Índice

BGL-UBA - Reference

Cosas a tener en cuenta

Flags de Compilación

```
g++ -std=c++11 -DACMTUYO -O2 -Wshadow -Wextra -D_GLIBCXX_DEBUG -Wall -c "%f"
g++ -std=c++11 -DACMTUYO -O2 -Wshadow -Wall -Wextra -D_GLIBCXX_DEBUG -o"%e" "%f"
time "./%e"
```

1. Estructuras

1.1. Fenwick Tree

```
// TRABAJAR CON UN VECTOR INDEXADO EN 1 EN "fenwick" (DE TAMANO N+1)
void add (tint k, tint x, vector<tint> &fenwick) { // Suma x al indice k
  tint n = fenwick.size() -1;
  while (k <= n) {
    fenwick[k] += x;
    k += (k & -k);
  }
}
// Devuelve la suma en el rango [1..k] (inclusive)
tint sum (tint k, vector<tint> &fenwick) {
  tint s = 0;
  while (k >= 1) {
    s += fenwick[k];
    k -= (k & -k);
  }
  return s;
}
```

1.2. Trie

```
const int MAXN = 60000;

struct TrieNode {
  map<char, int> sig;
  bool final = false;
  void reset() { sig.clear(); final = false; }
};

TrieNode trie[MAXN];
int trie_n = 1;

void resetTrie() {
  trie_n = 1;
  trie[0].reset();
}

void insertar(string st) {
  int pos = 0;
  for(int i=0; i<(int)st.size(); i++) {
    if (trie[pos].sig.find(st[i]) == trie[pos].sig.end()) {
      trie[pos].sig[st[i]] = trie_n;
      trie[trie_n].reset();
      trie_n++;
    }
    pos = trie[pos].sig[st[i]];
  }
  trie[pos].final = true;
}

bool buscar(string st) {
  int pos = 0;
  for(int i=0; i<(int)st.size(); i++) {
    if (trie[pos].sig.find(st[i]) == trie[pos].sig.end())
      return false;
    pos = trie[pos].sig[st[i]];
  }
  return (trie[pos].final == true);
}
```

1.3. Segment Tree

```
// Nodo del segment tree
struct Nodo {
  tint x;
  Nodo (tint xx) { x = xx; }
};
// Operacion del segment tree : tiene que ser ASOCIATIVA
Nodo op (Nodo n1, Nodo n2) {
  return Nodo(n1.x+n2.x);
}
vector<Nodo> buildSegTree (vector<Nodo> &v ) {
  // Completa el tamanho
  tint k = 4, n = v.size();
  while (k < 2*n)
    k <<= 1;
  // Rellena las hojas
  vector<Nodo> seg (k, Nodo(0));
  forn(i,n)
    seg[(k >> 1)+i] = v[i];
  // Completa los padres
  while (k > 0) {
```

```
    seg[(k-1) >> 1] = op(seg[k-1],seg[k-2]);
    k -= 2;
  }
  return seg;
}
// i es el indice de [0,n) en el arreglo original
// Nodo es lo que queremos poner ahora como hoja
void update(tint i, Nodo nodo,vector<Nodo> &seg) {
  tint k = seg.size()/2 + i;
  seg[k] = nodo;
  while (k > 0) {
    seg[k >> 1] = op(seg[k],seg[k^1]);
    k >>= 1;
  }
}
Nodo queryAux(tint k, tint l, tint r, tint i, tint j, vector<Nodo> &seg) {
  if (i <= l && r <= j)
    return seg[k];
  if (r <= i or l >= j)
    return Nodo(0); // Aca va el NEUTRO de la funcion "op"
  Nodo a = queryAux(2*k,l,(l+r) >> 1,i,j,seg);
  Nodo b = queryAux(2*k+1,(l+r) >> 1,r,i,j,seg);
  return op(a,b);
}
// i,j son los indices del arreglo del que se hace la query
// la query se hace en [i,j)
Nodo query(tint i, tint j, vector<Nodo> &seg) {
  return queryAux(1,0,seg.size() >> 1,i,j,seg);
}
// USO:
int main() {
  tint n = 15;
  vector<Nodo> v (n, Nodo(0));
  forn(i,n)
    v[i] = Nodo((3*(i+1))%7 - 9*(i-4)%13);
  vector<Nodo> seg = buildSegTree(v);
  forn(i,n)
    cout << v[i].x << "␣"; // 13 7 7 14 1 -5 -5 2 -4 -4 3 -10 -3 -3 -9
  cout << endl;
  cout << query(3,11,seg).x << "\n"; // Devuelve 2
  update(6,Nodo(0),seg);
  cout << query(3,11,seg).x << "\n"; // Devuelve 7
  return 0;
}
```

1.4. Order Statistic Tree (GCC)

```
#include <iostream>
#include <ext/pb_ds/assoc_container.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;

int main() {
  ordered_set X;
  X.insert(1); X.insert(2); X.insert(4); X.insert(8); X.insert(16);

  // find_by_order(i): iterador al lugar donde se encuentra el i-esimo
  cout << *X.find_by_order(1) << endl; // 2
  cout << *X.find_by_order(2) << endl; // 4
  cout << *X.find_by_order(4) << endl; // 16
  cout << (X.end() == X.find_by_order(6)) << endl; // true

  // order_of_key(x): orden donde iria el elemento x
  cout << X.order_of_key(-5) << endl;  // 0
  cout << X.order_of_key(1) << endl;   // 0
  cout << X.order_of_key(3) << endl;   // 2
  cout << X.order_of_key(4) << endl;   // 2
  cout << X.order_of_key(400) << endl; // 5
  return 0;
}
```

