un subgrafo completamente conexo, lo mismo con los segundos $(d+1)$ vertices, entonces se ponen l arcos uniendo l nodos del primer subgrafo con k del segundo.

2.5 Stable Marriages Problem.

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.

2.6 How to split graph $\mathbf{G}$ in one or two subgraphs with even degree for every node.

Siempre existe una solucion.

Algoritmo Recursivo.

Para un grafo con $n \leq 3$ nodos es facil encontrar la solucion.

Para un grafo con $n > 3$ nodos. Se escoge cualquier nodo u con el degree impar y se elimina del grafo, de no existir, el propio grafo es solucion. Se cambia el subgrafo de los nodos vecinos de u por su complemento. Se hace la llamada recursiva con el grafo transformado. Al tener u un numero impar de vecinos, entonces los subgrafos solucion tendran uno de ellos una cantidad par de nodos vecino de u, llamemosle $Geven$ y el otro una cantidad impar de nodos vecinos de u, llamemosle $Godd$, entonces al cambiar en $Geven$ y en $Godd$ los subgrafos vecinos de u por sus complementos, en $Godd$ no habra cambios en los degrees de los nodos, pero en $Geven$ los nodos vecinos de u tendran degree impar, entonces se adiciona u a $Geven$ para arreglar los degrees de sus vecinos y el degree de u sera par tambien.

2.7 Euler tours.

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.

2.8 How to make a graph given the degrees paritys of every nodes and the set of possibles arcs to put. $O(n+m)$

```
string s; //s[i] = 'o' for odd parity OR 'e' for even
int n, m;
vector<int> g[MAXN];
bool mk[MAXN];
vector<par> sol;
bool ok( int u ){
  mk[u] = true;
  int d = 0;
  for( int i = 0; i < g[u].size(); i++ )
    if( !mk[ g[u][i] ] && !ok( g[u][i] ) ){
      d++;
      sol.push_back( par( u , g[u][i] ) );
    }
  int r = ( s[u] == 'o' ) ? 1 : 0;
  return d%2 == r;
}
bool solve( int u ){
  bool yes = true;
  for( int i = 0; i < n; i++ )
    if( !mk[i] && !ok(i) ){
      yes = false; break;
    }
  return yes;
}
```

2.9 Bridges, Articulations Points, Biconnected Componentes, BC-Tree.

Biconnected Graphs. In graph theory, a biconnected graph is a connected and "nonseparable" graph, meaning that if any one vertex were to be removed, the graph will remain connected. Therefore a biconnected graph has no articulation vertices.
Lemma: In a biconnected graph with $n \geq 3$ vertices, for any three different vertices a, b, c, there is a simple path to from a to b going through c.

Biconnected component (Block). In graph theory, a biconnected component is a maximal biconnected subgraph.

Block-Cut tree (BC-tree). The structure of the blocks and cutpoints of a connected graph can be described by a tree called the block-cut tree or BC-tree. This tree has a vertex for each block and for each articulation point of the given graph. There is an edge in the block-cut tree for each pair of a block and an articulation point that belongs to that block.

```
template<const int MAXN,const int MAXM>
struct BridgesAPointsBComponents{
  int dfstime[MAXN], low[MAXN], dfsnum, n;
  vector<int> g[MAXN];
  bool mk[MAXM];
  vector< pair<int,int> > &edges;
  vector<int> apoints, bridges;

  stack<int> stk;
  vector< vector<int> > bcomps;
  BridgesAPointsBComponents( int n,
          vector< pair<int,int> > &edges ) :
          edges(edges), n(n){
    for( int u = 0; u <= n; u++ )
      low[u] = dfstime[u] = 0;
    for( int e = 0; e < edges.size(); e++ ){
      mk[e] = false;
      g[ edges[e].first ].push_back(e);
      g[ edges[e].second].push_back(e);
    }
    dfsnum = 0;
    findAll();
  }
  int ady(int e,int u){
    return (edges[e].first == u ? edges[e].second :
                                  edges[e].first);
  }
  void dfs( int u ){
    bool isAp = false;
    dfstime[u] = low[u] = ++dfsnum;
    stk.push(u);

    for( auto e : g[u] ){
      int v = ady( e , u );
      if( !dfstime[v] ){
        mk[e] = true;
        dfs( v );
        low[u] = min( low[u] , low[v] );
        if( low[v] > dfstime[u] ){
          bridges.push_back(e);
        }
        if( low[v] >= dfstime[u] ){
          isAp = ( dfstime[u] > 1 || dfstime[v] > 2 );
          bcomps.push_back({u});
          while (bcomps.back().back() != v){
            bcomps.back().push_back(stk.top());
            stk.pop();
          }
        }
      }
      else if( !mk[e] ){
        low[u] = min( low[u] , dfstime[v] );
      }
    }
    if( isAp ){
      apoints.push_back(u);
    }
  }
  void findAll(){
    for( int u = 1; u <= n; u++ ){
      if( !dfstime[u] ){
        dfsnum = 0;
        dfs(u);
      }
    }
  }
};
```

2.10 Dominator Tree, $O(m \log n)$.

```
struct graph{
  int n;
  vector<vector<int>> adj, radj;

  graph(int n) : n(n), adj(n), radj(n) {}

  void add_edge(int src, int dst){
    adj[src].push_back(dst);
    radj[dst].push_back(src);
  }
  vector<int> rank, semi, low, anc;
  int eval(int v){
    if (anc[v] < n && anc[anc[v]] < n){
      int x = eval(anc[v]);
      if (rank[semi[low[v]]] > rank[semi[x]])
        low[v] = x;
      anc[v] = anc[anc[v]];
    }
```

```
    return low[v];
  }
  vector<int> prev, ord;
  void dfs(int u){
    rank[u] = ord.size();
    ord.push_back(u);
    for (auto v : adj[u]){
      if (rank[v] < n)
        continue;
      dfs(v);
      prev[v] = u;
    }
  }
  // idom[u] is an immediate dominator of u
  vector<int> idom;

  void dominator_tree(int r){
    idom.assign(n, n);
    prev = rank = anc = idom;
    semi.resize(n);
    iota(semi.begin(), semi.end(), 0);
    low = semi;
    ord.clear();
    dfs(r);
    vector<vector<int>> dom(n);
    for(int i =(int) ord.size() - 1; i >= 1; --i){
      int w = ord[i];
      for (auto v : radj[w]){
        int u = eval(v);
        if (rank[semi[w]] > rank[semi[u]])
          semi[w] = semi[u];
      }
      dom[semi[w]].push_back(w);
      anc[w] = prev[w];
      for (int v : dom[prev[w]]){
        int u = eval(v);
        idom[v] = (rank[prev[w]] > rank[semi[u]]
          ? u : prev[w]);
      }
      dom[prev[w]].clear();
    }
    for (int i = 1; i < (int) ord.size(); ++i){
      int w = ord[i];
      if (idom[w] != semi[w])
        idom[w] = idom[idom[w]];
    }
  }
  vector<int> dominators(int u){
    vector<int> S;
    for (; u < n; u = idom[u])
      S.push_back(u);
    return S;
  }
};
```

2.11 2-SAT

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 $(\bar{x}, y)$ and $(\bar{y}, x)$. The formula is satisfiable iff x and $\bar{x}$ are in distinct SCCs, for all x. To find a satisfiable assignment, consider the graph's SCCs in topological order from, assigning `true' to all variables of the current SCC (if it hasn't been previously assigned `false'), and `false' to all inverses.

3 Graph (Flow)

3.1 Maximum Flow.

```
void addNotDirectedEdge( int u, int v, ll c ){
  ady[E] = v; flow[E] = 0; cap[E] = c;
  nxt[E] = last[u]; last[u] = E++;
  ady[E] = u; flow[E] = 0; cap[E] = c;
  nxt[E] = last[v]; last[v] = E++;
}
```

3.2 Arcs of Minimum Cut.

From the source vertex, do a dfs along edges in the residual network (i.e., non-saturated edges and back edges of edges that have flow), and mark all vertices that can be reached this way. The cut consists of all edges that go from a marked to an unmarked vertex.

3.3 Maximum Flow of Minimum Cost.

```
template<typename flow_t,typename cost_t,
         const int MAXN,const int MAXM,
         const flow_t oof,const cost_t ooc>
struct MinCostMaxFlow{
  int last[MAXN], ady[2*MAXM],
      nxt[2*MAXM], uback[MAXN],
      eback[MAXN],n,E;
  flow_t cap[2*MAXM], flow[2*MAXM];
  cost_t cost[2*MAXM], cst[MAXN];
  MinCostMaxFlow( int n ) : n(n){
    E = 2;
    for( int i = 0; i <= n; i++ ) last[i] = 0;
  }
  void addDirectedEdge( int u, int v, flow_t c,
                        cost_t cst ){
    ady[E] = v; cap[E] = c; flow[E] = 0;
    cost[E] = +cst;
    nxt[E] = last[u]; last[u] = E++;

    ady[E] = u; cap[E] = 0; flow[E] = 0;
    cost[E] = -cst;
    nxt[E] = last[v]; last[v] = E++;
  }
  void addNotDirectedEdge(int u,int v,flow_t c,
                          cost_t cst ){
    addDirectedEdge( u , v , c , cst );
    addDirectedEdge( v , u , c , cst );
  }
  bool Dijkstra( int s, int t ){
    for( int i = 0; i <= n; i++ )
      cst[i] = ooc;
    cst[s] = 0;
    priority_queue< pair<cost_t,int> > pq;
    pq.push( { 0 , s } );
    while( !pq.empty() ){
      int u = pq.top().second;
      pq.pop();
      //NUNCA utilizar el arreglo bool
      //mk[] para este dijkstra--->da WA
      if( u == t ) return true;
      for( int e = last[u]; e; e = nxt[e] ){
        int v = ady[e];
        cost_t w = cost[e];
        if( flow[e] < cap[e] &&
            cst[v] > cst[u] + w ){
          cst[v] = cst[u] + w;
          uback[v] = u;
          eback[v] = e;
          pq.push( { -cst[v] , v } );
        }
      }
    }
    return (cst[t] != ooc);
  }
  pair<cost_t,flow_t> minCostMaxFlow(int s,int t,
                                     flow_t MAXFLOW ){
    flow_t maxflow = 0;
    cost_t mincost = 0;
    while( maxflow < MAXFLOW && Dijkstra(s,t) ){
      flow_t f = oof;
      for( int u = t; u != s; u = uback[u] ){
        int e = eback[u];
        f = min( f , cap[e] - flow[e] );
      }
      f = min( f , MAXFLOW - maxflow );
      maxflow += f;
      mincost += cst[t] * (cost_t)f;
      for( int u = t; u != s; u = uback[u] ){
        int e = eback[u];
        flow[e] += f;
        flow[e^1] -= f;
      }
    }
    return { mincost , maxflow };
  }
};
```

3.4 Gomory-Hu Tree.

The Gomory–Hu tree of an undirected graph with capacities is a weighted tree that represents the minimum s-t cuts for all s-t pairs in the graph. The Gomory–Hu tree can be constructed in $O(|V|-1) \cdot MaxFlow\ Computations$. The minimum s-t cuts for all s-t pairs in the graph are eguals to the minimum s-t cuts en the tree. The minimum s-t cut in a tree es the minimum arc in path s-t.

```
vector<pair<int,int> > ght[MAXN]; /** El arbol **/
int p[MAXN];
void gomory_hu_tree(){
  fill( p , p + n + 1, 1 );
  for( int u = 2; u <= n; u++ ){
    for( int e = 2; e < E; e++ ) flow[e] = 0;
    int mxf = maxflow( u , p[u] );
    ght[u].push_back( { p[u] , mxf } );
    ght[ p[u] ].push_back( { u , mxf } );
    for( int v = u+1; v <= n; v++ )
      if( lev[v] && p[v] == p[u] ) p[v] = u;
  }
}
```

3.5 Maximum Flow with edge Demands.

Dado un grafo dirigido donde cada lado tienen una capacidad c_e y una demanda d_e, calcular el maximo flujo tal que por cada arco e, pasa un flujo $d_e \leq flow_e \leq c_e$.

```
struct MaxFlowEdgeDemands{
  MaxFlow<flow_t,MAXN,MAXM,oo> mxf;
  int n;
  flow_t inDemands[MAXN],
         outDemands[MAXN], D = 0;
  MaxFlowEdgeDemands( int n ) : n(n){
    mxf = MaxFlow<flow_t,MAXN,MAXM,oo>(n+2);
    for( int i = 0; i <= n; i++ )
      inDemands[i] = outDemands[i] = 0;
  }
  void addDirectedEdge( int u, int v,
                        flow_t demand,flow_t cap){
    mxf.addDirectedEdge( u , v , cap - demand);
    outDemands[u] += demand;
    inDemands[v] += demand;
    D += demand;
  }
  pair<bool,flow_t> maxFlow( int s, int t ){
    int S = n+1, T = n+2;
    mxf.addDirectedEdge( t , s , oo );
    for( int u = 0; u <= n; u++ ){
      if( inDemands[u] )
        mxf.addDirectedEdge(S,u,inDemands[u]);
      if( outDemands[u] )
        mxf.addDirectedEdge(u,T,outDemands[u]);
    }
    flow_t feasibleFlow = mxf.maxFlow(S,T);
    if( feasibleFlow != D ) return {false,0};
    for( int u = 0; u <= n; u++ ){
      if( outDemands[u] )
        mxf.last[u] = mxf.nxt[ mxf.last[u] ];
      if( inDemands[u] )
        mxf.last[u] = mxf.nxt[ mxf.last[u] ];
    }
    flow_t flow = mxf.maxFlow( s , t );
    return { true , flow };
  }
};
```

3.6 Maximum Flow Problems.

Codeforces-E. Biologist. En un laboratorio se quiere realizar un experimento de cambio de sexo a perros. Existen n perros numerados $1,2,\ldots,n$. Realizar el experimento al perro i le cuesta al laboratorio v_i pesos. El laboratorio tiene m patrocinadores, el patrocinador j esta dispuesto a pagar w_j pesos si se le complace, cada patrocinador j escoge un SOLO sexo y varios perros, el patrocinador estara complacido si todos los perros escogidos terminan con el sexo escogido por el. ¿Cuál es el subconjunto de perros a cambiarle el sexo que maximice la ganancia (o minimice la perdida)?

Solucion: Sean M el conjunto de perros que son masculinos, F el conjunto de perros que son femeninos, PM el conjunto de patrocinadores que escogieron el sexo masculino y PF el conjunto de patrocinadores que escogieron el sexo femenino. Se construye una red R con un nodo por cada elemento de los conjuntos M, F, PM y PF y ademas dos nodos s y t. Se crean las aristas:

(s, i, v_i) para todo perro i en M,

(s, i, w_i) para todo patrocinador i en PM,

(i, t, v_i) para todo perro i en F,

(i, t, w_i) para todo patrocinador i en PF,

(i, j, ∞) para todo i en M, y para todo j en $PM \cup PF$,

(j, i, ∞) para todo i en F, y para todo j en $PM \cup PF$,

$$solucion = \sum_{j \in (PM \cup PF)} w_j - mincut(s,t)$$

El subconjunto de perros solucion serian los perros de M a la DERECHA del corte, y los perros de F a la IZQUIERDA del corte. El subconjunto de patrocinadores complacidos serian los patrocinadores de PM a la izquierda del corte y los patrocinadores de PF a la derecha del corte.

4 Graph (Matching)

4.1 Tutte theorem.

A graph $G = (V, E)$, has a perfect matching if and only if for every subset U of V, the subgraph induced by $V - U$ has at most $|U|$ connected components with an odd number of vertices.

4.2 Hall's Theorem

Hall's Theorem. There exists a system of distinct representatives for a family of sets $S_1, S_2, \ldots, S_m$ if and only if the union of any k of these sets contains at least k elements for all $k = 1,2,\ldots,m$.

Hall's Condition. Given a set A, let $N(A)$ be the set of neighbours of A. Then the bipartite graph G with bipartitions X and Y has a perfect matching if and only if $|N(A)| \geq |A|$ for all subsets A of X.

