Índice

1. Teoría de números

1.1. Big mod

```
//retorna (b^p)mod(m)
// 0 <= b,p <= 2147483647
// 1 <= m <= 46340
long f(long b, long p, long m){
  long mask = 1;
  long pow2 = b % m;
  long r = 1;

  while (mask){
    if (p & mask)
      r = (r * pow2) % m;
    pow2 = (pow2*pow2) % m;
    mask <<= 1;
  }
  return r;
}
```

1.2. Criba de Eratóstenes

Marca los números primos en un arreglo. Algunos tiempos de ejecución:

SIZE	Tiempo (s)
100000	0.004
1000000	0.078
10000000	1.550
100000000	14.319

```
#include <iostream>

const int SIZE = 1000000;

//criba[i] = false si i es primo
bool criba[SIZE+1];

void buildCriba(){
  memset(criba, false, sizeof(criba));

  criba[0] = criba[1] = true;
  for (int i=2; i<=SIZE; i += 2){
    criba[i] = true;
  }

  for (int i=3; i<=SIZE; i += 2){
    if (!criba[i]){
      for (int j=i+i; j<=SIZE; j += i){
        criba[j] = true;
      }
    }
  }
}
```

1.3. Divisores de un número

Este algoritmo imprime todos los divisores de un número (en desorden) en O($\sqrt{n}$). Hasta 4294967295 (máximo *unsigned long*) responde instantaneamente. Se puede forzar un poco más usando *unsigned long long* pero más allá de 10^{12} empieza a responder muy lento.

```
for (int i=1; i*i<=n; i++) {
  if (n%i == 0) {
    cout << i << endl;
    if (i*i<n) cout << (n/i) << endl;
  }
}
```

2. Grafos

2.1. Algoritmo de Dijkstra

El peso de todas las aristas debe ser no negativo.

```
#include <iostream>
#include <algorithm>
#include <queue>

using namespace std;
```

```
struct edge{
  int to, weight;
  edge() {}
  edge(int t, int w) : to(t), weight(w) {}
  bool operator < (const edge &that) const {
    return weight > that.weight;
  }
};

int main(){
  int N, C=0;
  scanf("%d", &N);
  while (N-- && ++C){
    int n, m, s, t;
    scanf("%d%d%d%d", &n, &m, &s, &t);
    vector<edge> g[n];
    while (m--){
      int u, v, w;
      scanf("%d%d%d", &u, &v, &w);
      g[u].push_back(edge(v, w));
      g[v].push_back(edge(u, w));
    }

    int d[n];
    for (int i=0; i<n; ++i) d[i] = INT_MAX;
    d[s] = 0;
    priority_queue<edge> q;
    q.push(edge(s, 0));
    while (q.empty() == false){
      int node = q.top().to;
      int dist = q.top().weight;
      q.pop();

      if (dist > d[node]) continue;
      if (node == t) break;

      for (int i=0; i<g[node].size(); ++i){
        int to = g[node][i].to;
        int w_extra = g[node][i].weight;

        if (dist + w_extra < d[to]){
          d[to] = dist + w_extra;
          q.push(edge(to, d[to]));
        }
      }
    }
    printf("Case #%d: ", C);
    if (d[t] < INT_MAX) printf("%d\n", d[t]);
    else printf("unreachable\n");
  }
  return 0;
}
```

