Resumen de algoritmos para torneos de programación

Andrés Mejía

20 de septiembre de 2008

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

Field-testing:

- *SPOJ* - 2912 - Super Primes
- *Live Archive* - 3639 - Prime Path

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

SIZE	Tiempo (s)
100000	0.003
1000000	0.060
10000000	0.620
100000000	7.650

```
#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=4; i<=SIZE; i += 2){
    criba[i] = true;
  }
  for (int i=3; i*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. Combinatoria

2.1. Cuadro resumen

Fórmulas para combinaciones y permutaciones:

Tipo	*¿Se permite la repetición?*	*Fórmula*
r-permutaciones	No	$\frac{n!}{(n-r)!}$
r-combinaciones	No	$\frac{n!}{r!(n-r)!}$
r-permutaciones	Sí	n^r
r-combinaciones	Sí	$\frac{(n+r-1)!}{r!(n-1)!}$

Tomado de *Matemática discreta y sus aplicaciones*, Kenneth Rosen, 5ta edición, McGraw-Hill, página 315.

2.2. Combinaciones, coeficientes binomiales, triángulo de Pascal

Complejidad: $O(n^2)$

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \\ 1 & n = k \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{en otro caso} \end{cases}$$

```
const int N = 30;
long long choose[N+1][N+1];
  /* Binomial coefficients */
  for (int i=0; i<=N; ++i) choose[i][0] = choose[i][i] = 1;
  for (int i=1; i<=N; ++i)
    for (int j=1; j<i; ++j)
      choose[i][j] = choose[i-1][j-1] + choose[i-1][j];
```

Nota: $\binom{n}{k}$ está indefinido en el código anterior si $n > k$. ¡La tabla puede estar llena con cualquier basura del compilador!

2.3. Permutaciones con elementos indistinguibles

El número de permutaciones diferentes de n objetos, donde hay n_1 objetos indistinguibles de tipo 1, n_2 objetos indistinguibles de tipo 2, ..., y n_k objetos indistinguibles de tipo k, es

$$\frac{n!}{n_1!n_2!\cdots n_k!}$$

Ejemplo: Con las letras de la palabra `PROGRAMAR` se pueden formar $\frac{9!}{2!\cdot 3!} = 30240$ permutaciones diferentes.

2.4. Desordenes, desarreglos o permutaciones completas

Un desarreglo es una permutación donde ningún elemento i está en la posición i-ésima. Por ejemplo, *4213* es un desarreglo de 4 elementos pero *3241* no lo es porque el 2 aparece en la posición 2.
Sea D_n el número de desarreglos de n elementos, entonces:

$$D_n = \begin{cases} 1 & n = 0 \\ 0 & n = 1 \\ (n-1)(D_{n-1} + D_{n-2}) & n \geq 2 \end{cases}$$

Usando el principio de inclusión-exclusión, también se puede encontrar la fórmula

$$D_n = n!\left[1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \cdots + (-1)^n\frac{1}{n!}\right] = n!\sum_{i=0}^{n}\frac{(-1)^i}{i!}$$

3. Grafos

3.1. 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;
}
```

3.2. Minimum spanning tree: Algoritmo de Prim

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

3.3. 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)){
        //printf("Joining %d with %d using weight %d\n", u, v, w);
        total -= w;
        merge(u, v);
      }
    }
    cout << total << endl;

  }
  return 0;
}
```

3.4. Algoritmo de Floyd-Warshall

Complejidad: $O(n^3)$
Se asume que no hay ciclos de costo negativo.

```
/*
  g[i][j] = Distancia entre el nodo i y el j.
 */
unsigned long long g[101][101];

void floyd(){
  //Llenar g
  //...

  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]);
      }
    }
  }
  /*
    Acá se cumple que g[i][j] = Longitud de la ruta más corta de i a j.
   */
}
```

3.5. Algoritmo de Bellman-Ford

Si no hay ciclos de coste negativo, encuentra la distancia más corta entre un nodo y todos los demás. Si sí hay, permite saberlo.
El coste de las aristas sí puede ser negativo.