4.3 Vertex covers and independent sets.

Vertex covers and independent sets. Let M, C, I be a maximum matching(MM), a minimum vertex cover(MVC), and a maximum independent set(MIS). 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 $C = (A \cap T) \cup (B \cap S)$.

4.4 Minimum Vertex Cover.

```
bool min_cover1[MAXN],min_cover2[MAXN];
int cola[MAXN];
int min_cover(){
  int enq = 0, deq = 0;
  fill(min_cover1,min_cover1+n+1,false);
  fill(from,from+m+1,0);
  fill(min_cover2,min_cover2+m+1,0);
  int max_matching = kuhn();
  for(int i = 1; i <= m; i++){
    if( from[i] )
```

```
    min_cover1[ from[i] ] = true;
  }
  for(int i = 1; i <= n; i++){
    if( !min_cover1[i] )
      cola[enq++] = i;
  }
  while( deq < enq ){
    int u = cola[deq++];
    for( int i = 0; i < g[u].size(); i++ ){
      int v = g[u][i];
      if( from[v] ){
        min_cover2[v] = true;
        if( min_cover1[ from[v] ] ){
          cola[enq++] = from[v];
          min_cover1[ from[v] ] = false;
        }
      }
    }
  }
  int mnv = 0;
  for( int i = 1; i <= n; i++ ){
    if( min_cover1[i] )
      mnv++;
  }
  for( int i = 1; i <= m; i++ ){
    if( min_cover2[i] )
      mnv++;
  }
  return mnv;
}
```

4.5 *Hungarian.*

```
const int oo = ( 1 << 29 );
const int MAXN = 51;
const int MAXV = GREATER THAT ANY TOTAL SUM OF WEITHS;
int n;
int w[MAXN][MAXN];
int usedx[MAXN], usedy[MAXN];
int from[MAXN];
int labelx[MAXN], labely[MAXN];
bool kuhn( int x ){
  usedx[x] = true;
  for( int y = 1; y <= n; y++ ){
    if( !usedy[y] &&
        labelx[x] + labely[y] == w[x][y] ){
      usedy[y] = true;
      if( !from[y] || kuhn( from[y] ) ){
        from[y] = x;
        return true;
      }
    }
  }
  return false;
}
int hungarian(){
  for( int i = 1; i <= n; i++ ){
    from[i] = 0;
    labely[i] = 0;
  }
  for( int i = 1; i <= n; i++ ){
    labelx[i] = 0;
    for( int j = 1; j <= n; j++ )
      labelx[i] = max(labelx[i],w[i][j]);
  }
  for( int k = 1; k <= n; k++ ){
    while( true ){
      for( int i = 1; i <= n; i++ )
        usedx[i] = usedy[i] = false;
      if( kuhn(k) ) break;
      int exc = oo;
      for( int i = 1; i <= n; i++ ){
        if( usedx[i] ){
          for( int j = 1; j <= n; j++ ){
            if( !usedy[j] )
              exc=min(exc,
                      labelx[i]+labely[j]-w[i][j]);
          }
        }
      }
      if( exc == 0 || exc == oo ) break;
      for( int i = 1; i <= n; i++ ){
        if( usedx[i] ) labelx[i] -= exc;
        if( usedy[i] ) labely[i] += exc;
      }
    }
  }
  int res = 0;
  for( int i = 1; i <= n; i++ ){
    res += labely[i];
    res += labelx[ from[i] ];
  }
  return res;
}
int main() {
  cin >> n;
  for( int i = 1; i <= n; i++ ){
    for( int j = 1; j <= n; j++ )
      cin >> w[i][j];
  }
  int mx = hungarian();
  for( int i = 1; i <= n; i++ ){
    for( int j = 1; j <= n; j++ ){
      w[i][j] = MAXV - w[i][j];
    }
  }
  int mn = ( n * MAXV ) - hungarian();
  cout << mx << ' ' << mn << '\n';
}
```

4.6 *Matching Problems.*

Maximum $|X| - |\boldsymbol{neighbourhood}(X)|$ in bipartite graph. Dado un grafo bipartito con n nodos en una parte y m nodos en la otra parte . Se quiere encontrar un subconjunto de nodos X de la primera parte, tal que si Y es el conjunto de todos los nodos vecinos de X, $|X| - |Y|$ sea maximo.

Solucion: Se halla el $mincover$, X estara formado por todos los nodos que no pertenezcan al $mincover$.

Nodes in all maximum matchings. Dado un grafo bipartito se quiere hallar todos los nodos de la primera parte que pertenecen a todos los maximum matching.

Solucion: Se calcula maximum matching, las aristas que pertenezcan al matching se orientan de la segunda parte a la primera, y las que no pertenezcan al matching se orientan de la primera parte a la segunda parte. Un nodo de la primera parte siempre estara en cualquier matching si y solo si NO existe ningun camino desde algun nodo que no este en el matching de la primera parte hacia el.

Edges that belongs to some maximum matching. Encontrar las aristas que pertencen al menos a un maximum matching.

Solucion. Hallar cualquier maximum matching. Una arista pertenece al menos a un maximum matching si cumple alguna de las siguientes condiciones:

1. Pertenece al maximum matching encontrado.
2. Une a un nodo en el matching con otro que no esta en el matching.
3. Pertenece a algun ciclo alternante ($arcInMatching \rightarrow arcOutmatching \rightarrow arcInMatching$). Una forma de implementacion es direccionar las aristas del matching hacia un lado, y las que no pertenecen al matching hacia el otro, entonces hallar las SCC, las aristas que unen a dos nodos de una misma SCC cumplen esta condicion.
4. Las aristas que pertenecen a algun camino alternante simple que empieze en un nodo fuera del matching. La condicion 2 es un subcaso de esta condicion. Una forma de implementacion es hacer dos dfs(o bfs), uno con las aristas del matching direccionadas hacia la derecha y las demas aristas direccionadas hacia la izquierda, el otro con las aristas direccionadas de forma contraria. Las aristas recorridas en alguno de los dfs(o bfs) cumplen la condicion.

4.7 *Poset*

Poset. (Conjunto parcialmente ordenado) es un conjunto P y una relacion binaria $\leq$ tal que para todo a, b, c en P se cumple que:

1. $a \leq a$ $(Reflexion)$

2. $a \leq b$ y $b \leq c$ implica $a \leq c$ $(Transitividad)$

3. $a \leq b$ y $b \leq a$ implica $a = b$ $(Antisimetria)$

Un par de elementos a, b son $comparables$ si $a \leq b$ ó $b \leq a$. De otra forma son $incomparables$.

Un poset sin elementos incomparables, es un ordenamiento lineal o total.

$a < b$ si $a \leq b$ y $a \neq b$.

Una $chain$ es una secuencia de $a_1 < a_2 < \cdots < a_s$.

Un conjunto A es un $antichain$ si todo par de elementos en A son $incomparables$.

Theorem 1. Sea P un poset finito, entonces la minima cantidad m de $chains$, tal que $P = \bigcup_{i=1}^{m} C_i$ es igual al tamaño de la maxima $antichain$ que se pueda formar de P.

Theorem 2. Sea P un poset finito, entonces la minima cantidad m de $antichains$, tal que $P = \bigcup_{i=1}^{m} A_i$ es igual al tamaño de la maxima $chain$ que se pueda formar de P.

5 Graph (Tree)

5.1 Prüfer Code.

Prüfer Code.

Algoritmo para convertir un árbol en una secuencia de Prüfer. La secuencia de Prüfer de un árbol etiquetado se puede generar removiendo iterativamente los vértices del árbol hasta que queden solamente dos vértices. Específicamente, consideremos un árbol etiquetado T con vértices $\{1, 2, \ldots, n\}$. En el paso i, remover la hoja con la menor etiqueta y hacer que el i-ésimo elemento de la secuencia de Prüfer sea la etiqueta del nodo vecino de la hoja removida. La secuencia del árbol etiquetado es claramente única y tiene longitud $n - 2$.

Algoritmo para convertir una secuencia de Prüfer en un árbol. Sea $\{a_1, a_2, \ldots, a_n\}$ una secuencia de Prüfer. El árbol tendra $n + 2$ nodos, numerados $1, \ldots, n + 2$. El grado de cada nodo será igual al número de veces que éste aparezca en la secuencia más 1. Luego, por cada número a_i en la secuencia, encontrar el nodo j de menor etiqueta con grado 1, y agregar la arista (j, a_i) al árbol. Decrementar los grados de los nodos a_i y j. Al recorrer la secuencia quedaran solo dos nodos u, v con grado 1, entonces se agrega la arista (u, v) al árbol.

5.2 Transformation of a tree into 2-edge-connected graph adding minimum number of edges.

Root the tree at any non-leaf vertex and run dfs. Each time you encounter a leaf, append its number to list $Leaves$. At the end, let $s = Leaves.sz()/2$. For each valid i, connect $Leaves[i]$ and $Leaves[i + s]$.

5.3 Maximum number of disjoint-vertex paths on tree.

Este problema es similar a encontrar en el arbol la minima cantidad de aristas a quitar tal $degree_u \leq 2$ para todo nodo u.

```
int d[MAXN];//despues de llamar a dfs(1,-1),
            //sol tendra las aristas que se quedan.
vector< pair<int,int> > sol;
void dfs( int u, int p ){
  for( int i = 0; i < g[u].size(); i++ )
    if( g[u][i] != p ){
        int v = g[u][i];
        dfs( v , u );
        if( d[u] < 2 && d[v] < 2 ){
            d[u]++;
            d[v]++;
            sol.push_back( { u , v } );
        }
    }
}
```

5.4 Isomorphism Trees.

```
#define all(c) (c).begin(), (c).end()
struct tree{
  int n;
  vector<vector<int>> adj;

  tree(int n) : n(n), adj(n) {}
  void add_edge(int src, int dst){
    adj[src].push_back(dst);
    adj[dst].push_back(src);
  }
  vector<int> centers(){
    vector<int> prev;
    int u = 0;
    for (int k = 0; k < 2; ++k){
      queue<int> q;
      prev.assign(n, -1);
      for (q.push(prev[u] = u); !q.empty(); q.pop()){
        u = q.front();
        for (auto v : adj[u]){
          if (prev[v] >= 0)
            continue;
          q.push(v);
          prev[v] = u;
        }
      }
    }
    vector<int> path = { u };
    while (u != prev[u]){
      path.push_back(u = prev[u]);
    }
    int m = path.size();
    if (m % 2 == 0)
      return {path[m/2-1], path[m/2]};
    else
      return {path[m/2]};
  }

  vector<vector<int>> layer;
  vector<int> prev;

  int levelize(int r){
    prev.assign(n, -1);
    prev[r] = n;
    layer = {{r}};
    while (1){
      vector<int> next;
      for (int u : layer.back()){
        for (int v : adj[u]){
          if (prev[v] >= 0)
            continue;
          prev[v] = u;
          next.push_back(v);
        }
      }
      if (next.empty()) break;
      layer.push_back(next);
    }
    return layer.size();
  }
};
bool isomorphic(tree S, int s, tree T, int t){
  if(S.n != T.n) return false;
  if(S.levelize(s) != T.levelize(t)) return false;

  vector<vector<int>> longcodeS(S.n + 1),
                      longcodeT(T.n + 1);
  vector<int> codeS(S.n), codeT(T.n);
```

```
  for(int h = (int) S.layer.size() - 1; h >= 0; --h){
    map<vector<int>, int> bucket;
    for (int u : S.layer[h]){
        sort(all(longcodeS[u]));
        bucket[longcodeS[u]] = 0;
    }
    for (int u : T.layer[h]){
        sort(all(longcodeT[u]));
        bucket[longcodeT[u]] = 0;
    }
    int id = 0;
    for (auto &p : bucket)
        p.second = id++;
    for (int u : S.layer[h]){
        codeS[u] = bucket[longcodeS[u]];
        longcodeS[S.prev[u]].push_back(codeS[u]);
    }
    for (int u : T.layer[h]){
        codeT[u] = bucket[longcodeT[u]];
        longcodeT[T.prev[u]].push_back(codeT[u]);
    }
  }
  return codeS[s] == codeT[t];
}
bool isomorphic(tree S, tree T){
  auto x = S.centers(), y = T.centers();
  if (x.size() != y.size())
    return false;
  if (isomorphic(S, x[0], T, y[0]))
    return true;
  return x.size() > 1 && isomorphic(S, x[1], T, y[0]);
}
```

5.5 DSU on Tree.

Tecnica Offline para resolver subtree querys y path querys.

```
bool big[MAXN];
void add( int u, int p, bool ok ){
  if(ok) //adicionar
  else //restar
  for( auto v : g[u] ){
    if( v != p && !big[v] )
      add( v , u , ok );
  }
}
void solve( int u, int p, bool keep = true ){
  int mxsz = 0;
  int bigchild = -1;
  for( auto v : g[u] ){
    if( v != p && mxsz < sz[v] ){
        mxsz = sz[v];
        bigchild = v;
    }
  }
  for( auto v : g[u] ){
    if( v != p && v != bigchild )
      solve( v , u , false );
  }
  if( bigchild != -1 ){
    solve( bigchild , u );
    big[bigchild] = true;
  }
  add( u , p , true );
  for( auto q : querys[u] ){
    //Responder las querys en el nodo u.
  }
  if( bigchild != -1 ){
    big[bigchild] = false;
  }
  if( !keep ){
    add( u , p , false );//restar
  }
}
```

5.6 HLDecomposition + Euler Tour.

```
int n;
vector<int> g[MAXN];
int cnt[MAXN], head[MAXN], parent[MAXN],
    tin[MAXN], tout[MAXN], nods[MAXN]
    depth[MAXN];
void dfs( int u, int p ){
  cnt[u] = 1;
  depth[u] = depth[p] + 1;
  for( auto &v : g[u] ){
    dfs( v , u );
    cnt[u] += cnt[v];
    if( cnt[ g[u][0] ] < cnt[v] )
      swap( g[u][0] , v );
  }
}
void tour( int u, int p, int &dfstime ){
  head[u] = (p==0 || g[p][0]!=u) ? u : head[p];
  tin[u] = ++dfstime;
  nods[dfstime] = u;
  for( auto v : g[u] ){
    tour( v , u , dfstime );
  }
  tout[u] = dfstime;
}
void HLD( int &ro ){
  for( int u = 1; u <= n; u++ ){
    if( !parent[u] )
      dfs(u,0);
  }
  int dfstime = 0;
  for( int u = 1; u <= n; u++ ){
    if( !parent[u] )
       tour(u,0,dfstime);
  }
  buildSt( ro , 1 , n , nods );
}
void subtreeQuery(int u){
  solveSt(1,1,n,tin[u],tout[u]);
}
void pathQuery( int u, int v ){
  int sol;
  while(head[u] != head[v]){
    if(depth[ head[u] ] < depth[ head[v] ]) swap(u,v);
      sol = joinSol( sol ,
               query_st(1,1,n,tin[head[u]],tin[u])
             );
      u = parent[ head[u] ];
  }
  if(depth[u] > depth[v]) swap(u,v);
  sol = joinSol(sol,query_st(1,1,n,tin[u],tin[v]));
  return sol;
}
```

5.7 Centroide Decomposition

```
//La cantidad de nodos de un subarbol
int sz[MAXN];
//Para ir marcando los centroides encontrados
bool isCentroide[MAXN];
//lev[c] = La profundidad del centroide c en el arbol
//de centroides.
int lev[MAXN];
//len[u] = La cantidad de nodos en el subarbol
//que tiene a u como centroide
int len[MAXN];
//parent[c] = El padre del centroide c en el arbol de
//centroides
int parent[MAXN];
//d[u][ lev[c] ] = La distancia desde el nodo u hasta
//su ancestro c en el arbol de centroides.
int d[MAXN][MAXLG];
//tin[u][ lev[c] ] = El tiempo en que es visitado
//el nodo u en un euler tour por el subarbol que
//tiene a c como centroide.
tin[MAXN][MAXLG];
//tout[u][ lev[c] ] = El tiempo en que es visitado
//el ultimo nodo descendiente de u en un euler tour
//por el subarbol que tiene a c como centroide.
tout[MAXN][MAXLG];
//node[c][ tin[u][ lev[c] ] ] = u
vector<int> node[MAXN];
//d2[c][ tin[u][ lev[c] ] ] = d[u][ lev[c] ]
vector<int> d2[MAXN];
```

Transform Original Tree in Binary Tree.