1.5. BWT + Suffix Array

```
const tint maxN = 100000; // largo del string

tint p[maxN], r[maxN], h[maxN],t,n;

bool sacmp (tint a, tint b)
{
  return p[(a+t)%n] < p[(b+t)%n];
}

void bwt(const string &s, tint nn)
{
  n = nn;
  tint bc[256]; // bc histograma de letras de s
  forn(i,256)
  bc[i] = 0;
  forn(i,n)
  ++bc[tint(s[i])];
```

```
  forn(i,255)
  bc[i+1] += bc[i];
  forn(i,n)
  r[--bc[tint(s[i])]] = i;
  forn(i,n)
  p[i] = bc[tint(s[i])];

  tint lnb,nb = 1;
  for (t = 1; t < n; t *= 2)
  {
    lnb = nb;
    nb = 0;
    for (tint i = 0, j = 1; i < n; i = j++)
    {
      // calcula el siguiente bucket
      while (j < n && p[r[j]] == p[r[i]])
      j++;
      if (j-i > 1)
      {
        sort(r+i,r+j,sacmp);
        tint pk,opk = p[(r[i]+t)%n];
        tint q = i, v = i;
        for(; i < j; i++)
        {
          if ( (( pk = p[(r[i]+t)%n]) != opk) && !(q <= opk && pk < j) )
          {
            opk = pk;
            v = i;
          }
          p[r[i]] = v;
        }
      }
      nb++;
    }
    if (lnb == nb)
    break;
  }
  // prim = p[0]
}
```

1.6. Longest Common Prefix + Modo de Uso

```
void lcp (const string &s)
{
  tint q = 0, j;
  forn(i,n)
  {
    if (p[i])
    {
      j = r[p[i]-1];
      while (q < n && s[(i+q)%n] == s[(j+q)%n])
      q++;
      h[p[i]-1] = q;
      if (q > 0)
      q--;
    }
  }

}

int main()
{
  // Uso de SUFFIX ARRAY y LCP : Agregar '$' al final del string
  // Uso de BWT (ROTACIONES) : Usar el string normal

  string s = "banana$";
  tint nn = s.size();
  forn(i,nn)
    cout << s.substr(nn-i-1,nn) << "␣:␣" << i << endl; // TESTEAR Suffix Array
    //~ cout << s.substr(i,nn-i) + s.substr(0,i) << " : " << i << endl; //
        TESTEAR BWT

  bwt(s,nn);
  cout << "---------" << endl;


  forn(i,nn)
    cout << s.substr(r[i],nn) << "␣:␣" << r[i] << endl; // TESTEAR Suffix Array
    //~ cout << s.substr(r[i],nn) + s.substr(0,r[i]) << " : " << r[i] << endl;
        // TESTEAR BWT

  lcp(s);
  cout << "---------" << endl;
  forn(i,nn-1)
    cout << h[i] << endl;


  return 0;
}
```

2. Grafos

2.1. Dijsktra

```
const tint INFINITO = 1e15;

// parent : Inicializar (n,{}) : Guarda donde se realiza la minima distancia
// ladj : Por cada vertice, un par {indice,peso}

void dijkstra (tint comienzo, vector<vector<pair<tint,tint> > > &ladj,
vector<tint> &distance, vector<vector<tint> > &parent) {
  priority_queue <pair<tint,tint> > q; // {-peso,indice}
  tint n = distance.size();
  forn(i,n)
    distance[i] = (i != comienzo)*INFINITO;
  vector<tint> procesado (n,0);
  q.push({0,comienzo});
  while (!q.empty()) {
    tint actual = q.top().second;
    q.pop();
    if (!procesado[actual]) {
      procesado[actual] = 1;
      for (auto vecino : ladj[actual]) {
        if (distance[actual] + vecino.second < distance[vecino.first]) {
          distance[vecino.first] = distance[actual] + vecino.second;
          q.push({-distance[vecino.first],vecino.first});
          parent[vecino.first] = {actual};
        }
        else if (distance[actual] + vecino.second == distance[vecino.first])
          parent[vecino.first].push_back(actual);
      }
    }
  }
}
// En distance quedan las minimas distancias desde comienzo
```

2.2. TopoSort y Kosaraju

```
typedef vector<tint> vi;
void dfsTopo(vector<vi> &g, tint s, vi &vis, vi &ord, vi &comp) {
  vis[s] = true;
  for(auto ad : g[s]) if (!vis[ad]) dfsTopo(g, ad, vis, ord, comp);
  ord.push_back(s);
  comp.push_back(s);
}
vi topoSort(vector<vi> &g) { // Devuelve el orden topologico
  int N = g.size();
  vi vis, ord, aux;
  vis.assign(N, 0);
  forn(i,N) if (!vis[i]) dfsTopo(g, i, vis, ord, aux);
  reverse(ord.begin(), ord.end());
  return ord;
}
// Devuelve las componentes en orden topologico
vector<vi> kosaraju(vector<vi> &graf) {
  vi ord = topoSort(graf);
  // Invertimos el grafo
  tint N = graf.size();
  vector<vi> grafInv(N, vi());
  forn(i,N) for(auto j : graf[i]) grafInv[j].push_back(i);

  vi vis(N, false), aux;
  vector<vi> comps;
  for (auto o : ord)
  if (!vis[o]) {
    vi comp; dfsTopo(grafInv, o, vis, aux, comp);
    comps.push_back(comp);
  }
  return comps;
}
```

2.3. SPFA

```
const tint maxN = 16384; // cantidad de nodos
const tint INFINITO = 1e15; // suma de modulos de las aristas o algo asi

tint best[maxN];
bool adentro[maxN];
// ladj : {indice,peso}
void spfa (tint start, vector<vector<pair<tint,tint> > > &ladj) {
  tint n = ladj.size();
  forn(i,n)
    best[i] = (i != start)*INFINITO;
  vector<tint> vecinos = {start}, nuevosVecinos;
  while (!vecinos.empty()) {
    tint actual = vecinos.back();
    vecinos.pop_back();
    adentro[actual] = false;
    for (auto vecino : ladj[actual]) {
      if (best[actual] + vecino.second < best[vecino.first]) {
        best[vecino.first] = best[actual] + vecino.second;
        if (!adentro[vecino.first]) {
          nuevosVecinos.push_back(vecino.first);
          adentro[vecino.first] = 1;
```

```
        }
      }
    }
    if (vecinos.empty())
      vecinos.swap(nuevosVecinos);
  }
}
```

