

Índice

1. Estructuras	2
1.1. Fenwick Tree	2
1.2. Trie	2
1.3. Segment Tree	2
1.4. Order Statistic Tree (GCC)	3
2. Grafos	4
2.1. Dijkstra	4
2.2. TopoSort y Kosaraju	4
2.3. 2-SAT (Jonaz)	4
2.4. Puentes, Puntos de Articulacion y Biconexas (Jonaz)	4
2.5. SPFA	4
2.6. Ciclo Hamiltoniano Minimo	5
2.7. Dinic (aguanta multiejes y autoejes)	5
2.8. Flujo de Costo Mínimo	6
3. Arboles	7
3.1. Union-Find (Guty)	7
3.2. Union-Find (Jonaz)	7
3.3. Kruskal (usa UF de Jonaz)	7
3.4. LCA - Segment Tree (Jonaz)	7
3.5. Binary Lifting (saltitos potencia de 2)	7
4. Strings	9
4.1. ToString/ToNumber	9
4.2. Tablita de Bordes	9
4.3. Knuth-Morris-Pratt (KMP) (Jonaz)	9
4.4. Subsecuencia Comun mas larga (Guty o Jonaz)	9
4.5. Edit-Distance	9
4.6. Substring Palindromo (esPalindromo(s[i..j]))	10
4.7. Suffix Array	10
4.8. Longest Common Prefix	10
5. Matematica	11
5.1. PotLog	11
5.2. Criba	11
5.3. Euclides Extendido (Guty)	11
5.4. Eliminacion Gaussiana (Código ruso)	11
5.5. Rabin-Miller	12
5.6. Pollard-Rho	13
5.7. FFT	13

5.8. Regla de Simpson (Integracion Numerica)	14
6. Geometria	15
6.1. Tipo PUNTO y Operaciones	15
6.2. Area de Poligono	15
6.3. Punto en Poligono	15
6.4. Interseccion de Segmentos	15
6.5. Angulo Entre Puntos y Distancia entre Segmentos	16
6.6. Convex-Hull (2D) (Jonaz)	16
6.7. Sweep Line Facil (Interseccion de Segmentos/Closest Pair)	16
6.8. Sweep Line Dificil (Union de Rectángulos)	16
6.9. Radial Sweep	16
6.10. Minimum Bounding Circle	16
7. Varios	17
7.1. Operaciones de bits	17
7.2. Longest Increasing Subsequence (LIS)	17
7.3. Maximum Subarray Sum	17
7.4. Rotar 90° una matriz (sentido horario)	17
7.5. Random + Imprimir Doubles + Leading Zeroes	17
7.6. Sliding Window RMQ	17
7.7. Ternary Search	18

BGL-UBA - Reference

Cosas a tener en cuenta

Flags de Compilación

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

1. Estructuras

1.1. Fenwick Tree

```
1 // TRABAJAR CON UN VECTOR INDEXADO EN 1 EN "fenwick" (DE TAMANO N+1)
2 void add (tint k, tint x, vector<tint> &fenwick) { // Suma x al indice k
3     tint n = fenwick.size() -1;
4     while (k <= n) {
5         fenwick[k] += x;
6         k += (k & -k);
7     }
8 }
9 // Devuelve la suma en el rango [1..k] (inclusive)
10 tint sum (tint k, vector<tint> &fenwick) {
11     tint s = 0;
12     while (k >= 1) {
13         s += fenwick[k];
14         k -= (k & -k);
15     }
16     return s;
17 }
```

1.2. Trie

```
1 const int MAXN = 60000;
2
3 struct TrieNode {
4     map<char, int> sig;
5     bool final = false;
6     void reset() { sig.clear(); final = false; }
7 };
8
9 TrieNode trie[MAXN];
10 int trie_n = 1;
11
12 void resetTrie() {
13     trie_n = 1;
14     trie[0].reset();
15 }
```

```
15 }
16
17 void insertar(string st) {
18     int pos = 0;
19     for(int i=0; i<(int)st.size(); i++) {
20         if (trie[pos].sig.find(st[i]) == trie[pos].sig.end()) {
21             trie[pos].sig[st[i]] = trie_n;
22             trie[trie_n].reset();
23             trie_n++;
24         }
25         pos = trie[pos].sig[st[i]];
26     }
27     trie[pos].final = true;
28 }
29
30 bool buscar(string st) {
31     int pos = 0;
32     for(int i=0; i<(int)st.size(); i++) {
33         if (trie[pos].sig.find(st[i]) == trie[pos].sig.end())
34             return false;
35         pos = trie[pos].sig[st[i]];
36     }
37     return (trie[pos].final == true);
38 }
```

