Resumen de algoritmos para torneos de programación

Loperamida Clorhidrato 2mg

16 de octubre de 2010

Índice

1. BIT

1.1. Simple BIT

```
// In this implementation, the tree is represented by a vector<int>.
// Elements are numbered by 0, 1, ..., n-1.
// tree[i] is sum of elements with indexes i&(i+1)..i, inclusive.
// (Note: this is a bit different from what is proposed in Fenwick's artic
// To see why it makes sense, think about the trailing 1's in binary
// representation of indexes.)

// Creates a zero-initialized Fenwick tree for n elements.
vector<int> create(int n) { return vector<int>(n, 0); }

// Returns sum of elements with indexes a..b, inclusive
int query(const vector<int> &tree, int a, int b) {
  if (a == 0) {
    int sum = 0;
    for (; b >= 0; b = (b & (b + 1)) - 1)
      sum += tree[b];
    return sum;
  } else {
    return query(tree, 0, b) - query(tree, 0, a-1);
  }
}

// Increases value of k-th element by inc.
void increase(vector<int> &tree, int k, int inc) {
  for (; k < (int)tree.size(); k |= k + 1)
```

```
    tree[k] += inc;
}

int main(){
  vector<int> f = create(100000);
  for(int i = 0 ; i < 100000; ++i)
    increase(f, i, i);
  for(int i = 0; i < 100; ++i){
    //In this case it will return the sum = (i)(i+1)/2
    D(query(f,0,i));
  }
  return 0;
}
```

.....

2. DP

2.1. Kadane

```
const int MAXN = 22;

int cube[MAXN][MAXN][MAXN];
int mat[MAXN][MAXN];
int arr[MAXN];

int n;

// Returns the maximum sum inside an array
// The sum best = Sum i in [from, to]
int kadane(){
  int best=1<<31,current=0,from=0,to=0,aa=0;
  for(int i=0;i<MAXN;++i){
    current += arr[i];
    if ( current > best ){ best=current; from=aa; to=i;}
    if ( current < 0 ){ current = 0; aa = i+1;}
  }
  return best;
}
// Returns the submatrix with maximum sum
// The sum is inside the matrix (xi,y1) - (x2, y2)
// A is the matrix, N the size
int kadane2D () {
  vector<int>pr(102,0);
  int S = 1<<31, s=0, k,l,x1=0,x2=0,y1=0,y2=0,j,t;
  for(int z=0;z < N;++z){
    pr = vector<int>(MAXN,0);
    for(int x=z;x<N;++x){
      t=0;s = 1<<31;j=k=l=0;
      for(int i=0;i<N;++i) {
        pr[i]=pr[i]+a[x][i]; t=t+pr[i];
        if (t>s){ s = t; k = i; l = j;}
        if(t<0){ t=0; j=i+1;}
      }
      if (s > S) { S = s; x1 = x; y1 = k; x2 = z;  y2 = l;}
    }
  }
  return S;
}

// Easier to implement. Less information
int best2D(){
  int ans = 0;
  for(int i=0; i<n; ++i){
    memset(arr, 0, sizeof arr);
    for (int j=i; j<n; ++j){
      for (int k=0; k<n; ++k) arr[k] += mat[j][k];
      int sum = 0;
      for (int k=0; k<n; ++k){
        sum += arr[k];
        ans = max(ans, sum);
        if (sum < 0) sum = 0;
      }
    }
  }
  return ans;
}


// Cube has the actual input. If all numbers in cube are negative
// the maximum sum is the biggest of the numbers
int kadane3D(){
  int ans = 0;
```

```
  for (int i=0; i<n; ++i){
    memset(mat, 0, sizeof mat);
    for (int j=i; j<n; ++j){
      for (int ii=0; ii<n; ++ii){
        for (int jj=0; jj<n; ++jj){
          mat[ii][jj] += cube[j][ii][jj];
        }
      }
      ans = max(ans, ());
    }
  }
  return ans;
}
```

.....

2.2. LIS

```
#define INF 2<<30-1

int main(){
  int n;
  while(scanf("%d", &n)==1){
    vector<long>S(n);
    vector<long>M(n+1,INF);
    for(int i=0;i<n;++i) scanf("%ld", &S[i]);
    M[0]=0;
    int _m = 0;
    for(int i=0; i<S.size();++i){
      int d = upper_bound(M.begin(),M.begin()+n,S[i]) - M.begin();
      if(S[i]!=M[d-1]){
        M[d] = S[i];
        _m = max(_m,d);
        //parent[S[i]] = M[d-1];
      }
    }
    printf("%d\n",max(1,_m));
  }
  return 0;
}
```

.....

