

Repositorio en c++

Universidad de la amazonia

19 de septiembre de 2017

Índice

1. Estructuras de datos	2	5. Programacion dinamica	11
1.1. tablas aditivas	2	5.1. Subconjuntos de un conjunto	11
1.2. disjoint set union find	2	5.2. Problema de la mochila	12
1.3. union find con compresion de caminos	2	5.3. Longest Increment Subsequence	12
1.4. segment tree	3	5.4. Max Range Sum	12
2. Grafos	4	5.5. Subset Sum	12
2.1. Dijkstra	4	6. Cadenas	12
2.2. Bellman-Ford	4	6.1. KMP	12
2.3. Floyd Warshall	5	7. Tips and formulas(ufps)	13
2.4. Kosaraju	5	7.1. ASCII Table	13
2.5. Kruskal	6	7.2. Formulas	14
2.6. Topological sort	6	7.3. Sequences	16
3. Matematicas	6	7.4. Time Complexities	17
3.1. MCD y MCM	6	8. Extras	18
3.2. Exponenciacion binaria	7	8.1. Formulas extra	18
3.3. Multiplicacion modular	7	8.2. Secuencias	18
3.4. Exponenciacion modular	7		
3.5. Algoritmo extendido de euclides	7		
3.6. Inverso multiplicativo modular	8		
3.7. Phi de euler	8		
3.8. Rho de pollard	8		
3.9. BigInteger c++	9		
4. Otros	11		
4.1. Busqueda binaria	11		
4.2. Raiz babilonica	11		

1. Estructuras de datos

1.1. tablas aditivas

Construccion $O(n)$

```
1 void build(){
2     //matriz inicial tab, tabla aditiva se construye en tab2
3     memset(tab2, 0, sizeof(tab2));
4     tab2[1][1] = tab[0][0];
5     for (int i = 2; i < 6; i++) tab2[i][1] = tab2[i-1][1] + tab[i -
        1][0];
6     for (int j = 2; j < 6; j++) tab2[1][j] = tab2[1][j-1] + tab[0][j
        - 1];
7
8     for (int i = 2; i < 6; i++)
9     for (int j = 2; j < 6; j++)
10         tab2[i][j] = tab2[i][j - 1] + tab2[i - 1][j] + tab[i -
            1][j - 1] - tab2[i - 1][j - 1];
11     //ejemplo: en matriz de 5*5 buscar acumulado de 2,2 hasta 5,5
12     //tab2[5][5] - tab2[2][5] - tab2[5][2] + tab2[2][2]
13 }
```

1.2. disjoint set union find

Construccion $O(n)$

asocia elementos en conjuntos de arboles.

```
1 struct union_find{
2     int padre[100];
3     int rango[100];
4     vector<int> grupo[100];
5
6     void iniciar(int n){
7         for (int i = 0; i < n; i++) {
8             padre[i] = i;
9             rango[i] = 0;
10            grupo[i].clear();
11            grupo[i].push_back(i);
12        }
13    }
14
15    int raiz(int x){
```

```
16        if(padre[x] == x) return x;
17        return raiz(padre[x]);
18    }
19
20    void unir(int x, int y){
21        x = raiz(x);
22        y = raiz(y);
23        if(x == y) return;
24
25        if(rango[x] > rango[y]){
26            padre[y] = x;
27            grupo[x].insert(grupo[x].begin(), grupo[y].begin(), grupo[y].
                end());
28            grupo[y].clear();
29            return;
30        }
31
32        padre[x] = y;
33        grupo[y].insert(grupo[y].begin(), grupo[x].begin(), grupo[x].end
            ());
34        grupo[x].clear();
35        if(rango[y] == rango[x]) rango[y]++;
36    }
37
38    bool MismoGrupo(int x, int y){
39        return raiz(x) == raiz(y);
40    }
41
42    void grupo_n(int n){
43        cout << "elementos en el grupo de " << n << endl;
44        n = raiz(n);
45        for(int i = 0; i < grupo[n].size(); i++) cout << grupo[n][i] <<
            "\n";
46        cout << endl;
47    }
48 };
```

1.3. union find con compresion de caminos

asocia elementos de manera simple.

```
1 struct union_find{
```

```

2  int padre[100];
3  int rango[100];
4
5  void iniciar(int n){
6      for (int i = 0; i < n; i++) {
7          padre[i] = i;
8          rango[i] = 0;
9      }
10 }
11
12 int raiz(int x){
13     if(x == padre[x] ) return x;
14     else return padre[x] = raiz(padre[x]);
15 }
16
17 void unir(int x, int y){
18     x = raiz(x);
19     y = raiz(y);
20     if(x == y) return;
21
22     if(rango[x] > rango[y]){
23         padre[y] = x;
24         return;
25     }
26
27     padre[x] = y;
28     if(rango[y] == rango[x]) rango[y]++;
29 }
30
31 bool MismoGrupo(int x, int y){
32     return raiz(x) == raiz(y);
33 }
34 };

```

1.4. segment tree

Ejemplo de RMQ (Range Minium Query)
 Contruccion $O(n)$
 Consulta $O(\log n)$
 Update $O(\log n)$

```

1  struct segment_tree{