1.3. Segment Tree

```
1 // Nodo del segment tree
2 struct Nodo {
3     tint x;
4     Nodo (tint xx) { x = xx; }
5 };
6 // Operacion del segment tree : tiene que ser ASOCIATIVA
7 Nodo op (Nodo n1, Nodo n2) {
8     return Nodo(n1.x+n2.x);
9 }
10 vector<Nodo> buildSegTree (vector<Nodo> &v ) {
11     // Completa el tamanho
12     tint k = 4, n = v.size();
13     while (k < 2*n)
14         k <<= 1;
15     // Rellena las hojas
16     vector<Nodo> seg (k, Nodo(0));
17     forn(i,n)
18         seg[(k >> 1)+i] = v[i];
19     // Completa los padres
20     while (k > 0) {
```

```

21     seg[(k-1) >> 1] = op(seg[k-1],seg[k-2]);
22     k -= 2;
23 }
24 return seg;
25 }
26 // i es el indice de [0,n) en el arreglo original
27 // Nodo es lo que queremos poner ahora como hoja
28 void update(tint i, Nodo nodo,vector<Nodo> &seg) {
29     tint k = seg.size()/2 + i;
30     seg[k] = nodo;
31     while (k > 0) {
32         seg[k >> 1] = op(seg[k],seg[k^1]);
33         k >>= 1;
34     }
35 }
36 Nodo queryAux(tint k, tint l, tint r, tint i, tint j, vector<Nodo> &seg) {
37     if (i <= l && r <= j)
38         return seg[k];
39     if (r <= i or l >= j)
40         return Nodo(0); // Aca va el NEUTRO de la funcion "op"
41     Nodo a = queryAux(2*k,l,(l+r) >> 1,i,j,seg);
42     Nodo b = queryAux(2*k+1,(l+r) >> 1,r,i,j,seg);
43     return op(a,b);
44 }
45 // i,j son los indices del arreglo del que se hace la query
46 // la query se hace en [i,j)
47 Nodo query(tint i, tint j, vector<Nodo> &seg) {
48     return queryAux(1,0,seg.size() >> 1,i,j,seg);
49 }
50 // USO:
51 int main() {
52     tint n = 15;
53     vector<Nodo> v (n, Nodo(0));
54     forn(i,n)
55         v[i] = Nodo((3*(i+1))%7 - 9*(i-4)%13);
56     vector<Nodo> seg = buildSegTree(v);
57     forn(i,n)
58         cout << v[i].x << " "; // 13 7 7 14 1 -5 -5 2 -4 -4 3 -10 -3 -3 -9
59     cout << endl;
60     cout << query(3,11,seg).x << "\n"; // Devuelve 2
61     update(6,Nodo(0),seg);
62     cout << query(3,11,seg).x << "\n"; // Devuelve 7
63     return 0;
64 }

```

1.4. Order Statistic Tree (GCC)

```

1  #include <iostream>
2  #include <ext/pb_ds/assoc_container.hpp>
3
4  using namespace std;
5  using namespace __gnu_pbds;
6
7  typedef tree<int, null_type, less<int>, rb_tree_tag,
8  tree_order_statistics_node_update> ordered_set;
9
10 int main() {
11     ordered_set X;
12     X.insert(1); X.insert(2); X.insert(4); X.insert(8); X.insert(16);
13
14     // find_by_order(i): iterador al lugar donde se encuentra el i-esimo
15     cout << *X.find_by_order(1) << endl; // 2
16     cout << *X.find_by_order(2) << endl; // 4
17     cout << *X.find_by_order(4) << endl; // 16
18     cout << (X.end() == X.find_by_order(6)) << endl; // true
19
20     // order_of_key(x): orden donde iria el elemento x
21     cout << X.order_of_key(-5) << endl; // 0
22     cout << X.order_of_key(1) << endl; // 0
23     cout << X.order_of_key(3) << endl; // 2
24     cout << X.order_of_key(4) << endl; // 2
25     cout << X.order_of_key(400) << endl; // 5
26     return 0;
27 }

```

2. Grafos

2.1. Dijkstra

```

1  const tint INFINITO = 1e15;
2
3  // parent : Inicializar (n,{}) : Guarda donde se realiza la minima distancia
4  // ladj : Por cada vertice, un par {indice,peso}
5
6  void dijkstra (tint comienzo, vector<vector<pair<tint,tint> > > &ladj,
7  vector<tint> &distance, vector<vector<tint> > &parent) {
8      priority_queue<pair<tint,tint> > q; // {-peso,indice}
9      tint n = distance.size();
10     forn(i,n)
11         distance[i] = (i != comienzo)*INFINITO;
12     vector<tint> procesado (n,0);
13     q.push({0,comienzo});
14     while (!q.empty()) {
15         tint actual = q.top().second;
16         q.pop();
17         if (!procesado[actual]) {
18             procesado[actual] = 1;
19             for (auto vecino : ladj[actual]) {
20                 if (distance[actual] + vecino.second < distance[vecino.first]) {
21                     distance[vecino.first] = distance[actual] + vecino.second;
22                     q.push({-distance[vecino.first],vecino.first});
23                     parent[vecino.first] = {actual};
24                 }
25                 else if (distance[actual] + vecino.second == distance[vecino.first])
26                     parent[vecino.first].push_back(actual);
27             }
28         }
29     }
30 }
31 // En distance quedan las minimas distancias desde comienzo

```

2.2. TopoSort y Kosaraju

```

1  typedef vector<tint> vi;
2  void dfsTopo(vector<vi> &g, tint s, vi &vis, vi &ord, vi &comp) {
3      vis[s] = true;
4      for(auto ad : g[s]) if (!vis[ad]) dfsTopo(g, ad, vis, ord, comp);
5      ord.push_back(s);
6      comp.push_back(s);
7  }
8  vi topoSort(vector<vi> &g) { // Devuelve el orden topologico
9      int N = g.size();

```

```

10     vi vis, ord, aux;
11     vis.assign(N, 0);
12     forn(i,N) if (!vis[i]) dfsTopo(g, i, vis, ord, aux);
13     reverse(ord.begin(), ord.end());
14     return ord;
15 }
16 // Devuelve las componentes en orden topologico
17 vector<vi> kosaraju(vector<vi> &graf) {
18     vi ord = topoSort(graf);
19     // Invertimos el grafo
20     tint N = graf.size();
21     vector<vi> grafInv(N, vi());
22     forn(i,N) for(auto j : graf[i]) grafInv[j].push_back(i);
23
24     vi vis(N, false), aux;
25     vector<vi> comps;
26     for (auto o : ord)
27         if (!vis[o]) {
28             vi comp; dfsTopo(grafInv, o, vis, aux, comp);
29             comps.push_back(comp);
30         }
31     return comps;
32 }