3. Geometría

3.1. Andes

```
// Returns true if pXq is inside aXb
bool cabe(long p, long q, long a, long b){
  long x,y,z,q; if(p<q) swap(p,q); if(a<b) swap(a,b);
  if(p<=a && q<=b) return true;
  if(p==q) return b>=q;
  x = 2*p*q*a; y=p*p-q*q; z=p*p+q*q; w=z-a*a;
  return p>a && 1.0*b*z >= x+y*sqrt(w) - 1e-10;
}

// Centroide (centro de masa) de un polno
// pt[i][0] = pt[i].x  |  pt[i][1] = pt[i].y
// Area will return positive or negative
double area(vector<vector<double> > &pt){
  double r = 0.0; int t = pt.size();
  for(int i = 0, j = 1; i<t; i++, j = j+1 == t? 0 : j+1){
    r+= (pt[i][0] * pt[j][1] - pt[i][1] * pt[j][0]);
  }
  return r/2.0;
}

pair<double, double> centroide(vector<vector<double> > &pt){
  double d = area(pt) * 6.0;
  double p[2];
  p[0] = p[1] = 0.0;
  for(int i = 0, j = 1, t = pt.size(); i<t; i++,
    j = j+1 ==t ? 0 : j+1)
    for(int k = 0; k<2;k++)
    p[k] += (pt[i][k] + pt[j][k]) * \
    (pt[i][0] * pt[j][1] - pt[j][0] * pt[i][1]);
  return pair<double, double>(pt[0]/d, pt[1]/d);
}
```

.....