```

```

2  vector<int> st, A;
3  int n;
4
5  int mov_izq(int index){
6      return index << 1;
7  }
8
9  int mov_der(int index){
10     return (index << 1) + 1;
11 }
12
13 void construir(int pos, int izq, int der){
14     if(izq == der){
15         st[pos] = A[der];
16         return;
17     }
18
19     construir(mov_izq(pos), izq, (izq + der) >> 1);
20     construir(mov_der(pos), ((izq + der) >> 1) + 1, der);
21     int aux1 = mov_izq(pos), aux2 = mov_der(pos);
22     st[pos] = min(st[aux1], st[aux2]);
23 }
24
25 void iniciar(vector<int> arr){//metodo a invocar
26     A = arr;
27     n = A.size();
28     st.assign(n*4, 0);
29     construir(1, 0, n - 1);
30 }
31
32 int rmq(int pos, int izq, int der, int i, int j){
33     if(i > der || j < izq) return -1;
34     if(i <= izq && j >= der) return st[pos];
35
36     int aux1 = rmq(mov_izq(pos), izq, (izq + der) >> 1, i, j);
37     int aux2 = rmq(mov_der(pos), ((izq + der) >> 1) + 1, der, i,
38         j);
39     if(aux1 == -1) return aux2;
40     if(aux2 == -1) return aux1;
41
42     return min(aux1, aux2);

```

```

42     }
43
44     int RMQ(int i, int j){//metodo a invocar
45         return rmq(1, 0, n-1, i, j);
46     }
47
48     int cambiar(int pos, int izq, int der, int index, int nuevo){
49         if(index > der || index < izq) return st[pos];
50         if(der == index && izq == index){
51             A[index] = nuevo;
52             return st[pos] = nuevo;
53         }
54
55         int aux1 = cambiar(mov_izq(pos), izq, (izq + der) >> 1, index
56             , nuevo);
57         int aux2 = cambiar(mov_der(pos), ((izq + der) >> 1) + 1, der,
58             index, nuevo);
59         return st[pos] = min(aux1, aux2);
60     }
61
62     int update(int index, int num){//metodo a invocar
63         return cambiar(1, 0, n-1, index, num);
64     }
65 };

```

2. Grafos

2.1. Dijkstra

Ruta minima $O((n + m)\log n)$

```

1 typedef pair<int, int> ii;//peso, nodo
2 typedef vector<ii> vii;
3 typedef vector<vii> vvii;
4 typedef vector<int> vi;
5 #define inf 1000000000
6
7 vi dijkstra(vvii &grafo, int nodo, int tam){
8     vi dis(tam + 1, inf);
9     priority_queue<ii> cola;
10    cola.push(ii(-0, nodo));

```

```

11    int peso, aux;
12    ii par, par2;
13
14    while(cola.size()){
15        par = cola.top();
16        cola.pop();
17        peso = -par.first;
18        nodo = par.second;
19
20        if(dis[nodo] <= peso) continue;
21        dis[nodo] = peso;
22
23        for(int i = 0; i < grafo[nodo].size(); i++){
24            par2 = grafo[nodo][i];
25            aux = dis[nodo] + par2.first;
26            if(dis[par2.second] > aux){
27                cola.push(ii(-aux, par2.second));
28            }
29        }
30    }
31
32    return dis;
33 }

```

2.2. Bellman-Ford

Ruta minima con pesos negativos $O(n^2)$

```

1 typedef pair<int, int> ii;
2 typedef pair<int, ii> iii;//(peso, nodo padre, nodo hijo)
3 typedef vector<int> vi;
4
5 #define mpdii(a, b, c) iii(a, ii(b, c))
6 #define inf 1000000000
7
8 vector<iii> grafo; //lista de incidencia
9 int padre[10];//opcional
10
11 bool BellmanFord(vector<iii> &lista, int nodos, int inicio, vi &dis){
12
13     for(int i = 0; i < nodos; i++){
14         dis[i] = inf;

```

```

15     padre[i] = i;
16 }
17 dis[inicio] = 0;
18 int aux;
19
20 for (int i = 0; i < nodos; i++)
21     for (int j = 0; j < lista.size(); j++) {
22         aux = dis[lista[j].second.first] + lista[j].first;
23         if (dis[lista[j].second.second] > aux) {
24             dis[lista[j].second.second] = aux;
25             padre[lista[j].second.second] = lista[j].second.first;
26         }
27     }
28
29     for(int j = 0; j < nodos; j++){
30         aux = dis[lista[j].second.first] + lista[j].first;
31         if(dis[lista[j].second.second] > aux)
32             return false;//existe ciclo!!!
33     }
34     return true;
35 }

```

2.3. Floyd Warshall

Ruta minima de toda la matriz, recomendable si $n \leq 100$
 $O(n^3)$

```

1  #define inf 1000000000//10^9
2  using namespace std;
3  vector<vector<int>> matriz(10, vector<int>(10, inf));
4
5  void FloydWarshall(vector<vector<int>> &grafo, int nodos){
6      int aux;
7      //hacemos las diagonales cero
8      for(int i = 0; i < nodos; i++) grafo[i][i] = 0;
9
10     for(int k = 0; k < nodos; k++)
11         for(int i = 0; i < nodos; i++)
12             for(int j = 0; j < nodos; j++){
13                 aux = grafo[i][k] + grafo[k][j];
14                 if(grafo[i][j] > aux) grafo[i][j] = aux;
15             }