2.4. Ciclo Hamiltoniano Minimo

```
const tint INFINITO = 1e15;

tint minimumHamiltonianCycle (vector<vector<tint> > &d) {
  tint r = d.size(), minHam = INFINITO;
  if (r > 1) {
    vector<vector<tint> > dp ((1 << r), vector<tint> (r,INFINITO));
    dp[1][0] = 0;
    for(tint mask = 1; mask < (1 << r); mask += 2)
    forn(i,r)
      if ( (i > 0) && (mask & (1 << i)) && (mask & 1) )
        forn(j,r)
          if ((i != j) && (mask & (1 << j)))
            dp[mask][i] = min(dp[mask][i],dp[mask ^ (1 << i)][j] + d[j][i]);

    forsn(i,1,r)
      minHam = min(minHam,dp[(1 << r) - 1][i] + d[i][0]);
  }
  else
    minHam = d[0][0];
  return minHam;
}
```

2.5. Dinic (aguanta multiejes y autoejes)

```
const tint maxN = 512;
const tint INFINITO = 1e15;
struct Arista {
  tint start,end,capacity,flow;
  Arista (tint ss, tint ee, tint cc, tint ff) {
    start = ss;
    end = ee;
    capacity = cc;
    flow = ff;
  }
};

vector<Arista>  red; // Red residual
vector<tint> ladj [maxN]; // (guarda vecinos como indices en red)

tint n, s, t; // #Nodos, source, sink
tint ultimoVecino [maxN]; // ultimo vecino visitado en dfs
tint nivel [maxN]; // Nivel del bfs

void agregarArista (tint ss, tint ee, tint c) {
  ladj[ss].push_back( tint (red.size())); // guardamos el indice
  red.push_back(Arista(ss,ee,c,0));
  ladj[ee].push_back( tint (red.size()));
  red.push_back(Arista(ee,ss,c,0));
}

bool bfs () {
  forn(i,n+1)
    nivel[i] = -1;
  vector<tint> vecinos = {s}, nuevosVecinos;
  nivel[s] = 0;
  while (!vecinos.empty() && nivel[t] == -1) {
    tint actual = vecinos.back();
    vecinos.pop_back();
    for (auto iArista : ladj[actual]) {
      tint vecino = red[iArista].end;
      // Si bajo en uno el nivel y puedo mandar flujo en la red residual
      if (nivel[vecino] == -1 && red[iArista].flow < red[iArista].capacity) {
        nivel[vecino] = nivel[actual] + 1;
        nuevosVecinos.push_back(vecino);
      }
    }
    if (vecinos.empty()) {
      swap(vecinos,nuevosVecinos);
      nuevosVecinos = {};
    }
  }
  return (nivel[t] != -1);
}

tint dfs (tint actual, tint flujo) {
  if (flujo <= 0)
    return 0;
  else if (actual == t)
    return flujo;
  else {
    while (ultimoVecino[actual] < tint(ladj[actual].size())) {
      tint id = ladj[actual][ultimoVecino[actual]];
      if (nivel[red[id].end] == nivel[actual] + 1) {
```

```
60          tint pushed = dfs(red[id].end,min(flujo,red[id].capacity-red[id].flow));
61          if (pushed > 0) {
62            red[id].flow += pushed;
63            red[id^1].flow -= pushed;
64            return pushed;
65          }
66        }
67        ultimoVecino[actual]++;
68      }
69      return 0;
70    }
71  }
72
73  tint dinic () {
74    tint flujo = 0;
75    while (bfs()) {
76      forn(i,n+1)
77        ultimoVecino[i] = 0;
78      tint pushed = dfs(s,INFINITO);
79
80      while (pushed > 0) {
81        flujo += pushed;
82        pushed = dfs(s,INFINITO);
83      }
84    }
85    return flujo;
86  }
```

3. Arboles

3.1. Union-Find (Guty)

```
const tint maxN = 131072;
vector<tint> caminito;
tint representante[maxN];
tint tamanho[maxN];

void inicializar (tint n) {
  forn(i,n) {
    representante[i] = i;
    tamanho[i] = 1;
  }
}

tint find (tint x) {
  caminito = {};
  while (x != representante[x]) {
    caminito.push_back(x);
    x = representante[x];
  }
  for (auto z : caminito)
    representante[z] = x;
  return x;
}

bool same (tint a, tint b) { return (find(a) == find(b)); }

void unite (tint a, tint b) {
  a = find(a);
  b = find(b);
  if (tamanho[a] < tamanho[b])
    swap(a,b);
  tamanho[a] += tamanho[b];
  representante[b] = a;
}
```

3.2. Union-Find (Jonaz)

```
class UF {
private: vector<int> p, rank; int comps;
public:
  UF(int N) {
    rank.assign(N, 0); comps = N;
    p.assign(N, 0); forn(i,N) p[i] = i;
  }
  int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
  bool sameSet(int i, int j) { return findSet(i) == findSet(j); }
  void unionSet(int i, int j) {
    if (!sameSet(i,j)) {
      int x = findSet(i), y = findSet(j);
      if (rank[x] > rank[y]) p[y] = x;
      else {
        p[x] = y;
        if (rank[x] == rank[y]) rank[y]++;
      }
      comps--;
    }
  }
  int components() { return comps; }
};
```

3.3. Kruskal (usa UF de Jonaz)

```
struct Arista {
  tint peso, start, end;
  Arista(tint s, tint e, tint p) : peso(p), start(s), end(e) {}
  bool operator<(const Arista& o) const {
    return make_tuple(peso, start, end) < make_tuple(o.peso, o.start, o.end);
}};
// Devuelve el peso del AGM, y en 'agm' deja las aristas del mismo.
tint kruskal(vector<Arista> &ars, tint size, vector<Arista> &agm) {
  sort(ars.begin(), ars.end());
  tint min_peso = 0;
  UF uf(size);
  for(auto &a : ars) {
    if (!uf.sameSet(a.start, a.end)) {
      min_peso += a.peso;
      uf.unionSet(a.start, a.end);
      agm.push_back(a);
      if ((tint)agm.size() == size-1) break; // Esto es que ya tiene V-1 aristas
  }}
  return min_peso;
}
```