2.2. Minimum spanning tree: Algoritmo de Prim

```
#include <stdio.h>
#include <string>
```

```cpp
#include <set>
#include <vector>
#include <queue>
#include <iostream>
#include <map>

using namespace std;

typedef string node;
typedef pair<double, node> edge;
typedef map<node, vector<edge> > graph;


int main(){
  double length;
  while (cin >> length){
    int cities;
    cin >> cities;
    graph g;
    for (int i=0; i<cities; ++i){
      string s;
      cin >> s;
      g[s] = vector<edge>();
    }
    int edges;
    cin >> edges;
    for (int i=0; i<edges; ++i){
      string u, v;
      double w;
      cin >> u >> v >> w;
      g[u].push_back(edge(w, v));
      g[v].push_back(edge(w, u));
    }

    double total = 0.0;
    priority_queue<edge, vector<edge>, greater<edge> > q;
    q.push(edge(0.0, g.begin()->first));
    set<node> visited;
    while (q.size()){
      node u = q.top().second;
      double w = q.top().first;
      q.pop(); //!!

      if (visited.count(u)) continue;

      visited.insert(u);
      total += w;
      vector<edge> &vecinos = g[u];
      for (int i=0; i<vecinos.size(); ++i){
        node v = vecinos[i].second;
        double w_extra = vecinos[i].first;
        if (visited.count(v) == 0){
          q.push(edge(w_extra, v));
        }
      }
    }
```

```
    if (total > length){
      cout << "Not enough cable" << endl;
    }else{
      printf("Need %.1lf miles of cable\n", total);
    }

  }
  return 0;
}
```

2.3. Minimum spanning tree: Algoritmo de Kruskal + Union-Find

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

/*
Algoritmo de Kruskal, para encontrar el árbol de recubrimiento de mínima suma.
*/

struct edge{
  int start, end, weight;
  bool operator < (const edge &that) const {
    //Si se desea encontrar el árbol de recubrimiento de máxima suma, cambiar el < por
un >
    return weight < that.weight;
  }
};



/* Union find */
int p[10001], rank[10001];
void make_set(int x){ p[x] = x, rank[x] = 0; }
void link(int x, int y){ rank[x] > rank[y] ? p[y] = x : p[x] = y, rank[x] == rank[y] ?
rank[y]++: 0; }
int find_set(int x){ return x != p[x] ? p[x] = find_set(p[x]) : p[x]; }
void merge(int x, int y){ link(find_set(x), find_set(y)); }
/* End union find */


int main(){
  int c;
  cin >> c;
  while (c--){
    int n, m;
    cin >> n >> m;
    vector<edge> e;
    long long total = 0;
    while (m--){
      edge t;
      cin >> t.start >> t.end >> t.weight;
      e.push_back(t);
      total += t.weight;
    }
```

```
    sort(e.begin(), e.end());
    for (int i=0; i<=n; ++i){
      make_set(i);
    }
    for (int i=0; i<e.size(); ++i){
      int u = e[i].start, v = e[i].end, w = e[i].weight;
      if (find_set(u) != find_set(v)){
        //xprintf("Joining %d with %d using weight %d\n", u, v, w);
        total -= w;
        merge(u, v);
      }
    }
    cout << total << endl;

  }
  return 0;
}
```

2.4. Algoritmo de Floyd

```
#include <iostream>
#include <climits>
#include <algorithm>

using namespace std;

unsigned long long g[101][101];

int main(){
  int casos;
  cin >> casos;
  bool first = true;
  while (casos--){
    if (!first) cout << endl;
    first = false;

    int n, e, t;
    cin >> n >> e >> t;
    for (int i=0; i<n; ++i){
      for (int j=0; j<n; ++j){
        g[i][j] = INT_MAX;
      }
      g[i][i] = 0;
    }

    int m;
    cin >> m;
    while (m--){
      int i, j, k;
      cin >> i >> j >> k;
      g[i-1][j-1] = k;
    }

    for (int k=0; k<n; ++k){
      for (int i=0; i<n; ++i){
        for (int j=0; j<n; ++j){
          g[i][j] = min(g[i][j], g[i][k] + g[k][j]);
```