```

```

16 }

```

2.4. Kosaraju

Componentes fuertemente conexas grafos si y no dirigidos
 $O(2(n + m))$

```

1  typedef vector<int> vi;
2
3  vector<vi> grafo(5), transpuesto(5), comp;
4  stack<int> pila;
5  bool vis[5];
6
7  void dfs(int n, vector<vi> lista, bool f, vi &grupo){
8      vis[n] = true;
9      if(!f) grupo.push_back(n);
10
11     for (int i = 0; i < lista[n].size(); i++)
12         if (!vis[lista[n][i]]) dfs(lista[n][i], lista, f, grupo);
13
14     if(f) pila.push(n);
15 }
16
17 void kosaraju(){
18     memset(vis, false, sizeof(vis));
19     vi no_se_utiliza;
20     for (int i = 0; i < 5; i++)
21         if(!vis[i]) dfs(i, grafo, true, no_se_utiliza);
22
23     memset(vis, false, sizeof(vis));
24     int n;
25     while(pila.size()) {
26         n = pila.top();
27         pila.pop();
28         if (!vis[n]) {
29             vi vec;
30             dfs(n, transpuesto, false, vec);
31             comp.push_back(vec);
32         }
33     }
34
35     for (int i = 0; i < comp.size(); i++){

```

```

36     for (int j = 0; j < comp[i].size(); j++) cout << comp[i][j] << "␣
        ";
37     cout << endl;
38 }
39 }

```

2.5. Kruskal

Arbol generador minimo, se necesita de un union-find $O(m \log n)$, sin contar el ordenamiento.

```

1  typedef pair<int, int> ii;
2  typedef pair<int, ii> piii; //peso, origen y destino
3  #define mpiii(a, b, c) piii(a, ii(b, c))
4  //ejemplo de insertar: grafo.push_back(mpiii(7, 0, 1))
5  //de nodo 0 a 1 con peso 7.
6
7  vector<piii> grafo; //lista de incidencia
8  union_find arbol;
9
10 int kruskal(vector<piii> lista, int nodos, union_find &uf){
11     sort(lista.begin(), lista.end());
12     uf.iniciar(nodos);
13     int acum = 0, ejes = 0, n = nodos - 1;
14
15     for (int i = 0; i < lista.size(); i++) {
16         if (!uf.MismoGrupo(lista[i].second.first, lista[i].second.second)
17             ) {
18             ejes++;
19             uf.unir(lista[i].second.first, lista[i].second.second);
20             acum += lista[i].first;
21             if(ejes == n) return acum;
22         }
23     }
24     return -1;
25 }

```

2.6. Topological sort

$O(m + n)$

```

1  vi res; //guarda la respuesta
2  vi ent; //cantidad de aristas entrantes de cada nodo
3
4  bool topological_sort(vii &lis, int tam){
5      res.clear();
6      queue<int> s;
7      for(int i = 1; i <= tam; i++){
8          if(!ent[i]) s.push(i);
9      }
10
11     int n, m;
12     while(s.size()){
13         n = s.front();
14         s.pop();
15         res.push_back(n);
16
17         for(int i = 0; i < lis[n].size(); i++){
18             m = lis[n][i];
19             ent[m]--;
20             if(!ent[m]) s.push(m);
21         }
22     }
23     if(res.size() != tam) return false; //contiene ciclo!!!
24     else return true;
25 }

```

3. Matematicas

3.1. MCD y MCM

Maximo comun divisor(MCD) y minimo comun multiplo(MCM)

```

1  int mcd(int a, int b){ //algoritmo de euclides
2      return a? mcd(b % a, a): b;
3  }
4
5  int mcm(int a, int b) {
6      return a*b/mcd(a,b);
7  }

```

3.2. Exponenciacion binaria

$O(\log n)$

```
1 typedef long long int lli;
2
3 lli exp_bin (lli a, lli n) {
4     lli res = 1;
5     while (n) {
6         if (n & 1) res *= a;
7         a *= a;
8         n >>= 1;
9     }
10    return res;
11 }
```

3.3. Multiplicacion modular

Encuentra $(a*b) \bmod c$, la operacion puede generar overflow si se realiza directamente, el metodo mulmod evita el overflow usando un ciclo, pero se puede usar el tipo de dato int128 de c++11 para poder calcular de manera directa, pero el int128 no se puede leer o imprimir directamente.

```
1 typedef long long int lli; //metodo normal
2 lli mulmod (lli a, lli b, lli c) {
3     lli x = 0, y = a%c;
4     while (b > 0){
5         if (b % 2 == 1) x = (x+y) % c;
6         y = (y*2) % c;
7         b /= 2;
8     }
9     return x % c;
10 }
11
12 typedef __int128 bi; //metodo con __int128
13 lli mulmod_2(bi a, bi b, bi c){
14     return (lli) ((a*b) % c);
15 }
16
17 int main(){
18     lli a, b, c;
19     cin >> a >> b >> c;
20     cout << mulmod_2((bi) a, (bi) b, (bi) c) << endl;
```

```
21     return 0;
22 }
```

3.4. Exponenciacion modular

Encuentra $(a^b) \bmod c$, se necesita implementar previamente multiplicacion modular.

```
1 lli expmod (lli b, lli e, lli m){ //O(log b)
2     if(!e) return 1;
3     lli q = expmod(b,e/2,m); q = mulmod(q,q,m);
4     return e%2? mulmod(b,q,m) : q;
5 }
```