3.4. Binary Lifting (saltitos potencia de 2)

```
const tint maxN = 32768; // cantidad de nodos
const tint maxK = 16;  // lg(cantidadDeNodos)
const tint NEUTRO = 1e8; // neutro de la operacion (ejemplo: minimo)

tint d[maxN]; // profundidad
pair<tint,tint> p[maxN][maxK];
```

```cpp
// {ancestro a distancia 2^k, Lo que queremos entre los 2^k ancestros}

void dfs(tint actual, vector<vector<pair<tint,tint> > > &ladj, tint padre) {
  d[actual] = d[padre]+1;
  for (auto x : ladj[actual])
    if (x.first != padre) {
      p[x.first][0] = {actual,x.second};
      dfs(x.first,ladj,actual);
    }
}

tint subir(tint a, tint c, tint &ans, bool tomaMinimo) {
  tint k = 0;
  while (c > 0) {
    if (c % 2) {
      if (tomaMinimo)
        ans = min(ans,p[a][k].second);
      a = p[a][k].first;
    }
    k++;
    c /= 2;
  }
  return a;
}

tint answer (tint a, tint b) {
  // IGUALAMOS LAS ALTURAS
  if (d[a] < d[b])
    swap(a,b);
  tint w = d[a] - d[b], ans = NEUTRO;
  a = subir(a,w,ans,true);

  // HACEMOS LA BINARY PARA BUSCAR EL LCA
  tint cInf = 0, cSup = maxN;
  while (cSup - cInf > 1) {
    tint ra = a, rb = b;
    tint c = (cSup+cInf)/2;
    ra = subir(ra,c,ans,false);
    rb = subir(rb,c,ans,false);
    if (ra == rb)
      cSup = c;
    else
      cInf = c;
  }
  // SUBIMOS LO QUE HAGA FALTA PARA LLEGAR AL LCA
  cSup *= (a != b);
  a = subir(a,cSup,ans,true);
  b = subir(b,cSup,ans,true);
  return ans;
}

// INICIALIZACION
int main() {
  forn(i,maxN)
  forn(k,maxK)
    p[i][k] = {-1,NEUTRO};
  // HACEMOS EL PRIMER PASO EN FUNCION DEL GRAFO
  vector<vector<pair<tint,tint> > > ladj (maxN); // listaDeAdyacencia del arbol
  d[0] = -1;
  dfs(0,ladj,0);
  // LLENADO DE LA TABLA
  forsn(k,1,maxK)
  forn(i,maxN) {
    tint ancestro = p[i][k-1].first;
    if (ancestro >= 0)
      p[i][k] = {p[ancestro][k-1].first,
                 min(p[i][k-1].second,p[ancestro][k-1].second) };
  }
}
```

4. Strings

4.1. ToString/ToNumber

```cpp
#include <iostream>
#include <string>
#include <sstream>

tint toNumber (string s)
{
  tint Number;
  if ( ! (istringstream(s) >> Number) )
    Number = 0; // el string vacio lo manda al cero
  return Number;
}

string toString (tint number)
{
    ostringstream ostr;
    ostr << number;
    return  ostr.str();
}
```

4.2. Bordes & Knuth-Morris-Pratt (KMP)

```cpp
const int MAXN = 1e7;
int kmp_table[MAXN];
string s, t;
// Deja en kmp_table[i] el borde de longitud i de s
// en kmp_table[s.size()] el maximo borde
void fill_table() {
  int pos = 2, cnd = 0;
  kmp_table[0] = -1;
  kmp_table[1] = 0;
  while(pos <= int(s.size())) {
    if (s[pos-1] == s[cnd])
      kmp_table[pos++] = ++cnd;
    else if (cnd>0)
      cnd = kmp_table[cnd];
    else kmp_table[pos++] = 0;
  }
}
// Matchear s (chico) en t (grande)
// Devuelve todos los indices donde matchea
vector<int> kmp2() {
  vector<int> r;
  int m = 0, i = 0;
  while(m+i < int(t.size())) {
    if (s[i] == t[m+i]) {
      if (i == int(s.size()-1)) r.push_back(m);
      i++;
    } else {
      m = m + i - kmp_table[i];
      if (kmp_table[i] > -1) i = kmp_table[i];
      else i = 0;
    }
  }
  return r;
}

int main() {
  s = "abracadabra";
  fill_table();
  for(int i=0; i<int(s.size()+1); i++) cout << kmp_table[i] << "␣";
  cout << endl; // -1 0 0 0 1 0 1 0 1 2 3 4
  s = "abra";
  fill_table();
  t = "abracadabracadabra";
  for(auto e : kmp2()) cout << e << "␣";
  cout << endl; // 0 7 14
  return 0;
}
```

4.3. Subsecuencia Comun mas larga

```cpp
const tint maxN = 128;

tint p[maxN][maxN];
// Llamar lcs(s1,s2,n,m)
tint lcs(string &s1, string &s2,tint i,tint j)
{
  if (p[i][j] == -1)
  {
    if (i == 0 or j == 0)
    p[i][j] = 0;
    else
    {
      if (s1[i-1] == s2[j-1])
      p[i][j] = 1 + lcs(s1,s2,i-1,j-1);
      else
      {
        p[i][j] = max(p[i][j],lcs(s1,s2,i-1,j));
        p[i][j] = max(p[i][j],lcs(s1,s2,i,j-1));
      }
```

```
    }
  }
  return p[i][j];
}
```

