```cpp
//############ Camino Euleriano(Dirigido) ########################
#define MAXV 5000
#define MAXE 200000
struct DirectedEulerGraph {
  int V,ne,last[MAXV],to[MAXE],next[MAXE],cur[MAXV];
  int in[MAXV],out[MAXV];
  int start,end;
  vector<int> path;
  DirectedEulerGraph(){}
  void clear(int V_){
    V = V_; ne = 0;
    memset(last,-1,sizeof last);
    memset(in,0,sizeof in);
    memset(out,0,sizeof out);
  }
  void add_edge(int u, int v){
    to[ne] = v; next[ne] = last[u]; last[u] = ne++;
    ++out[u]; ++in[v];
    start = u;
  }
  bool check(){
    int cont = 0,aux = start;
    start = end = -1;

    for(int i = 0;i < V;++i){
      if(in[i] == out[i]) ++cont;
      else if(out[i] == in[i] + 1) start = i;
      else if(in[i] == out[i] + 1) end = i;
      else return false;
    }
    if(cont == V){
      start = end = aux;
      return true;
    }
    return (cont == V - 2 && start != -1 && end != -1);
  }
  bool build(){
    stack<int> S;
    S.push(start);
    memcpy(cur,last,sizeof last);
    path.clear();
    while(!S.empty()){
      int u = S.top();

      if(cur[u] == -1){
        path.push_back(u);
        S.pop();
      }else{
        int v = to[ cur[u] ];
        cur[u] = next[ cur[u] ];
        S.push(v);
      }
    }
    reverse(path.begin(),path.end());
    return path.size() == ne + 1;
  }
};

//############ Camino Euleriano(Dirigido) ########################
You should keep an array root[q] which gives you the index of the
    interval of the root ( [0, n ) ) after performing each query and a
    number i r = 0 which is its index in the initial segment tree (
    ans of course, an array s[MAXNODES] which is the sum of elements in
    that node). Also you should have a NEXT_FREE_INDEX = 1 which is
    always the next free index for a node.
First of all, you need to build the initial segment tree :
(In these codes, all arrays and queries are 0-based)

void build(int id = ir,int l = 0,int r = n){
  if(r - l < 2){
    s[id] = a[l];
    return ;
  }
  int mid = (l+r)/2;
  L[id] = NEXT_FREE_INDEX ++;
  R[id] = NEXT_FREE_INDEX ++;
  build(L[id], l, mid);
  build(R[id], mid, r);
  s[id] = s[L[id]] + s[R[id]];
}

(So, we should call build() )

Update function : (its return value, is the index of the interval in the
     new version of segment tree and id is the index of old one)
```

```
int upd(int p, int v,int id,int l = 0,int r = n){
  int ID =  NEXT_FREE_INDEX ++; // index of the node in new version of segment tree
  if(r - l < 2){
    s[ID] = (a[p] += v);
    return ID;
  }
  int mid = (l+r)/2;
  L[ID] = L[id], R[ID] = R[id]; // in case of not updating the interval of left child or right child
  if(p < mid)
    L[ID] = upd(p, v, L[ID], l, mid);
  else
    R[ID] = upd(p, v, R[ID], mid, r);
  return ID;
}

(For the first query (with index 0) we should run root[0]   = upd (p, v,    ir) and for the rest of them, for  j  - th  query se should run root[j]   = upd (p,    v,    root[ j  - 1 ]) )

Function for ask queries :

int sum(int x,int y,int id,int l = 0,int r = n){
  if(x >= r or l >= y)  return 0;
  if(x <= l && r <= y)  return s[id];
  int mid = (l+r)/2;