3.5. Algoritmo extendido de euclides

Encuentra dos numeros x e y tal que: $\text{MCD}(a, b) = ax + by$

```
1 int gcd_ex (int a, int b, int & x, int & y) {
2     if (a == 0) {
3         x = 0; y = 1;
4         return b;
5     }
6     int x1, y1;
7     int d = gcd_ex (b%a, a, x1, y1);
8     x = y1 - (b / a) * x1;
9     y = x1;
10    return d;
11 }
12
13 int main(){
14     int n, m, x, y, res;
15
16     while(cin >> n >> m){
17         res = gcd_ex(n, m, x, y);
18         cout << "gcd=" << res << ",x=" << x << ",y=" << y <<
19             endl;
20     }
21 }
```

3.6. Inverso multiplicativo modular

Encuentra un x tal que $(a * x)$ es congruente a 1 con modulo p , entonces:
 $(a * x) \bmod p = 1 \bmod p$
necesita del algoritmo extendido de euclides
 $O(\log m)$

```
1 void inverso(int a, int m){
2     int x, y;
3
4     int g = gcd_ex (a, m, x, y);
5     if (g != 1)
6         cout << "no_solution";
7     else {
8         x = (x % m + m) % m;
9         cout << x << endl;
10    }
11 }
```

3.7. Phi de euler

Devuelve la cantidad de coprimos de un numero n
 $O(\sqrt{n})$

```
1 int phi (int n) {
2     int result = n;
3     for (int i=2; i*i<=n; ++i)
4         if (n % i == 0) {
5             while (n % i == 0)
6                 n /= i;
7             result -= result / i;
8         }
9     if (n > 1)
10        result -= result / n;
11    return result;
12 }
```

3.8. Rho de pollard

Factorizacion rapida, requiere de implementar previamente exponenciacion modular, multiplicacion modular y el MCD
 $O(\sqrt[4]{n})$

```
1 bool es_primo_prob (ll n, int a) {
2     if (n == a) return true;
3     ll s = 0, d = n-1;
4     while (d % 2 == 0) s++, d/=2;
5
6     ll x = expmod(a,d,n);
7     if ((x == 1) || (x+1 == n)) return true;
8
9     for(int i = 0; i < s-1; i++){
10        x = mulmod(x, x, n);
11        if (x == 1) return false;
12        if (x+1 == n) return true;
13    }
14    return false;
15 }
16
17 bool rabin (ll n){ //devuelve true si n es primo
18     if (n == 1) return false;
19     const int ar[] = {2,3,5,7,11,13,17,19,23};
20     for(int j = 0; j < 9; j++)
21         if (!es_primo_prob(n,ar[j]))
22             return false;
23     return true;
24 }
25
26 ll rho(ll n){
27     if( (n & 1) == 0 ) return 2;
28     ll x = 2 , y = 2 , d = 1;
29     ll c = rand() % n + 1;
30     while( d == 1 ){
31         x = (mulmod( x , x , n ) + c) % n;
32         y = (mulmod( y , y , n ) + c) % n;
33         y = (mulmod( y , y , n ) + c) % n;
34         if( x - y >= 0 ) d = mcd( x - y , n );
35         else d = mcd( y - x , n );
36     }
37     return d==n? rho(n):d;
38 }
39
40 map<ll, ll> prim;
41
```



```

42 void factRho (ll n){
43     if (n == 1) return;
44     if (rabin(n)){
45         prim[n]++; //se agrega el primo n a la solucion
46         return;
47     }
48     ll factor = rho(n);
49     factRho(factor);
50     factRho(n/factor);
51 }
52
53 int main(){
54     ll n;
55     while(scanf("%lld", &n), n > 0){
56         prim.clear();
57         factRho(n);
58
59         for(map<ll, ll>::iterator it = prim.begin(); it != prim.end()
60             ; it++){
61             cout << "el_" << (it)->first << " aparece_" << (it)->
62                 second << " veces\n";
63         }
64     }
65     return 0;
66 }

```

3.9. BigInteger c++

```

1  #include <bits/stdc++.h>
2
3  using namespace std;
4  typedef vector<int> vi;
5
6  struct biginteger{
7      vi num;
8
9      void iniciar(string c){
10         num.clear();
11         int tam = c.length();
12         for(int i = tam - 1; i > -1; i--) num.push_back(c[i] - '0');
13         quitar_zeros_izq();

```

```

14     }
15
16     void iniciar(int c){
17         num.clear();
18         while(c > 0){
19             num.push_back(c % 10);
20             c /= 10;
21         }
22     }
23
24     void imprimir(){
25         for(int i = num.size() - 1; i > -1; i--) printf("%d", num[i]);
26         printf("\n");
27     }
28
29     void quitar_zeros_izq(){
30         int q = num.size();
31         while(q > 1 && !num[--q]) num.pop_back();
32     }
33
34     biginteger suma(biginteger b2){
35         vi b = b2.num;
36         biginteger res;
37         res.num.assign(num.begin(), num.end());
38         int aux = 0, pos = b.size(), tam = num.size();
39
40         for(int i = 0; i < pos; i++){
41             if(i < tam){
42                 aux += res.num[i] + b[i];
43                 res.num[i] = aux % 10;
44             }else{
45                 aux += b[i];
46                 res.num.push_back(aux % 10);
47             }
48             aux /= 10;
49         }
50
51         while(aux > 0){
52             if(pos >= tam)
53                 res.num.push_back(aux % 10);
54             else{