4.4. Edit-Distance

```
// Minima distancia entre strings si lo que se puede es: INSERTAR, REMOVER, MODIFICAR, SWAPS ADYACENTES
const tint maxN = 1024; // maximo largo de los strings
const tint INFINITO = 1e15;
string s1,s2;
tint dist[maxN][maxN];

tint f(tint i, tint j)
{
  // Si un string es vacio, hay que borrar todo el otro
  if (i == -1 or j == -1)
    return max(i,j)+1;
  if (dist[i][j] == INFINITO)
  {
    tint mini = INFINITO;
      // Lo mejor de borrar el i-esimo de s1 o insertar al final de s1 a s2[j]
    mini = min(mini,min(f(i-1,j)+1,f(i,j-1)+1));
    if (s1[i] == s2[j]) // Si coinciden, dejo como esta y resuelvo lo  anterior
      mini = min(mini,f(i-1,j-1));
    else // Modificar s1[i] a s2[j] y resolver lo anterior
      mini = min(mini,f(i-1,j-1)+1);

    // Borramos los intermedios y swapeamos los ultimos 2 si funciona, lo hago y resuelvo lo anterior
    forn(k,i)
    {
      if (i >= 1 && j >= 1 && s1[i] == s2[j-1] && s1[i-k-1] == s2[j])
        mini = min(mini,f(i-k-2,j-2)+k+1);
    }
    dist[i][j] = mini;
  }
  return dist[i][j];
}
// USO:
int main()
{
  tint n = s1.size(), m = s2.size();
  forn(i,n)
  forn(j,m)
    dist[i][j] = INFINITO;
  cout << f(n-1,m-1) << "\n";
  return 0;
}
```

4.5. Substring Palindromo (esPalindromo(s[i..j]))

```
// Asumo i < j
bool esPalindromo (tint i, tint j, vector<vector<tint> > &r, tint n)
{
  if (i+j >= n)
    return (r[i+j][n-i] - r[i+j][n-j-1]) == j-i+1;
  else
    return (r[i+j][j+1] - r[i+j][i]) == j-i+1;
}
// USO:
int main()
{
    tint n = s.size(); // s nuestro string
    vector<vector<tint> > v (n, vector<tint> (n,0));
    forn(i,n)
    forn(j,n)
      v[i][j] = (s[i] == s[j]);
    vector<vector<tint> > r (2*n-1,vector<tint> (n+1,0));

    forn(i,2*n-1)
    {
      tint sum = 0, x = min(i,n-1), y = 0;
      if (i >= n)
        y = i-n+1;
      forn(j,n)
      {
        if (x >=0 && y < n)
          sum += v[x--][y++];
        r[i][j+1] = sum;
      }
    }
    // Ahora podemos preguntar si es palindromo s[i..j]
}
```

5. Matematica

5.1. PotLog

```
const tint nmod = 1000000007; // o el primo que deseamos
tint potLogMod (tint x, tint y) // Calcula: (x^y) mod nmod
{
  tint ans = 1;
  while (y > 0)
  {
    if (y % 2)
      ans = (x * ans) % nmod;
    x = (x * x) % nmod;
    y /= 2;
  }
  return ans;
}
tint invMod(tint a) // nmod PRIMO. Devuelve b tal que: (a*b) = 1 (mod nmod)
{
  return potLogMod(a,nmod-2);
}
```

5.2. Criba

```
const tint maxN = 1000500;
tint p[maxN + 1] = {1, 1};
tint phi[maxN];

map<tint,tint> factorizar (tint n)
{
  map<tint,tint> f;
  while (n > 1)
  {
    f[p[n]]++;
    n /= p[n];
  }
  return f;
}
// USO:
int main()
{
  // CRIBA COMUN : (p[n] = mayor primo que divide a n (n >= 2) )
  for (tint i = 1; i <= maxN; ++i)
    if (p[i] == 1)
      for (tint j = i; j <= maxN; j += i)
        //if (p[j] == 1 or i == 1) // Con esta linea da el menor primo
        p[j] = i;
  // CALCULA PHI(N): #Coprimos con N
  for (tint i = 0; i < maxN; i++)
    phi[i] = i;
  for (tint i = 1; i < maxN; i++)
  for (tint j = 2 * i; j < maxN; j += i)
    phi[j] -= phi[i];
  return 0;
}
```

5.3. Numero Combinatorio

```
const tint maxN = 1024;

tint binom[maxN][maxN];

tint comb(tint n, tint m)
{
  if (m < 0 or m > n)
  return 0;
  if (m == 0 or m == n)
  return 1;
  if (n >= maxN)
  return comb(n-1,m-1) + comb(n-1,m);
  if (binom[n][m] == -1)
  binom[n][m] = comb(n-1,m-1) + comb(n-1,m);
  return binom[n][m];
}

// En el main:
//  forn(i,maxN)
//  forn(j,maxN)
//    binom[i][j] = -1;

tint nBolasEnkCajas (tint n, tint k)
{
  return comb(n+k-1,n);
}
```

5.4. Euclides Extendido

```
tint gcd (tint a, tint b, tint &x, tint &y)
{
  if ( a == 0 )
  {
    x = 0 ; y = 1 ;
    return b ;
  }
```

```
  tint x1, y1 ;
  tint d = gcd (b % a, a, x1, y1) ;
  x = y1 - ( b / a ) * x1 ;
  y = x1 ;
  return d;
}

// Nota si gcd(a,m) == 1 => 1 = a*x+m*y => 1 = a*x (mod m)
// O sea, que "x" es el inverso de "a" modulo "m"
```

5.5. Teorema Chino del Resto

```

pair<tint,tint> tcr (vector<tint> &r, vector<tint> &m) // x = r_i (mod m_i)
{
  tint p = 0, q = 1, n = r.size();
  forn(i,n)
  {
    p = modulo(p-r[i],q);
    tint x,y;
    tint d = gcd(m[i],q,x,y);
    if (p % d)
    return {-1,-1}; // sistema incompatible
    q = (q / d)*m[i];
    p = modulo(r[i]+m[i]*x*(p/d), q); //  OVERFLOW?: __int128 o mult()
    // modulo(r[i]+modulo(modulo(m[i]*x,q)*(p/d),q), q);
  }
  return {p,q}; // x = p (mod q)
}
```