3.2. Brasileros

```
const int INF = 0x3F3F3F3F;
const int NULO = -1;
```

```
const double EPS = 1e-10;

//If x==y, returns 0
//If x>y, returns 1
//If x<y, returns -1
int cmp(double x,double y=0, double tol=EPS){
  return( x<=y + tol) ? (x + tol < y) ? -1 : 0 : 1;
}

struct point {
double x,y;
point(double x=0, double y=0):x(x),y(y){}
point operator+ (const point &q) {return point (x + q.x, y+q.y);}
point operator- (const point &q){return point (x - q.x, y-q.y);}
point operator* (const double &t){return point(x*t , y*t);}
point operator/ (const double &t){return point(x/t , y/t);}
double operator* (const point &q){return x*q.x+y*q.y;} //Dot Pr
double operator% (const point &q){return x*q.y-y*q.x;} //Cross Pr

  int cmp(point q) const {
    if(int t= ::cmp(x,q.x)) return t;
    return ::cmp(y,q.y);
  }

  bool operator ==(const point &q) const { return cmp(q) == 0; }
  bool operator != (const point &q) const { return cmp(q) != 0; }
  bool operator < (const point &q) const { return cmp(q) < 0; }

  friend ostream& operator <<(ostream& o, point p){
    return o<<"("<<p.x<<", "<<p.y<<")";
  }

    //Distancia entre dos puntos
  double Distance(const point &o) const{
    double d1 = x-o.x, d2=y-o.y;
    return sqrt(d1*d1+d2*d2);
  }

  static point pivot;
};
```

```
typedef vector<point> polygon;
typedef pair<point, double> circle;

point point::pivot(0,0);

double abs(point p) { return hypot(p.x,p.y); }
double arg(point p) { return atan2(p.y,p.x); }

/**
* Calcula el signo de giro entre dos vectores definidos
* por (p-r) y (q-r)
**/
inline int turn(point &p, point &q, point &r){
  return ::cmp((p-r)%(q-r));
}

int ccw (point p, point q, point r) {
  return cmp((p-r)%(q-r));
}

double angle(point p, point q, point r) {
  point u = p-q, v=r-q;
  return atan2(u%v, u*v);
}

//Decide si q esta sobre el segmento PR
bool between(point p, point q, point r){
  return ccw(p,q,r)==0 && cmp((p-q)*(r-q))<=0;
}

//Decide si dos segmentos PQ y RS tienen puntos en comun
bool seg_intersect(point p, point q, point r, point s){
  point A = q-p, B=s-r, C=r-p, D=s-q;
  int a = cmp(A%C) + 2 * cmp(A%D);
  int b = cmp(B%C) + 2 * cmp(B%D);
  if(a==3 || a== -3 || b == 3 || b == -3) return false;
  if(a ||b || p == r || p == s || q == r || q == s) return true;
  int t = (p<r) + (p<s) + (q<r) + (q<s);
  return t!=0 && t!=4;
}
```

```
//Calcula la distancia de un punto R al segmento PQ
double seg_distance(point p, point q, point r){
  point A = r - q, B = r - p, C = q - p;
  double a = A * A, b = B * B, c = C * C;
  if (cmp(b,a+c)>=0) return  sqrt(a);
  else if (cmp(a, b+c) >=0) return sqrt(b);
  return fabs(A % B) / sqrt (c);
}

//Califica un punto P con ralacion al poligono T
//Retorna 0,                -1,                          1
//En el exterior, en la frontera, en el interior respectivamente
int in_poly(point p, polygon &T){
  double a = 0; int N = T.size();
  for(int i=0; i < N; ++i) {
    if (between(T[i], p, T[(i+1) % N])) return -1;
    a+=angle(T[i],p,T[(i+1) % N]);
  }
  return cmp(a) != 0;
}

//Comparacion radial
bool radial_lt(point p, point q){
  point P = p-point::pivot, Q = q - point::pivot;
  double R = P % Q;
  if(::cmp(R)) return R > 0;
  return ::cmp(P*P, Q*Q) < 0;
}

//Destruye la lista de puntos T
polygon convex_hull(vector<point> &T){
  int j=0, k, n=T.size(); polygon U(n);

  point::pivot = *min_element(all(T));
  sort(all(T), radial_lt);
  for(k = n-2; k>=0 && ccw(T[0], T[n-1], T[k])==0; k--) ;
  reverse((k+1) + all(T));

  for(int i=0; i< n; ++i){
  //cambia >= por > para mantener los puntos colineales
    while (j > 1 && ccw(U[j-1], U[j-2], T[i]) >= 0) j--;
    U[j++] = T[i];
  }
  U.erase(j + all(U)); // U.erase(j+U.begin(), U.end() )
  return U;
}

//Returns the cuadrant where the point is
//Retorna 5 si el punto es (0, 0)
int quadrant(const point &p) {
  if(::cmp(p.x)==0 && ::cmp(p.y)==0) return 5;
  if(::cmp(p.y) == 1) {
    if(::cmp(p.x)==1) return 1;
    return 2;
  }
  if(::cmp(p.y)==0){
    if(::cmp(p.x)==1 || ::cmp(p.x)==0) return 1;
    return 3;
  }
  if(::cmp(p.x)==-1) return 3;
  return 4;
}

//Comparator to sort the points by their angle
bool PolarCom(point &p, point &q) {
  point P = P - point::pivot, Q = q - point::pivot;
  int q1 = quadrant(P), q2 = quadrant(Q);
  if(q1!=q2) return q1<q2;
  double R = P%Q;
  if(::cmp(R)) return R>0;
  return ::cmp(P*P, Q*Q) < 0;
}

//Calcula el area de un poligono T
double poly_area(polygon &T){
  double s = 0; int n = T.size();
  for(int i=0; i < n; ++i)
    s+= T[i] % T[(i+1)%n];
  return  fabs(s)/2.0;
}

//Encuentra el punto de interseccion de dos rectas PQ y RS
```

```
point line_intersect(point p, point q, point r, point s){
  point a = q - p, b = s - r, c = point(p % q, r % s);
  return
    point(point(a.x , b.x) % c, point(a.y , b.y) % c) / (a % b);
}

//Encuentra el menor circulo que contiene todos los puntos dados
bool in_circle(circle C, point p){
  return cmp(abs(p - C.first), C.second) <= 0;
}

point circumcenter(point p, point q, point r) {
  point a = p - r;
  point b = q - r;
  point c = point(a * (p+r) / 2, b * (q+r) / 2);
  return
    point(c % point(a.y, b.y), point(a.x, b.x) % c) / (a % b);
}

circle spanning_circle(vector<point> &T) {
  int n = T.size();
  random_shuffle(all(T));
  circle C(point(), -INFINITY);
  for(int i=0; i < n; i++) if (!in_circle(C, T[i])) {
    C = circle(T[i], 0);
    for(int j = 0; j < i; j++) if (!in_circle(C, T[j])) {
      C = circle((T[i]+T[j]) / 2, abs(T[i] - T[j])/2);
      for(int k = 0; k < j; k++) if (!in_circle(C, T[k])) {
        point o = circumcenter(T[i], T[j], T[k]);
        C = circle(o, abs(o - T[k]));
      }
    }
  }
  return C;
}
//Fin del spanning_circle
```

```
//Saca la interseccion de dos poligonos convexos P y Q.
//Tanto P como Q deben estar orientados positivamente

polygon poly_intersect(polygon &P, polygon &Q) {
  int m = Q.size(), n = P.size();
  int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;
  polygon R;
  while( (aa < n || ba < m) && aa < 2*n && ba < 2*m) {
    point p1 = P[a], p2 = P[(a+1) % n];
    point q1 = Q[b], q2 = Q[(b+1) % m];
    point A = p2 - p1, B = q2 - q1;
    int cross = cmp(A%B);
    int ha = ccw(p2, q2, p1);
    int hb=ccw(q2, p2, q1);
    if (cross == 0 && ccw(p1, q1, p2) == 0 && cmp(A*B) < 0) {
      if(between(p1, q1, p2)) R.pb(q1);
      if(between(p1, q2, p2)) R.pb(q2);
      if(between(q1, p1, q2)) R.pb(p1);
      if(between(q1, p2, q2)) R.pb(p2);
      if (R.size() < 2) return polygon ();
      inflag = 1; break;
    } else if(cross != 0 && seg_intersect(p1, p2, q1, q2)) {
      if (inflag==0) aa = ba = 0;
      R.pb(line_intersect(p1, p2, q1, q2));
      inflag = (hb > 0) ? 1 : -1;
    }
    if (cross == 0 && hb < 0 && ha < 0) return R;
    bool t = cross == 0 && hb ==0 && ha == 0;
    if (t  ? (inflag == 1):(cross >=0)?(ha <= 0):(hb > 0) ){
      if(inflag == -1) R.pb(q2);
      ba++; b++; b%=m;
    }else {
      if(inflag == 1) R.pb(p2);
      aa++;a++;a%=n;
    }
  }
  if (inflag == 0) {
    if (in_poly(P[0], Q)) return P;
    if (in_poly(Q[0], P)) return Q;
  }
  R.erase(unique(all(R)) , R.end());
```

```
  if (R.size() > 1 && R.front() == R.back()) R.pop_back();
  return R;
}
```