```

```

55     aux += res.num[pos];
56     res.num[pos++] = aux % 10;
57 }
58 aux /= 10;
59 }
60 res.quitar_zeros_izq();
61 return res;
62 }
63
64 biginteger multiplicar(bigInteger b2) {
65     vi y = b2.num;
66     int n = num.size() , m = y.size(), aux = 0, l = n - 1;
67     biginteger res;
68     res.num.assign(n + m - 1, 0);
69
70     for(int i = 0; i < n; i++){
71         for(int j = 0; j < m; j++){
72             if(i != l)
73                 res.num[i + j] += (num[i] * y[j]);
74             else{
75                 aux += res.num[i + j] + (num[i] * y[j]);
76                 res.num[i + j] = aux % 10;
77                 aux /= 10;
78             }
79         }
80         if(i != l){
81             aux += res.num[i];
82             res.num[i] = aux % 10;
83             aux /= 10;
84         }
85     }
86
87     while(aux){
88         res.num.push_back(aux % 10);
89         aux /= 10;
90     }
91     res.quitar_zeros_izq();
92     return res;
93 }
94
95 biginteger resta(bigInteger b2){//asumimos que b2 es menor

```

```

96     vi x = b2.num;
97     biginteger res;
98     res.num.assign(num.begin(), num.end());
99     int i;
100
101     for(i = 0; i < x.size(); i++){
102         if(x[i] > res.num[i]){
103             res.num[i] += 10;
104             res.num[i + 1]--;
105         }
106         res.num[i] -= x[i];
107     }
108     while(res.num[i] < 0){
109         res.num[i++] += 10;
110         res.num[i]--;
111     }
112     res.quitar_zeros_izq();
113     return res;
114 }
115
116 int comparar(bigInteger b){//1 mayor, 0 igual, -1 menor
117     if(num.size() > b.num.size()) return 1;
118     else
119         if(num.size() < b.num.size()) return -1;
120     else{
121         for(int i = num.size() - 1; i > -1; i--){
122             if(num[i] > b.num[i]) return 1;
123             else if(num[i] < b.num[i]) return -1;
124         }
125     }
126     return 0;
127 }
128 };
129 typedef biginteger bigint;
130
131 bigint operator+(bigint &x, bigint &y){return x.suma(y);}
132 bigint operator-(bigint &x, bigint &y){return x.resta(y);}
133 bigint operator*(bigint &x, bigint &y){return x.multiplicar(y);}
134
135 bigint operator+=(bigint &x, bigint &y){return x = x.suma(y);}
136 bigint operator-=(bigint &x, bigint &y){return x = x.resta(y);}

```

```

137 bigint operator*=(bigint &x, bigint &y){return x = x.multiplicar(y);}
138
139 bool operator<(bigint &x, bigint &y){return x.comparar(y) == -1;}
140 bool operator>(bigint &x, bigint &y){return x.comparar(y) == 1;}
141 bool operator==(bigint &x, bigint &y){ return x.comparar(y)
    == 0;}
142 bool operator<=(bigint &x, bigint &y){
143     int q = x.comparar(y);
144     return q == -1 || q == 0;
145 }
146 bool operator>=(bigint &x, bigint &y){
147     int q = x.comparar(y);
148     return q == 0 || q == 1;
149 }
150
151 int main(){
152     string n, m;
153     bigint b1, b2;
154
155     while(cin >> n >> m){
156         b1.iniciar(n);
157         b2.iniciar(m);
158         b1 = b1 * b2;
159         b1.imprimir();
160     }
161 }

```

4. Otros

4.1. Búsqueda binaria

$O(\log n)$

```

1 int f(int a, int b){
2     return ar[a] > b;
3 }
4
5 int busqueda_binaria(int min, int max, int v){
6     int epsilon = 1, med = 0;
7
8     while(max-min > epsilon){
9         med = (max+min)/2;
10        if(f(med,v))

```

```

11        max = med;
12        else
13            min = med;
14    }
15    return min;
16 }

```

4.2. Raiz babilonica

Encuentra la raíz cuadrada de un numero

```

1 double raiz(double x) {
2     double b = x, h = 0, apro = 1;
3     while (apro > 1e-8) {
4         b = (h + b) / 2;
5         h = x / b;
6         apro = abs(h - b);
7     }
8     return b;
9 }

```

5. Programacion dinamica

5.1. Subconjuntos de un conjunto

$O(2^n)$

```

1 void mask(int n, int ar[]){
2     int l = 1 << n;
3
4     for(int i = 0; i < l; i++){
5         for(int j = 0; j < n; j++){
6             if(i & (1 << j)){
7                 printf("%d_", ar[j]);
8             }
9         }
10        printf("\n");
11    }
12 }

```

5.2. Problema de la mochila

```
1 vi ben;//beneficio
2 vi cos;//costo
3 int knapsack(int cap, vi &cos, vi &ben, int n) {
4     int dp[n+1][cap+1];
5
6     for(int i = 0; i <= n; i++){
7         for(int j = 0; j <= cap; j++){
8             if(i == 0 || j == 0) dp[i][j] = 0;
9             else if(cos[i - 1] <= j)
10                 dp[i][j] = max(ben[i - 1] + dp[i - 1][j - cos[i - 1]], dp[i - 1][j]);
11             else
12                 dp[i][j] = dp[i - 1][j];
13         }
14     }
15     return dp[n][cap];
16 }
```