6 Graph (Dynamic Connectivity)

6.1 Dynamic MST

6.1.1 Solucion Offline:Divide and Conquer. $O(q \log^2 q)$.

```
struct DynamicMST{
  int n;
  DSU dsu;
  DynamicMST( int nn ){
    n = nn;
```

```
    dsu = DSU(n);
  }
  void solve( int l, int r, vector<arc> &arcs,
              arc *qs, ll cost, ll *sol ){
    int t = dsu.changes.size();
    if( l == r ){
      for(int i=0;i<arcs.size();i++){
        if( arcs[i].u == qs[l].u &&
            arcs[i].v == qs[l].v )
          arcs[i].w = qs[l].w;
      }
      sort( arcs.begin() , arcs.end() );
      for( auto e : arcs ){
        if( dsu.merge( e.u , e.v ) )
          cost += e.w;
      }
      sol[l] = cost;
    }
    else{
      sort( arcs.begin() , arcs.end() );
      map< pair<int,int> ,int> dic;
      for( int i = l; i <= r; i++ ){
        dsu.merge( qs[i].u , qs[i].v );
        dic[ {qs[i].u,qs[i].v} ]=qs[i].w;
      }
      vector<arc> used;
      for( auto e : arcs ){
        if( dsu.merge( e.u , e.v ) )
          used.push_back(e);
      }
      dsu.rollback(t);
      for( auto e : used ){
        dsu.merge( e.u , e.v );
        cost += e.w;
      }
      int t2 = dsu.changes.size();
      vector<arc> arcs2;
      for( auto e : arcs ){
        if( !dic.count( { e.u , e.v } ) ){
          if( dsu.merge( e.u , e.v ) ){
            arcs2.push_back(e);
          }
        }
        else{
          arcs2.push_back(e);
        }
      }
      dsu.rollback(t2);
      int mid = (l + r) / 2;
      solve( l , mid , arcs2 , qs , cost , sol );
      solve( mid+1 , r , arcs2 , qs , cost , sol );
      for( int i = 0; i < arcs.size(); i++ ){
        if( dic.count( {arcs[i].u , arcs[i].v} ) )
          arcs[i].w = dic[ {arcs[i].u,arcs[i].v} ];
      }
    }
    dsu.rollback( t );
  }
};
```

6.1.2 Solucion Online solo adcionando aristas: Link-Cut Tree (Fast Implementation) .

```
//2015 USP Try-outs
//I. The Kunming-Singapore Railway
struct Node {
  int id, mai, val, idm;
  Node *left, *right, *parent;
  bool evert;
  Node(){
    left = right = parent = 0;
    evert = false;
  }
  Node(int x, int v){
    left = right = parent = 0;
    evert = false;
    id = idm = x;
    mai = val = v;
  }
  bool is_root() {
    return parent == 0 || (parent->left != this &&
                           parent->right != this);
  }
  void update() {
    if (evert) {
      evert = false;
      swap(left, right);
      if (left != 0) left->evert ^= 1;
      if (right != 0) right->evert ^= 1;
    }
  }
  void refresh(){
    mai = val;
    idm = id;
    if(left && left->mai > mai)
      mai = left->mai, idm = left->idm;
    if(right && right->mai > mai)
      mai = right->mai, idm = right->idm;
  }
};
void add_edge(Node* p, Node* u, bool is_left) {
  if (u != 0) u->parent = p;
  if (is_left)
    p->left = u;
  else
    p->right = u;
}
void rotate(Node* u) {
  Node* p = u->parent;
  Node* q = p->parent;
  bool proot = p->is_root();
  bool is_left = (u == p->left);
  add_edge(p, is_left ? u->right : u->left, is_left);
  add_edge(u, p, !is_left);
  if (!proot)
    add_edge(q, u, p == q->left);
  else
    u->parent = q;
  p->refresh();
}
void splay(Node* u) {
  while (!u->is_root()) {
    Node* p = u->parent;
    Node* q = p->parent;
    if (!p->is_root()) q->update();
    p->update();
    u->update();
    if (!p->is_root()) {
      if ((p->left == u) != (q->left == p)) // zig zag
        rotate(u);
      else
        rotate(p); // zig zig
    }
    rotate(u); // zig
  }
  u->update();
  u->refresh();
}
Node* access(Node* v) {
  Node* prev = 0;
  for (Node* u = v; u != 0; u = u->parent) {
    splay(u);
    u->right = prev;
    prev = u;
  }
  splay(v);
  return prev;
}
void reroot(Node* v) {
  access(v);
  v->evert ^= 1;
}
bool connected(Node* u, Node* v) {
  if (u == v) return true;
  access(u);
  access(v);
  return u->parent != 0;
}
void link(Node* v, Node* u) {
  reroot(u);
  u->parent = v;
}
void cut(Node* u) {
  access(u);
  u->left->parent = 0;
  u->left = 0;
}
void cut(Node* u, Node* v) {
  reroot(u);
  cut(v);
}
int find(Node* u, Node* v){
      reroot(u);
      access(v);
      return v->idm;
}
const int MAXN = 100100;
Node vet[MAXN];
pair<int,int> edges[MAXN];
int main(){
```

```cpp
  int T; cin >> T;
  while(T--){
    int n, m, q;
    cin >> n >> m >> q;
    for(int i=0; i<n; i++ ) vet[i] = Node(-1,-1);
    int ans = 0;
    for( int i = n; i < n+m+q; i++ ){
      int u, v, w;
      cin >> u >> v >> w; u--; v--;
      edges[i] = { u , v };
      if(u!=v){
        if(!connected(&vet[u],&vet[v])){
          vet[i] = Node(i,w);
          link(&vet[v],&vet[i]);
          link(&vet[u],&vet[i]);
          ans += w;
        }else{
          int id = find(&vet[u],&vet[v]);
          if(vet[id].val > w){
            ans -= vet[id].val;
            int iu = edges[id].first,
                iv = edges[id].second;
            cut(&vet[iu],&vet[id]);
            cut(&vet[iv],&vet[id]);
            vet[i] = Node(i,w);
            link(&vet[v],&vet[i]);
            link(&vet[u],&vet[i]);
            ans += w;
          }
        }
      }
      if(i >= m+n) cout << ans << '\n';
    }
  }
}
```

7 Data Structures

7.1 unordered_map

```cpp
struct HASH{
  size_t operator()(const pair<int,int>&x)const{
    return hash<ll>()
              (((ll)x.first)^(((ll)x.second)<<32));
  }
};
int main(){
    unordered_map<pair<int,int>,int>mp;
    /**
    With this two lines unordered_map become
    about 10 times faster. You can replace 32768
    with another suitable power of two.
    (it depends on number of insert s you will do).
    **/
    mp.reserve(32768);
    mp.max_load_factor(0.25);
}
```

7.2 Ordered Statistics Set.

```cpp
#include <bits/stdc++.h>
using namespace std;
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<
  int,//data type of Key
  null_type,
  less<int>,//Key comparison functor
  rb_tree_tag,
  tree_order_statistics_node_update>
ordered_set;
int main(){
ordered_set X;
X.insert(16);
X.insert(1);
X.insert(8);
X.insert(2);
X.insert(4);

cout<< *X.find_by_order(0) <<'\n'; // 1
cout<< *X.find_by_order(1) <<'\n'; // 2
cout<< *X.find_by_order(2) <<'\n'; // 4
cout<< *X.find_by_order(3) <<'\n'; // 8
cout<< *X.find_by_order(4) <<'\n'; // 16

cout<< (end(X)==X.find_by_order(5))<<'\n';//true
cout<< (end(X)==X.find_by_order(500))<<'\n';//true

cout<< X.order_of_key(-5) <<'\n';  // 0
cout<< X.order_of_key(1) <<'\n';   // 0
cout<< X.order_of_key(3) <<'\n';   // 2
cout<< X.order_of_key(4) <<'\n';   // 2
cout<< X.order_of_key(400) <<'\n'; // 5

//1 2 4 8 16
for(auto it : X){
cout << it << ' ';
}cout << '\n';
}
```

7.3 BestHistory and Best Lazy + Persistent SegmentTree with updates in ranges.

Problem SPOJ GSS2. Dado un array A, responder preguntas del tipo:

$q(L,R)$ → Cual es la suma maxima de un subarray con los limites en el intervalo $[L,R]$ sumando solo elementos distintos.

Solucion.

Sea $s(a,b) = \sum_{a\le i\le b} A[i]$, sumando solo elementos distintos.

Sea $bestmx(i,j) = \max_{i\le k\le j}(\, s(i,k)\,)$.

Entonces la solucion a cada query $q(L,R)$ es $sol(L,R) = \max_{L\le i\le R}(\, bestmax(i,R)\,)$.

Una implementacion offline de la solucion es ordenar las query por el final, entonces hacer un sweepline por el array, para responder las preguntas en cada prefijo R, y con un SegmentTree con BestHistory_BestLazy se mantiene la informacion necesaria para responder, para cada nuevo R en el seewpline se actualiza el SegmentTree sumando $A[R]$ solo a las despues de la ultima ocurrencia de $A[R]$. Para hacer la implementacion online se utiliza un SegmentTree Persistente.

Implementacion Online.

```cpp
const int MAXN = 100010, MAXZ = 200;
const ll oo = (1ll<<60);
int ls[MAXN*MAXZ], rs[MAXN*MAXZ], SZ;
ll mx[MAXN*MAXZ], bestmx[MAXN*MAXZ],
   lazy[MAXN*MAXZ], bestlazy[MAXN*MAXZ];
int newnode(){
  int t = ++SZ;
  lazy[t] = bestlazy[t] = mx[t]
          = bestmx[t] = ls[t] = rs[t] = 0;
  return t;
}
void buildSt( int &t, int l, int r ){
  t = newnode();
  mx[t] = lazy[t] = bestmx[t] = bestlazy[t] = 0;

  if( l == r ) return;
  int mid = (l+r)>>1;
  buildSt( ls[t] , l , mid );
  buildSt( rs[t] , mid+1, r );
}
int copynode( int t ){
  //return t;  Para hacerlo no persitente
  int clone = newnode();
  ls[clone] = ls[t];
  rs[clone] = rs[t];
  mx[clone] = mx[t];
  bestmx[clone] = bestmx[t];
  lazy[clone] = lazy[t];
  bestlazy[clone] = bestlazy[t];
  return clone;
}
void putTag( int t, ll laz, ll bestlaz ){
  bestmx[t]=max(bestmx[t],mx[t]+bestlaz);
  mx[t] += laz;
  bestlazy[t]=max(bestlazy[t],lazy[t]+bestlaz);
  lazy[t] += laz;
}
void pushDown( int t ){
```

```
  if( lazy[t] || bestlazy[t] ){
    ls[t] = copynode( ls[t] );
    rs[t] = copynode( rs[t] );
    putTag( ls[t] , lazy[t] , bestlazy[t] );
    putTag( rs[t] , lazy[t] , bestlazy[t] );
    lazy[t] = bestlazy[t] = 0;
  }
}
int addSt(int t,int l,int r,int lq,int rq,ll upd){
  if( r < lq || rq < l ) return t;
  t = copynode(t);
  if( lq <= l && r <= rq ){
    putTag( t , upd , upd );
  }
  else{
    pushDown(t);
    int mid = (l+r)>>1;
    ls[t]=addSt(ls[t] ,l ,mid ,lq ,rq ,upd );
    rs[t]=addSt(rs[t] , mid+1 ,r ,lq ,rq ,upd );
    mx[t]=max( mx[ ls[t] ] ,mx[ rs[t] ] );
    bestmx[t]=max(bestmx[ ls[t] ],bestmx[ rs[t] ]);
  }
  return t;
}
ll getBestMX(int t,int l,int r,int lq,int rq,
             ll bestlaz = 0ll){
  if( r < lq || rq < l ) return 0;
  if( lq <= l && r <= rq ){
    return max( bestmx[t] , mx[t] + bestlaz );
  }
  int mid = (l+r)>>1;
  return max(getBestMX(ls[t],l,mid,lq,rq,
               max(bestlazy[t],lazy[t]+bestlaz)),
             getBestMX(rs[t],mid+1,r,lq,rq,
               max(bestlazy[t],lazy[t]+bestlaz))
            );
}
int main(){
  SZ = 0;
  int n; cin >> n;
  vector<int> ro( n+1 , 0 );//SegmentTree roots
  buildSt( ro[0] , 1 , n );
  map<ll,int> dic;
  for( int i = 1; i <= n; i++ ){
    ro[i] = ro[i-1];
    ll x; cin >> x;
    ro[i] = addSt(ro[i],1,n , dic[x]+1 , i , x );
    dic[x] = i;
  }
  int q; cin >> q;
  while( q-- ){
    int l, r; cin >> l >> r;
    if( l > r ) swap( l , r );
    cout << getBestMX(ro[r],1,n,l,r) << '\n';
  }
}
```

7.4 Persistent Treap.

```
const int MAX_CNTNODES = 11000000;
int sz[MAX_CNTNODES], value[MAX_CNTNODES],
    ls[MAX_CNTNODES], rs[MAX_CNTNODES];
int CNT_NODE = 0;
int newnode( int v ){
  int t = ++CNT_NODE;
  value[t] = v;
  sz[t] = 1;
  return t;
}
int copynode( int t ){
  if( !t ) return 0;
  int ct = newnode( value[t] );
  ls[ct] = ls[t];
  rs[ct] = rs[t];
  sz[ct] = sz[t];
  return ct;
}
const int elio = 2000000000;
inline int randint( int r = elio ){
  static unsigned int seed = 239017u;
  seed = seed * 1664525u + 1013904223u;
  return seed % r;
}
bool hey( int A, int B ) {
  return (ll)(randint()) * (ll)(sz[A] + sz[B]) <
         (ll)sz[A] * (ll)elio;
}
void split( int t, int key, int &L, int &R,
            bool copie = true ){
  if( !t ){ L = R = 0; return; }
  if( sz[ls[t]] + 1 <= key ){
    L = copie ? copynode(t) : t;
    split( rs[L],key-(sz[ls[t]] + 1),rs[L],R,copie );
    update(L);
  }
  else{
    R = copie ? copynode(t) : t;
    split( ls[R] , key , L , ls[R] , copie );
    update(R);
  }
}
int merge( int L, int R, bool copie = true ){
  int t = NULL;
  if( !L || !R ){
    if( copie )
      t = !L ? copynode(R) : copynode(L);
    else
      t = !L ? R : L;
    return t;
  }
  if( hey( L , R ) ){
    t = copie ? copynode(L) : L;
    rs[t] = merge( rs[t] , R , copie );
  }
  else{
    t = copie ? copynode(R) : R;
    ls[t] = merge( L , ls[t] , copie );
  }
  update(t);
  return t;
}
void build_treap( int &t, int l, int r, int *vs ){
  if( l == r ){
    t = newnode(vs[l]);
    return;
  }
  int mid = ( l + r ) >> 1;
  t = newnode(vs[mid]);
  if( mid > l )
    build_treap( ls[t] , l , mid-1 , vs );
  if( mid < r )
    build_treap( rs[t] , mid+1 , r , vs );
  update(t);
}
```

7.5 Classic Problems.

7.5.1 Cantidad de numeros mayores que $\boldsymbol{k}$ en un intervalo $[\boldsymbol{L},\boldsymbol{R}]$ en un array $\boldsymbol{A}$.