```

2.3. 2-SAT (Jonaz)

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

2.5. SPFA

```

1  const tint maxN = 16384; // cantidad de nodos
2  const tint INFINITO = 1e15; // suma de modulos de las aristas o algo asi
3
4  tint best[maxN];
5  bool adentro[maxN];
6  // ladj : {indice,peso}
7  void spfa (tint start, vector<vector<pair<tint,tint> > > &ladj) {
8      tint n = ladj.size();
9      forn(i,n)
10         best[i] = (i != start)*INFINITO;
11     vector<tint> vecinos = {start}, nuevosVecinos;
12     while (!vecinos.empty()) {
13         tint actual = vecinos.back();
14         vecinos.pop_back();
15         adentro[actual] = false;

```

```

16     for (auto vecino : ladj[actual]) {
17         if (best[actual] + vecino.second < best[vecino.first]) {
18             best[vecino.first] = best[actual] + vecino.second;
19             if (!adentro[vecino.first]) {
20                 nuevosVecinos.push_back(vecino.first);
21                 adentro[vecino.first] = 1;
22             }
23         }
24     }
25     if (vecinos.empty())
26         vecinos.swap(nuevosVecinos);
27 }
28 }

```

2.6. Ciclo Hamiltoniano Minimo

```

1  const tint INFINITO = 1e15;
2
3  tint minimumHamiltonianCycle (vector<vector<tint> > &d) {
4      tint r = d.size(), minHam = INFINITO;
5      if (r > 1) {
6          vector<vector<tint> > dp ((1 << r), vector<tint> (r,INFINITO));
7          dp[1][0] = 0;
8          for(tint mask = 1; mask < (1 << r); mask += 2)
9              forn(i,r)
10                 if ( (i > 0) && (mask & (1 << i)) && (mask & 1) )
11                     forn(j,r)
12                         if ((i != j) && (mask & (1 << j)))
13                             dp[mask][i] = min(dp[mask][i], dp[mask ^ (1 << i)][j] + d[j][i]);
14
15         forsn(i,1,r)
16             minHam = min(minHam, dp[(1 << r) - 1][i] + d[i][0]);
17     }
18     else
19         minHam = d[0][0];
20     return minHam;
21 }

```

2.7. Dinic (aguantate multiejes y autoejes)

```

1  const tint maxN = 512;
2  const tint INFINITO = 1e15;
3  struct Arista {
4      tint start,end,capacity,flow;
5      Arista (tint ss, tint ee, tint cc, tint ff) {
6          start = ss;
7          end = ee;

```

```

8      capacity = cc;
9      flow = ff;
10     }
11 };
12
13 vector<Arista> red; // Red residual
14 vector<tint> ladj [maxN]; // (guarda vecinos como indices en red)
15
16 tint n, s, t; // #Nodos, source, sink
17 tint ultimoVecino [maxN]; // ultimo vecino visitado en dfs
18 tint nivel [maxN]; // Nivel del bfs
19
20 void agregarArista (tint ss, tint ee, tint c) {
21     ladj[ss].push_back( tint (red.size())); // guardamos el indice
22     red.push_back(Arista(ss,ee,c,0));
23     ladj[ee].push_back( tint (red.size()));
24     red.push_back(Arista(ee,ss,c,0));
25 }
26
27 bool bfs () {
28     forn(i,n+1)
29         nivel[i] = -1;
30     vector<tint> vecinos = {s}, nuevosVecinos;
31     nivel[s] = 0;
32     while (!vecinos.empty() && nivel[t] == -1) {
33         tint actual = vecinos.back();
34         vecinos.pop_back();
35         for (auto iArista : ladj[actual]) {
36             tint vecino = red[iArista].end;
37             // Si bajo en uno el nivel y puedo mandar flujo en la red residual
38             if (nivel[vecino] == -1 && red[iArista].flow < red[iArista].capacity) {
39                 nivel[vecino] = nivel[actual] + 1;
40                 nuevosVecinos.push_back(vecino);
41             }
42         }
43     }
44     if (vecinos.empty()) {
45         swap(vecinos,nuevosVecinos);
46         nuevosVecinos = {};
47     }
48     return (nivel[t] != -1);
49 }
50
51 tint dfs (tint actual, tint flujo) {
52     if (flujo <= 0)
53         return 0;

```

```
54     else if (actual == t)
55         return flujo;
56     else {
57         while (ultimoVecino[actual] < tint(ladj[actual].size())) {
58             tint id = ladj[actual][ultimoVecino[actual]];
59             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 }
```

2.8. Flujo de Costo Mínimo

3. Arboles

3.1. Union-Find (Guty)

```

1  const tint maxN = 131072;
2  vector<tint> caminito;
3  tint representante[maxN];
4  tint tamanho[maxN];
5
6  void inicializar (tint n) {
7      forn(i,n) {
8          representante[i] = i;
9          tamanho[i] = 1;
10     }
11 }
12
13 tint find (tint x) {
14     caminito = {};
15     while (x != representante[x]) {
16         caminito.push_back(x);
17         x = representante[x];
18     }
19     for (auto z : caminito)
20         representante[z] = x;
21     return x;
22 }
23
24 bool same (tint a, tint b) { return (find(a) == find(b)); }
25
26 void unite (tint a, tint b) {
27     a = find(a);
28     b = find(b);
29     if (tamanho[a] < tamanho[b])
30         swap(a,b);
31     tamanho[a] += tamanho[b];
32     representante[b] = a;
33 }