5.3. Longest Increment Subsequence

Subsecuencia creciente mas larga

$O(n \log n)$

```
1 int LIS(int arr[]){
2     int res = 0;
3     vector<int> vec(1, 1);
4
5     for(int i = 0; i < 1; i++){
6         for(int j = i + 1; j < 1; j++){
7             if(arr[i] < arr[j]) vec[j]=max(vec[j], vec[i]+1);
8             res = max(res, vec[i]);
9         }
10     }
11
12     return res;
13 }
```

5.4. Max Range Sum

$O(n)$

```
1 int main(){
```

```
2     int n, num, res, aux;
3
4     while(scanf("%d", &n), n){
5         res = aux = 0;
6         for(int i = 0; i < n; i++){
7             scanf("%d", &num);
8             aux += num;
9             res = max(aux, res);
10            if(aux < 0) aux = 0;
11        }
12
13        if(res > 0) printf("MRS_=%d\n", res);
14        else printf("negativo.\n");
15    }
16    return 0;
17 }
```

5.5. Subset Sum

```
1 bool dp[5][50]; //filas cantidad de numeros
2 //columnas rango maximo a evaluar
3
4 void pre(vi &num){
5     memset(dp, false, sizeof(dp));
6
7     for(int i = 0; i < num.size(); i++){
8         if(i) for(int j = 1; j < 50; j++){
9             if(dp[i - 1][j]) dp[i][j + num[i]] = true;
10
11             dp[i][num[i]] = true;
12         }
13     }
14 }
```

6. Cadenas

6.1. KMP

Encuentra si una cadena es subcadena de otra

```
1 string cad, pat;
2 int tabla[1000];
3
```

```

4 void preCalcular(){
5     int i = 0, j = -1;
6     tabla[0] = -1;
7
8     while(i < pat.length()){
9         while(j >= 0 && pat[i] != pat[j]) j = tabla[j];
10        i++;
11        j++;
12        tabla[i] = j;
13    }
14 }
15
16 void kmp(){
17     int i = 0, j = 0;
18     while(i < cad.length()){
19         while(j >= 0 && cad[i] != pat[j]) j = tabla[j];
20        i++;
21        j++;
22        if(j == pat.length()){
23            printf("%s_esta_en_el_indice_%d_de_la_cadena:_%s\n", pat.
24                c_str(), i - j, cad.c_str());
25            j = tabla[j];
26        }
27    }
28 }

```

7. Tips and formulas(ufps)

7.1. ASCII Table

Caracteres ASCII con sus respectivos valores numéricos.

No.	ASCII	No.	ASCII
0	NUL	16	DLE
1	SOH	17	DC1
2	STX	18	DC2
3	ETX	19	DC3
4	EOT	20	DC4
5	ENQ	21	NAK
6	ACK	22	SYN
7	BEL	23	ETB
8	BS	24	CAN
9	TAB	25	EM
10	LF	26	SUB
11	VT	27	ESC
12	FF	28	FS
13	CR	29	GS
14	SO	30	RS
15	SI	31	US

No.	ASCII	No.	ASCII
32	(space)	48	0
33	!	49	1
34	"	50	2
35	#	51	3
36	\$	52	4
37	%	53	5
38	&	54	6
39	'	55	7
40	(56	8
41)	57	9
42	*	58	:
43	+	59	;
44	,	60	i
45	-	61	=
46	.	62	¿
47	/	63	?

No.	ASCII	No.	ASCII
64	@	80	P
65	A	81	Q
66	B	82	R
67	C	83	S
68	D	84	T
69	E	85	U
70	F	86	V
71	G	87	W
72	H	88	X
73	I	89	Y
74	J	90	Z
75	K	91	[
76	L	92	\
77	M	93]
78	N	94	^
79	O	95	-

No.	ASCII	No.	ASCII
96	`	112	p
97	a	113	q
98	b	114	r
99	c	115	s
100	d	116	t
101	e	117	u
102	f	118	v
103	g	119	w
104	h	120	x
105	i	121	y
106	j	122	z
107	k	123	{
108	l	124	
109	m	125	}
110	n	126	~
111	o	127	

7.2. Formulas

PERMUTACIÓN Y COMBINACIÓN	
Combinación (Coeficiente Binomial)	Número de subconjuntos de k elementos escogidos de un conjunto con n elementos. $\binom{n}{k} = \binom{n}{n-k} = \frac{n!}{k!(n-k)!}$
Combinación con repetición	Número de grupos formados por n elementos, partiendo de m tipos de elementos. $CR_m^n = \binom{m+n-1}{n} = \frac{(m+n-1)!}{n!(m-1)!}$
Permutación	Número de formas de agrupar n elementos, donde importa el orden y sin repetir elementos $P_n = n!$
Permutación múltiple	Elegir r elementos de n posibles con repetición n^r
Permutación con repetición	Se tienen n elementos donde el primer elemento se repite a veces , el segundo b veces , el tercero c veces, \dots $PR_n^{a,b,c,\dots} = \frac{P_n}{a!b!c!\dots}$
Permutaciones sin repetición	Número de formas de agrupar r elementos de n disponibles, sin repetir elementos $\frac{n!}{(n-r)!}$
DISTANCIAS	