```
struct edge{
  int u, v, w;
};

edge * e; //e = Arreglo de todas las aristas
long long d[300]; //Distancias
int n; //Cantidad de nodos
int m; //Cantidad de aristas

/*
  Retorna falso si hay un ciclo de costo negativo.

  Si retorna verdadero, entonces d[i] contiene la distancia más corta entre el s y el
nodo i.
 */
bool bellman(int &s){
  //Llenar e
  e = new edge[n];
  //...

  for (int i=0; i<n; ++i) d[i] = INT_MAX;
  d[s] = 0LL;

  for (int i=0; i<n-1; ++i){
    bool cambio = false;
    for (int j=0; j<m; ++j){
      int u = e[j].u, v = e[j].v;
      long long w = e[j].w;
      if (d[u] + w < d[v]){
        d[v] = d[u] + w;
        cambio = true;
```

```
      }
    }
    if (!cambio) break; //Early-exit
  }

  for (int j=0; j<m; ++j){
    int u = e[j].u, v = e[j].v;
    long long w = e[j].w;
    if (d[u] + w < d[v]) return false;
  }

  delete [] e;
  return true;
}
```

3.6. 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;
}
```

3.7. 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];

/*
  cap[i][j] = Capacidad de la arista (i, j).
  flow[i][j] = Flujo actual de i a j.
```

```
  prev[i] = Predecesor del nodo i en un camino de aumentación.
 */

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;
}
```

4. Programación dinámica

4.1. 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];
}
```

4.2. Partición de troncos

Este problema es similar al problema de *Matrix Chain Multiplication*. Se tiene un tronco de longitud n, y m puntos de corte en el tronco. Se puede hacer un corte a la vez, cuyo costo es igual a la longitud del tronco. ¿Cuál es el mínimo costo para partir todo el tronco en pedacitos individuales?

Ejemplo: Se tiene un tronco de longitud 10. Los puntos de corte son 2, 4, y 7. El mínimo costo para partirlo es 20, y se obtiene así:

- Partir el tronco (0, 10) por 4. Vale 10 y quedan los troncos (0, 4) y (4, 10).
- Partir el tronco (0, 4) por 2. Vale 4 y quedan los troncos (0, 2), (2, 4) y (4, 10).
- No hay que partir el tronco (0, 2).
- No hay que partir el tronco (2, 4).
- Partir el tronco (4, 10) por 7. Vale 6 y quedan los troncos (4, 7) y (7, 10).
- No hay que partir el tronco (4, 7).
- No hay que partir el tronco (7, 10).
- El costo total es $10 + 4 + 6 = 20$.

El algoritmo es $O(n^3)$, pero optimizable a $O(n^2)$ con una tabla adicional:

```
/*
  O(n^3)

 dp[i][j] = Mínimo costo de partir la cadena entre las particiones i e j, ambas
incluídas.
 */
int dp[1005][1005];
int p[1005];

int cubic(){
  int n, m;
  while (scanf("%d%d", &n, &m)==2){
    p[0] = 0;
    for (int i=1; i<=m; ++i){
      scanf("%d", &p[i]);
    }
    p[m+1] = n;
    m += 2;

    for (int i=0; i<m; ++i){
      dp[i][i+1] = 0;
    }

    for (int i=m-2; i>=0; --i){
      for (int j=i+2; j<m; ++j){
        dp[i][j] = p[j]-p[i];
        int t = INT_MAX;
        for (int k=i+1; k<j; ++k){
          t = min(t, dp[i][k] + dp[k][j]);
        }
        dp[i][j] += t;
      }
    }

    printf("%d\n", dp[0][m-1]);
```

```
  }
  return 0;
}


/*
  O(n^2)

  dp[i][j] = Mínimo costo de partir la cadena entre las particiones i e j, ambas
incluídas.
  pivot[i][j] = Índice de la partición que usé para lograr dp[i][j].
 */
int dp[1005][1005], pivot[1005][1005];
int p[1005];

int quadratic(){
  int n, m;
  while (scanf("%d%d", &n, &m)==2){
    p[0] = 0;
    for (int i=1; i<=m; ++i){
      scanf("%d", &p[i]);
    }
    p[m+1] = n;
    m += 2;

    for (int i=0; i<m-1; ++i){
      dp[i][i+1] = 0;
    }
    for (int i=0; i<m-2; ++i){
      dp[i][i+2] = p[i+2] - p[i];
      pivot[i][i+2] = i+1;
    }

    for (int d=3; d<m; ++d){ //d = longitud
      for (int j, i=0; (j = i + d) < m; ++i){
        dp[i][j] = p[j] - p[i];
        int t = INT_MAX, s;
        for (int k=pivot[i][j-1]; k<=pivot[i+1][j]; ++k){
          int x = dp[i][k] + dp[k][j];
          if (x < t) t = x, s = k;
        }
        dp[i][j] += t, pivot[i][j] = s;
      }
    }

    printf("%d\n", dp[0][m-1]);
  }
  return 0;
}
```