```

3.2. Union-Find (Jonaz)

```

1  class UF {
2  private: vector<int> p, rank; int comps;
3  public:
4      UF(int N) {
5          rank.assign(N, 0); comps = N;
6          p.assign(N, 0); forn(i,N) p[i] = i;
7      }

```

```

8      int findSet(int i) { return (p[i] == i) ? i : (p[i] = findSet(p[i])); }
9      bool sameSet(int i, int j) { return findSet(i) == findSet(j); }
10     void unionSet(int i, int j) {
11         if (!sameSet(i,j)) {
12             int x = findSet(i), y = findSet(j);
13             if (rank[x] > rank[y]) p[y] = x;
14             else {
15                 p[x] = y;
16                 if (rank[x] == rank[y]) rank[y]++;
17             }
18             comps--;
19         }
20     }
21     int components() { return comps; }
22 };

```

3.3. Kruskal (usa UF de Jonaz)

```

1  struct Arista {
2      tint peso, start, end;
3      Arista(tint s, tint e, tint p) : peso(p), start(s), end(e) {}
4      bool operator<(const Arista& o) const {
5          return make_tuple(peso, start, end) < make_tuple(o.peso, o.start, o.end);
6      };
7      // Devuelve el peso del AGM, y en 'agm' deja las aristas del mismo.
8      tint kruskal(vector<Arista> &ars, tint size, vector<Arista> &agm) {
9          sort(ars.begin(), ars.end());
10         tint min_peso = 0;
11         UF uf(size);
12         for(auto &a : ars) {
13             if (!uf.sameSet(a.start, a.end)) {
14                 min_peso += a.peso;
15                 uf.unionSet(a.start, a.end);
16                 agm.push_back(a);
17                 if ((tint)agm.size() == size-1) break; // Esto es que ya tiene V-1 aristas
18             }
19         }
20         return min_peso;
21     }

```

3.4. LCA - Segment Tree (Jonaz)

3.5. Binary Lifting (saltitos potencia de 2)

```

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

```

```

4
5 tint d[maxN]; // profundidad
6 pair<tint,tint> p[maxN][maxK];
7 // {ancestro a distancia 2^k, Lo que queremos entre los 2^k ancestros}
8
9 void dfs(tint actual, vector<vector<pair<tint,tint> > > &ladj, tint padre) {
10     d[actual] = d[padre]+1;
11     for (auto x : ladj[actual])
12         if (x.first != padre) {
13             p[x.first][0] = {actual,x.second};
14             dfs(x.first,ladj,actual);
15         }
16 }
17
18 tint subir(tint a, tint c, tint &ans, bool tomaMinimo) {
19     tint k = 0;
20     while (c > 0) {
21         if (c % 2) {
22             if (tomaMinimo)
23                 ans = min(ans,p[a][k].second);
24             a = p[a][k].first;
25         }
26         k++;
27         c /= 2;
28     }
29     return a;
30 }
31
32 tint answer (tint a, tint b) {
33     // IGUALAMOS LAS ALTURAS
34     if (d[a] < d[b])
35         swap(a,b);
36     tint w = d[a] - d[b], ans = NEUTRO;
37     a = subir(a,w,ans,true);
38
39     // HACEMOS LA BINARY PARA BUSCAR EL LCA
40     tint cInf = 0, cSup = maxN;
41     while (cSup - cInf > 1) {
42         tint ra = a, rb = b;
43         tint c = (cSup+cInf)/2;
44         ra = subir(ra,c,ans,false);
45         rb = subir(rb,c,ans,false);
46         if (ra == rb)
47             cSup = c;
48         else
49             cInf = c;

```

```

50     }
51     // SUBIMOS LO QUE HAGA FALTA PARA LLEGAR AL LCA
52     cSup *= (a != b);
53     a = subir(a,cSup,ans,true);
54     b = subir(b,cSup,ans,true);
55     return ans;
56 }
57
58 // INICIALIZACION
59 int main() {
60     forn(i,maxN)
61         forn(k,maxK)
62             p[i][k] = {-1,NEUTRO};
63     // HACEMOS EL PRIMER PASO EN FUNCION DEL GRAFO
64     vector<vector<pair<tint,tint> > > ladj (maxN); // listaDeAdyacencia del arbol
65     d[0] = -1;
66     dfs(0,ladj,0);
67     // LLENADO DE LA TABLA
68     forsn(k,1,maxK)
69         forn(i,maxN) {
70             tint ancestro = p[i][k-1].first;
71             if (ancestro >= 0)
72                 p[i][k] = {p[ancestro][k-1].first,
73                             min(p[i][k-1].second,p[ancestro][k-1].second) };
74         }
75 }

```


4. Strings

4.1. ToString/ToNumber

```

1 #include <iostream>
2 #include <string>
3 #include <sstream>
4
5 tint toNumber (string s)
6 {
7     tint Number;
8     if ( ! (istringstream(s) >> Number) )
9         Number = 0; // el string vacio lo manda al cero
10    return Number;
11 }
12
13 string toString (tint number)
14 {
15     ostringstream ostr;
16     ostr << number;
17     return ostr.str();
18 }
```