Continúa en la siguiente columna

Distancia Euclidean	$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
Distancia Manhattan	$d_M(P_1, P_2) = x_2 - x_1 + y_2 - y_1 $
CIRCUNFERENCIA Y CÍRCULO	
Considerando r como el radio, α como el ángulo del arco o sector, y (R, r) como radio mayor y menor respectivamente.	
Área	$A = \pi * r^2$
Longitud	$L = 2 * \pi * r$
Longitud de un arco	$L = \frac{2 * \pi * r * \alpha}{360}$
Área sector circular	$A = \frac{\pi * r^2 * \alpha}{360}$
Área corona circular	$A = \pi(R^2 - r^2)$
TRIÁNGULO	
Considerando b como la longitud de la base, h como la altura, letras minúsculas como la longitud de los lados, letras mayúsculas como los ángulos, y r como el radio de circunferencias asociadas.	
Área conociendo base y altura	$A = \frac{1}{2}b * h$

Continúa en la siguiente columna

Área conociendo 2 lados y el ángulo que forman	$A = \frac{1}{2}b * a * \sin(C)$
Área conociendo los 3 lados	$A = \sqrt{p(p-a)(p-b)(p-c)}$ con $p = \frac{a+b+c}{2}$
Área de un triángulo circunscrito a una circunferencia	$A = \frac{abc}{4r}$
Área de un triángulo inscrito a una circunferencia	$A = r(\frac{a+b+c}{2})$
Área de un triángulo equilátero	$A = \frac{\sqrt{3}}{4}a^2$
RAZONES TRIGONOMÉTRICAS	
Considerando un triángulo rectángulo de lados a, b y c , con vértices A, B y C (cada vértice opuesto al lado cuya letra minúscula coincide con el) y un ángulo α con centro en el vertice A . a y b son catetos, c es la hipotenusa:	
$\sin(\alpha) = \frac{\text{cateto opuesto}}{\text{hipotenusa}} = \frac{a}{c}$	

Continúa en la siguiente columna

$\cos(\alpha) = \frac{\text{cateto adyacente}}{\text{hipotenusa}} = \frac{b}{c}$	
$\tan(\alpha) = \frac{\text{cateto opuesto}}{\text{cateto adyacente}} = \frac{a}{b}$	
$\sec(\alpha) = \frac{1}{\cos(\alpha)} = \frac{c}{b}$	
$\csc(\alpha) = \frac{1}{\sin(\alpha)} = \frac{c}{a}$	
$\cot(\alpha) = \frac{1}{\tan(\alpha)} = \frac{b}{a}$	
PROPIEDADES DEL MÓDULO (RESIDUO)	
Propiedad neutro	$(a \% b) \% b = a \% b$
Propiedad asociativa en multiplicación	$(ab) \% c = ((a \% c)(b \% c)) \% c$
Propiedad asociativa en suma	$(a + b) \% c = ((a \% c) + (b \% c)) \% c$
CONSTANTES	
Pi	$\pi = \text{acos}(-1) \approx 3,14159$

Continúa en la siguiente columna

e	$e \approx 2,71828$
Número áureo	$\phi = \frac{1 + \sqrt{5}}{2} \approx 1,61803$

7.3. Sequences

Listado de secuencias mas comunes y como hallarlas.

Estrellas octangulares	0, 1, 14, 51, 124, 245, 426, 679, 1016, 1449, 1990, 2651, ...
	$f(n) = n * (2 * n^2 - 1).$
Euler totient	1, 1, 2, 2, 4, 2, 6, 4, 6, 4, 10, 4, 12, 6,...
	$f(n)$ = Cantidad de números naturales $\leq n$ coprimos con n.
Números de Bell	1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, ...
	Se inicia una matriz triangular con $f[0][0] = f[1][0] = 1$. La suma de estos dos se guarda en $f[1][1]$ y se traslada a $f[2][0]$. Ahora se suman $f[1][0]$ con $f[2][0]$ y se guarda en $f[2][1]$. Luego se suman $f[1][1]$ con $f[2][1]$ y se guarda en $f[2][2]$ trasladandose a $f[3][0]$ y así sucesivamente. Los valores de la primera columna contienen la respuesta.
Números de Catalán	1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, ...
	$f(n) = \frac{(2n)!}{(n+1)!n!}$
Números de Fermat	3, 5, 17, 257, 65537, 4294967297, 18446744073709551617, ...
	$f(n) = 2^{(2^n)} + 1$
Números de Fibonacci	0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...
	$f(0) = 0; f(1) = 1; f(n) = f(n-1) + f(n-2)$ para $n > 1$