a) Online with updates, $\boldsymbol{O}(\,(\boldsymbol{n}+\boldsymbol{q})\ \boldsymbol{log^2\,n}\,)$. SegmentTree sobre las pocisiones de A y en cada $nodo(l,r)$ se guarda un SegmenTree Dinamico sobre el dominio de los numeros, que guardara por cada numero la cantidad de ocurrencias del mismo en $A[l,\dots,r]$.

b) Online with updates, $\boldsymbol{O}(\,(\boldsymbol{n}+\boldsymbol{q})\ \boldsymbol{log^2\,n}\,)$. SegmentTree sobre las posiciones de A y en cada $nodo(l,r)$ se guarda un ordered_statistic_set(de pares porque los numeros se pueden repetir) con todos los numeros en $A[l,\dots,r]$.

c) Online with updates, $\boldsymbol{O}(\,(\boldsymbol{n}+\boldsymbol{q})\ \boldsymbol{log^2\,n}\,)$. SegmentTree sobre las posiciones de A y en cada $nodo(l,r)$ se guarda un treap con todos los numeros en $A[l,\dots,r]$.

d) Online with updates, $\boldsymbol{O}(\,(\boldsymbol{n}+\boldsymbol{q})\ \boldsymbol{log^2\,n}\,)$. SegmentTree Dinamico sobre el dominio de los numeros de A, y en cada $nodo(l,r)$ se guarda un ordered_statistic_set con las posiciones i tal que $l \le A[i] \le r$.

e) Online with updates, $\boldsymbol{O}(\,(\boldsymbol{n}+\boldsymbol{q})\ \boldsymbol{log^2\,n}\,)$. SegmentTree Dinamico sobre el dominio de los numeros de A, y en cada $nodo(l,r)$ se guarda un treap con las posiciones i tal que $l \le A[i] \le r$.

f) Online with updates, $\boldsymbol{O}(\,(\boldsymbol{n}+\boldsymbol{q})\ \boldsymbol{log^2\,n}\,)$. SegmentTree Dinamico sobre el dominio de los numeros de A, y en cada $nodo(l,r)$ se guarda un SegmentTree Dinamico sobre las posiciones i tal que $l \le A[i] \le r$.

g) Online without updates, $\boldsymbol{O}(\,(\boldsymbol{n}+\boldsymbol{q})\ \boldsymbol{log^2\,n}\,)$. SegmentTree sobre las pocisiones de A, y en cada $nodo(l,r)$ se guarda un vector con los numeros del intervalo $A[l,\dots,r]$ ordenados. Para la construccion se utiliza el merge del $mergesort$.

h) **Online without updates, $O((n+q)\ log\ n)$.** SegmentTree sobre las posiciones de A, y en cada $nodo(l,r)$ se guarda un vector ord con los numeros del intervalo $A[l,\dots,r]$ ordenados, y ademas dos vectores paralelos con los lower_bound en los hijos izquierdo y derecho del nodo ($Fractional\ Cascading$). Para la construccion se utiliza el merge del $mergesort$.

i) **Online without updates, $O((n+q)\ log\ n)$.** Persistent SegmentTree sobre los prefijo de A, por cada prefijo i la version del mismo es un SegmentTree Dinamico sobre el dominio de los numeros en A, y en cada $nodo(l,r)$ se guarda la cantidad de j tal que $j \le i$ y $l \le A[j] \le r$.

j) **Offline without update, $O((n+q)\ log\ n)$.** La misma idea de 7.5.1 i) pero ordenando las querys por el final, eliminado la necesidad de la persitencia. Si lo valores son pequennos o estan comprimidos se puede utilizar un BIT.

7.5.2 K-ésimo elemento en un intervalo $[L,R]$ en un array A.

a) **Into 7.5.1, $O(log\ maxvalue \times (7.5.1))$.** Si por cada query se hace una busqueda binaria en el dominio de los numeros, entonces el problema se resume en 7.5.1.

b) Online with updates, $O((n+q)\ log^2\ n)$. Las mismas estructuras de $[7.5.1\ d), 7.5.1\ e), 7.5.1\ f)]$, estas versiones del 7.5.1 al estar el SegmentTree principal sobre el dominio de los numeros de A permite prescindir de la busqueda binaria, e ir haciendo la misma en el propio recorrido por los nodos del SegmenTree.

c) Online without updates, $O((n+q)\ log\ n)$. La misma estructura de 7.5.1 i), junto con la de 7.5.2 b), pero recorrer las dos versiones en paralelo.

7.5.3 Cantidad de numeros distintos en un intervalo $[L,R]$ en un array A.

a) **Into 7.5.1, $O(7.5.1)$.** Construir un array $next$, donde $next[i] = \min j : i < j, A[i] = A[j]$. Entonces resolver un query $[L,R]$ es similar a resolver 7.5.1 en el arreglo $next$, el intervalo $[L,R]$ y $k = R$.

b) **Online without updates, $O((n+q)\ log\ n)$.** Persistent SegmentTree sobre los prefijo de A, por cada prefijo i la version del mismo es un SegmentTree sobre las pocisiones de A, y en cada $nodo(l,r)$ se guarda la cantidad de posiciones que estan activas, para cada valor solo estara activa la posicion de la ultima ocurrencia.

c) Offline without updates, $O((n+q)\ log\ n)$. La misma idea de 7.5.3 b) pero ordenando las querys por el final, eliminado la necesidad de la persitencia. Si lo valores son pequeños o estan comprimidos se puede utilizar un BIT.

7.5.4 Subarray de suma maxima en intervalo $[L,R]$ en un array A.

a) Online with updates, $O((n+q)\ log\ n)$. SegmentTree sobre las posiciones en A y en cada $nodo(l,r)$ se guardan, sol la solucion del intervalo, mxL la maxima suma de una subsecuencia prefijo del intervalo, mxR la maxima suma de una subsecuencia sufijo del intervalo, sum la suma del intervalo.
Merge:

$$sum[nod] = sum[ls] + sum[rs]$$
$$mxL[nod] = max(mxL[ls], sum[ls] + mxL[rs])$$
$$mxR[nod] = \max(mxR[rs], sum[rs] + mxR[ls])$$
$$sol[nod] = \max(sol[ls], sol[rs], mxR[ls] + mxL[rs])$$

b) Online with updates, $O((n+q)\ log\ n)$. Trabajar sobre un arreglo S, siendo $S[i] = \sum_{k \le i} A[k]$, mantener un segmentTree sobre las posiciones en S, y en cada $nodo(l,r)$ guardar sol la solucion del intervalo, mn el valor minimo del intervalo, mx el valor maximo del intervalo.
Merge:

$$sum[nod] = sum[ls] + sum[rs]$$
$$mn[nod] = min(mn[ls], mn[rs])$$
$$mx[nod] = \max(mx[rs], mx[ls])$$
$$sol[nod] = \max(sol[ls], sol[rs], mx[rs] - mn[ls])$$

c) Idea of 7.3.

8 Strings

8.1 Z Function

Complexity: $O(n)$.

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

8.2 Lyndon Decomposition.

Una cadena S es llamada simple si solo si ella misma es estrictamente menor que todos sus cyclic-shifts. Lyndon-Descomposition consiste en descomponer la cadena en subcadenas $S_1, S_2, \dots, S_p$ que sean simples, y a su vez obviamente $S1 \ge S2 \ge, \dots, \ge Sp$.

Complexity: $O(n)$.

```
void lyndon_decomposition(string s){
  int n = (int) s.length();
  int i = 0;
  while(i<n){
    int j=i+1, k=i;
    while(j<n && s[k] <= s[j]){
      if(s[k] < s[j]) k = i;
      else ++k;
      ++j;
    }
    while(i <= k){
      cout << s.substr(i,j-k) << '-';
      i += j - k;
    }
  }
  cout << '\n';
}
```

8.2.1 Minimum Cyclic Shift.

Complexity: $O(n)$.

```
string min_cyclic_shift(string s){
  s += s;
```

```
  int n = (int) s.length();
  int i = 0, ans = 0;
  while (i < n/2){
    ans = i;
    int j = i+1, k = i;
    while(j<n && s[k] <= s[j]){
      if(s[k] < s[j]) k = i;
      else ++k;
      ++j;
    }
    while (i <= k) i += j - k;
  }
  return s.substr(ans, n / 2);
}
```

8.3 Manacher.

Complexity: $O(n)$.

```
//   a|b|b|a|b
//d1 0|0|0|1|0
//d2 0|0|2|0|0
int d1[MAXN];//impar
int d2[MAXN];//par
int manager(string s){
  //cantidad total de palindromes en la palabra
  int cant=0;
  int n=s.size();
  int l,l2,r,r2,k,i;
  l=l2=0; r=r2=-1;
  for(i=0;i<n;i++){
    /**palindromes de length impar*/
    k = (i>r ? 0 : min(d1[l+r-i],r-i)) + 1;
    while(i+k<n && i-k>=0 && s[i-k]==s[i+k]) k++;
    d1[i]=--k;
    if(i+k>r) l=i-k, r=i+k;
    /**palindromes de length par*/
    k = (i>r2 ? 0 : min(d2[l2+r2-i+1],r2-i+1)) + 1;
    while((i+k-1)<n && (i-k)>=0 && s[i+k-1]==s[i-k])
       ++k;
    d2[i]=--k;
    if( (i+k-1)>r2 ) l2=i-k, r2=i+k-1;

    cant+=(d1[i]+d2[i]);
  }
  return cant;
}
```

8.4 Burrows-Wheeler inverse transform.

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

8.5 Tandem Repeats.

Complexity: $O(n \log n)$.

```
void output_tandem (string &s, int shift, bool left,
                    int cntr, int l, int l1, int l2) {
  int pos;
  if(left)
    pos = cntr-l1;
  else
    pos = cntr-l1-l2-l1+1;
  cout << "[" << shift + pos << ".."
       << shift + pos+2*l-1 << "] = "
       << s.substr(pos, 2*l) << endl;
}
void output_tandems(string &s,int shift,bool left,
                    int cntr, int l, int k1, int k2){
  for (int l1=1; l1<=l; ++l1) {
    if (left && l1 == l)  break;
    if (l1 <= k1 && l-l1 <= k2)
      output_tandem(s, shift,left,cntr, l, l1, l-l1);
  }
}
inline int get_z(vector<int> &z, int i) {
  return 0<=i && i<(int)z.size() ? z[i] : 0;
}
void find_tandems (string s, int shift = 0) {
  int n = (int) s.length();
  if (n == 1)  return;
  int nu = n/2,  nv = n-nu;
  string u = s.substr (0, nu), v = s.substr (nu);
  string ru = string (u.rbegin(), u.rend()),
    rv = string (v.rbegin(), v.rend());

  find_tandems (u, shift);
  find_tandems (v, shift + nu);

  vector<int> z1 = z_function (ru),
       z2 = z_function (v + '#' + u),
    z3 = z_function (ru + '#' + rv),
    z4 = z_function (v);
  for (int cntr=0; cntr<n; ++cntr) {
    int l, k1, k2;
    if (cntr < nu) {
      l = nu - cntr;
      k1 = get_z (z1, nu-cntr);
      k2 = get_z (z2, nv+1+cntr);
    }
    else {
      l = cntr - nu + 1;
      k1 = get_z (z3, nu+1 + nv-1-(cntr-nu));
      k2 = get_z (z4, (cntr-nu)+1);
    }
    if (k1 + k2 >= l)
      output_tandems(s,shift,cntr<nu,cntr, l, k1,k2);
  }
}
```

8.6 Suffix Array.

```
// 0 < *min_element(v.begin(), v.end()) !!!
vector <int> get_suffix_array(vector <int> v,
                              const int alpha) {
  v.push_back(0); int n = v.size(), classes = alpha;
  vector <int> cnt(max(n, alpha));
  vector <int> p(n), np(n), c(n), nc(n);
  for (int i = 0; i < n; i++) p[i] = i, c[i] = v[i];
  for (int len = 1; len < 2 * n; len <<= 1) {
    int hlen = len >> 1;
    for (int i = 0; i < n; i++)
      np[i] = (p[i] - hlen + n) % n;
    for (int i = 0; i < classes; i++) cnt[i] = 0;
    for (int i = 0; i < n; i++) cnt[c[i]]++;
    for (int i = 1; i < classes; i++)
      cnt[i] += cnt[i - 1];
    for (int i = n - 1; i >= 0; i--)
      p[--cnt[c[np[i]]]] = np[i];
    classes = 0;
    for (int i = 0; i < n; i++) {
      if (i == 0 || c[p[i]] != c[p[i - 1]] ||
         c[(p[i] + hlen)%n] != c[(p[i - 1] + hlen)%n])
        classes++;
      nc[p[i]] = classes - 1;
    }
    for (int i = 0; i < n; i++) c[i] = nc[i];
  }
  for (int i = 0; i < n - 1; i++) p[i] = p[i + 1];
  p.pop_back(); return p;
}

vector<int> get_lcp(vector<int>& v, vector<int>& sa) {
  int n = v.size(); vector <int> rank(n), lcp(n);
  for (int i = 0; i < n; i++) rank[sa[i]] = i;
  for (int i = 0, l = 0; i < n; i++) {
    if (rank[i] == n - 1) { l = 0; continue; }
    l = max(0, l - 1); int j = sa[rank[i] + 1];
    while (i + l < n && j + l < n &&
           v[i + l] == v[j + l]) l++;
    lcp[rank[i]] = l;
  }
  return lcp;
}
```

8.7 Suffix Automata.

```
struct sa_node {
  int max_length;
  map <char, sa_node *> go; sa_node * suffix_link;
  vector<pair<char, sa_node *>> inv_suffix_link;
  int id, pos; bool ok, visited;
  sa_node() {}
};
const int N = 500 * 1000 + 10;
const int SIZE = 2 * N + 10;

sa_node all[SIZE];
```

```
sa_node * first_free;
inline sa_node * get_new(int max_length,
                         int id, int pos) {
  first_free->max_length = max_length;
  first_free->go = map <char, sa_node *> ();
  first_free->suffix_link = nullptr;
  first_free->inv_suffix_link =
      vector<pair<char, sa_node*>>();
  first_free->id = id; first_free->pos = pos;
  first_free->visited = false;
  return first_free++;
}
inline sa_node * sa_init() {
  first_free = all; return get_new(0, 0, 0);
}
inline sa_node* get_clone(sa_node* u, int max_length){
  first_free->max_length = max_length;
  first_free->go = u->go;
  first_free->suffix_link = u->suffix_link;
  first_free->inv_suffix_link =
            vector<pair<char, sa_node*>>();
  first_free->id = u->id;
  first_free->pos = u->pos;
  first_free->ok = u->ok;
  first_free->visited = false;
  return first_free++;
}
inline sa_node * add(sa_node * root, sa_node * p,
                     char c, int id, int pos){
  if (p->go.find(c) != p->go.end()) {
    sa_node * q = p->go[c];
    if (p->max_length + 1 == q->max_length) return q;
    sa_node* clone_q = get_clone(q,p->max_length + 1);
    q->suffix_link = clone_q;
    while (p != nullptr && p->go[c] == q){
        p->go[c] = clone_q, p = p->suffix_link;
    }
    return clone_q;
  }
  sa_node * l = get_new(p->max_length + 1, id, pos);
  while (p != nullptr && p->go.find(c) == p->go.end())
    p->go[c] = l, p = p->suffix_link;
  if (p == nullptr) {
    l->suffix_link = root;
  } else {
    sa_node * q = p->go[c];
    if (p->max_length + 1 == q->max_length) {
      l->suffix_link = q;
    } else {
      auto clone_q = get_clone(q, p->max_length + 1);
      l->suffix_link = q->suffix_link = clone_q;
      while (p != nullptr && p->go[c] == q){
        p->go[c] = clone_q; p = p->suffix_link;
      }
    }
  }
  return l;
}
sa_node * build(vector <string> & s) {
  int n = s.size();
  sa_node * root = sa_init();
  sa_node * last = root;
  for (int i = 0; i < n; i++) {
    reverse(s[i].begin(), s[i].end());
    for (int p = 0; p < (int)s[i].size(); p++)
      last = add(root, last, s[i][p], i, p);
    last = root;
  }
  for(sa_node * now = all; now != first_free; now++) {
    if (now->suffix_link != nullptr) {
      now->suffix_link->inv_suffix_link.push_back(
      make_pair(s[now->id][now->pos -
      now->suffix_link->max_length], now));
    }
  }
  for (sa_node * now = all; now != first_free; now++)
    sort(now->inv_suffix_link.begin(),
         now->inv_suffix_link.end());
}
```