4.2. Tablita de Bordes

```

1 // Complejidad O(N)
2 // Devuelve arreglo, en la posicion i la longitud del maximo borde hasta st[i].
3 // En el ejemplo "abracadabra" devuelve 0 0 0 1 0 1 0 1 2 3 4
4 vector<int> calcularBordes(string st) {
5     int i=1, j=0, n=st.size();
6     vector<int> bordes(n, 0);
7     while(i<n) {
8         while(j>0 && st[i] != st[j])
9             j = bordes[j-1];
10        if (st[i] == st[j])
11            j++;
12        bordes[i++] = j;
13    }
14    return bordes;
15 }
16 // String Matching con Bordes:
17 // Concatenar S+'$'+T (donde $ no aparece en S ni T)
18 // S chico, T grande
19 // Calcular bordes, siempre que el borde maximo para alguna
20 // posicion correspondiente a T sea de longitud |S|
21 // entonces hay un substring en T que coincide con S.
```

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

4.4. Subsecuencia Comun mas larga (Guty o Jonaz)

4.5. Edit-Distance

```

1 // Minima distancia entre strings si lo que se puede es: INSERTAR, REMOVE,
   // MODIFICAR, SWAPS ADYACENTES
2 const tint maxN = 1024; // maximo largo de los strings
3 const tint INFINITO = 1e15;
4 string s1,s2;
5 tint dist[maxN][maxN];
6
7 tint f(tint i, tint j)
8 {
9     // Si un string es vacio, hay que borrar todo el otro
10    if (i == -1 or j == -1)
11        return max(i,j)+1;
12    if (dist[i][j] == INFINITO)
13    {
14        tint mini = INFINITO;
15        // Lo mejor de borrar el i-esimo de s1 o insertar al final de s1 a s2[j]
16        mini = min(mini,min(f(i-1,j)+1,f(i,j-1)+1));
17        if (s1[i] == s2[j]) // Si coinciden, dejo como esta y resuelvo lo anterior
18            mini = min(mini,f(i-1,j-1));
19        else // Modificar s1[i] a s2[j] y resolver lo anterior
20            mini = min(mini,f(i-1,j-1)+1);
21
22        // Borrarnos los intermedios y swapeamos los ultimos 2 si funciona, lo hago y
           // resuelvo lo anterior
23        forn(k,i)
24        {
25            if (i >= 1 && j >= 1 && s1[i] == s2[j-1] && s1[i-k-1] == s2[j])
26                mini = min(mini,f(i-k-2,j-2)+k+1);
27        }
28        dist[i][j] = mini;
29    }
30    return dist[i][j];
31 }
32 // USO:
33 int main()
34 {
35     tint n = s1.size(), m = s2.size();
36     forn(i,n)
37     forn(j,m)
```

```
38     dist[i][j] = INFINITO;
39     cout << f(n-1,m-1) << "\n";
40     return 0;
41 }
```

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

```
1 // Asumo i < j
2 bool esPalindromo (tint i, tint j, vector<vector<tint> > &r, tint n)
3 {
4     if (i+j >= n)
5         return (r[i+j][n-i] - r[i+j][n-j-1]) == j-i+1;
6     else
7         return (r[i+j][j+1] - r[i+j][i]) == j-i+1;
8 }
9 // USO:
10 int main()
11 {
12     tint n = s.size(); // s nuestro string
13     vector<vector<tint> > v (n, vector<tint> (n,0));
14     forn(i,n)
15         forn(j,n)
16             v[i][j] = (s[i] == s[j]);
17     vector<vector<tint> > r (2*n-1,vector<tint> (n+1,0));
18
19     forn(i,2*n-1)
20     {
21         tint sum = 0, x = min(i,n-1), y = 0;
22         if (i >= n)
23             y = i-n+1;
24         forn(j,n)
25         {
26             if (x >= 0 && y < n)
27                 sum += v[x--][y++];
28             r[i][j+1] = sum;
29         }
30     }
31     // Ahora podemos preguntar si es palindromo s[i..j]
32 }
```

4.7. Suffix Array

4.8. Longest Common Prefix

5. Matematica

5.1. PotLog

```

1 | const tint nmod = 1000000007; // o el primo que deseamos
2 | tint potLogMod (tint x, tint y) // Calcula: (x^y) mod nmod
3 | {
4 |     tint ans = 1;
5 |     while (y > 0)
6 |     {
7 |         if (y % 2)
8 |             ans = (x * ans) % nmod;
9 |         x = (x * x) % nmod;
10 |        y /= 2;
11 |    }
12 |    return ans;
13 | }
14 | tint invMod(tint a) // nmod PRIMO. Devuelve b tal que: (a*b) = 1 (mod nmod)
15 | {
16 |     return potLogMod(a,nmod-2);
17 | }
```

5.2. Criba

```

1 | const tint maxN = 1000500;
2 | tint p[maxN + 1] = {1, 1};
3 | tint phi[maxN];
4 |
5 | map<tint,tint> factorizar (tint n)
6 | {
7 |     map<tint,tint> f;
8 |     while (n > 1)
9 |     {
10 |         f[p[n]]++;
11 |         n /= p[n];
12 |     }
13 |     return f;
14 | }
15 | // USO:
16 | int main()
17 | {
18 |     // CRIBA COMUN : (p[n] = mayor primo que divide a n (n >= 2) )
19 |     for (tint i = 1; i <= maxN; ++i)
20 |         if (p[i] == 1)
21 |             for (tint j = i; j <= maxN; j += i)
22 |                 //if (p[j] == 1 or i == 1) // Con esta linea da el menor primo
23 |                     p[j] = i;
```

```

24 | // CALCULA PHI(N): #Coprimos con N
25 | for (tint i = 0; i < maxN; i++)
26 |     phi[i] = i;
27 | for (tint i = 1; i < maxN; i++)
28 |     for (tint j = 2 * i; j < maxN; j += i)
29 |         phi[j] -= phi[i];
30 | return 0;
31 | }
```