  return sum(x, y, L[id], l, mid) +
         sum(x, y, R[id], mid, r);
}

(So, we should print the value of sum(x,  y  ,  root  [i]) )

################## UTIL PARA SEG TREE ####################
merge(v[2 * id].begin(), v[2 * id].end(), v[2 * id + 1].begin(), v[2 * id + 1].end(), back_inserter(v[id]));

################## SUFFIX ARRAY ##########################
#define MAXN 1000005
int n,t;  //n es el tama o de la cadena
int p[MAXN],r[MAXN],h[MAXN];
//p es el inverso del suffix array, no usa indices del suffix array ordenado
//r es el suffix array, el primer elmento siempre sera el indice de $
//h el el tama o del lcp entre el i-esimo y el i+1-esimo elemento de suffix array ordenado
string s;
void fix_index(int *b, int *e) {
   int pkm1, pk, np, i, d, m;
   pkm1 = p[*b + t];
   m = e - b; d = 0;
   np = b - r;
   for(i = 0; i < m; i++) {
      if (((pk = p[*b+t]) != pkm1) && !(np <= pkm1 && pk < np+m)) {
         pkm1 = pk;
         d = i;
      }
      p[*(b++)] = np + d;
   }
}
bool comp(int i, int j) {
   return p[i + t] < p[j + t];
}
void suff_arr() {
   int i, j, bc[256];
   t = 1;
   for(i = 0; i < 256; i++) bc[i] = 0;  //alfabeto
   for(i = 0; i < n; i++) ++bc[int(s[i])]; //counting sort inicial del alfabeto
   for(i = 1; i < 256; i++) bc[i] += bc[i - 1];
   for(i = 0; i < n; i++) r[--bc[int(s[i])]] = i;
   for(i = n - 1; i >= 0; i--) p[i] = bc[int(s[i])];
   for(t = 1; t < n; t *= 2) {
      for(i = 0, j = 1; i < n; i = j++) {
         while(j < n && p[r[j]] == p[r[i]]) ++j;
         if (j - i > 1) {
            sort(r + i, r + j, comp);
            fix_index(r + i, r + j);
         }
      }
   }
}
void lcp() {
   int tam = 0, i, j;
   for(i = 0; i < n; i++)if (p[i] > 0) {
      j = r[p[i] - 1];
```

```
      while(s[i + tam] == s[j + tam]) ++tam;
      h[p[i] - 1] = tam;
      if (tam > 0) --tam;
   }
   h[n - 1] = 0;
}
int main(){
   s="margarita$";//OJO NO OLVIDAR EL $
   n=s.size();
   suff_arr();
   lcp();
   for(int i=0;i<n;i++)cout<<r[i]<<"␣";cout<<endl;
   for(int i=0;i<n;i++)cout<<h[i]<<"␣";cout<<endl;
   return 0;
}

##-> Dado un string cuantos substring diferentes hay

cin >> s;
int dev=s.size()*(s.size()+1)/2;
s += '$';  //un caracter menor a todos para que no afecte el resultado
n = s.size();
suff_arr();
lcp();
for(int i=0;i<n;i++)dev-=h[i];
cout<<dev<<endl;

##-> El LCS es el longest common substrings de 2 strings
//leer s y s2
int tam1=strlen(s);
int tam2=strlen(s2);
s[tam1]='$';
for(int i=tam1+1;i<tam1+tam2+1;i++)
 s[i]=s2[i-tam1-1];
n = tam1+tam2+1;
suff_arr();
lcp();
int dev=0;
for(int i=0;i<n;i++)
 if( (r[i]<tam1 && r[i+1]>tam1)|| (r[i+1]<tam1 && r[i]>tam1))
    dev=max(dev,h[i]);
printf("%d\n",dev);

##-> Dado un string devuelve la rotacion menor lexicografica

scanf("%s",&s);
//un caracter menor a todos para que no afecte el resultado
int tam1=strlen(s);
for(int i=tam1;i<2*tam1;i++)s[i]=s[i-tam1];
n=2*tam1;
suff_arr();
char dev[tam1];
for(int i=0;i<n;i++)
 if(r[i]<tam1){
    for(int j=r[i];j<r[i]+tam1;j++)
       dev[j-r[i]]=s[j];
    break;
 }
for(int i=0;i<tam1;i++)
printf("%c",dev[i]);
printf("\n");

##-> Contar los substrings q se repiten al menos una vez.