8.8 Palindromic Tree.

```
const int MAXN = 100100;
struct node {
  int next[26];
  int len;
  int sufflink;
  node(){ fill( next , next + 26 , 0 ); }
};
string s;
node tree[MAXN];
int SZ;
int CUR;
inline int find_s( int cur, int pos ){
  while( pos - 1 - tree[cur].len < 0 ||
          s[pos-1-tree[cur].len] != s[pos] ){
    cur = tree[cur].sufflink;
  }
  return cur;
}
bool addLetter( int pos ) {
  CUR = find_s( CUR , pos );
  int ch = s[pos] - 'a';

  if( tree[CUR].next[ch] ){
    CUR = tree[CUR].next[ch];
    return false;
  }

  SZ++;
  tree[SZ].len = tree[CUR].len + 2;
  tree[CUR].next[ch] = SZ;

  if( tree[SZ].len == 1 ){
    tree[SZ].sufflink = 2;
    CUR = SZ;
    return true;
  }

  tree[SZ].sufflink =
     tree[find_s(tree[CUR].sufflink ,pos )].next[ch];
  CUR = SZ;
  return true;
}
void initTree() {
  tree[1].len = -1; tree[1].sufflink = 1;
  tree[2].len = 0; tree[2].sufflink = 1;
  SZ = 2;
  CUR = 2;
}
int main(){
  int n = s.size();
  initTree();
  int sol = 0;
  for( int i = 0; i < n; i++ ){
    if( addLetter( i ) )
      sol++;
    cout << sol << " \n"[i+1==n];
  }
}
```

9 Geometry

9.1 Point.

9.1.1 Definitions and operators.

```
typedef long long LL;
const double INF = 1e17, EPS = 1e-10, PI = acos(-1);
// be careful with EPS ////////////////////////////
// !!! implement sign for LL and double ///////////
// most of the functions work for both ////////////
inline int sign(const LL x)
{ return (x < 0) ? -1 : (x > 0); } //<-sign/////////
inline int sign(const double x)
{ return x < -EPS ? -1 : x > EPS; } //<-sign////////
inline bool is_in(double a, double b, double x) {
  if (a > b) swap(a, b);
  return (a - EPS <= x && x <= b + EPS);
} // <- x inside range [a, b] /////////////////////
struct point {
  double x, y;
  point(double _x = 0, double _y = 0)
  : x(_x), y(_y) {}
}; ///////////////////////////////////////////////
bool operator < (const point &P, const point &Q) {
  if (sign(P.y - Q.y) != 0) return P.y < Q.y;
  return (sign(P.x - Q.x) == -1);
} ///////////////////////////////////////////////
struct compare_x {
  bool operator () (const point &P, const point &Q) {
    if (sign(P.x - Q.x) != 0) return P.x < Q.x;
      return P.y < Q.y;
  }
}; ///////////////////////////////////////////////
inline void read(point &P) { cin >> P.x >> P.y; } ////
point operator + (const point &P, const point &Q)
```

```
{ return point(P.x + Q.x, P.y + Q.y); } /////////////
point operator - (const point &P, const point &Q)
{ return point(P.x - Q.x, P.y - Q.y); } /////////////
point operator * (const point &P, const double k)
{ return point(P.x * k, P.y * k); } /////////////////
point operator / (const point &P, const double k) {
  assert(sign(k) != 0);
  return point(P.x / k, P.y / k);
} ///////////////////////////////////////////////////
inline int half_plane(const point &P) {
  if (sign(P.y) != 0) return sign(P.y);
  return sign(P.x);
} ///////////////////////////////////////////////////
inline double dot(const point &P, const point &Q)
{ return P.x * Q.x + P.y * Q.y; } ///////////////////
inline double cross(const point &P, const point &Q)
{ return P.x * Q.y - P.y * Q.x; } ///////////////////
inline double norm2(const point &P)
{ return dot(P, P); } ///////////////////////////////
inline double norm(const point &P)
{ return sqrt(dot(P, P)); } /////////////////////////
inline double dist2(const point &P, const point &Q)
{ return norm2(P - Q); } ////////////////////////////
inline double dist(const point &P, const point &Q)
{ return sqrt(dot(P - Q, P - Q)); } /////////////////
point rotate_point(point P, double angle) {
  return point(P.x * cos(angle) - P.y * sin(angle),
               P.y * cos(angle) + P.x * sin(angle));
} ///////////////////////////////////////////////////
point rotate_90_ccw(point & P)
{ return point(-P.y, P.x); } ////////////////////////
point rotate_90_cw(point & P)
{ return point(P.y, -P.x); } ////////////////////////
void normalize(point &P) { // for vectors
  assert(sign(P.x) != 0 || sign(P.y) != 0);
  // LL g = __gcd(abs(P.x), abs(P.y));
  // P.x /= g; P.y /= g;
  if (P.x < 0 || (P.x == 0 && P.y < 0))
    P.x = -P.x; P.y = -P.y;
} ///////////////////////////////////////////////////
struct compare_angle {
  point O;
  compare_angle(point _O = point(0, 0))
  { O = _O; }
  bool operator () (const point &P, const point &Q){
    if (half_plane(P - O) != half_plane(Q - O))
      return half_plane(P-O) < half_plane(Q-O);
    int c = sign(cross(P - O, Q - O));
    if (c != 0) return (c > 0);
    return dist2(P, O) < dist2(Q, O);
  }
};
```

9.2 *Segment and line.*

9.2.1 Point inside segment.

```
inline bool is_in(point A, point B, point P) {
  if (sign(cross(B - A, P - A)) != 0)
    return false;
  return(is_in(A.x, B.x, P.x) && is_in(A.y, B.y,P.y));
}
```

9.2.2 Project point.

```
point project(point P, point P1, point &P2)
{ return P1+(P2-P1)*(dot(P2-P1, P-P1)/norm2(P2-P1)); }
```

9.2.3 Reflect Point.

```
point reflect(point P, point P1, point P2)
{ return project(P, P1, P2) * 2.0 - P; }
```

9.2.4 Distance from point to line.

```
double point_to_line(point P, point A, point B)
{ return abs(cross(B - A, C - A) / norm(B - A)); }
```

9.2.5 Distance from point to segment.

```
double point_to_segment(point P, point A, point B) {
  if (sign(dot(P - A, B - A)) <= 0)
    return dist(P, A);
  if (sign(dot(P - B, A - B)) <= 0)
    return dist(P, B);
  return point_to_line(P, A, B);
}
```

9.2.6 Lines intersection.

```
point intersect(point A, point B, point C, point D)
{ return A+(B-A)*(cross(C-A, C-D)/cross(B-A,C-D)); }
```

9.2.7 Segments intersection.

```
bool intersect(point A, point B, point C, point D) {
  if (sign(cross(B - A, C - A)) == 0) {
    if (sign(cross(B - A, C - A)) == 0) {
      if (B < A) swap(A, B);
      if (D < C) swap(C, D);
      if (C < A) swap(A, C), swap(B, D);
      return C < B || C == B;
    }
    return false;
  }
  if (sign(cross(C-A,B-A))*sign(cross(D-A,B-A))==1)
    return false;
  if (sign(cross(A-C,D-C))*sign(cross(B-C,D-C))==1)
    return false;
  return true;
} // segment AB intersects segment CD
```

9.2.8 Segments Distance.

```
double segment_to_segment(point A, point B,
                          point C, point D) {
  if (intersect(A, B, C, D)) return 0.0;
  return min(min(point_to_segment(A, C, D),
                 point_to_segment(B, C, D)),
             min(point_to_segment(C, A, B),
                 point_to_segment(D, A, B)));
}
```

9.3 *Polygon.*

9.3.1 Polygon area., $O(n)$

```
inline double signed_area_2(const vector <point> &G) {
  double res = 0; int n = G.size();
  for (int i = 0; i < n; i++)
    res += cross(G[i], G[(i + 1) % n]);
  return res;
}
inline double abs_area(const vector <point> &G)
{ return abs(0.5 * signed_area_2(G)); }
```

9.3.1.1 Pick Theorem.

```
// A = I + B / 2 - 1
```

9.3.2 Is Convex? $O(n)$.

```
inline bool is_convex(const vector <point> &G) {
  int n = G.size(); assert(n >= 3);
  for (int i = 0; i < n; i++) {
    int j = (i + 1) % n; int k = (i + 2) % n;
    if (sign(cross(G[j] - G[i], G[k] - G[i])) < 0)
      return false;
  }
  return true;
}
```

9.3.3 Normalize Convex, $O(n)$.

```
void normalize_convex(vector <point> &G) {
  G.erase(unique(G.begin(), G.end()), G.end());
  while (G.size() > 1 && G[0] == G.back())
    G.pop_back();
  rotate(G.begin(),
         min_element(G.begin(), G.end()), G.end());
  int ptr = 1;
  for (int i = 1; i < G.size(); i++) {
    if (is_in(G[i], G[i-1],
                     G[i+1 == G.size() ? 0 : i+1]))
      continue;
    G[ptr++] = G[i];
  } G.resize(ptr);
}
```

9.3.4 Point inside polygon, $O(n)$.

```
const int OUT = 0, ON = 1, IN = 2;
int inside_polygon(point P, const vector <point> &G) {
  int n = G.size(), cnt = 0;
  for (int i = 0; i < n; i++) {
    point A = G[i], B = G[(i + 1) == n ? 0 : i+1];
    if (is_in(P, A, B)) return ON;
```

```
    if (B.y < A.y) swap(A, B);
    if (P.y < A.y || B.y <= P.y || A.y == B.y)
      continue;
    if (sign(cross(B - A, P - A)) > 0) cnt++;
  } return ((cnt & 1) ? IN : OUT);
}
```

9.3.5 Point inside convex polygon, $O(log\ n)$.

```
// O(log(n)) !!! apply normalize_convex before !!! ///
int inside_convex(point P, const vector <point> &G) {
  int n = G.size(); assert(n >= 3);
  if (sign(cross(P - G[0], G[1] - G[0])) > 0)
    return OUT;
  if (sign(cross(G[n - 1] - G[0], P - G[0])) > 0)
    return OUT;
  if (sign(cross(P - G[0], G[1] - G[0])) == 0)
    return (is_in(P, G[0], G[1]) ? ON : OUT);
  if (sign(cross(G[n - 1] - G[0], P - G[0])) == 0)
    return (is_in(P, G[0], G[n - 1]) ? ON : OUT);
  int lo = 2, hi = n - 1, pos = hi;
  while (lo <= hi) {
    int mid = (lo + hi) >> 1;
    if (sign(cross(P - G[0], G[mid] - G[0])) >= 0)
      pos = mid, hi = mid - 1;
    else lo = mid + 1;
  }
  int s = sign(cross(G[pos] - G[pos - 1],
                     P - G[pos - 1]));
  if(s == 0) return ON; return ((s > 0) ? IN : OUT);
}
```

9.3.6 Convex Hull, $O(n\ log\ n)$.

```
vector <point> convex_hull(vector <point> pts) {
  if (pts.size() <= 2) return pts;
  sort(pts.begin(), pts.end());
  int n = pts.size(), t = 0;
  vector <point> ch(2 * n);
  for (int i = 0; i < n; i++) {
    while (t > 1 &&
           sign(cross(ch[t-1]-ch[t-2],
                      pts[i]-ch[t-2])) <= 0)
      t--;
    ch[t++] = pts[i];
  }
  int pt = t;
  for (int i = n - 2; i >= 0; i--) {
    while (t > pt &&
           sign(cross(ch[t-1]-ch[t-2],
                      pts[i]-ch[t-2])) <= 0)
      t--;
    ch[t++] = pts[i];
  }
  if (ch[0] == ch[t - 1]) t--;
  ch.resize(top);
  return ch;
}
```

9.3.7 Segment intersects convex polygon, $O(log\ n)$.

```
inline int get_upper_point(const vector <point> &G) {
  int n = G.size(), upper = 0;
  while (upper + 1 < n && G[upper] < G[upper + 1])
    upper++;
  return upper;
}
/// O(log(n)) !!! apply normalize_convex and ////////
/// get_upper_point before !!! // test //////////////
bool convex_to_segment_intersection(vector<point>& G,
                        int upper, point A, point B) {
  if (sign(cross(G[0] - A, B - A)) *
      sign(cross(G[upper] - A, B - A)) <= 0)
    return true;
  int n = G.size(); if (B < A) swap(A, B);
  if (cross(B - A, G[0] - A) > 0) {
    int lo = 1, hi = upper, id = 0;
    while (lo <= hi) {
      int mid = (lo + hi) >> 1;
      if (cross(G[mid] - A, B - A) >=
          cross(G[mid - 1] - A, B - A))
        id = mid, lo = mid + 1;
      else hi = mid - 1;
    }
    return (cross(G[id] - A, B - A) >= 0);
  }
  int lo = upper, hi = ((int)G.size()) - 1, id = 0;
  while (lo <= hi) {
    int mid = (lo + hi) >> 1;
    if (cross(B - A, G[mid]) >=
        cross(B - A, G[(mid + 1) % n]))
      id = mid, hi = mid - 1;
    else lo = mid + 1;
  }
  return (cross(B - A, G[id] - A) >= 0);
}
```

9.3.8 Minkosky sum, $O(n + m)$.

```
vector <point> minkowsky_sum(vector <point> a,
                             vector <point> b) {
  int na = a.size(), nb = b.size();
  if (na == 0 || nb == 0) return {};
  normalize_convex(a); normalize_convex(b);
  vector <point> s; s.push_back(a[0] + b[0]);
  int pa = 0, pb = 0;
  while (pa != na && pb != nb) {
    point va = a[(pa + 1) % na] - a[pa];
    point vb = b[(pb + 1) % nb] - b[pb];
    if (sign(cross(va, vb)) >= 0) {
      point p = s.back() + va;
      s.push_back(p); pa++;
    } else {
      point p = s.back() + vb;
      s.push_back(p); pb++;
    }
  }
  while (pa != na) {
    point va = a[(pa + 1) % na] - a[pa];
    point p = s.back() + va;
    s.push_back(p); pa++;
  }
  while (pb != nb) {
    point vb = b[(pb + 1) % nb] - b[pb];
    point p = s.back() + vb;
    s.push_back(p); pb++;
  }
  assert(s.back() == s[0]);
  normalize_convex(s);
  return s;
}
```

9.3.9 Rotating calipers [further pair of points], $O(n)$.

```
// O(n) Returns the d ^ 2
// !!! apply normalize(G) before use
// for width:
// same idea but using distance from point to line
LL convex_diameter_2(vector <point> & G) {
  int n = G.size(), p0 = 0, p1 = 0;
  for (int i = 1; i < n; i++) {
    // < compare y first then x
    if (G[i] < G[p0]) p0 = i;
    if (G[p1] < G[i]) p1 = i;
  }
  LL res = dist2(G[p0], G[p1]);
  int c0 = p0, c1 = p1; do {
    point v1 = G[p0+1 == n ? 0 : p0 + 1] - G[p0];
    point v2 = G[p1] - G[p1+1 == n ? 0 : p1 + 1];
    int s = sign(cross(v1, v2));
    if (s == 1) {
      p0 = p0 + 1 == n ? 0 : p0 + 1;
    } else if (s == -1) {
      p1 = p1 + 1 == n ? 0 : p1 + 1;
    } else {
      p0 = p0 + 1 == n ? 0 : p0 + 1;
      p1 = p1 + 1 == n ? 0 : p1 + 1;
    }
    res = max(res, dist2(G[p0], G[p1]));
  } while (c0 != p0 || c1 != p1);
  return res;
}
```