.....

3.3. Java

```
import java.awt.geom.*;

public class geojava {
  private static final double EPS = 1e-10;

  private static int cmp(double x, double y) {
    return (x<= y + EPS) ? (x + EPS < y) ? -1 : 0 : 1;
  }

  //Point Class
  private static class Point implements Comparable {
    public double x, y;

    public Point(double x, double y){
      this.x = x;
      this.y = y;
    }

    public Point(){
      this.x = this.y = 0.0;
    }

    public double dotProduct(Point o){
      return this.x * o.x + this.y * o.y;
    }

    public double crossProduct(Point o){
      return this.x*o.y - this.y*o.x;
    }

    public Point add(Point o){
      return new Point(this.x + o.x, this.y + o.y);
    }

    public Point substract(Point o){
      return new Point(this.x - o.x, this.y - o.y);
    }

    public Point multiply (double m){
      return new Point(this.x * m, this.y * m);
    }

    public Point divide (double m){
      return new Point (this.x/m, this.y/m);
    }

    //@Override
    public int compareTo(Object k){
      if(k instanceof Point){
        Point o = (Point)k;
        if (this.x < o.x) return -1;
        if (this.x > o.x) return 1;
        if (this.y < o.y) return -1;
        if (this.y > o.y) return 1;
        return 0;
      }
      return -5; //No es un punto!
    }

    //Euclidean distance Between Two Points

    double distance(Point o){
      double d1 = x-o.x, d2 = y-o.y;
      return Math.sqrt(d1*d1+d2*d2);
    }
  }//End of point class

  private static double angle(Point p, Point q,
    Point r){
    Point u = p.substract(r), v = q.substract(r);
    return Math.atan2(u.crossProduct(v), u.dotProduct(v));
  }

  private static int turn (Point p, Point q, Point r){
    return cmp((p.substract(r)).crossProduct(q.substract(r)),0.0);
```

```
}

private static boolean between(Point p, Point q,
  Point r){
  return turn(p,r,q)==0 &&
    cmp((p.substract(r)).dotProduct(q.substract(r)),0.0)<=0;
}

private static int inPolygon(Point p, Point[] polygon,
  int polygonSize){
  double a = 0; int N = polygonSize;
  for(int i=0;i < N; ++i){
    if(between(polygon[i], polygon[(i+1)%N], p))
      return -1;
    a+=angle(polygon[i], polygon[(i+1)%N], p);
  }
  return (cmp(a,0.0)==0)? 0 : 1;
}

private static Point GetIntersection(Line2D.Double l1,
  Line2D.Double l2){
  double A1 = l1.y2 - l1.y1;
  double B1 = l1.x1 - l1.x2;
  double C1 = A1 * l1.x1 + B1*l1.y1;

  double A2 = l2.y2 - l2.y1;
  double B2 = l2.x1 - l2.x2;
  double C2 = A2 * l2.x1 + B2*l2.y1;

  double det = A1*B2 - A2*B1;

  if(det==0){
    //Lines are parallel. Check if they are on the same line
    double m1 = A1/B1;
    double m2 = A2/B2;
    //Check whether their slopes are the same or not,
    //or if they are vertical
    if(cmp(m1,m2)==0 || (B1==0 && B2==0)) {
      //Cuidado con la implementación aqui!
      if((l1.x2==l2.x1 && l1.y2 == l2.y1) ||
        (l1.x2==l2.x2 && l1.y2 == l2.y2))
          return new Point(l1.x2, l1.y2);

        if((l1.x2==l2.x1 && l1.y2==l2.y1) ||
          (l1.x2==l2.x2 && l1.y2==l2.y2))
          return new Point(l1.x2, l1.y2);
      }
      return null;
    }
    double x = (B2*C1 - B1*C2)/det;
    double y = (A1*C2 - A2*C1)/det;
    return new Point(x,y);
  }
}
```

.....