```
        }
      }
    }

    int r=0;
    e -= 1;
    for (int i=0; i<n; ++i){
      r += ((g[i][e] <= t) ? 1 : 0);
    }

    cout << r << endl;
  }
  return 0;
}
```

2.5. Puntos de articulación

```
#include <vector>
#include <set>
#include <map>
#include <algorithm>
#include <iostream>
#include <iterator>

using namespace std;

typedef string node;
typedef map<node, vector<node> > graph;
typedef char color;

const color WHITE = 0, GRAY = 1, BLACK = 2;

graph g;
map<node, color> colors;
map<node, int> d, low;

set<node> cameras;

int timeCount;

void dfs(node v, bool isRoot = true){
  colors[v] = GRAY;
  d[v] = low[v] = ++timeCount;
  vector<node> neighbors = g[v];
  int count = 0;
  for (int i=0; i<neighbors.size(); ++i){
    if (colors[neighbors[i]] == WHITE){ //  (v, neighbors[i]) is a tree edge
      dfs(neighbors[i], false);
      if (!isRoot && low[neighbors[i]] >= d[v]){
        cameras.insert(v);
      }
      low[v] = min(low[v], low[neighbors[i]]);
      ++count;
    }else{ // (v, neighbors[i]) is a back edge
      low[v] = min(low[v], d[neighbors[i]]);
    }
  }
```

```
  if (isRoot && count > 1){ //Is root and has two neighbors in the DFS-tree
    cameras.insert(v);
  }
  colors[v] = BLACK;
}

int main(){
  int n;
  int map = 1;
  while (cin >> n && n > 0){
    if (map > 1) cout << endl;
    g.clear();
    colors.clear();
    d.clear();
    low.clear();
    timeCount = 0;
    while (n--){
      node v;
      cin >> v;
      colors[v] = WHITE;
      g[v] = vector<node>();
    }

    cin >> n;
    while (n--){
      node v,u;
      cin >> v >> u;
      g[v].push_back(u);
      g[u].push_back(v);
    }

    cameras.clear();

    for (graph::iterator i = g.begin(); i != g.end(); ++i){
      if (colors[(*i).first] == WHITE){
        dfs((*i).first);
      }
    }

    cout << "City map #"<<map<<": " << cameras.size() << " camera(s) found" <<
endl;
    copy(cameras.begin(), cameras.end(), ostream_iterator<node>(cout,"\n"));
    ++map;
  }
  return 0;
}
```

2.6. Máximo flujo: Método de Ford-Fulkerson, algoritmo de Edmonds-Karp

El algoritmo de Edmonds-Karp es una modificación al método de Ford-Fulkerson. Este último utiliza DFS para hallar un camino de aumentación, pero la sugerencia de Edmonds-Karp es utilizar BFS que lo hace más eficiente en algunos grafos.

```
int cap[MAXN+1][MAXN+1], flow[MAXN+1][MAXN+1], prev[MAXN+1];

int fordFulkerson(int n, int s, int t){
  int ans = 0;
```

```
  for (int i=0; i<n; ++i) fill(flow[i], flow[i]+n, 0);
  while (true){
    fill(prev, prev+n, -1);
    queue<int> q;
    q.push(s);
    while (q.size() && prev[t] == -1){
      int u = q.front();
      q.pop();
      for (int v = 0; v<n; ++v)
        if (v != s && prev[v] == -1 && cap[u][v] > 0 && cap[u][v] - flow[u][v] > 0)
          prev[v] = u, q.push(v);
    }

    if (prev[t] == -1) break;

    int bottleneck = INT_MAX;
    for (int v = t, u = prev[v]; u != -1; v = u, u = prev[v]){
      bottleneck = min(bottleneck, cap[u][v] - flow[u][v]);
    }
    for (int v = t, u = prev[v]; u != -1; v = u, u = prev[v]){
      flow[u][v] += bottleneck;
      flow[v][u] = -flow[u][v];
    }
    ans += bottleneck;
  }
  return ans;
}
```

3. Programación dinámica

3.1. Longest common subsequence