5.6. Eliminacion Gaussiana (Código ruso)

```
// Declarar EPS e INF al principio adecuadamente.
tint gauss ( vector < vector < double > > a, vector < double > & ans ) {
  tint n = a. size ( ) ;
  tint m = a[0].size() -1;
  vector<tint> where (m, -1);
  for ( tint col = 0, row = 0 ; col < m && row < n; ++col)
  {
    int sel = row ;
    for ( tint i = row ; i < n ; ++i)
      if ( abs (a[i][col]) > abs (a[sel][col]) )
        sel = i ;
    if ( abs(a[sel][col]) < EPS )
      continue ;
    for ( tint i = col ; i <= m ; ++i)
      swap (a[sel][i], a[row][i]);
    where[col] = row;
    for ( tint i = 0 ; i < n ; ++i)
      if ( i != row )
      {
        ldouble c = a[i][col] / a[row][col] ;
        for ( tint j = col ; j <= m ; ++ j)
          a[i][j] -= a[row][j] * c;
      }
    ++row;
  }

  ans.assign(m, 0);
  for ( tint i = 0 ; i < m ; ++i )
    if ( where[i] != - 1 )
      ans [i] = a[where[i]][m] / a [where[i]][i];
  for ( tint i = 0 ; i < n ; ++i )
  {
    ldouble sum = 0 ;
    for ( tint j = 0 ; j < m ; ++j )
      sum += ans[j]*a[i][j];
    if ( abs(sum - a[i][m] ) > EPS )
      return 0 ;
  }

  for ( int i = 0 ; i < m ; ++ i )
    if ( where[i] == - 1)
      return INF ;
  return 1 ;
}

// 0 -> No hay solucion
// 1 -> Hay solucion unica. La devuelve en "ans"
// INF -> Hay infinitas soluciones
```

5.7. Rabin-Miller

```
// USA: "PotLog", pero pasandole el modulo como parametro
#include <random>
const tint semilla = 38532164;
mt19937 gen(semilla);

tint mult(tint a, tint b, tint m)
{
  int largestBit = 0;
  while( (b >> largestBit) != 0)
    largestBit++;
  tint ans = 0;
```

```
  for(tint currentBit = largestBit - 1; currentBit >= 0; currentBit--)
  {
    ans = (ans + ans);
    if (ans >= m)
      ans -= m;

    if ( (b >> currentBit) & 1)
    {
      ans += a;
      if (ans >= m)
        ans -= m;
    }
  }
  return ans;
}

bool esPrimoRM (tint n)
{
  if (n <= 1)
    return false;
  else if (n <= 3)
    return true;
  else if (n % 2 == 0)
    return false;
  else
  {
    uniform_int_distribution<tint> dis(2, n-2);
    tint kOrig = 0, m = n-1;
    while (m % 2 == 0)
    {
      kOrig++;
      m /= 2;
    }
    bool esPrimo = true;
    vector<tint> testigos = {2,3,5,7,11,13,17,19,23,29,31,37};
    for (auto a : testigos)
    {
      if (a < n)
      {
        tint b = potLogMod(a,m,n), k = kOrig;
        if (b == 1 or b == n-1)
          continue;
        else
        {
          forn(j,k)
          {
            b = mult(b,b,n);
            if (b == n-1)
              break;
            else if (b == 1)
            {
              esPrimo = false;
              break;
            }
          }
          if (b != n-1)
          {
            esPrimo = false;
            break;
          }
        }
      }
    }
    return esPrimo;
  }
}
```

5.8. Pollard-Rho

```
// USA: Rabin-Miller
tint gcd (tint a, tint b)
{
  if (a == 0)
    return b;
  return gcd (b % a, a);
}
void factorizar (tint n, map<tint,tint> &f)
{
  while (n > 1)
  {
    if (esPrimoRM(n))
    {
      f[n]++;
      n /= n;
    }
    else
    {
      uniform_int_distribution<tint> dis(1, n-1);
      tint a = dis(gen), b = dis(gen), x = 2, y = 2, d;
      do
      {
        x = (mult(x,x,n) + mult(a,x,n) + b) % n;
        y = (mult(y,y,n) + mult(a,y,n) + b) % n;
```

```
        y = (mult(y,y,n) + mult(a,y,n) + b) % n;
        d = gcd(abs(x-y),n);
      }
      while (d == 1);
      if (d != n)
      {
        factorizar(d,f);
        n /= d;
      }

    }
  }
}
```

5.9. FFT

```
// USA : "PotLog" e "InvMod" con nmod = mod
const tint mod = (1 << 21)*11 + 1 ; // es re primo
const tint root = 38;
const tint root_1 = 21247462;
const tint root_pw = 1 << 21 ; // largo del arreglo
/*
 * const tint mod = 7340033;
 * const tint root = 5 ;
 * const tint root_1 = 4404020 ;
 * const tint root_pw = 1 << 20 ;
 */

tint modulo (tint n)
{
  return ((n % mod) + mod) % mod;
}
void fft (vector <tint> &a, bool invert )
{
  tint n = a. size();
  for (tint i = 1 , j = 0 ; i < n ; ++ i )
  {
    tint bit = n >> 1 ;
    while(j >= bit)
    {
      j -= bit ;
      bit >>= 1;
    }
    j += bit ;
    if ( i < j )
      swap (a[i],a[j]);
  }
  for (tint len = 2 ; len <= n ; len <<= 1)
  {
    tint wlen = root;
    if (invert)
      wlen = root_1;
    for (tint i = len ; i < root_pw ; i <<= 1)
      wlen = modulo(wlen * wlen);
    for (tint i = 0 ; i < n ; i += len )
    {
      tint w = 1 ;
      forn(j,len/2)
      {
        tint u = a[i+j], v = modulo(a[i+j+len/2] * w) ;
        a[i+j] = modulo(u+v);
        a[i+j + len/2] = modulo(u - v);
        w = modulo(w * wlen) ;
      }
    }
  }

  if (invert)
  {
    tint nrev = invMod(n);
    forn(i,n)
      a[i] = modulo(a[i] * nrev) ;
  }
}

void multiply (const vector<tint> &a, const vector<tint> &b, vector<tint> &res)
{
  vector<tint> fa(a.begin(), a.end() ), fb(b.begin(), b.end() );
  tint n = 1 ;
  while (n < max(tint(a.size()), tint(b.size())))
    n <<= 1;
  n <<= 1 ;
  fa.resize(n), fb.resize(n);
  fft (fa, false) , fft(fb, false);
  forn(i,n)
    fa[i] *= fb[i];
  fft(fa, true);
  res = fa;
}
// USO:
int main()
{
  vector<tint> a = {1,0,0,1};
```

```
78   vector<tint> b = {1,0,0,1};
79   vector<tint> res;
80   multiply(a,b,res);
81   for (auto x : res)
82     cout << x << "␣" ; // 1 0 0 2 0 0 1 0
83   cout << endl;
84   return 0;
85 }
```