4. Grafos

4.1. Topological Sort (BFS)

```
/** Creates an edge from u to v. This represents that task
u comes before task v **/
void add_edge(int u, int v){
  g[u].push_back(v);
  d[v]++;
}

int d[MAXN]; //d[i] is the number dependencies

vector<int> top_sort(graph &g, int *d){
  vector<int> order;
  int n = g.size();
  queue<int> q;
  set <int> inside;
  for(int i=0; i<n; ++i)
  if(d[i]==0){
    q.push(i);
    inside.insert(i);
    order.push_back(i);
  }
  while(q.size()){
    int actual = q.front();
```

```
    q.pop();
    inside.erase(actual);
    for(int i=0;i<g[actual].size();++i){
      int next = g[actual][i];
      d[next]--;
      if(d[next]==0){
        if(inside.count(next)) {
          return vector<int>(1,INT_MAX); // There's a cycle
        }
        q.push(next); inside.insert(next); order.push_back(next);
      }
    }
  }
  ir(order.size()!=n)return vector<int>(1,INT_MAX);
  return order;
}
```

.....

4.2. Longest Path in DAG

```
struct node {
  int weight;
  int index;
};
bool visited[MAXNODES];
bool can_go(node n);//retorna true si se puede visitar ese nodo
node best;
int dfs(node root)
{
  memset(visited, false, sizeof visited);
  stack<node> s;
  s.push(root);
  int ans = 0;
  while(s.size())
  {
    node actual = s.top();
    visited[actual.index] = true;
    s.pop();
    int weight = actual.weight;
    if(weight > ans)
    {
      ans = weight;
      best = actual;
    }
      //for para cada vecino)
    if(can_go(vecino))
      s.push(vecino);
  }
  return ans;
}
int max_path_dag()
{
  node root;
  root.weight = 0;
  root.index = 0; // cualquier node del dag funciona
  int t = dfs(root);
  best.weight = 0;
  int ans = dfs(best);
  return ans;
}
```

.....

5. LCA y RMQ

5.1. LCA y RMQ

```
// RMQ will find the POSITION of the
// smallest integer inside an array A
// between A[i] and A[j] (inclusive)

// <f(x), g(x)>
// f is the construction
// g is the query

// First implementation. <O(n^2), O(1)>
const int MAXN = 11;
int M[MAXN][MAXN], A[MAXN], N = 10;
// DP approach: M[i][j] = position of the RQM from i to j
// if A[ M[i][j-1] ] < A[j] then the RQM(i,j) is the same
// else A[j] is smaller than the last smaller so RQM(i,j) = j
void preprocess1(){
  int i,j;
```

```
  for(i = 0; i<N; ++i) M[i][i] = i;
  for(i = 0; i<N; ++i)
    for(j = i+1; j<N; ++j)
      // Leave the <= if you want the leftmost position
    if(A[M[i][j-1]] <= A[j])
    M[i][j] = M[i][j] = M[i][j-1];
  else
    M[i][j] = j;
}
```

.....

6. String Matching

6.1. KMP

```
// Computes the jumping function
vector<int> kmp_table(string &P){
  int i = 0, j = -1;
  int m = P.size();
  vector<int> f(m+1);
  f[0] = -1;
  while( i < m ){
    while(j>=0 and P[i] != P[j]) j = f[j];
    f[++i] = ++j;
  }
  return f;
}

void kmp(string &T, string &P){
  vector<int> pi = kmp_table(P);
  int n = T.size(), m = P.size();
  int q = 0;
  for(int i = 0; i<n; ){
    while(q > -1 and P[q] != T[i])  q = pi[q];
    i++; q++;
    if(q >= m ){
      any = true;
      printf("%d\n", i - q);
      q = pi[q];
    }
  }
}

int main(){
  int T, C=1;
  string s,t;
  while(scanf("%d", &T)){
    if(T==0) break;
    printf("Test case #%d\n", C++);
    cin >> s;
    vector<int>pi = kmp_table(s);
    for(int i=1;i<=T;++i)
      if(pi[i] > 0)
        if(i % (i-pi[i]) == 0)
          printf("%d %d\n", i, i/(i-pi[i]));
    puts("");
  }
  return 0;
}
```

.....