Continúa en la siguiente columna

Números de Lucas	2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, ...
	$f(0) = 2; f(1) = 1; f(n) = f(n-1) + f(n-2)$ para $n > 1$
Números de Pell	0, 1, 2, 5, 12, 29, 70, 169, 408, 985, 2378, 5741, 13860, ...
	$f(0) = 0; f(1) = 1; f(n) = 2f(n-1) + f(n-2)$ para $n > 1$
Números de Tribonacci	0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, 274, 504, ...
	$f(0) = f(1) = 0; f(2) = 1; f(n) = f(n-1) + f(n-2) + f(n-3)$ para $n > 2$
Números factoriales	1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880, ...
	$f(0) = 1; f(n) = \prod_{k=1}^n k$ para $n > 0$.
Números piramidales cuadrados	0, 1, 5, 14, 30, 55, 91, 140, 204, 285, 385, 506, 650, ...
	$f(n) = \frac{n * (n+1) * (2 * n + 1)}{6}$
Números primos de Mersenne	3, 7, 31, 127, 8191, 131071, 524287, 2147483647, ...
	$f(n) = 2^{p(n)} - 1$ donde p representa valores primos iniciando en $p(0) = 2$.
Números tetraedrales	0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, ...
	$f(n) = \frac{n * (n+1) * (n+2)}{6}$
Números triangulares	0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66, 78, 91, 105, ...
	$f(n) = \frac{n(n+1)}{2}$

Continúa en la siguiente columna

OEIS A000127	1, 2, 4, 8, 16, 31, 57, 99, 163, 256, 386, 562, ...
	$f(n) = \frac{(n^4 - 6n^3 + 23n^2 - 18n + 24)}{24}$.
Secuencia de Narayana	1, 1, 1, 2, 3, 4, 6, 9, 13, 19, 28, 41, 60, 88, 129, ...
	$f(0) = f(1) = f(2) = 1; f(n) = f(n-1) + f(n-3)$ para todo $n > 2$.
Secuencia de Silvestre	2, 3, 7, 43, 1807, 3263443, 10650056950807, ...
	$f(0) = 2; f(n+1) = f(n)^2 - f(n) + 1$
Secuencia de vendedor perezoso	1, 2, 4, 7, 11, 16, 22, 29, 37, 46, 56, 67, 79, 92, 106, ...
	Equivale al triangular(n) + 1. Máxima número de piezas que se pueden formar al hacer n cortes a un disco. $f(n) = \frac{n(n+1)}{2} + 1$
Suma de los divisores de un número	1, 3, 4, 7, 6, 12, 8, 15, 13, 18, 12, 28, 14, 24, ...
	Para todo $n > 1$ cuya descomposición en factores primos es $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$ se tiene que: $f(n) = \frac{p_1^{a_1+1} - 1}{p_1 - 1} * \frac{p_2^{a_2+1} - 1}{p_2 - 1} * \dots * \frac{p_k^{a_k+1} - 1}{p_k - 1}$

7.4. Time Complexities

Aproximación del mayor número n de datos que pueden procesarse para cada una de las complejidades algoritmicas. Tomar esta tabla solo como referencia.

Complexity	n
$O(n!)$	11
$O(n^5)$	50
$O(2^n * n^2)$	18
$O(2^n * n)$	22
$O(n^4)$	100
$O(n^3)$	500
$O(n^2 \log_2 n)$	1.000
$O(n^2)$	10.000
$O(n \log_2 n)$	10^6
$O(n)$	10^8
$O(\sqrt{n})$	10^{16}
$O(\log_2 n)$	-

$O(1)$

-

8. Extras

8.1. Formulas extra

formula de triangulos degenerados:

$$\frac{(a+b-c)*(a+c-b)*(b+c-a)}{a*b*c}$$

Si el resultado es mayor que 0.5, es posible formar el triangulo.

Ecuacion de la recta que pasa por dos puntos:

$$\frac{x-x_1}{x_2-x_1} = \frac{y-y_1}{y_2-y_1}$$

Distancia de un punto a una recta:

Teniendo una recta con formula de la forma: $ax + by + c$ la distancia minima a un punto p de la forma (x, y) la distancia minima esta dada por la formula:

$$d = \frac{ax+by+c}{\sqrt{a^2+b^2}}$$

Formula de numeros fibonacci:

$$f(n) = \frac{1}{\sqrt{5}} * \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$$

8.2. Secuencias

Primos:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103
107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211
223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331
337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449
457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587
593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709
719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853
857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991
997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093
1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217
1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319
1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453
1459 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567
1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669
1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801
1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933
1949 1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063
2069 2081 2083 2087 2089 2099 2111 2113 2129 2131 2137 2141 2143 2153 2161 2179
2203 2207 2213 2221 2237 2239 2243 2251 2267 2269 2273 2281 2287 2293 2297 2309
2311 2333 2339 2341 2347 2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417
2423 2437 2441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 2557
2579 2591 2593 2609 2617 2621 2633 2647 2657 2659 2663 2671 2677 2683 2687 2689
2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797
2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897 2903 2909 2917 2927

Fibonacci:

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269
2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155
165580141 267914296 433494437 701408733 1134903170 1836311903

Factoriales:

1 2 6 24 120 720 5040 40320 362880 3628800 39916800 479001600 6227020800
87178291200 1307674368000 20922789888000 355687428096000 6402373705728000
121645100408832000

Potencias de dos: de 1 hasta 63

1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768
65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432	67108864	134217728	268435456	536870912	1073741824	2147483648
4294967296	8589934592	17179869184	34359738368	68719476736	137438953472	274877906944	549755813888	1099511627776	2199023255552	4398046511104	8796093022208	17592186044416	35184372088832	70368744177664	140737488355328
281474976710656	562949953421312	1125899906842624	2251799813685248	4503599627370496	9007199254740992	18014398509481984	36028797018963968	72057594037927936	144115188075855872	288230376151711744	576460752303423488	1152921504606846976	2305843009213693952	4611686018427387904	9223372036854775808