Analisis: Notamos que si el lcp(i,i+1) con lcp(i+1,i+2)  aumenta quiere
    decir que encontramos h[i+1]-h[i] palabras nuevas (prefijos)   SUM(
    max(h[i+1]-h[i],0))

##-> LCS de n cadenas
int M[100011][20];
int ind[100011];
int ultimo[10];
void rmq(){
    for(int i=0;i<n;i++)
        M[i][0]=h[i];
    for(int j=1;(1<<j)<=n;j++)
        for(int i=0;i+(1<<j)-1<n;i++){
            if(M[i][j-1] < M[i+(1<<(j-1))][j-1]){
                M[i][j]=M[i][j-1];
            }else{
                M[i][j]=M[i+(1<<(j-1))][j-1];
            }
        }
}
int query(int x,int y){
    if(x==y)return h[x];
    int k=log(y-x+1);
```

```
    while((1<<k)< y-x+1)k++;
    k--;
    return min(M[x][k],M[y-(1<<k)+1][k]  );
}
int main(){
    memset(ultimo,-1,sizeof(ultimo));
    int k;
    cin>>k;
    int pos=0;
    s="";
    for(int i=0;i<k;i++){
        string aux;
        cin>>aux;
        s+=aux;
        s+='0'+i;
        for(int j=pos;j+1<s.size();j++)ind[j]=i;
        pos=s.size();
    }
    n=s.size();
    suff_arr();
    lcp();
    rmq();

    int maxi=0;
    for(int i=k;i<s.size();i++){
        ultimo[ind[r[i]]]=i;
        int minimo=100000000;
        for(int j=0;j<k;j++)
            minimo=min(minimo,ultimo[j]);
        if(minimo==-1)continue;
        int minimo2=query(minimo,i-1);
        maxi=max(minimo2,maxi);
    }
    cout<<maxi<<endl;
}

######################## Full Euclides ########################
typedef vector<int> VI;
typedef pair<int, int> PII;
// return a % b (positive value)
int mod(int a, int b) {
  return ((a%b) + b) % b;
}
```

```
// computes gcd(a,b)
int gcd(int a, int b) {
  while (b) { int t = a%b; a = b; b = t; }
  return a;
}
// computes lcm(a,b)
int lcm(int a, int b) {
  return a / gcd(a, b)*b;
}
// (a^b) mod m via successive squaring
int powermod(int a, int b, int m) {
  int ret = 1;
  while (b) {
    if (b & 1) ret = mod(ret*a, m);
    a = mod(a*a, m);
    b >>= 1;
  }
  return ret;
}

// returns g = gcd(a, b); finds x, y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
  int xx = y = 0;
  int yy = x = 1;
  while (b) {
    int q = a / b;
    int t = b; b = a%b; a = t;
    t = xx; xx = x - q*xx; x = t;
    t = yy; yy = y - q*yy; y = t;
  }
  return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
  int x, y;
  VI ret;
  int g = extended_euclid(a, n, x, y);
  if (!(b%g)) {
    x = mod(x*(b / g), n);
    for (int i = 0; i < g; i++)
      ret.push_back(mod(x + i*(n / g), n));
  }
```

```
  return ret;
}

// computes b such that ab = 1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
  int x, y;
  int g = extended_euclid(a, n, x, y);
  if (g > 1) return -1;
  return mod(x, n);
}

// Chinese remainder theorem (special case): find z such that
// z % m1 = r1, z % m2 = r2.  Here, z is unique modulo M = lcm(m1, m2).
// Return (z, M).  On failure, M = -1.
PII chinese_remainder_theorem(int m1, int r1, int m2, int r2) {
  int s, t;
  int g = extended_euclid(m1, m2, s, t);
  if (r1%g != r2%g) return make_pair(0, -1);
  return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2) / g, m1*m2 / g);
}

// Chinese remainder theorem: find z such that
// z % m[i] = r[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (m[i]).  Return (z, M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &m, const VI &r) {
  PII ret = make_pair(r[0], m[0]);
  for (int i = 1; i < m.size(); i++) {
    ret = chinese_remainder_theorem(ret.second, ret.first, m[i], r[i]);
    if (ret.second == -1) break;
  }
  return ret;
}

// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x, int &y) {
  if (!a && !b)
  {
    if (c) return false;
    x = 0; y = 0;
    return true;
  }
  if (!a)
  {
    if (c % b) return false;
    x = 0; y = c / b;
    return true;
  }
  if (!b)
  {
    if (c % a) return false;
    x = c / a; y = 0;
    return true;
  }
  int g = gcd(a, b);
  if (c % g) return false;
  x = c / g * mod_inverse(a / g, b / g);
  y = (c - a*x) / b;
  return true;
}

int main() {
  // expected: 2
  cout << gcd(14, 30) << endl;

  // expected: 2 -2 1
  int x, y;
  int g = extended_euclid(14, 30, x, y);
  cout << g << " " << x << " " << y << endl;

  // expected: 95 451
  VI sols = modular_linear_equation_solver(14, 30, 100);
  for (int i = 0; i < sols.size(); i++) cout << sols[i] << " ";
  cout << endl;

  // expected: 8
  cout << mod_inverse(8, 9) << endl;

  // expected: 23 105
  //           11 12
  PII ret = chinese_remainder_theorem(VI({ 3, 5, 7 }), VI({ 2, 3, 2 }));
  cout << ret.first << " " << ret.second << endl;
  ret = chinese_remainder_theorem(VI({ 4, 6 }), VI({ 3, 5 }));
  cout << ret.first << " " << ret.second << endl;
```

```
415
416   // expected: 5 -15
417   if (!linear_diophantine(7, 2, 5, x, y)) cout << "ERROR" << endl;
418   cout << x << "␣" << y << endl;
419   return 0;
420 }
```