5.3. Euclides Extendido (Guty)

```

1 | tint gcd ( int a, int b, int & x, int & y )
2 | {
3 |     if ( a == 0 )
4 |     {
5 |         x = 0 ; y = 1 ;
6 |         return b ;
7 |     }
8 |     tint x1, y1 ;
9 |     tint d = gcd ( b % a, a, x1, y1 ) ;
10 |    x = y1 - ( b / a ) * x1 ;
11 |    y = x1 ;
12 |    return d;
13 | }
14 |
15 | // Nota si gcd(a,m) == 1 => 1 = a*x+m*y => 1 = a*x (mod m)
16 | // 0 sea, que "x" es el inverso :-)
```

5.4. Eliminacion Gaussiana (Código ruso)

```

1 | // Declarar EPS e INF al principio adecuadamente.
2 | tint gauss ( vector < vector < double > > a, vector < double > & ans ) {
3 |     tint n = a. size ( ) ;
4 |     tint m = a[0].size() -1;
5 |     vector<tint> where (m, -1);
6 |     for ( tint col = 0, row = 0 ; col < m && row < n; ++col)
7 |     {
8 |         int sel = row ;
9 |         for ( tint i = row ; i < n ; ++i)
10 |             if ( abs (a[i][col]) > abs (a[sel][col]) )
11 |                 sel = i ;
12 |         if ( abs(a[sel][col]) < EPS )
13 |             continue ;
14 |         for ( tint i = col ; i <= m ; ++i)
15 |             swap (a[sel][i], a[row][i]);
16 |         where [col] = row;
17 |         for ( tint i = 0 ; i < n ; ++i)
```

```

18     if ( i != row )
19     {
20         ldouble c = a[i][col] / a[row][col] ;
21         for ( tint j = col ; j <= m ; ++ j )
22             a[i][j] -= a[row][j] * c;
23     }
24     ++row;
25 }
26
27 ans.assign(m, 0);
28 for ( tint i = 0 ; i < m ; ++ i )
29     if ( where[i] != - 1 )
30         ans [i] = a[where[i]][m] / a [where[i]][i];
31 for ( tint i = 0 ; i < n ; ++ i )
32 {
33     ldouble sum = 0 ;
34     for ( tint j = 0 ; j < m ; ++ j )
35         sum += ans[j]*a[i][j];
36     if ( abs(sum - a[i][m] ) > EPS )
37         return 0 ;
38 }
39
40 for ( int i = 0 ; i < m ; ++ i )
41     if ( where[i] == - 1 )
42         return INF ;
43 return 1 ;
44 }
45
46 // 0 -> No hay solucion
47 // 1 -> Hay solucion unica. La devuelve en "ans"
48 // INF -> Hay infinitas soluciones

```

5.5. Rabin-Miller

```

1 // USA: "PotLog", pero pasandole el modulo como parametro
2 #include <random>
3 const tint semilla = 38532164;
4 mt19937 gen(semilla);
5
6 tint mult(tint a, tint b, tint m)
7 {
8     int largestBit = 0;
9     while( (b >> largestBit) != 0)
10         largestBit++;
11     tint ans = 0;
12     for(tint currentBit = largestBit - 1; currentBit >= 0; currentBit--)
13     {

```

```

14         ans = (ans + ans);
15         if (ans >= m)
16             ans -= m;
17
18         if ( (b >> currentBit) & 1)
19         {
20             ans += a;
21             if (ans >= m)
22                 ans -= m;
23         }
24     }
25     return ans;
26 }
27
28 bool esPrimoRM (tint n)
29 {
30     if (n <= 1)
31         return false;
32     else if (n <= 3)
33         return true;
34     else if (n % 2 == 0)
35         return false;
36     else
37     {
38         uniform_int_distribution<tint> dis(2, n-2);
39         tint kOrig = 0, m = n-1;
40         while (m % 2 == 0)
41         {
42             kOrig++;
43             m /= 2;
44         }
45         bool esPrimo = true;
46         vector<tint> testigos = {2,3,5,7,11,13,17,19,23,29,31,37};
47         for (auto a : testigos)
48         {
49             if (a < n)
50             {
51                 tint b = potLogMod(a,m,n), k = kOrig;
52                 if (b == 1 or b == n-1)
53                     continue;
54                 else
55                 {
56                     forn(j,k)
57                     {
58                         b = mult(b,b,n);
59                         if (b == n-1)

```

```

60         break;
61     else if (b == 1)
62     {
63         esPrimo = false;
64         break;
65     }
66 }
67 if (b != n-1)
68 {
69     esPrimo = false;
70     break;
71 }
72 }
73 }
74 }
75 return esPrimo;
76 }
77 }

```

5.6. Pollard-Rho

```

1 // USA: Rabin-Miller
2 tint gcd (tint a, tint b)
3 {
4     if (a == 0)
5         return b;
6     return gcd (b % a, a);
7 }
8 void factorizar (tint n, map<tint,tint> &f)
9 {
10     while (n > 1)
11     {
12         if (esPrimoRM(n))
13         {
14             f[n]++;
15             n /= n;
16         }
17         else
18         {
19             uniform_int_distribution<tint> dis(1, n-1);
20             tint a = dis(gen), b = dis(gen), x = 2, y = 2, d;
21             do
22             {
23                 x = (mult(x,x,n) + mult(a,x,n) + b) % n;
24                 y = (mult(y,y,n) + mult(a,y,n) + b) % n;
25                 y = (mult(y,y,n) + mult(a,y,n) + b) % n;
26                 d = gcd(abs(x-y),n);