9.3.10 Centroid, $O(n)$.

```
point centroid(const vector <point> &g) {
  int n = g.size();
  point C(0, 0); double area = 0.0;
  for (int i = 1; i < n - 1; i++) {
    int j = (i + 1 == n) ? 0 : i + 1;
    double a = cross(g[i] - g[0], g[j] - g[0]);
    area += a;
    C = C + (g[0] + g[i] + g[j]) * a;
  }
  C = C / (3.0 * area);
  return C;
}
```

9.4 Circle.

9.4.1 Count common tangents.

```
int count_common_tangents(point C1, LL r1,
                          point C2, LL r2) {
  LL d2 = dist2(C1, C2);
  if (d2 > (r1 + r2) * (r1 + r2)) return 4;
  if (d2 == (r1 + r2) * (r1 + r2)) return 3;
  if (d2 < (r1 - r2) * (r1 - r2)) return 0;
  if (d2 == (r1 - r2) * (r1 - r2)) return 1;
  return 2;
}
```

9.4.2 Circumcenter.

```
point circle_center(point A, point B, point C) {
  assert(abs(cross(B - A, C - A)) > EPS);//no colinear
  return intersect((A+B)/2.0,
                   (A+B)/2.0 + rotate_90_ccw(B - A),
                   (B+C)/2.0,
                   (B+C)/2.0 + rotate_90_ccw(C - B));
}
```

9.4.3 Point circle tangents.

```
pair<point,point>point_circle_tangent(point P,
                                      point C,
                                      double r) {
  double d = dist(P, C), a = asin(r / d);
  double l = sqrt(d * d - r * r);
  point A = P + rotate_point((C - P) * (l / d), a);
  point B = P + rotate_point((C - P) * (l / d), -a);
  return {A, B};
}
```

9.4.4 Circle circle tangents.

```
vector<pair<point, point>> common_tangents(point C1,
                                           double r1,
                                           point C2,
                                           double r2){
  double d = dist(C1, C2);
  assert(!(d <= EPS && abs(r1 - r2) <= EPS));
  if (r2 > r1) swap(C1, C2), swap(r1, r2);
  if (r1 > d + r2 + EPS)
    return {};
  if (abs(r1 - d - r1) <= EPS)
    return {{C1 + (C2 - C1) * (r1 / d),
             C1 + (C2 - C1) * (r1 / d)}};
  vector <pair <point, point> > answer;
  {
    auto t = point_circle_tangent(C2, C1, r1 - r2);
    auto V_first = rotate_point((t.first - C2) *
                  (r2 / dist(t.first, C2)), 0.5 * PI);
    point V_second = rotate_point((t.second - C2) *
                  (r2 / dist(t.second, C2)), -0.5 * PI);
    answer.push_back(make_pair(C2 + V_first,
                               t.first + V_first));
    answer.push_back(make_pair(C2 + V_second,
                               t.second + V_second));
  }
  if (abs(d - r1 - r2) <= EPS) {
    answer.push_back(make_pair(C1 + (C2 - C1) *
            (r1 / d), C1 + (C2 - C1) * (r1 / d)));
  } else if (d > r1 + r2 + EPS) {
    auto t = point_circle_tangent(C2, C1, r1 + r2);
    point V_first = rotate_point((t.first - C2) *
            (r2 / dist(t.first, C2)), -0.5 * PI);
    point V_second = rotate_point((t.second - C2) *
            (r2 / dist(t.second, C2)), 0.5 * PI);
    answer.push_back(make_pair(C2 + V_first,
                               t.first + V_first));
    answer.push_back(make_pair(C2 + V_second,
                               t.second + V_second));
  }//TODO avoid rotate(P, 0.5 PI), use rotate_90_ccw
  return answer;
}
```

9.4.5 Line-circle intersection.

```
vector<point> line_circle_intersect(point A,point B,
                                    point C,double r){
  point PC = project(C, A, B);
  double d = dist(C, PC);
  if (d > r + EPS) return {};
  if (sign(d - r) == 0) return {PC};
  double l = sqrt(r * r - d * d);
  point v = (B - A) * (l / dist(A, B));
  return {PC + v, PC - v};
}
```

9.4.6 Circle circle intersection.

```
vector <point> circle_circle_intersect(point C1,
                                       double r1,
                                       point C2,
                                       double r2) {
  if (r2 > r1) swap(r2, r1), swap(C2, C1);
  double d = dist(C1, C2);
  assert(!(d <= EPS && abs(r1 - r2) <= EPS));
  if (d > r1 + r2 + EPS || r1 > d + r2 + EPS)return{};
  if (abs(d - (r1 + r2)) <= EPS ||
      abs(r1 - (d + r2)) <= EPS)
      return {C1 + (C2 - C1) * (r1 / d)};
  double a = (r1 * r1 - r2 * r2 + d * d) / (2.0 * d);
  double b = sqrt(r1 * r1 - a * a);
  point P = C1 + (C2 - C1) * (a / d);
  point V = rotate_point(C2 - C1, 0.5 * PI) * (b / d);
  return {P + V, P - V};
}
```

9.5 Closest pair of points.

```
double closest_pair_of_points(vector <point> pts) {
  sort(pts.begin(), pts.end(), compare_x());
  multiset <point> S;
  int n = pts.size(); double res = INF;
  for (int i = 0, last = 0; i < n; i++) {
    while (last<i && pts[i].x-pts[last].x >= res+EPS)
      S.erase(S.find(pts[last++]));
    auto lo = S.lower_bound(
              point(-INF, pts[i].y - res - EPS));
    auto hi = S.upper_bound(
              point(INF, pts[i].y + res + EPS));
    while (lo != hi)
      res = min(res, dist(pts[i], *lo)), lo++;
    S.insert(pts[i]);
  } return res;
}
```

9.6 Half planes intersection.

9.6.1 Convex cut, 1 cut $O(n)$.

```
// cut G, left side of PQ (Q - P).
vector <point> convex_cut(const vector <point> &G,
                          point P, point Q) {
  int n = G.size(); vector <point> res;
  for (int i = 0; i < n; i++) {
    int si = sign(cross(Q - P, G[i] - P));
    if (si >= 0) res.push_back(G[i]);
    int j = (i + 1 == n) ? 0 : i + 1;
    int sj = sign(cross(Q - P, G[j] - P));
    if (si * sj == -1)
      res.push_back(intersect(P, Q, G[i],G[j]));
  }
  return res;
}
```

9.6.2 Half planes intersection, $O(n \log n)$.

```
struct line { // a * x + b * y + c = 0
  double a, b, c, angle;
  line() {}
  line(double _a, double _b, double _c)
  : a(_a), b(_b), c(_c) { set_angle(); /* !!! */}
  line(point P, point Q) {
    double dx = Q.x - P.x,  dy = Q.y - P.y;
    double len = sqrt(dx * dx + dy * dy);
    dx /= len; dy /= len; a = -dy; b = dx;
    // c = -cross(point(Q - P, P));
    c = -(a * P.x + b * P.y);
    set_angle(); /// !!!
  }
  inline int side(const point &P) {
    return sign(a * P.x + b * P.y + c);
  }
  inline void set_angle() {
    angle = atan2(-a, b);
  }
}; // half plane ///////////////////////////////////
inline point intersect(const line &a, const line &b){
  double det = a.a * b.b - a.b * b.a;
  // assert(abs(det) > EPS); // !!!
```

```
  double det_x = (-a.c) * b.b - a.b * (-b.c);
  double det_y = a.a * (-b.c) - (-a.c) * b.a;
  return point(det_x / det, det_y / det);
} ///////////////////////////////////////////////////
vector<point>half_planes_intersection(vector<line>all)
{
  const double INF = 1e9; /// segun el problema
all.push_back(line(point(-INF,-INF),point(INF,-INF)));
all.push_back(line(point(INF, -INF),point(INF,INF)));
all.push_back(line(point(INF, INF),point(-INF,INF)));
all.push_back(line(point(-INF,INF),point(-INF,-INF)));
    int n = all.size();
  sort(all.begin(), all.end(),
  [&](const line &a, const line &b) {
    if (sign(a.angle - b.angle) != 0)
      return a.angle < b.angle;
    return a.c > b.c;
  });
  int ptr = 1;
  for (int i = 1; i < n; i++) {
    if (sign(all[i].angle - all[ptr - 1].angle) == 0)
      continue;
    // TODO cambiar eps para comparar angulos
    all[ptr++] = all[i];
  }
  if (ptr > 1 &&
    sign(all[0].angle -
             all[ptr-1].angle - 2.0 *PI)==0){
    if (all[ptr - 1].c < all[0].c)
      swap(all[ptr - 1], all[0]);
    ptr--;
  }
  all.resize(ptr); n = all.size();
  vector <line> Q(n);
  int head = 0, tail = 0;
  for (int i = 0; i < n; i++) {
    while (head + 1 < tail &&
           all[i].side(
             intersect(Q[tail - 2], Q[tail - 1]))!=1)
      tail--;
    while (head + 1 < tail &&
           all[i].side(
             intersect(Q[head], Q[head + 1])) != 1)
      head++;
    Q[tail++] = all[i];
  }
  while (head + 1 < tail &&
         Q[head].side(
           intersect(Q[tail - 1], Q[tail - 2])) != 1)
    tail--;
  while (head + 1 < tail &&
         Q[tail - 1].side(
           intersect(Q[head], Q[head + 1])) != 1)
    head++; /// not sure
  vector <point> hull(tail - head);
  for (int i = head; i < tail; i++) {
    int j = (i + 1 == tail) ? head : i + 1;
    hull[i - head] = intersect(Q[i], Q[j]);
  }
  return hull;
}
```

10 Dynamic

10.1 Convex Hull Trick

```
// DP[i] = min(d[j] + b[j] * a[i]) : (j < i)
// sufficient condition: b[j] >= b[j + 1]
// (b[j] <= b[j + 1] para minimizar)
const int N = 100005;
struct line {
  long long m, n;
  long long y(long long x) {
    return m * x + n;
  }
};
inline long double inter(line a, line b) {
  return (long double)(b.n - a.n) /
         (long double)(a.m - b.m);
}
struct ConvexHullTrick {
  line ch[N]; int size;
  void clear() { size = 0; }
  void add(line l) {
    while (size > 1 &&
      inter(l, ch[size - 1]) < inter(l, ch[size - 2]))
      size--;
    ch[size++] = l;
  }
  long long get_min(long long x) {
    int id = 0, lo = 1, hi = size - 1;
    while (lo <= hi) {
      int mid = (lo + hi) >> 1;
      if (ch[mid].y(x) < ch[mid - 1].y(x)) {
        id = mid; lo = mid + 1;
      } else {
        hi = mid - 1;
      }
    }
    return ch[id].y(x);
  }
} CH;
int n;
long long a[N], b[N];
long long dp[N];
int main() {
  cin >> n;
  for (int i = 0; i < n; i++)
    cin >> a[i];
  for (int i = 0; i < n; i++)
    cin >> b[i];
  CH.clear(); CH.add((line) {b[0], 0});
  for (int i = 1; i < n; i++) {
    dp[i] = CH.get_min(a[i]);
    CH.add((line) {b[i], dp[i]});
  }
  cout << dp[n - 1] << "\n";
}
```

10.2 Convex Hull Trick - General Case

```
bool query_flag = false;
struct line {
  long long m, c;
  mutable function<const line*()> succ;
  bool operator<(const line& o) const {
    if (!query_flag) return m < o.m;
    // cambiar a > para minimizar
    const line* s = succ();
    if (!s) return false;
    return (c - s->c) < (s->m - m) * o.m;
    // cambiar a > para minimizar
  }
};
struct maximum_hull : multiset<line> {
  bool bad(iterator y) {
    auto x = (y==begin())?end():prev(y),z=next(y);
    if (x == end() && z == end()) return false;
    else if (x == end())
      return y->m == z->m && y->c <= z->c;
    else if (z == end())
      return y->m == x->m && y->c <= x->c;
    else return (x->c - y->c) * (z->m - y->m) >=
                (y->c - z->c) * (y->m - x->m);
  }
  void insert_line(long long m, long long c) {
    auto y = insert({ m, c, nullptr });
    y->succ = [=] {
      return next(y) == end() ? nullptr : &*next(y);
    };
    if (bad(y)) { erase(y); return; }
    iterator z;
    while ((z = next(y)) != end() && bad(z)) erase(z);
    while (y != begin() && bad(z = prev(y))) erase(z);
  }
  long long eval(long long x) {
    if (empty()) return numeric_limits<ll>::min();
    query_flag = true;
    auto l = *lower_bound({ x, 0, nullptr });
    query_flag = false;
    return l.m * x + l.c;
  }
};
```

10.3 Divide & Conquer Optimization

```
// dp[i][j] = min(dp[i - 1][k] + C[k][j]) : (k < j)
// sufficient condition: pos[i][j] <= pos[i][j + 1]
const int N = 1005, K = 205, INF = 1e9;
int n, k, u[N][N], sum[N][N];
int dp[K][N], pos[K][N];

inline int cost(int l, int r) {
  return (sum[r][r] - sum[r][l - 1] -
          sum[l - 1][r] + sum[l - 1][l - 1]) >> 1;
}
void solve(int g, int l, int r, int pl, int pr) {
```

```
  if (l > r) return; int mid = (l + r) >> 1;
  dp[g][mid] = INF; pos[g][mid] = -1;
  for (int i = pl; i < mid && i <= pr; i++)
    if(dp[g][mid] > dp[g - 1][i] + cost(i + 1, mid)) {
      dp[g][mid] = dp[g - 1][i] + cost(i + 1, mid);
      pos[g][mid] = i;
    }
  solve(g, l, mid - 1, pl, pos[g][mid]);
  solve(g, mid + 1, r, pos[g][mid], pr);
}
int main() {
  cin >> n >> k;
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      cin >> u[i][j];
  for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
      sum[i][j] = u[i][j] + sum[i - 1][j]
      + sum[i][j - 1] - sum[i - 1][j - 1];
  for (int i = 1; i <= n; i++)
    dp[1][i] = cost(1, i), pos[1][i] = 0;
  int ans = dp[1][n];
  for (int i = 2; i <= k; i++)
    solve(i, i, n, i - 1, n),
    ans = min(ans, dp[i][n]);
  cout << ans << "\n";
}
```

10.4 Knuth Optimization

```
// f[i][j] = min(f[i][k] + f[k][j] + c[i][j]):
// [i][j] -> interval [i, j)
// sufficient condition:
// p[i][j - 1] <= p[i][j] <= p[i + 1][j]
// Example Optimal Binary Search Tree
auto c = [&](int l, int r, int m) {
  assert(l <= m && m < r);
  return sv[r] - sv[l] - v[m];
}; // sv[i] = v[0] + v[1] + ... + v[i - 1]
for (int i = 0; i < n; i++) {
  f[i][i + 1] = 0;
  p[i][i + 1] = i;
}
for (int s = 2; s <= n; s++) {
  for (int l = 0; l + s <= n; l++) {
    int r = l + s;
    f[l][r] = -1;
    for (int m = p[l][r - 1]; m <= p[l + 1][r]; m++) {
      int x = f[l][m] + f[m + 1][r] + c(l, r, m);
      if (f[l][r] == -1 || x <= f[l][r]) {
        f[l][r] = x;
        p[l][r] = m;
      }
    }
  }
}
```

10.5 Longest Common Increasing Subsecuence.

Complexity $O(n^2)$.

```
const int MAXN = 1010;
void LCIS(int n,int *a,int m,int *b,int &sz,int *sol){
  vector< int > lcis( m , 0 );
  vector< int > back( m , -1 );
  for( int i = 0; i < n; i++ ){
    int cur = 0;
    int last = -1;
    for( int j = 0; j < m; j++ ){
      if( a[i] == b[j] && cur + 1 > lcis[j] ){
        lcis[j] = cur + 1;
        back[j] = last;
      }
      else if( b[j] <= a[i] && cur < lcis[j] ){
        cur = lcis[j];
        last = j;
      }
    }
  }
  sz = 0;
  int ind = -1;
  for( int j = 0; j < m; j++ )
    if( sz < lcis[j] ){
      sz = lcis[j];
      ind = j;
    }
  for( int i = sz-1; ind != -1; i--, ind = back[ind] )
    sol[i] = b[ind];
}
```