5. Geometría

5.1. Área de un polígono

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

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

```
//P es un polígono ordenado anticlockwise.
//Si es clockwise, retorna el area negativa.
//Si no esta ordenado retorna pura mierda.
//P[0] != P[n-1]
double PolygonArea(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;
}
```

5.2. 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:

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

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

5.3. Convex hull: Graham Scan

Complejidad: $O(n \log_2 n)$

```
/*
  Graham Scan.
 */
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
#include <math.h>
#include <stdio.h>

using namespace std;

const double pi = 2*acos(0);

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;
}

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);
}

inline double dist(const point &a, const point &b){
  return sqrt(distsqr(a, b));
}

//retorna > 0 si c esta a la izquierda del segmento AB
//retorna < 0 si c esta a la derecha del segmento AB
//retorna == 0 si c es colineal con 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){
  int t = cross(pivot, that, self);
  if (t < 0) return true;
  if (t == 0){
    //Self < that si está más cerquita
    return (distsqr(pivot, self) < distsqr(pivot, that));
  }
  return false;
}

vector<point> graham(vector<point> p){
  //Metemos el más abajo más a la izquierda en la posición 0
  for (int i=1; i<p.size(); ++i){
    if (p[i].y < p[0].y || (p[i].y == p[0].y && p[i].x < p[0].x ))
      swap(p[0], p[i]);
  }

  pivot = p[0];
  sort(p.begin(), p.end(), angleCmp);

  //Ordenar por ángulo y eliminar repetidos.
  //Si varios puntos tienen el mismo angulo el más lejano queda después en la lista
  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]);
  }
```

```
  //chull contiene los puntos del convex hull ordenados anti-clockwise.
  //No contiene ningún punto repetido.
  //El primer punto no es el mismo que el último, i.e, la última arista
  //va de chull[chull.size()-1] a chull[0]
  return chull;
}
```

5.4. 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;
```

```
}
```

5.5. 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);
}
```

5.6. 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);
}
```

5.7. Determinar si un polígono es convexo

```
/*
  Returns positive if a-b-c make a left turn.
  Returns negative if a-b-c make a right turn.
  Returns 0.0 if a-b-c are colineal.
 */
double turn(const point &a, const point &b, const point &c){
  double z = (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x);
  if (fabs(z) < 1e-9) return 0.0;
```

```
  return z;
}

/*
  Returns true if polygon p is convex.
  False if it's concave or it can't be determined
  (For example, if all points are colineal we can't
  make a choice).
 */
bool isConvexPolygon(const vector<point> &p){
  int mask = 0;
  int n = p.size();
  for (int i=0; i<n; ++i){
    int j=(i+1)%n;
    int k=(i+2)%n;
    double z = turn(p[i], p[j], p[k]);
    if (z < 0.0){
      mask |= 1;
    }else if (z > 0.0){
      mask |= 2;
    }
    if (mask == 3) return false;
  }
  return mask != 0;
}
```

5.8. Determinar si un punto está dentro de un polígono convexo

```
/*
  Returns true if point a is inside convex polygon p.
  Note that if point a lies on the border of p it
  is considered outside.