```

```

27     }
28     while (d == 1);
29     if (d != n)
30     {
31         factorizar(d,f);
32         n /= d;
33     }
34 }
35 }
36 }
37 }

```

5.7. FFT

```

1 // USA : "PotLog" e "InvMod" con nmod = mod
2 const tint mod = (1 << 21)*11 + 1 ; // es re primo
3 const tint root = 38;
4 const tint root_1 = 21247462;
5 const tint root_pw = 1 << 21 ; // largo del arreglo
6 /*
7  * const tint mod = 7340033;
8  * const tint root = 5 ;
9  * const tint root_1 = 4404020 ;
10  * const tint root_pw = 1 << 20 ;
11  */
12
13 tint modulo (tint n)
14 {
15     return ((n % mod) + mod) % mod;
16 }
17 void fft (vector <tint> &a, bool invert )
18 {
19     tint n = a. size();
20     for (tint i = 1 , j = 0 ; i < n ; ++ i )
21     {
22         tint bit = n >> 1 ;
23         while(j >= bit)
24         {
25             j -= bit ;
26             bit >>= 1;
27         }
28         j += bit ;
29         if ( i < j )
30             swap (a[i],a[j]);
31     }
32     for (tint len = 2 ; len <= n ; len <= 1)
33     {

```

```

34     tint wlen = root;
35     if (invert)
36         wlen = root_1;
37     for (tint i = len ; i < root_pw ; i <= 1)
38         wlen = modulo(wlen * wlen);
39     for (tint i = 0 ; i < n ; i += len )
40     {
41         tint w = 1 ;
42         forn(j,len/2)
43         {
44             tint u = a[i+j], v = modulo(a[i+j+len/2] * w) ;
45             a[i+j] = modulo(u+v);
46             a[i+j + len/2] = modulo(u - v);
47             w = modulo(w * wlen) ;
48         }
49     }
50 }
51
52 if (invert)
53 {
54     tint nrev = invMod(n);
55     forn(i,n)
56         a[i] = modulo(a[i] * nrev) ;
57 }
58 }
59
60 void multiply (const vector<tint> &a, const vector<tint> &b, vector<tint> &res)
61 {
62     vector<tint> fa(a.begin(), a.end() ), fb(b.begin(), b.end() );
63     tint n = 1 ;
64     while (n < max(tint(a.size()), tint(b.size())))
65         n <<= 1;
66     n <<= 1 ;
67     fa.resize(n), fb.resize(n);
68     fft(fa, false) , fft(fb, false);
69     forn(i,n)
70         fa[i] *= fb[i];
71     fft(fa, true);
72     res = fa;
73 }
74 // USO:
75 int main()
76 {
77     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 }

```

5.8. Regla de Simpson (Integracion Numerica)

```

1 // f (x) una funcion definida
2 ldoube a, b; // extremos de integracion
3 const int N = 1000*1000; // cantidad de nodos en la malla
4 ldoube s = 0; // resultado de la integral
5 ldoube h = (b - a) / N;
6 forn(i,N)
7 {
8     ldoube 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

```

1  const ldouble epsilon = 1e-10;
2  const ldouble pi = acos(-1);
3
4  struct Punto
5  {
6      ldouble x,y;
7      Punto (ldouble xx, ldouble yy)
8      {
9          x = xx;
10         y = yy;
11     }
12     Punto()
13     {
14         x = 0.0;
15         y = 0.0;
16     }
17 };
18 Punto operator + (Punto p1, Punto p2)
19 {
20     return Punto(p1.x+p2.x,p1.y+p2.y);
21 }
22 Punto operator - (Punto p1, Punto p2)
23 {
24     return Punto(p1.x-p2.x,p1.y-p2.y);
25 }
26 Punto operator * (ldouble lambda, Punto p)
27 {
28     return Punto(lambda*p.x, lambda*p.y);
29 }
30 ldouble operator * (Punto p1, Punto p2)
31 {
32     return p1.x*p2.x+p1.y*p2.y;
33 }
34 ldouble operator ^ (Punto p1, Punto p2)
35 {
36     return p1.x*p2.y - p1.y*p2.x;
37 }
38 Punto operator ~ (Punto p)
39 {
40     return Punto(-p.y,p.x);
41 }
42 ldouble norma (Punto p)

```

```

43 {
44     return sqrt(p.x*p.x+p.y*p.y);
45 }
46 bool operator < (Punto p1, Punto p2)
47 {
48     return make_pair(p1.x,p1.y) < make_pair(p2.x,p2.y);
49 }
50 bool operator == (Punto p1, Punto p2)
51 {
52     return ((abs(p1.x-p2.x) < epsilon) && (abs(p1.y-p2.y) < epsilon));
53 }

```

6.2. Area de Poligono

```

1  ldouble areaTriangulo (Punto p1, Punto p2, Punto p3)
2  {
3      return abs((p1-p3)^(p1-p2))/2.0;
4  }
5
6  ldouble areaPoligono(vector<Punto> &polygon)
7  {
8      ldouble area = 0.0;
9      tint n = polygon.size();
10     forn(i,n)
11         area += polygon[i]^polygon[(i+1)%n];
12     return abs(area)/2.0;
13 }

```

6.3. Punto en Poligono