11 Number Theory

11.1 Utiles (add, mul, power).

```
typedef long long LL;
inline void add(LL & a, LL b, LL M)
{ a += b; if (a >= M) a -= M; }

inline LL mul(LL a, LL b, LL m) {
  // return (__int128)a * b % m; // !!!!
  // a %= m; b %= m;
  if (m <= 2e9) return a * b % m;
  LL q = (long double)a * b / m;
  LL r = a * b - q * m; r %= m;
  if (r < 0) r += m;
  return r;
} // to avoid overflow, m < 1e18
inline LL power(LL x, LL n, LL m) {
  if (x == 0 || m == 1) return 0;
  LL y = 1 % m; x %= m;
  while (n > 0) {
    if (n & 1) y = mul(y, x, m);
    x = mul(x, x, m); n >>= 1;
  }
  return y;
}
```

11.2 Extended GCD.

11.2.1 Extended GCD. $[ax + by = gcd(a, b)]$.

```
LL egcd(LL a, LL b, LL & x, LL & y) {
  if (a == 0) {x = 0; y = 1; return b;}
  LL g = egcd(b % a, a, y, x);
  x -= (b / a) * y; return g;
}
```

11.2.2 Chinese Remainder Theorem.

```
pair <LL, LL> crt(LL r1, LL m1, LL r2, LL m2) {
  LL d = __gcd(m1, m2);
  if (r1 % d != r2 % d) return {-1, -1};
  LL rd = r1 % d;
  r1 /= d; m1 /= d; r2 /= d; m2 /= d;
  if (m1 < m2) {swap(r1, r2); swap(m1, m2);}
  LL k = (r2 - r1) % m2; if (k < 0) k += m2;
  LL x, y; egcd(m1, m2, x, y);
  x %= m2; if (x < 0) x += m2;
  k *= x; k %= m2;
  return {m1 * m2 * d, (k * m1 + r1) * d + rd};
}
```

11.2.3 Modular Inverse.

```
LL inverse(LL n, LL m) {
  LL x, y; LL g = egcd(n, m, x, y);
  if (g != 1) return -1;
  x %= m; if (x < 0) x += m;
  assert((n * x % m) == 1); return x;
}
```

11.3 Fast Sieve, $O(n)$.

```
bool ready = false;
const int P = 1000 * 1000;
bool _isPrime[P]; int minPrime[P];
vector <int> primes;

inline bool fastSieve() { // O(n)
  for (int i = 0; i < P; i++)
    _isPrime[i] = true;
  _isPrime[0] = _isPrime[1] = false;
  primes.reserve(P); //
  for (int i = 2; i < P; i++) {
    if (_isPrime[i])
      primes.push_back(i), minPrime[i] = i;
    for (auto p : primes) {
      if (p * i >= P) break;
      _isPrime[p * i] = false; minPrime[p * i] = p;
      if (i % p == 0) break;
```

```
    }
  }
}
```

11.4 Primality.

11.4.1 Miller Rabin primality test.

```
inline bool witness(LL x, LL n, LL s, LL d) {
  LL cur = power(x, d, n);
  if (cur == 1) return false;
  for (int r = 0; r < s; r++) {
    if (cur == n - 1) return false;
    cur = mul(cur, cur, n);
  }
  return true;
}
bool isPrime(long long n) {
  if (!ready) fastSieve(), ready = true; // !!!
  if (n < P) return _isPrime[n];
  if (n < 2) return false;
  for (int x : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29}) {
    if (n == x) return true;
    if (n % x == 0) return false;
  }
  if (n < 31 * 31) return true;
  int s = 0; LL d = n - 1;
  while (!(d & 1)) s++, d >>= 1;
  // for n int: test = {2, 7, 61}
  // for n<3e18:test={2, 3, 5, 7, 11, 13, 17, 19, 23}
  static vector <LL> test = {2, 325, 9375, 28178,
  450775, 9780504, 1795265022};
  for (long long x : test) {
    if (x % n == 0) return true;
    if (witness(x, n, s, d)) return false;
  }
  return true;
} // ends Miller Rabin primality test !!!
```

11.4.2 Next Prime.

```
LL nextPrime(LL x) {
  for (x = max(2LL, x + 1); x % 6; x++)
    if (isPrime(x)) return x;
  for (; true; x += 6) {
    if (isPrime(x + 1)) return x + 1;
    if (isPrime(x + 5)) return x + 5;
  }
}
```

11.4.3 Count Primes, $O(n^{3/4})$.

```
// O(n^0.75) same idea for other functions over primes
LL countPrimes(LL n) {
  int r = sqrt((double)n);
  while ((LL)(r + 1) * (r + 1) <= n) r++;
  vector <LL> values; // all [n / x]
  for (LL x = 1; x <= n; x = n / (n / x) + 1)
    values.push_back(n / x);
  function <int(LL)> pos = [&](LL x) {
    if (x > r) return (int)(n / x) - 1;
    return (int)(values.size() - x);
  };
  // auto primes = getPrimes(r);
  if (!ready) fastSieve(), ready = true;
  vector <LL> cnt(values.size());
  for (auto v : values)
    cnt[pos(v)] = v - 1;
  for (auto p : primes)
    for (auto v : values) {
      if ((LL)p * p > v) break;
      cnt[pos(v)] -= cnt[pos(v / p)];
      cnt[pos(v)] += cnt[pos(p - 1)];
    }
  return cnt[pos(n)];
} // ends prime counting
```

11.5 Factorization.

```
void rho(LL n, LL c, vector <LL> & fp) {
  if (n == 1) return;
  if (n < P) { // use sieve
    if (!ready)
      fastSieve(), ready = true; // !!!
    while (n > 1) {
      int p = minPrime[n];
      while (n > 1 && minPrime[n] == p)
        fp.push_back(p), n /= p;
    }
    return;
  }
  if (isPrime(n)) {
    fp.push_back(n);
    return;
  }
  if (!(n & 1)) {
    fp.push_back(2);
    rho(n / 2, c, fp);
    return;
  }
  LL x = 2, s = 2, p = 1, l = 1;
  function <LL(LL)> f = [&c, &n] (LL x) {
    return (LL)(((__int128)x * x + c) % n);
  };
  while (true) {
    x = f(x);
    LL g = __gcd(abs(x - s), n);
    if (g != 1) {
      rho(g, c + 1, fp);
      rho(n / g, c + 1, fp);
      return;
    }
    if (p == l) s = x, p <<= 1, l = 0;
    l++;
  }
}
vector <pair <LL, int>> factorize(LL n) {
  vector <LL> p; rho(n, 1, p);
  sort(p.begin(), p.end());
  vector <pair <LL, int>> f;
  for (int i = 0, j = 0; i < p.size(); i = j) {
    while (j < p.size() && p[i] == p[j]) j++;
    f.emplace_back(p[i], j - i);
  }
  return f;
} // ends pollar rho factorization
```

11.6 Euler Phi.

```
LL phi(LL n) { // euler phi
  auto f = factorize(n);
  LL r = n; for (auto p : f)
    r -= r / p.first;
  return r;
}
```

11.7 Primitive root.

```
LL primitiveRoot(LL n) {
  if (n <= 0) return -1;
  if (n == 1 || n == 2 || n == 4)
    return n - 1;
  auto f = factorize(n);
  if (f[0].first == 2 &&
     (f[0].second != 1 || f.size() != 2))
    return -1;
  if (f[0].first != 2 && f.size() != 1)
    return -1;
  int phin = phi(n); f = factorize(phin);
  for (int g = 2; g < n; g++) {
    if (power(g, phin, n) != 1) continue;
    bool ok = true;
    for (auto & p : f)
      if (power(g, phin / p.first, n) == 1) {
        ok = false; break;
      }
    if (ok) return g;
  }
  assert(false); return -1;
}
```

11.8 Discrete logarithm. $a^x = b \pmod c$.

```
// a ^ x = b (mod c)
LL discreteLogarithm(LL a, LL b, LL m) {
  if (b == 1) return 0;
  unordered_map <LL, LL> M;
  int c = (int)sqrt((double)m) + 1;
  LL v = power(a, c, m), pv = 1, w = b;
  for (int i = 1; i <= c; i++)
    pv = mul(pv, v, m), M[pv] = i;
  LL ans = -1;
  for (int i = 0; i < c; i++) {
    if (M.find(w) != M.end()) {
      int lg = M[w] * c - i;
```

```
      if (ans == -1 || lg < ans)
        ans = lg;
    }
    w = mul(w, a, m);
  }
  return ans;
} // TODO swap steps, calc min answer faster
```

11.9 Congrunce Equations, $ax = b \pmod m$.

```
// ax = b (mod m)
vector<LL> congruence_equation(LL a, LL b, LL m) {
  vector<LL> ret; LL g = __gcd(a, m), x;
  if (b % g != 0) return ret;
  a /= g; b /= g; x = inverse(a, m / g);
  for (int k = 0; k < g; ++k)
    ret.push_back((x * b + m / g * k) % m);
  return ret;
}
```

11.10 Discrete Root, $x^n = a \pmod m$.

```
// Same idea used in discrete logarithm using
// primitive root (g ^ n) ^ x') = g ^ (a')
// + congruence equation n * x' = a' (modulo phi(m))
// gcd(n, phi(m)) solutions
```

11.11 Discrete sqrt.

```
LL legendre(LL n, LL p) {
  return power(n, (p - 1LL) / 2LL, p);
}
// Tonelli Shanks
LL discreteSqrt(LL a, LL p) { // -1 if no solution
  if (a == 0) return 0; if (p == 2) return a;
  if (legendre(a, p) != 1) return -1;
  LL d = p - 1; int s = 0;
  while (!(d & 1)) s++, d >>= 1LL;
  LL q = 2;
  while (legendre(q, p) != p - 1) q++;
  LL t = power(a, (d + 1) / 2, p);
  LL r = power(a, d, p);
  while (r != 1) {
    int i = 0; LL v = 1;
    while (power(r, v, p) != 1) i++, v *= 2LL;
    LL e = power(2, s - i - 1, p);
    LL u = power(q, d * e, p);
    t = mul(t, u, p);
    r = mul(r, mul(u, u, p), p);
  }
  return t;
} // if y is a solution; then -y is too
```

11.12 Farey.

```
// todas las fracciones reducidas tal que
// el denominador es menor o igual que n
// a / b y c / d son consecutivas si y solo si:
// b * c - a * d = 1; TODO test
vector <pair <LL, LL> > farey(int n) {
  LL a = 0, b = 1, c = 1, d = n;
  vector <pair <LL, LL> > s;
  s.push_back({0, 1});
  while (c < n) {
    LL k = (n + b) / d;
    LL e = k * c - a;
    LL f = k * d - b;
    a = c; b = d;
    c = e; d = f;
    s.push_back({a, b});
  }
  return s;
}
```

11.13 Fibonacci sequence $O(\log n)$.

```
void _fib(LL n, LL m, LL & x, LL & y) {
  if (n == 1) {
    x = 1; y = 1;
  } else if (n & 1) {
    _fib(n - 1, m, y, x);
    y += x; if (y >= m) y -= m;
  } else {
    LL a, b; _fib(n >> 1, m, a, b);
    y = (mul(a, a, m) + mul(b, b, m)) % m;
    x = (mul(a, b, m) + mul(a, b - a + m, m)) % m;
  }
}
LL fib(LL n, LL m) { // O(log(n))
  assert(n > 0);
  LL x, y; _fib(n, m, x, y);
  return x;
} // ends fibonacci
```

11.14 Partitions.

```
vector <LL> partitions(int n, LL m) { // O(n ^ (3/2))
  vector <LL> p(n + 1); p[0] = 1;
  for (int i = 1; i <= n; i++) {
    for (int j = 1; i >= (3 * j * j - j) / 2; j++) {
      LL u = p[i - (3 * j * j - j) / 2];
      if (!(j & 1)) u = m - u;
      add(p[i], u, m);
      if (i >= (3 * j * j + j) / 2) {
        LL v = p[i - (3 * j * j + j) / 2];
        if (!(j & 1)) v = m - v;
        add(p[i], v, m);
      }
    }
  }
  return p;
} // end partitions implementation
```

11.15 Linear recurrence.

```
// O(n * n * log(n))
template <const int M, const int B = 63> // B bits
struct linearRec {
  int n;
  vector <int> f; // f={f(1), f(2), ..., f(n)}
  // f(k)=t[0]*f(k-1)+t[1]*f(k-2)+...+t[n-1]*f(k-n)
  vector <int> t;
  vector <vector <int> > fn; // for bin power
  vector<int>add(vector<int>& a, vector<int>& b) {
    vector <int> r(2 * n + 1);
    for (int i = 0; i <= n; i++)
      for (int j = 0; j <= n; j++)
        add(r[i + j], mul(a[i], b[j], M), M);
    for (int i = 2 * n; i > n; i--) {
      for (int j = 0; j < n; j++)
        add(r[i - 1 - j], mul(r[i], t[j], M), M);
      r[i] = 0;
    }
    r.erase(r.begin() + n + 1, r.end()); return r;
  }
  linearRec(vector <int> & _f, vector <int> & _t)
  : f(_f), t(_t) {
    n = f.size(); vector <int> a(n + 1);
    a[1] = 1; fn.push_back(a);
    for (int i = 1; i < B; i++)
      fn.push_back(add(fn[i - 1], fn[i - 1]));
  }
  int calc(long long k) {
    vector <int> a(n + 1); a[0] = 1;
    for (int i = 0; i < B; i++)
      if (k & (1LL << i)) a = add(a, fn[i]);
    int r = 0; for (int i = 0; i < n; i++) {
      r += (long long)a[i + 1] * f[i] % M;
      if (r >= M) r -= M;
    }
    return r;
  }
}; // ends linear recurrence solver
```

11.16 Latice Points under al line.

Lattice points below segment:

solves for sigma $[\frac{(a+b*i)}{m}]$ *where* $0 \le i < n$.

```
LL solve(LL n, LL a, LL b, LL m) {
  if (b == 0) return n * (a / m);
  if (a >= m) return n * (a/m) + solve(n,a % m, b, m);
  if (b >= m)
    return (n-1)*n/2*(b/m) + solve (n, a, b % m, m);
  return solve((a + b * n) /m, (a + b * n) % m, m, b);
}
```

12 Maths.

12.1 Simpson Rule.

```
template <class F, int N = 1 << 20>
double simpson(F f, double l, double r) {
  double h = (r - l) / (double)N, s = 0.0;
  for (int i = 0; i <= N; i++) {
    double x = l + h * i;
    s += f(x) * ((i == 0 || i == N) ? 1 :
                ((i & 1) == 0) ? 2 : 4);
  }
  s *= h / 3.0;
  return s;
}
```

