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);
  }
}                                                          (inclusive)
tint sum (tint k, vector<tint> &fenwick) // Devuelve la suma en el rango [1..k]
{
  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);
}
```

```cpp
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);
}
```

```cpp
// 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;
}
```

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 (Jonaz)

2.3. Kosaraju (Jonaz)

2.4. 2-SAT (Jonaz)

2.5. Puentes, Puntos de Articulacion y Biconexas (Jonaz)

2.6. 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);
  }
}
```

2.7. 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;
}
```

2.8. 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)
      {
        tint pushed = dfs(red[id].end,min(flujo,red[id].capacity-red[id].flow));
        if (pushed > 0)
        {
          red[id].flow += pushed;
          red[id^1].flow -= pushed;
          return pushed;
        }
      }
      ultimoVecino[actual]++;
    }
    return 0;
  }
}

tint dinic ()
{
  tint flujo = 0;
  while (bfs())
  {

    forn(i,n+1)
      ultimoVecino[i] = 0;
    tint pushed = dfs(s,INFINITO);

    while (pushed > 0)
    {
      flujo += pushed;
      pushed = dfs(s,INFINITO);
    }
  }
  return flujo;
}
```

2.9. Flujo de Costo Mnimo

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) Union by Rank & Path ompression

```
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. LCA - Segment Tree (Jonaz)

3.5. 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]; // {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

```
#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. Tablita de Bordes (Jonaz)

4.3. Knuth-Morris-Pratt (KMP) (Jonaz)

4.4. Subsecuencia Comun mas larga (Guty o Jonaz)

4.5. 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;
}
```

4.6. 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]
}
```

4.7. Suffix Array

4.8. Longest Common Prefix

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. Euclides Extendido (Guty)

5.4. Teorema Chino del Resto (Guty)

5.5. Eliminacion Gaussiana

5.6. 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;
  }
}
```

5.7. 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;
      }

    }
  }
}
```

5.8. FFT

```cpp
// 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};
  vector<tint> b = {1,0,0,1};
  vector<tint> res;
  multiply(a,b,res);
  for (auto x : res)
    cout << x << "␣" ; // 1 0 0 2 0 0 1 0
  cout << endl;
  return 0;
}
```

5.9. Regla de Simpson (Integracion Numerica) (Guty)

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) (Jonaz)

6.7. Sweep Line Facil (Interseccion de Segmentos/Closest Pair)

6.8. Sweep Line Dificil (Union de Rectngulos)

6.9. Radial Sweep

6.10. Minimum Bounding Circle

7. Varios

7.1. Longest Incresing Subsequence (LIS)

```
tint LIS(vector<tint> &v) {
  if (v.empty()) return 0;

  tint l = 0;             // ultimo lugar de tails hasta ahora
  vi tails(v.size(), 0);  // candidatos de final de sub secuencias
  tails[l] = v[0];

  forsn(i,1,v.size()) {
    // con upper_bound es no-decreciente
    tint me = lower_bound(tails.begin(),tails.begin()+l+1, v[i])-tails.begin();
    tails[me] = v[i];
    if (me > l) l = me;
  }
  return l + 1;
}
```

7.2. Maximum Subarray Sum

```

tint maximumSum (vector<tint> &a) // a no vacio
{
  tint maxTotal = a[0], maxAca = a[0], n = a.size();
  forsn(i,1,n)
  {
    maxAca = max(a[i],maxAca + a[i]);
    maxTotal = max(maxTotal,maxAca);
  }
  return maxTotal;
}
```

7.3. Rotar 90 una matriz (sentido horario)

```
void rotar (vector<string> &origi)
{
  tint n = origi.size();
  string aux (n,'x');
  vector<string> rotado (n,aux);
  forn(i,n)
  forn(j,n)
    rotado[j][n-i-1] = origi[i][j];
  origi = rotado;
}
```

7.4. Random + Imprimir Doubles

```
#include <iostream>
#include <random>
#include <iomanip>

using namespace std;

random_device rd;
mt19937 gen(rd());
uniform_int_distribution<int> dis1(1, 10000);
uniform_real_distribution<long double> dis2(1, 10000);

int main()
{
  cout <<  dis1(gen) << "\n";
  cout <<  fixed << showpoint << setprecision(16) << dis2(gen) << "\n";
  return 0;
}
```

7.5. Slding Window RMQ

```

void agrandarVentana (tint &r, deque<pair<tint,tint> > &rmq, vector<tint> &v)
{
  while (!rmq.empty() && rmq.back().first >= v[r])
    rmq.pop_back();
  rmq.push_back({v[r],r});
  r++;

}

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;
}
```

7.6. 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);
}
```