```

1  bool adentroPoligono(vector<Punto> &polygon, Punto p) // polygon EN EL SENTIDO
   DE LAS AGUJAS
2  {
3      bool adentro = true;
4      tint n = polygon.size();
5      forn(i,n)
6          adentro &= (((p-polygon[i])^(p-polygon[(i+1)%n])) < 0);
7      return adentro;
8  }

```

6.4. Interseccion de Segmentos

```

1  struct Segmento
2  {
3      Punto start,end,dir;
4      Segmento (Punto ss, Punto ee)
5      {

```

```

6   start = ss;
7   end = ee;
8   dir = ee-ss;
9   }
10  };
11  // res.second == 0 -> NO HAY INTERSECCION
12  // res.second == 1 -> INTERSECAN EN UN PUNTO (que esta en res.first)
13  // res.second == 2 -> SON COLINEALES E INTERSECAN EN TODO UN SEGMENTO (Da un
    extremo, si queremos el otro, correr otra vez con "otroExtremo" = true)
14  pair<Punto,tint> interSeg (Segmento s1, Segmento s2, bool otroExtremo )
15  {
16      if ((abs(s1.dir ^ s2.dir)) < epsilon) // son colineales
17      {
18          vector<pair<Punto,tint> > aux = {{s1.start - epsilon*s1.dir,1},
19                                           {s1.end + epsilon*s1.dir,1},
20                                           {s2.start - epsilon*s2.dir,2},
21                                           {s2.end + epsilon*s2.dir,2}};
22
23          sort(aux.begin(),aux.end());
24          if (aux[0].second != aux[1].second)
25              return make_pair(aux[1+otroExtremo].first,2);
26          else
27              return make_pair(Punto(),0);
28      }
29      else
30      {
31          ldouble alfa = ((s2.start-s1.start)^s2.dir) / (s1.dir^s2.dir);
32          if (0 <= alfa && alfa <= 1)
33              return make_pair(s1.start+alfa*s1.dir,1);
34          else
35              return make_pair(Punto(),0);
36      }
37  }

```

6.5. Angulo Entre Puntos y Distancia entre Segmentos

```

1  ldouble angEntre (Punto p1, Punto p2, Punto p3) // P1^P2P3
2  {
3      ldouble a = norma(p2-p3);
4      ldouble b = norma(p1-p3);
5      ldouble c = norma(p2-p1);
6      return acos((a*a+c*c-b*b)/(2*a*c));
7  }
8
9  ldouble dPuntoSeg (Punto p, Segmento s)
10 {
11     if (angEntre(p,s.start,s.end) > pi/2 or angEntre(p,s.end,s.start) > pi/2)
12         return min(norma(p-s.start),norma(p-s.end));

```

```

13     else
14         return abs( ((s.start-p)^(s.end-p)) / (norma(s.dir)) );
15 }
16
17 ldouble dEntreSeg(Segmento s1, Segmento s2)
18 {
19     ldouble a = min(dPuntoSeg(s1.start,s2),dPuntoSeg(s1.end,s2));
20     ldouble b = min(dPuntoSeg(s2.start,s1),dPuntoSeg(s2.end,s1));
21     return (interSeg(s1,s2,false).second == 0) * min(a,b);
22 }

```

6.6. Convex-Hull (2D) (Jonaz)

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

6.8. Sweep Line Dificil (Union de Rectángulos)

6.9. Radial Sweep

6.10. Minimum Bounding Circle

7. Varios

7.1. Operaciones de bits

```

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

```

7.2. Longest Increasing 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 90° 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. Sliding 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)

```

```

11 {
12     if (l == rmq.front().second)
13         rmq.pop_front();
14     l++;
15 }
16
17 pair<tint,tint> minimoVentana (deque<pair<tint,tint> > &rmq)
18 {
19     return rmq.front();
20 }
21 // USO: En todo momento tenemos el minimo entre [l,r)
22 int main()
23 {
24     deque<pair<tint,tint> > rmq; // {numero,indice}
25     tint l = 0, r = 0; // l . r
26     vector<tint> v = {1,2,3,4,3,2,2,3,4};
27     agrandarVentana(r,rmq,v);
28     agrandarVentana(r,rmq,v);
29     agrandarVentana(r,rmq,v);
30     agrandarVentana(r,rmq,v);
31     agrandarVentana(r,rmq,v);
32     achicarVentana(l,rmq);
33     achicarVentana(l,rmq);
34     cout << minimoVentana(rmq).first << endl; // {3,4}
35     return 0;
36 }

```

7.7. Ternary Search

```

1 // Ternary en ENTEROS
2 tint miniTernarySearch (tint a, tint b) // En [a,b] esta el minimo
3 {
4     tint l = a, r = b;
5     while (abs(r - l) > 5)
6     {
7         tint al = (2*l + r)/3;
8         tint br = (l + 2*r)/3;
9         if (f(al) > f(br)) // cambiar a "<" para maximo
10             l = al;
11         else
12             r = br;
13     }
14     tint ans = 1e16;
15     forsn(k,l,r+1)
16         ans = min(ans,f(k)); // cambiar por "max" para maximo
17     return ans;
18 }

```

```

19 //Ternary en FLOATING POINT
20 ldouble miniTernarySearch (ldouble tL, ldouble tR) // En [tL, tR] esta el minimo
21 {
22     while (abs(tR - tL) > epsilon)
23     {
24         ldouble tLThird = (2.0*tL + tR)/3.0;
25         ldouble tRThird = (tL + 2.0*tR)/3.0;
26         if (f(tLeftThird) > f(tRightThird)) // cambiar a "<" para maximo
27             tLeft = tLeftThird;
28         else
29             tRight = tRightThird;
30     }
31     return f((tLeft+tRight)/2.0);
32 }

```