```
#define MAX(a,b) ((a>b)?(a):(b))
int dp[1001][1001];

int lcs(const string &s, const string &t){
  int m = s.size(), n = t.size();
  if (m == 0 || n == 0) return 0;
  for (int i=0; i<=m; ++i)
    dp[i][0] = 0;
  for (int j=1; j<=n; ++j)
    dp[0][j] = 0;
  for (int i=0; i<m; ++i)
    for (int j=0; j<n; ++j)
      if (s[i] == t[j])
        dp[i+1][j+1] = dp[i][j]+1;
      else
        dp[i+1][j+1] = MAX(dp[i+1][j], dp[i][j+1]);
  return dp[m][n];
}
```

4. Geometría

4.1. Área de un polígono

Si P es un polígono simple (no se intersecta a sí mismo) su área está dada por:

$A(P) = \frac{1}{2}\sum_{i=0}^{n-1}(x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$

4.2. Centro de masa de un polígono

Si P es un polígono simple (no se intersecta a sí mismo) su centro de masa está dado por:

$$\bar{C}_x = \frac{\iint_R x\,dA}{M} = \frac{1}{6M}\sum_{i=1}^{n}(y_{i+1} - y_i)(x_{i+1}^2 + x_{i+1} \cdot x_i + x_i^2)$$

$$\bar{C}_y = \frac{\iint_R y\,dA}{M} = \frac{1}{6M}\sum_{i=1}^{n}(x_i - x_{i+1})(y_{i+1}^2 + y_{i+1} \cdot y_i + y_i^2)$$

Donde M es el área del polígono.
Otra posible fórmula equivalente:

$$\bar{C}_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$$

$$\bar{C}_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i \cdot y_{i+1} - x_{i+1} \cdot y_i)$$

4.3. Convex hull: Graham Scan

Complejidad: $O(n \log_2 n)$

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
#include <cmath>

using namespace std;

struct point{
  int x,y;
  point() {}
  point(int X, int Y) : x(X), y(Y) {}
};

point pivot;

ostream& operator<< (ostream& out, const point& c)
{
  out << "(" << c.x << "," << c.y << ")";
  return out;
}

//P es un polígono ordenado anticlockwise.
//Si es clockwise, retorna el area negativa.
//Si no esta ordenado retorna pura mierda
double area(const vector<point> &p){
  double r = 0.0;
  for (int i=0; i<p.size(); ++i){
    int j = (i+1) % p.size();
    r += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return r/2.0;
```

```
}

//retorna si c esta a la izquierda de el segmento AB
inline int cross(const point &a, const point &b, const point &c){
  return (b.x-a.x)*(c.y-a.y) - (c.x-a.x)*(b.y-a.y);
}

//Self < that si esta a la derecha del segmento Pivot-That
bool angleCmp(const point &self, const point &that){
  return( cross(pivot, that, self) < 0 );
}

inline int distsqr(const point &a, const point &b){
  return (a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y);
}

//vector p tiene los puntos ordenados anticlockwise
vector<point> graham(vector<point> p){
  pivot = p[0];
  sort(p.begin(), p.end(), angleCmp);
  //Ordenar por ángulo y eliminar repetidos.
  //Si varios puntos tienen el mismo angulo se borran todos excepto el que esté más lejos
  for (int i=1; i<p.size()-1; ++i){
    if (cross(p[0], p[i], p[i+1]) == 0){ //Si son colineales...
      if (distsqr(p[0], p[i]) < distsqr(p[0], p[i+1])){ //Borrar el mas cercano
        p.erase(p.begin() + i);
      }else{
        p.erase(p.begin() + i + 1);
      }
      i--;
    }
  }

  vector<point> chull(p.begin(), p.begin()+3);

  //Ahora sí!!!
  for (int i=3; i<p.size(); ++i){
    while ( chull.size() >= 2 && cross(chull[chull.size()-2], chull[chull.size()-1],
p[i]) < 0){
      chull.erase(chull.end() - 1);
    }
    chull.push_back(p[i]);
  }

  return chull;
}

int main(){
  int n;
  int tileNo = 1;
  while (cin >> n && n){
    vector<point> p;
    point min(10000, 10000);
    int minPos;
    for (int i=0; i<n; ++i){
      int x, y;
      cin >> x >> y;
```

```
      p.push_back(point(x,y));
      if (y < min.y || (y == min.y && x < min.x )){
        min = point(x,y);
        minPos = i;
      }
    }
    double tileArea = fabs(area(p));

    //Destruye el orden cw|ccw poligono, pero hay que hacerlo por que Graham necesita el
pivote en p[0]
    swap(p[0], p[minPos]);
    pivot = p[0];
    double chullArea = fabs(area(graham(p)));

    printf("Tile #%d\n", tileNo++);
    printf("Wasted Space =%.2f \%\n\n",  (chullArea - tileArea) * 100.0 / chullArea);

  }
  return 0;
}
```