6.2. Suffix Arrays O(n log n)

```
const int N = 1000001;
// Begins Suffix Arrays implementation
// O(n log n) - Manber and Myers algorithm

//Usage:
// Fill str with the characters of the string.
// Call SuffixSort(n) where n = str.size()
// That's it!

//Output:
// pos = The suffix array. It has n suffixes
//       Contains the suffixes sorted in lexicographical order.
//       Each suffix is represented as a single integer
//       (the position of str where it starts).
// rank = The inverse of the suffix array.
//        rank[i] = the index of the suffix str[i..n)
//           in the pos array.
//        (In other words, pos[i] = k <==> rank[k] = i)
//        With this array, you can compare two suffixes in O(1):
```

```
//        Suffix str[i..n) is smaller
//        than str[j..n) if and only if rank[i] < rank[j]

int str[N]; //input
int rank[N], pos[N]; //output
int cnt[N], next[N]; //internal
bool bh[N], b2h[N];

bool smaller_first_char(int a, int b){
  return str[a] < str[b];
}

void SuffixSort(int n){
  for (int i=0; i<n; ++i){
    pos[i] = i;
  }
  sort(pos, pos + n, smaller_first_char);
  for (int i=0; i<n; ++i){
    bh[i] = i == 0 || str[pos[i]] != str[pos[i-1]];
    b2h[i] = false;
  }

  for (int h = 1; h < n; h <<= 1){
    int buckets = 0;
    for (int i=0, j; i < n; i = j){
      j = i + 1;
      while (j < n && !bh[j]) j++;
      next[i] = j;
      buckets++;
    }
    if (buckets == n) break;
    for (int i = 0; i < n; i = next[i]){
      cnt[i] = 0;
      for (int j = i; j < next[i]; ++j){
        rank[pos[j]] = i;
      }
    }

    cnt[rank[n - h]]++;
    b2h[rank[n - h]] = true;
    for (int i = 0; i < n; i = next[i]){
      for (int j = i; j < next[i]; ++j){
        int s = pos[j] - h;
        if (s >= 0){
          int head = rank[s];
          rank[s] = head + cnt[head]++;
          b2h[rank[s]] = true;
        }
      }
      for (int j = i; j < next[i]; ++j){
        int s = pos[j] - h;
        if (s >= 0 && b2h[rank[s]]){
          for (int k = rank[s]+1; !bh[k] && b2h[k]; k++)
            b2h[k] = false;
        }
      }
    }
    for (int i=0; i<n; ++i){
      pos[rank[i]] = i;
      bh[i] |= b2h[i];
    }
  }
  for (int i=0; i<n; ++i){
    rank[pos[i]] = i;
  }
}
// End of suffix array algorithm

// Algorithm GetHeight
// input: A text A and its suffix array Pos
//    1 for i:=1 to n do
//    2     Rank[Pos[i]] := i
//    3 od
//    4 h:=0
//    5 for i:=1 to n do
//    6     if Rank[i] > 1 then
//    7        k := Pos[Rank[i]-1]
//    8        while A[i+h] = A[j+h] do
//    9           h := h+1
//    10       od
//    11       Height[Rank[i]] := h
//    12       if h > 0 then h := h-1 fi
```

```
//    13    fi
//    14 od
int height[N];
// height[i] =
//  length of the LCP of suffix pos[i] and pos[i-1]
// height[0] = 0
void getHeight(int n){
  for (int i=0; i<n; ++i) rank[pos[i]] = i;
  height[0] = 0;
  for (int i=0, h=0; i<n; ++i){
    if (rank[i] > 0){
      int j = pos[rank[i]-1];
      while (i + h < n && j + h < n && str[i+h] == str[j+h]) h++;
      height[rank[i]] = h;
      if (h > 0) h--;
    }
  }
}

// Gets the longest common prefix from Sx and Sy
//    in a string of lenght n
// lcp(x,y) = min(lcp(x,x+1), lcp(x+1, x+2), ... , lcp(y-1, y))
// Runs in O(|x-y|)
int lcp(int x, int y, int n){
  if(x > y) return lcp(y,x,n);
  if(x == y) return n-pos[x];
  int lc = n+1;
  for(int i = x+1; i<=y; ++i)
    if (height[i] != 0) lc = min(lc, height[i]);
    else return 0;
  return lc;
}

string s;

void print_suffix_array(){
  puts("Suffix Array");
  int n = s.size();
  string tmp;
  for(int i=0;i<n;++i){
    tmp = s.substr(pos[i]);
    printf("pos[%d] = %2d \t suffix = %s \t height[%d] = %d\n",
      i,pos[i],tmp.c_str(), i, height[i]);
  }
}

// You need a string W that represents the pattern
// Not really tested. Pseudo-tested
int match_prefix(int n){
  string W; // Fill this outside
  if(W[0] < s[pos[0]]) return -1; // Is not here!
  if(W[0] > s[pos[n-1]]) return -1; // Not here too!
  if(W == s.substr(pos[0])) return pos[0];
  // Binary search for the W pattern
  int l = 0, r = n-1, m;