12.2 NTT.

```
const int M = 7340033;
vector <int> root, invRoot;
bool ready = false;

inline void add(int & a, int b)
{ a += b; if (a >= M) a -= M; }
inline int mul(int a, int b)
{ return (long long)a * b % M; }

inline int power(int x, int n) {
  int y = 1; while (n) {
    if(n & 1) y =mul(y, x); x =mul(x, x); n >>=1;
  }
  return y;
}
void calcRoots() {
  ready = true; int a = 2;
  while (power(a, (M - 1) / 2) != M - 1) a++;
  for (int l = 1; (M - 1) % l == 0; l <<= 1) {
    root.push_back(power(a, (M - 1) / l));
    invRoot.push_back(power(root.back(), M - 2));
  }
}
void transform(vector <int> & P, int n, bool invert){
  if (!ready) calcRoots(), ready = true;
  int ln = 0; while ((1 << ln) < n) ln++;
  for (int i = 0; i < n; i++) {
    int x = i, y = 0;
    for (int j = 0; j < ln; j++)
      y = (y << 1) | (x & 1), x >>= 1;
    if (y < i) swap(P[y], P[i]);
  }
  for (int e = 1; (1 << e) <= n; e++) {
    int len = (1 << e), half = len >> 1;
    int step = invert ? invRoot[e] : root[e];
    for (int i = 0; i < n; i += len) {
      int w = 1;
      for (int j = 0; j < half; j++) {
        int u = P[i + j];
        int v = mul(P[i + j + half], w);
        P[i + j] = u; add(P[i + j], v);
        P[i + j + half] = u;
        add(P[i + j + half], M - v);
        w = mul(w, step);
      }
    }
  }
  if (invert) {
    int in = power(n, M - 2);
    for(int i =0; i < n; i++) P[i] =mul(P[i],in);
  }
}

vector <int> mul(vector <int> P, vector <int> Q){
  assert(P.size() > 0 && Q.size() > 0);
  int s = P.size() + Q.size() - 1, n = 1;
  while (n < s) n <<= 1;
  P.resize(n); Q.resize(n);
  if (P == Q) transform(P, n, false), Q = P;
  else
    transform(P,n,false), transform(Q,n,false);
  for (int i = 0; i < n; i++)
    P[i] = mul(P[i], Q[i]);
  transform(P, n, true); P.resize(s); return P;
}

vector <int> inverse(vector <int> P) {
  int s = P.size(); assert(s > 0);
  if (P[0] == 0) return {};
  int n = 1; while (n < P.size()) n <<= 1;
  P.resize(n);
  vector <int> Q(2 * n), R(2 * n), S(2 * n);
  R[0] = power(P[0], M - 2);
  for (int k = 2; k <= n; k *= 2) {
    for(int i =0; i < k; i++) S[i] = R[i];
    for(int i =0; i < min(k, n); i++) Q[i] = P[i];
    for(int i =min(k, n); i < 2 * k; i++) Q[i] =0;
    transform(S, 2 * k, false);
    transform(Q, 2 * k, false);
    for (int i = 0; i < 2 * k; i++)
      S[i] = mul(S[i], mul(S[i], Q[i]));
    transform(S, 2 * k, true);
    for (int i = 0; i < k; i++)
      add(R[i], R[i]), add(R[i], M - S[i]);
  }
  R.resize(s); return R;
}

vector <int> integral(vector <int> P) {
  assert(P.size() > 0); P.push_back(0);
  for (int i = P.size() - 1; i > 0; i--)
    P[i] = mul(P[i - 1], power(i, M - 2));
  P[0] = 0; return P;
}

vector <int> derivative(vector <int> P) {
  assert(P.size() > 0);
  if (P.size() == 1) return {0};
  for (int i = 0; i < P.size() - 1; i++)
    P[i] = mul(P[i + 1], i + 1);
  P.pop_back(); return P;
}

vector <int> log(vector <int> P) {
  int s = P.size(); assert(s > 0);
  assert(P[0] == 1);
  P = integral(mul(derivative(P), inverse(P)));
  P.resize(s); return P;
}

vector <int> exp(vector <int> P) {
  int s = P.size(),n = 1; assert(s >0 && P[0] ==0);
  while (n < s) n <<= 1; vector <int> R({1});
  for (int k = 2; k <= n; k <<= 1) {
    vector <int> Q(k); R.resize(k);
    for(int i =0; i < min(k, n); i++) Q[i] =P[i];
    auto logR = log(R);
    for(int i =0; i<k; i++) add(Q[i], M -logR[i]);
    add(Q[0], 1); R = mul(R, Q);
  }
  R.resize(s); return R;
}

//  P ^ a, a is a real number
//  for log P[0] == 1, if P[0] != 1 transform
//  P into c * (x ^ d) * Q and Q[0] = 1
//  P ^ a = (c ^ a) * (x ^ (d * a)) * exp(a * log(Q))
// a = 1 / 2, P[0] == 1
vector <int> sqrt(vector <int> P) {
  int n = P.size(), inv2 = power(2, M - 2);
  P = log(P);
  for (int i = 0; i < n; i++)
    P[i] = mul(P[i], inv2);
  P = exp(P); P.resize(n);
  return P;
}
```

12.3 CRT for NTT.

```
int MOD[3] = {1045430273, 1051721729, 1053818881};
int PRT[3] = {3, 6, 7};
int crt (int *a, int mod) {
  static int inv[3][3];
  for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
      inv[i][j] = (int) inverse (MOD[i], MOD[j]);
  static int x[3];
  for (int i = 0; i < 3; i++) {
    x[i] = a[i];
    for (int j = 0; j < i; j++) {
      int t = (x[i] - x[j] + MOD[i]) % MOD[i];
      if (t < 0) t += MOD[i];
      x[i] = int (1LL * t * inv[j][i] % MOD[i]);
    }
  }
  int sum = 1, ret = x[0] % mod;
  for (int i = 1; i < 3; i ++) {
    sum = int (1LL * sum * MOD[i - 1] % mod);
    ret += int (1LL * x[i] * sum % mod);
```

```
    if (ret >= mod) ret -= mod;
  }
  return ret;
}
```

12.4 XOR FFT.

```
template <class T>
void xor_fft(vector <T> & P, bool invert) {
  int n = P.size();
  int ln = 0; while ((1 << ln) < n) ln++;
  for (int bit = 0; bit < ln; bit++)
    for (int mask = 0; mask < n; mask++)
      if (!(mask & (1 << bit))) {
        int u = P[mask], v = P[mask | (1 << bit)];
        P[mask] = u + v; P[mask | (1 << bit)] = u - v;
      }
  if (invert) for (auto & x : P) x /= T(n);
}
```

12.5 Berlekamp-Massey.

```
// Given the first elements of a sequence
// returns its recursive equation
template <const int M>
struct BerlekampMassey {
  inline int power(int x, int n) {
    int y = 1 % M;
    while (n) {
      if (n & 1) {
        y = (long long)y * x % M;
      }
      x = (long long)x * x % M;
      n >>= 1;
    }
    return y;
  }
  inline bool inverse(int x) {
    return power(x, M - 2);
  }
  inline vector<int> shift(vector <int> & P, int d){
    vector <int> Q(d + (int)P.size());
    for (int i = 0; i < (int)P.size(); i++)
      Q[i + d] = P[i];
    return Q;
  }
  int calc(vector<int>& P, vector<int>& d, int pos){
    int res = 0;
    for (int i = 0; i < (int)P.size(); i++) {
      res += (long long)d[pos - i] * P[i] % M;
      if (res >= M) res -= M;
    }
    return res;
  }
  vector<int> sub(vector<int> P,const vector<int>&Q){
    if (Q.size() > P.size()) P.resize(Q.size());
    for (int i = 0; i < (int)Q.size(); i++) {
      P[i] -= Q[i]; if (P[i] < 0) P[i] += M;
    }
    return P;
  }
  vector <int> scale(const int c, vector <int> P) {
    for (auto & x : P) x = (long long)x * c % M;
    return P;
  }
  vector <int> solve(vector <int> f) {
    int n = f.size(); vector <int> s(1, 1), b(1,1);
    for (int i = 1, j = 0, ld = f[0]; i < n; i++) {
      int d = calc(s, f, i);
      if (d == 0) continue;
      int c = (long long)d * inverse(ld) % M;
      if (((int)s.size() - 1) * 2 <= i) {
        auto ob = b; b = s;
        s = sub(s, scale(c, shift(ob, i - j)));
        j = i; ld = d;
      }
      else s = sub(s, scale(c, shift(b, i - j)));
    }
//    while (s.size() > 0 && s.back() == 0)
//      s.pop_back(); // ???
    return s;
  }
};
```

13 Algebra

13.1 Tridiagonal Matrix.

```
// a[i]*x[i - 1] + b[i]*x[i] + c[i] * x[i + 1] = d[i]
// a[0] = 0, c[n - 1] = 0
template <class T>
vector<T> tridiagonal(vector <T> a, vector <T> b,
                      vector <T> c, vector <T> d){
  int n = d.size();
  c[0] /= b[0];
  for (int i = 1; i < n; i++)
    c[i] /= (b[i] - a[i] * c[i - 1]);
  d[0] /= b[0];
  for (int i = 1; i < n; i++)
    d[i] = (d[i] - a[i] * d[i - 1]) /
           (b[i] - a[i] * c[i - 1]);
  vector <T> x(n);
  x[n - 1] = d[n - 1];
  for (int i = n - 2; i >= 0; i--)
    x[i] = d[i] - c[i] * x[i + 1];
  return x;
}
```

13.2 Gauss elimination.

```
const double eps = 1e-9;
// A * X' = B'
vector <int> gauss(vector<vector<double>> A,
                   vector<double> B,
                   vector<double> &X) {
  int n = A.size(), m = A[0].size();
  assert(B.size() == n);
  for (auto & r : A) assert(r.size() == m);
  vector <int> where(m, -1);
  for (int r = 0, c = 0; r < n && c < m; c++){
    int p = r;
    for (int i = r + 1; i < n; i++)
      if (abs(A[i][c]) > abs(A[p][c])) p = i;
    if (abs(A[p][c]) < eps) continue;
    swap(A[r], A[p]); swap(B[r], B[p]);
    where[c] = r; double x = 1.0 / A[r][c];
    for (int i = 0; i < n; i++) {
      if (i == r) continue;
      double y = A[i][c] * x;
      for (int j = c; j < m; j++)
        A[i][j] = A[i][j] - A[r][j] * y;
      B[i] = B[i] - B[r] * y;
    }
    r++;
  }
  X.resize(m, 0);
  for (int i = 0; i < m; i++)
    if (where[i] != -1)
      X[i] = B[where[i]] / A[where[i]][i];

  for (int i = 0; i < n; i++) {
    double s = 0.0;
    for (int j = 0; j < m; j++)
      s = s + X[j] * A[i][j];
    if (abs(s - B[i]) >= eps)
      return {};
  }
  for(int i = 0; i < m; i++)
    if (where[i] != -1){
      for (int j = 0; j < n; j++) {
        if (j == i) continue;
        if(abs(A[where[i]][j]) > eps){
          where[i] = -1;
          break;
        }
      }
    } // !!!!!!!!!!!
  return where;
}
```

13.3 Matrix Inverse and Determinant.

```
vector<vector<double>>invert(vector<vector<double>>A,
                             double & D) {
  int n = A.size();
  for (auto & r : A) assert(r.size() == n);
  vector<vector<double>> B(n,vector<double>(n,0.0));
  D = 1.0;
  for (int r = 0; r < n; r++) B[r][r] = 1.0;
  for (int r = 0; r < n; r++) {
    int p = r;
    for (int i = r; i < n; i++)
      if (abs(A[i][r]) > abs(A[p][r])) p = i;
    if (p != r) D = -D, swap(A[r], A[p]);
    if (abs(A[r][r]) < eps) { D = 0; return {}; }
    double x = A[r][r]; D *= x;
```

```
    for (int c = 0; c < n; c++)
      A[r][c] /= x, B[r][c] /= x;
    for (int i = 0; i < n; i++) {
      if (i == r) continue;
      double x = A[i][r];
      for (int j = 0; j < n; j++)
        A[i][j] -= A[r][j] * x,
        B[i][j] -= B[r][j] * x;
    }
  }
  return B;
}
```

14 Others.

14.1 Python Things.

14.1.1 Datetime.

```
from datetime import datetime, date, time,timedelta
import calendar

### Clase 'datetime' ###
ahora = datetime.now()#fecha y hora actual
print("Fecha y Hora:", ahora)
print("Fecha yHora UTC:",ahora.utcnow())#fecha/horaUTC
print("Día:",ahora.day)#día
print("Mes:",ahora.month)#mes
print("Año:",ahora.year)#año
print("Hora:", ahora.hour)#hora
print("Minutos:",ahora.minute)#minuto
print("Segundos:", ahora.second)#segundo
print("Microsegundos:",ahora.microsecond)#microsegundo

### Comparar horas
hora1 = time( 10, 5, 0 )  # Asigna 10h 5m 0s
hora2 = time( 23, 15, 0 ) # Asigna 23h 15m 0s
print("Hora1 < Hora2:", hora1 < hora2)  # True

### Comparar fechas
# Asigna fecha actual
fecha1 = date.today()
#Suma a fecha actual 2 días
fecha2 = date.today() + timedelta(days=2)
print("Fecha1 > Fecha2:", fecha1 > fecha2) #False

### Operaciones con fechas y horas
hoy = date.today()
# Resta a fecha actual 1 día
ayer = hoy - timedelta(days=1)
# Suma a fecha actual 1 día
manana = hoy + timedelta(days=1)
# Resta las dos fechas
diferencia_en_dias = manana - hoy
print("Diferencia en dias:",diferencia_en_dias)

### Diferencia entre dos fechas (datetime)
fecha1 = datetime.now()
fecha2 = datetime(1995, 11, 5, 0, 0, 0)
diferencia = fecha1 - fecha2
print("Diferencia:", diferencia)
print("Entre las 2 fechas hay ", diferencia.days,
      "días y ", diferencia.seconds, "seg.")

###A partir de una fecha se obtiene tupla con año,
###n° semana y día de semana
print("Fecha", fecha1,
"Año,n° sem.,día sem.:",datetime.isocalendar(fecha1))

### Obtener día de la semana por su número
# 0 - lunes, 1 - martes, etc
dia_semana = datetime.weekday(fecha1)
```

14.1.2 Maths.

```
import math

math.e # euler
math.factorial(10)
math.gcd( 25,125 )

# parte entera
math.trunc( 1.999 )

# calcula hipotenusa
math.hypot( 12, 5 )

# math.log( x, [base] ); base = math.e default
math.log( 10, 2 )

math.log(1.0000025)  # returns 2.4999968749105643e-06
math.log1p(0.0000025)# returns 2.4999968750052084e-06
# log1p(x) = log(1 + x) -->
# es mas preciso para valores cercanos a 1

###### Complex #####
import cmath

c = complex( 1.0, 1.0 )
cpx_polar = cmath.polar( c ) #polar
cmath.phase( c ); # obtener base
cmath.rect( cpx_polar[0], cpx_polar[1]) #rectangular
abs( c ) # modulo
#####################################################
### Decimal

from decimal import *
getcontext().prec = 28 # precision
ans = Decimal( '1' )/ Decimal( '7' )
ans = Decimal( math.acos(0) )* Decimal( '2' ) # pi

data = list( map(Decimal, '1.34 1.87 3.45'.split()) )
Decimal(2).sqrt()
Decimal(1).exp()
Decimal('10').ln()

########## others
import itertools ## combinaciones, permutaciones
print( list( itertools.permutations( [1,2,3] ) ) )
```

14.2 Polish Chains.

Convertir expresion infija en posfija.
1. Poner al final de la infija un ')' y al principio de la pila '('.
2. Si se encuentra un '(' se agrega a la pila.
3. Si se encuentra un ')' ir sacando de la pila y agregando a la expresion posfija mientras no se encuentre un '('. El '(' se saca tambien.
4. Si se encuentra un operando se agrega a la expresion posfija.
5. Si se encuentra un operador se saca de la pila hacia la posfija mientras que los operadorades que encuentre sean de >= prioridad que el actual o hasta encontrar un '('. Luego se mete el operador en la pila.

Resolver Posfija.
1. Si encuentro operando se agrega a la pila
2. Si encuentro operador sacar los 2 ultimos elementos, realizar operacion y meter el resultado nuevamente en la pila.
Al final el resultado esta en el unico elemento que queda en la pila.

14.3 Bits Hacks.

- $x \;\&\; -x$ is the least bit in x.
- for (int x = m; x;) { --x &= m; ... }
  loops over all subset masks of m (except m itself).
- c = x&-x, r = x+c; (((r^x) >> 2)/c) | r is the next number after x with the same number of bits set.
- rep(b,0,K) rep(i,0,(1 << K))
  if (i & 1 << b) D[i] += D[i^(1 << b)];
  computes all sums of subsets.