4.4. Convex hull: Andrew's monotone chain

Complejidad: $O(n \log_2 n)$

```
// Implementation of Monotone Chain Convex Hull algorithm.
#include <algorithm>
#include <vector>
using namespace std;

typedef long long CoordType;

struct Point {
        CoordType x, y;

        bool operator <(const Point &p) const {
                return x < p.x || (x == p.x && y < p.y);
        }
};

// 2D cross product.
// Return a positive value, if OAB makes a counter-clockwise turn,
// negative for clockwise turn, and zero if the points are collinear.
CoordType cross(const Point &O, const Point &A, const Point &B)
{
        return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}

// Returns a list of points on the convex hull in counter-clockwise order.
// Note: the last point in the returned list is the same as the first one.
vector<Point> convexHull(vector<Point> P)
{
        int n = P.size(), k = 0;
        vector<Point> H(2*n);

        // Sort points lexicographically
        sort(P.begin(), P.end());
```

```
        // Build lower hull
        for (int i = 0; i < n; i++) {
                while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
                H[k++] = P[i];
        }

        // Build upper hull
        for (int i = n-2, t = k+1; i >= 0; i--) {
                while (k >= t && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
                H[k++] = P[i];
        }

        H.resize(k);
        return H;
}
```

4.5. Mínima distancia entre un punto y un segmento

```
struct point{
  double x,y;
};

inline double dist(const point &a, const point &b){
  return sqrt((a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}

inline double distsqr(const point &a, const point &b){
  return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
}

/*
  Returns the closest distance between point pnt and the segment that goes from point a
to b
  Idea by: http://local.wasp.uwa.edu.au/~pbourke/geometry/pointline/
 */
double distance_point_to_segment(const point &a, const point &b, const point &pnt){
  double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) / distsqr(a, b);
  point intersection;
  intersection.x = a.x + u*(b.x - a.x);
  intersection.y = a.y + u*(b.y - a.y);
  if (u < 0.0 || u > 1.0){
    return min(dist(a, pnt), dist(b, pnt));
  }
  return dist(pnt, intersection);
}
```

4.6. Mínima distancia entre un punto y una recta

```
/*
  Returns the closest distance between point pnt and the line that passes through points
a and b
  Idea by: http://local.wasp.uwa.edu.au/~pbourke/geometry/pointline/
 */
double distance_point_to_line(const point &a, const point &b, const point &pnt){
  double u = ((pnt.x - a.x)*(b.x - a.x) + (pnt.y - a.y)*(b.y - a.y)) / distsqr(a, b);
  point intersection;
  intersection.x = a.x + u*(b.x - a.x);
```

```
  intersection.y = a.y + u*(b.y - a.y);
  return dist(pnt, intersection);
}
```

5. Java

5.1. Entrada desde entrada estándar

Este primer método es muy fácil pero es mucho más ineficiente porque utiliza Scanner en vez de BufferedReader:

```
import java.io.*;
import java.util.*;

class Main{
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()){
            String s= sc.nextLine();
            System.out.println("Leí: " + s);
        }
    }
}
```