  We assume p is convex! The result is useless if p
  is concave.
 */
bool insideConvexPolygon(const vector<point> &p, const point &a){
  int mask = 0;
  for (int i=0; i<n; ++i){
    int j = (i+1)%n;
    double z = turn(p[i], p[j], a);
    if (z < 0.0){
      mask |= 1;
    }else if (z > 0.0){
      mask |= 2;
    }else if (z == 0.0) return false;
    if (mask == 3) return false;
  }
  return mask != 0;

}
```

5.9. Determinar si un punto está dentro de un polígono cualquiera

ADVERTENCIA: Código no probado.

```
/*********
 * Point *
```

```
 *********
 * A simple point class used by some of the routines below.
 * Anything else that supports .x and .y will work just as
 * well. There are 2 variants - double and int.
 **/
struct P { double x, y; P() {}; P( double q, double w ) : x( q ), y( w ) {}
};
struct P { int x, y; P() {}; P( int q, int w ) : x( q ), y( w ) {} };

/***************
 * Polar angle *
 ***************
 * Returns an angle in the range [0, 2*Pi) of a given Cartesian point.
 * If the point is (0,0), -1.0 is returned.
 * REQUIRES:
 * include math.h
 * define EPS 0.000000001, or your choice
 * P has members x and y.
 **/
double polarAngle( P p )
{
    if( fabs( p.x ) <= EPS && fabs( p.y ) <= EPS ) return -1.0;
    if( fabs( p.x ) <= EPS ) return ( p.y > EPS ? 1.0 : 3.0 ) * acos( 0 );
    double theta = atan( 1.0 * p.y / p.x );
    if( p.x > EPS ) return( p.y >= -EPS ? theta : ( 4 * acos( 0 ) + theta )
);
    return( 2 * acos( 0 ) + theta );
}

/************************
 * Point inside polygon *
 ************************
 * Returns true iff p is inside poly.
 * PRE: The vertices of poly are ordered (either clockwise or
 *      counter-clockwise.
 * POST: Modify code inside to handle the special case of "on an edge".
 * REQUIRES:
 * polarAngle()
 * include math.h
 * include vector
 * define EPS 0.000000001, or your choice
 **/
bool pointInPoly( P p, vector< P > &poly )
{
    int n = poly.size();
    double ang = 0.0;
    for( int i = n - 1, j = 0; j < n; i = j++ )
    {
        P v( poly[i].x - p.x, poly[i].y - p.y );
```

```
        P w( poly[j].x - p.x, poly[j].y - p.y );
        double va = polarAngle( v );
        double wa = polarAngle( w );
        double xx = wa - va;
        if( va < -0.5 || wa < -0.5 || fabs( fabs( xx ) - 2 * acos( 0 ) ) <
EPS )
        {
            // POINT IS ON THE EDGE
            assert( false );
            ang += 2 * acos( 0 );
            continue;
        }
        if( xx < -2 * acos( 0 ) ) ang += xx + 4 * acos( 0 );
        else if( xx > 2 * acos( 0 ) ) ang += xx - 4 * acos( 0 );
        else ang += xx;
    }
    return( ang * ang > 1.0 );
}
```

6. Misceláneo

6.1. El *parser* más rápido del mundo

- Cada no-terminal: un método
- Cada lado derecho: invocar los métodos de los no-terminales o
- Cada terminal: invocar proceso *match*
- Alternativas en una producción: se hace un *if*

No funciona con gramáticas recursivas por izquierda ó en las que en algún momento haya varias posibles escogencias que empiezan por el mismo caracter (En ambos casos la gramática se puede factorizar).

Ejemplo: Para la gramática:

$$A \longrightarrow (A)A$$

$$A \longrightarrow \epsilon$$

```
//A -> (A)A | Epsilon

#include <iostream>
#include <string>

using namespace std;

bool ok;
char sgte;
int i;
string s;

bool match(char c){
  if (sgte != c){
    ok = false;
  }
  sgte = s[++i];
```

```
}

void A(){
  if (sgte == '('){
    match('(');
    A(); match(')'); A();
  }else if (sgte == '$' || sgte == ')'){
    //nada
  }else{
    ok = false;
  }
}

int main(){
  while(getline(cin, s) && s != ""){
    ok = true;
    s += '$';
    sgte = s[(i = 0)];
    A();
    if (i < s.length()-1) ok = false; //No consumi toda la cadena
    if (ok){
      cout << "Accepted\n";
    }else{
      cout << "Not accepted\n";
    }