5.10. Regla de Simpson (Integracion Numerica)

```
1  // f (x) una funcion definidia
2  ldouble a, b; // extremos de integracion
3  const int N = 1000*1000; // cantidad de nodos en la malla
4  ldouble s = 0; // resultado de la integral
5  ldouble h = (b - a) / N;
6  forn(i,N)
7  {
8    ldouble x = a + h * i;
9    s += f(x) * ((i==0 || i==N) ? 1 : ((i&1)==0) ? 2 : 4);
10 }
11 s *= h / 3; // es el resultado de integrar f en (a,b)
```

6. Geometria

6.1. Tipo PUNTO y Operaciones

```
const ldouble epsilon = 1e-10;
const ldouble pi = acos(-1);

struct Punto
{
  ldouble x,y;
  Punto (ldouble xx, ldouble yy)
  {
    x = xx;
    y = yy;
  }
  Punto()
  {
    x = 0.0;
    y = 0.0;
  }
};
Punto operator + (Punto p1, Punto p2)
{
  return Punto(p1.x+p2.x,p1.y+p2.y);
}
Punto operator - (Punto p1, Punto p2)
{
  return Punto(p1.x-p2.x,p1.y-p2.y);
}
Punto operator * (ldouble lambda, Punto p)
{
  return Punto(lambda*p.x, lambda*p.y);
}
ldouble operator * (Punto p1, Punto p2)
{
  return p1.x*p2.x+p1.y*p2.y;
}
ldouble operator ^ (Punto p1, Punto p2)
{
  return p1.x*p2.y - p1.y*p2.x;
}
Punto operator ~ (Punto p)
{
  return Punto(-p.y,p.x);
}
ldouble norma (Punto p)
{
  return sqrt(p.x*p.x+p.y*p.y);
}
bool operator < (Punto p1, Punto p2)
{
  return make_pair(p1.x,p1.y) < make_pair(p2.x,p2.y);
}
bool operator == (Punto p1, Punto p2)
{
  return ((abs(p1.x-p2.x) < epsilon) && (abs(p1.y-p2.y) < epsilon));
}
```

6.2. Area de Poligono

```
ldouble areaTriangulo (Punto p1, Punto p2, Punto p3)
{
  return abs((p1-p3)^(p1-p2))/2.0;
}

ldouble areaPoligono(vector<Punto> &polygon)
{
  ldouble area = 0.0;
  tint n = polygon.size();
  forn(i,n)
    area += polygon[i]^polygon[(i+1)%n];
  return abs(area)/2.0;
}
```

6.3. Punto en Poligono

```
bool adentroPoligono(vector<Punto> &polygon, Punto p) // polygon EN EL SENTIDO
    DE LAS AGUJAS
{
  bool adentro = true;
  tint n = polygon.size();
  forn(i,n)
    adentro &= (((p-polygon[i])^(p-polygon[(i+1)%n])) < 0);
  return adentro;
}
```

6.4. Interseccion de Segmentos

```
struct Segmento
{
  Punto start,end,dir;
  Segmento (Punto ss, Punto ee)
  {
```

```
    start = ss;
    end = ee;
    dir = ee-ss;
  }
};
// res.second == 0 -> NO HAY INTERSECCION
// res.second == 1 -> INTERSECAN EN UN PUNTO (que esta en res.first)
// res.second == 2 -> SON COLINEALES E INTERSECAN EN TODO UN SEGMENTO (Da un
    extremo, si queremos el otro, correr otra vez con "otroExtremo" = true)
pair<Punto,tint> interSeg (Segmento s1, Segmento s2, bool otroExtremo )
{
  if ((abs(s1.dir ^ s2.dir)) < epsilon) // son colineales
  {
    vector<pair<Punto,tint> > aux = {{s1.start - epsilon*s1.dir,1},
                                     {s1.end   + epsilon*s1.dir,1},
                                     {s2.start - epsilon*s2.dir,2},
                                     {s2.end   + epsilon*s2.dir,2}};
    sort(aux.begin(),aux.end());
    if (aux[0].second != aux[1].second)
      return make_pair(aux[1+otroExtremo].first,2);
    else
      return make_pair(Punto(),0);
  }
  else
  {
    ldouble alfa = ((s2.start-s1.start)^s2.dir) / (s1.dir^s2.dir);
    if (0 <= alfa && alfa <= 1)
      return make_pair(s1.start+alfa*s1.dir,1);
    else
      return make_pair(Punto(),0);
  }
}
```

6.5. Angulo Entre Puntos y Distancia entre Segmentos

```
ldouble angEntre (Punto p1, Punto p2, Punto p3) // P1^P2P3
{
  ldouble a = norma(p2-p3);
  ldouble b = norma(p1-p3);
  ldouble c = norma(p2-p1);
  return acos((a*a+c*c-b*b)/(2*a*c));
}

ldouble dPuntoSeg (Punto p, Segmento s)
{
  if (angEntre(p,s.start,s.end) > pi/2 or angEntre(p,s.end,s.start) > pi/2)
    return min(norma(p-s.start),norma(p-(s.end)));
  else
    return abs( ((s.start-p)^(s.end-p)) / (norma(s.dir)) );
}

ldouble dEntreSeg(Segmento s1, Segmento s2)
{
  ldouble a = min(dPuntoSeg(s1.start,s2),dPuntoSeg(s1.end,s2));
  ldouble b = min(dPuntoSeg(s2.start,s1),dPuntoSeg(s2.end,s1));
  return (interSeg(s1,s2,false).second == 0) * min(a,b);
}
```