Este segundo es más rápido:

```
import java.util.*;
import java.io.*;
import java.math.*;

class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        String line = reader.readLine();
        StringTokenizer tokenizer = new StringTokenizer(line);
        int N = Integer.valueOf(tokenizer.nextToken());
        while (N-- > 0){
            String a, b;
            a = reader.readLine();
            b = reader.readLine();

            int A = a.length(), B = b.length();
            if (B > A){
                System.out.println("0");
            }else{
                BigInteger dp[][] = new BigInteger[2][A];
                /*
dp[i][j] = cantidad de maneras diferentes
en que puedo distribuir las primeras i
letras de la subsecuencia (b) terminando
en la letra j de la secuencia original (a)
*/

                if (a.charAt(0) == b.charAt(0)){
                    dp[0][0] = BigInteger.ONE;
                }else{
                    dp[0][0] = BigInteger.ZERO;
                }
                for (int j=1; j<A; ++j){
```

```
                    dp[0][j] = dp[0][j-1];
                    if (a.charAt(j) == b.charAt(0)){
                        dp[0][j] = dp[0][j].add(BigInteger.ONE);
                    }
                }

                for (int i=1; i<B; ++i){
                    dp[i%2][0] = BigInteger.ZERO;
                    for (int j=1; j<A; ++j){
                        dp[i%2][j] = dp[i%2][j-1];
                        if (a.charAt(j) == b.charAt(i)){
                            dp[i%2][j] = dp[i%2][j].add(dp[(i+1)%2][j-1]);
                        }
                    }
                }
                System.out.println(dp[(B-1)%2][A-1].toString());
            }
        }
    }
}
```

5.2. Entrada desde archivo

```
import java.io.*;
import java.util.*;
public class BooleanTree {
        public static void main(String[] args) throws FileNotFoundException {
                System.setIn(new FileInputStream("tree.in"));
                System.setOut(new PrintStream("tree.out"));
                Scanner reader = new Scanner(System.in);
                N = reader.nextInt();
                for (int c = 1; c <= N; ++c) {
                        int res = 100;
                        if (res < 1000)
                                System.out.println("Case #" + c + ": " + res);
                        else
                                System.out.println("Case #" + c + ": IMPOSSIBLE");
                }
        }
}
```

6. C++

6.1. Entrada desde archivo

```
#include <iostream>
#include <fstream>

using namespace std;

int _main(){
  freopen("entrada.in", "r", stdin);
  freopen("entrada.out", "w", stdout);

  string s;
  while (cin >> s){
    cout << "Leí " << s << endl;
```

```
  }
  return 0;
}


int main(){
  ifstream fin("entrada.in");
  ofstream fout("entrada.out");

  string s;
  while (fin >> s){
    fout << "Leí " << s << endl;
  }
  return 0;
}
```

6.2. Strings con caractéres especiales

```
#include <iostream>
#include <cassert>
#include <stdio.h>
#include <assert.h>
#include <wchar.h>
#include <wctype.h>
#include <locale.h>

using namespace std;

int main(){
  assert(setlocale(LC_ALL, "en_US.UTF-8") != NULL);
  wchar_t c;

  wstring s;
  while (getline(wcin, s)){
    wcout << L"Leí : " << s << endl;
    for (int i=0; i<s.size(); ++i){
      c = s[i];
      wprintf(L"%lc%lc\n", towlower(s[i]), towupper(s[i]));
    }
  }

  return 0;
}
```

Nota: Como alternativa a la función getline, se pueden utilizar las funciones fgetws y fputws, y más adelante swscanf y wprintf:

```
#include <iostream>
#include <cassert>
#include <stdio.h>
#include <assert.h>
#include <wchar.h>
#include <wctype.h>
#include <locale.h>


using namespace std;
```

```c
int main(){
  assert(setlocale(LC_ALL, "en_US.UTF-8") != NULL);
  wchar_t in_buf[512], out_buf[512];
  swprintf(out_buf, 512, L"¿Podrías escribir un número?, Por ejemplo %d. ¡Gracias,
pingüino español!\n", 3);
  fputws(out_buf, stdout);

  fgetws(in_buf, 512, stdin);
  int n;
  swscanf(in_buf, L"%d", &n);

  swprintf(out_buf, 512, L"Escribiste %d, yo escribo ¿ÔÏàÚÑ~\n", n);
  fputws(out_buf, stdout);

  return 0;
}
```