  while(r-l > 1){
    m = (l+r)/2;
    if(W >= s.substr(pos[m]))
      l = m;
    else
      r = m;
  }
  // r is the i-sth smallest suffix
  // that means that pos[r] is the actual index
  if(W != s.substr(pos[r], W.size())) return -1; // not here at all!
  printf("Matched at %d\n", r);
  return pos[r];
}

// Get the biggest repeated substring and how many times it appears
// First, get the biggest repeated string (biggest height[i])
// Then count it's repetitions
// GATTACA
void get_the_biggest_repeated_substring(){
  int n = s.size();
  for(int i=0;i<n;++i) str[i] = s[i];
  SuffixSort(n);
  getHeight(n);
  int longest = 0, position = -1;
  for(int i=1;i<n;++i){
    if(longest < height[i]) { longest = height[i]; position = i - 1;}
  }
```

```
  int cnt = 1;
  for(int i=position+1;i<n;++i){
    if(height[i] >= longest) cnt++;
    else break;
  }
  if(longest != 0)
    cout << s.substr(pos[position], longest);
    cout << " " << cnt << endl;
  else
    puts("No repetitions found!");
}

// If you have the i-th smaller suffix, Si,
//    it's length will be |Si| = n - pos[i]
// Now, height[i] stores the number of
//     common letters between Si and Si
//     (s.substr(pos[i]) and s.substr(pos[i-1]))
// so, you have |Si| - height[i] different strings
//    from these two suffixes => n - pos[i] - height[i]
void number_of_different_substrings(){
  int n = s.size();
  // Uncomment if reading s and not str
  // for (int i=0; i<n; ++i) str[i] = s[i];
  // Build suffix array and height array
  int ans = 0;
  for(int i=0;i<n;++i)
    ans += n-pos[i]-height[i];
  cout << ans << endl;
}

// Number of substrings that appear at least twice in the text.
// The trick is that all 'spare' substrings that can give us
// Lcp(i - 1, i) can be obtained by Lcp(i - 2, i - 1)
// due to the ordered nature of our array.
// And the overall answer is
// Lcp(0, 1) +
//     Sum(max[0, Lcp(i, i - 1) - Lcp(i - 2, i - 1)])
//     for 2 <= i < n
// File Recover
void number_of_repeated_substrings(){
  int n = s.size();
  if(n==1){ cout << 0 << endl;  return; }
  //Uncomment if reading s and not str
  //for (int i=0; i<n; ++i) str[i] = s[i];
  //build suffix array and height array
  int cnt = height[1];
  for(int i=2;i<n;++i){
    cnt += max(0, height[i] - height[i-1]);
  }
  cout << cnt << endl;
}

// Given a string s and an int m, find the size
// of the biggest substring repeated m times (find the rightmost pos)
// if a string is repeated m+1 times, then it's repeated m times too

// The answer is the maximum, over i, of the longest common prefix
// between suffix i+m-1 in the sorted array.
void repeated_m_times(int m){
  int n = strlen(str);
  SuffixSort(n);
  getHeight(n);
  int length = 0, position = -1, t;
  for(int i=0;i<=n-m;++i){
    if((t=lcp(i,i+m-1,n)) > length){
      length = t;
      position = pos[i];
    }else if(t == length) { position = max(position, pos[i]); }
  }

  // Here you'll get the rightmost position
  // (that means, the last time the substring appears)
  for(int i = 0; i < n; ){
    if(pos[i] + length > n) {++i; continue;}
    int ps = 0, j = i+1;
    while(j<n && height[j] >= length){
      ps = max(ps,pos[j]);
      j++;
    }
    if(j - i >= m) position = max(position, ps);
    i = j;
  }
```

```
  if(length != 0)
    printf("%d %d\n", length, position);
  else
    puts("none");
}


// Reads a string of lenght k. Then just double it (s = s+s)
// and find the suffix arrays.
// The answer is the smallest i for which s.size() - pos[i] >= k
// If you want the first appearence (and not the string)
//   you'll need the second cycle
void smallest_rotation(){
  scanf("%d %s", &k, &sss);
  s = string(sss) + string(sss);
  int n = s.size();
  for (int i=0; i<n; ++i) str[i] = s[i];
  SuffixSort(n);
  getHeight(n);
  int best = 0;
  for(int i=0;i<n;++i){
    if(n - pos[i] >= k){
      //Find the first appearence of the string
      while( n - pos[i] >= k){
        if(pos[i] < pos[best] && pos[i]!=0) best = i;
        i++;
      }
      break;
    }
  }
  if(pos[best] == k) puts("0");
  else printf("%d\n", pos[best]);
}
```