  }
}
```

7. Java

7.1. 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());
            }
        }
    }
}
```

7.2. 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");
                }
        }
}
```

7.3. Mapas y sets

Programa de ejemplo:

```
import java.util.*;

public class Ejemplo {
    public static void main(String[] args){
        /*
         * Mapas
         * Tanto el HashMap como el TreeMap funcionan, pero tienen diferentes detalles
         * y difieren en algunos métodos (Ver API).
         */
        System.out.println("Maps");
        //TreeMap<String, Integer> m = new TreeMap<String, Integer>();
        HashMap<String, Integer> m = new HashMap<String, Integer>();
        m.put("Hola", new Integer(465));
        System.out.println("m.size() = " + m.size());

        if (m.containsKey("Hola")){
            System.out.println(m.get("Hola"));
        }

        System.out.println(m.get("Objeto inexistente"));


        /*
         * Sets
         * La misma diferencia entre TreeSet y HashSet.
         */
        System.out.println("\nSets");
        /*
         * *OJO: El HashSet no está en orden, el TreeSet sí.
         */
        //HashSet<Integer> s = new HashSet<Integer>();
        TreeSet<Integer> s = new TreeSet<Integer>();
        s.add(3576); s.add(new Integer("54")); s.add(new Integer(1000000007));

        if (s.contains(54)){
            System.out.println("54 presente.");
        }

        if (s.isEmpty() == false){
            System.out.println("s.size() = " + s.size());
            Iterator<Integer> i = s.iterator();
            while (i.hasNext()){
                System.out.println(i.next());
                i.remove();
            }
```

```
            System.out.println("s.size() = " + s.size());
        }
    }
}
```

La salida de este programa es:

```
Maps
m.size() = 1
465
null

Sets
54 presente.
s.size() = 3
54
3576
1000000007
s.size() = 0
```

Si quiere usarse una clase propia como llave del mapa o como elemento del set, la clase debe implementar algunos métodos especiales: Si va a usarse un TreeMap ó TreeSet hay que implementar los métodos `compareTo` y `equals` de la interfaz `Comparable` como en la sección 7.4. Si va a usarse un HashMap ó HashSet hay más complicaciones.

Sugerencia: Inventar una manera de codificar y decodificar la clase en una String o un Integer y meter esa representación en el mapa o set: esas clases ya tienen los métodos implementados.

7.4. Colas de prioridad

Hay que implementar unos métodos. Veamos un ejemplo:

```
import java.util.*;


class Item implements Comparable<Item>{
    int destino, peso;

    Item(int destino, int peso){
        this.peso = peso;
        this.destino = destino;
    }
    /*
     * Implementamos toda la javazofia.
     */
    public int compareTo(Item otro){
        // Return < 0 si this < otro
        // Return   0 si this == otro
        // Return > 0 si this > otro
        return peso - otro.peso; /* Un nodo es menor que otro si tiene menos peso */
    }
    public boolean equals(Object otro){
        if (otro instanceof Item){
            Item ese = (Item)otro;
            return destino == ese.destino && peso == ese.peso;
        }
        return false;
    }
```

```
    public String toString(){
        return "peso = " + peso + ", destino = " + destino;
    }
}

class Ejemplo {
    public static void main(String[] args) {
        PriorityQueue<Item> q = new PriorityQueue<Item>();
        q.add(new Item(12, 0));
        q.add(new Item(4, 1876));
        q.add(new Item(13, 0));
        q.add(new Item(8, 0));
        q.add(new Item(7, 3));
        while (!q.isEmpty()){
            System.out.println(q.poll());
        }

    }
}
```

La salida de este programa es:

```
peso = 0, destino = 12
peso = 0, destino = 8
peso = 0, destino = 13
peso = 3, destino = 7
peso = 1876, destino = 4
```

Vemos que la función de comparación que definimos no tiene en cuenta `destino`, por eso no desempata cuando dos `Items` tienen el mismo `peso` si no que escoge cualquiera de manera arbitraria.

8. C++

8.1. 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;
}
```

8.2. 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;

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;
}
```