6.6. Convex-Hull (2D)

```

tint norma2Sqr (Punto p)
{
  return p*p;
}
tint areaTriangulo (Punto p1, Punto p2, Punto p3) // Doble, negativo si horario
{
  return (p1-p3)^(p1-p2);
}
bool porX (Punto p1, Punto p2)
{
  return make_pair(p1.x,p1.y) < make_pair(p2.x,p2.y);
}

Punto r;
bool operator < (Punto p1, Punto p2)
{
  if (areaTriangulo(r,p1,p2) == 0)
    return norma2Sqr(p1-r) < norma2Sqr(p2-r);
  else
    return areaTriangulo(r,p1,p2) > 0;
}

vector<Punto> chull(vector<Punto> &l)
{
  vector<Punto> res = l;
  if (l.size() < 3)
    return res;
  r = *(min_element(res.begin(), res.end(), porX));
  sort(res.begin(), res.end());
  vector<Punto> ch = {res[0],res[1]};
  tint i = 2, k = res.size();
  while(i < k)
    if (ch.size() >= 2 && areaTriangulo(ch[ch.size()-2],ch[ch.size()-1],res[i])
```

```
        <0) // poner <= para capturar alineados
      ch.pop_back();
    else
      ch.push_back(res[i++]);
  return ch;
}
```

7. Varios

7.1. Operaciones de bits

```
1
2  __builtin_clz(x) // the number of zeros at the beginning of the number
3  __builtin_ctz(x) // the number of zeros at the end of the number
4  __builtin_popcount(x)  // the number of ones in the number
5  __builtin_parity(x) // the parity (even or odd) of the number of ones
6
7  // Iterar sobre el subconjunto de la mascara "x"
8  int b = 0;
9  do
10 {
11   // process subset b
12 } while (b=(b-x)&x);
```

7.2. Longest Incresing Subsequence (LIS)

```
1  tint LIS(vector<tint> &v) {
2    if (v.empty()) return 0;
3
4    tint l = 0;             // ultimo lugar de tails hasta ahora
5    vi tails(v.size(), 0);  // candidatos de final de sub secuencias
6    tails[l] = v[0];
7
8    forsn(i,1,v.size()) {
9      // con upper_bound es no-decreciente
10     tint me = lower_bound(tails.begin(),tails.begin()+l+1, v[i])-tails.begin();
11     tails[me] = v[i];
12     if (me > l) l = me;
13   }
14   return l + 1;
15 }
```

7.3. Maximum Subarray Sum

```
1  tint maximumSum (vector<tint> &a) // a no vacio
2  {
3    tint maxTotal = a[0], maxAca = a[0], n = a.size();
4    forsn(i,1,n)
5    {
6      maxAca = max(a[i],maxAca + a[i]);
7      maxTotal = max(maxTotal,maxAca);
8    }
9    return maxTotal;
10 }
```

7.4. Rotar 90º una matriz (sentido horario)

```
1  void rotar (vector<string> &origi)
2  {
3    tint n = origi.size();
4    string aux (n,'x');
5    vector<string> rotado (n,aux);
6    forn(i,n)
7    forn(j,n)
8      rotado[j][n-i-1] = origi[i][j];
9    origi = rotado;
10 }
```

7.5. Random + Imprimir Doubles + Leading Zeroes

```
1  #include <iostream>
2  #include <random>
3  #include <iomanip>
4
5  using namespace std;
6
7  random_device rd;
8  mt19937 gen(rd());
9  uniform_int_distribution<int> dis1(1, 10000);
10 uniform_real_distribution<long double> dis2(1, 10000);
11
12 int main()
13 {
14   cout <<  dis1(gen) << "\n";
15   cout <<  fixed << showpoint << setprecision(16) << dis2(gen) << "\n";
16   cout << setfill('0') << setw(10) << 12345 << "\n"; // 0000012345
17
18   return 0;
19 }
```

7.6. Slding Window RMQ

```
1  void agrandarVentana (tint &r, deque<pair<tint,tint> > &rmq, vector<tint> &v)
2  {
3    while (!rmq.empty() && rmq.back().first >= v[r])
4      rmq.pop_back();
5    rmq.push_back({v[r],r});
6    r++;
7
8  }
9
10 void achicarVentana (tint &l, deque<pair<tint,tint> > &rmq)
```

```
{
  if (l == rmq.front().second)
    rmq.pop_front();
  l++;
}

pair<tint,tint> minimoVentana (deque<pair<tint,tint> > &rmq)
{
  return rmq.front();
}
// USO: En todo momento tenemos el minimo entre [l,r)
int main()
{
  deque<pair<tint,tint> > rmq; // {numero,indice}
  tint l = 0, r = 0;//  l   . r
  vector<tint> v = {1,2,3,4,3,2,2,3,4};
  agrandarVentana(r,rmq,v);
  agrandarVentana(r,rmq,v);
  agrandarVentana(r,rmq,v);
  agrandarVentana(r,rmq,v);
  agrandarVentana(r,rmq,v);
  achicarVentana(l,rmq);
  achicarVentana(l,rmq);
  cout << minimoVentana(rmq).first << endl; // {3,4}
  return 0;
}
```

7.7. Ternary Search

```
// Ternary en ENTEROS
tint miniTernarySearch (tint a, tint b) // En [a,b] esta el minimo
{
  tint l = a, r = b;
  while (abs(r - l) > 5)
  {
    tint al = (2*l + r)/3;
    tint br = (l + 2*r)/3;
    if (f(al) > f(br)) // cambiar a "<" para maximo
      l = al;
    else
      r = br;
  }
  tint ans = 1e16;
  forsn(k,l,r+1)
    ans = min(ans,f(k));  // cambiar por "max" para maximo
  return ans;
}
```

```
//Ternary en FLOATING POINT
ldouble miniTernarySearch (ldouble tL, ldouble tR) // En [tL, tR] esta el minimo
{
  while (abs(tR - tL) > epsilon)
  {
    ldouble tLThird = (2.0*tL + tR)/3.0;
    ldouble tRThird = (tL + 2.0*tR)/3.0;
    if (f(tLeftThird) > f(tRightThird)) // cambiar a "<" para maximo
      tLeft = tLeftThird;
    else
      tRight = tRightThird;
  }
  return f((tLeft+tRight)/2.0);
}
```