.....

7. Teoría de Números

7.1. Big Mod

```
long bigmod(long b, long p, long m){
  if (p == 0) return 1;
  // square(x) = x * x
  else if (p%2 == 0) return square(bigmod (b,p/2,m)) % m;
  else return ((b % m) * bigmod( b,p-1,m)) % m;
}
```

.....

7.2. GCD Extendido

```
int egcd(int a, int b, int &x, int &y){
  x = 0, y = 1;
  int lastx = 1, lasty = 0;
  int quot, temp;
  while(b != 0){
    quot = a/b;
    temp = b;
    b = a%b;
    a = temp;
    temp = x;
    x = lastx - quot*temp;
    lastx = temp;
    temp = y;
    y = lasty - quot*temp;
    lasty = temp;
  }
  x = lastx, y = lasty;
  return a;
}
```

.....

7.3. Fibonacci O(log n)

```
typedef unsigned long long uulong;
uulong fib(int n){
  uulong i=1,j=0,k=0,h=1,t=0;
  while(n>0){
    if (n%2==1){ t=j*h;  j=i*h + j*k + t;  i = i*k + t; }
    t = h*h; h = 2*k*h + t; k = k*k + t;
```

```
    n = floor(n/2);
  }
  return j;
}
```

.....

7.4. Función Phi de Euler

```
//Generate primes with Erathostenes
int fi(int n) {
  if(primes[n]) return n-1;
  int result = n;
  for(int i=2;i*i <= n;i++) {
    if (n % i == 0) result -= result / i;
    while (n % i == 0) n /= i;
  }
  if (n > 1) result -= result / n;
  return result;
}
```

.....

7.5. Descomposición en Factores Primos

```
typedef map<int,int> prime_map;
void sqeeze(prime_map &M, int &n, int p) {
  for ( ; n%p ==0 ;n/=p) M[p]++;
}
prime_map factor(int n){
  prime_map M;
  if(n<0) return factor(-n);
  if(n<2) return M;
  sqeeze(M, n, 2); sqeeze(M, n, 3);
  int maxP = sqrt(n) + 2;
  for(int p=5; p< maxP; p+=6){
    sqeeze(M , n, p); sqeeze(M, n, p+2);
  }
  if(n>1) M[n]++;
  return M;
}
```

.....

8. Segment Trees

```
/*
como cuestion util vale la pena mencionar que todo
arbol binario se puede representar en un arreglo de
la siguiente manera:
  la raiz es el elemento en posicion 0
  el hijo izquierdo de un nodo en posicion x es x * 2 + 1
  el hijo derecho de un nodo en posicion x es x * 2 + 2
de esta manera no necesitamos usar mas que un arreglo para
almacenar nuestro segment tree, que es un arbol binario

este segment tree tiene dos operaciones
update(pos, by): actualiza el arbol de manera que el elemento
  de la lista de numeros (no del arbol) en posicio pos
  aumente en un valor igual a by
sum(from, to): determina la suma de los elementos en la
  lista de numeros en el intervalo [from, to]
*/
//numero maximo de elementos en la lista de numeros
#define N 200000

int tree[N * 4];
int n; //numero de elementos en el input

//actualiza en el arbol un valor de la lista de numeros
void update(int pos, int by){
int node = 0, left = 0, right = n - 1; //intervalo inicial [0, n -1]

while(left != right)
//mientras tengamos que dividir el intervalo actual
  {
    tree[node] += by;
    int mid = (left + right) / 2;
    if(pos <= mid){
node = node * 2 + 1; //hijo de la izquierda [left, mid]
right = mid;
}else{
node = node * 2 + 2; //hijo de la derecha [mid + 1, right]
left = mid + 1;
}
}
```

```
tree[node] += by;
}

/*
determina la suma de los elementos del intervalo [from, to]
la funcion es tal que node representa el intervalo [left, right]
y [from, to] siempre es subconjunto de [left, right]
*/
int sum(int from, int to, int node = 0,
        int left = 0, int right = n - 1){
if(from == left && to == right)
// si el segmento [left, right] es parte de lo que queremos sumar
return tree[node];

int mid = (left + right) / 2;
int res = 0;

if(from <= mid) //si necesitamos ir por la izquierda
res += sum(from, min(to, mid), node * 2 + 1, left, mid);

if(to > mid) //si necesitamos ir por la derecha
res += sum(max(mid + 1, from), to, node * 2 + 2, mid + 1, right);

return res;
}

//inicializa todo el arbol con cero
int init(){
for(int i = 0; i < 4 * n; i++) tree[i] = 0;
}
```

.....