Repositorio en c++

Universidad de la amazonia

29 de agosto de 2017

Índice

1. Estructuras de datos

1.1. tablas aditivas

Construccion O(n)

```
void build(){
    //matriz inicial tab, tabla aditiva se construye en tab2
    memset(tab2, 0, sizeof(tab2));
    tab2[1][1] = tab[0][0];
    for (int i = 2; i < 6; i++) tab2[i][1] = tab2[i-1][1] + tab[
        i - 1][0];
    for (int j = 2; j < 6; j++) tab2[1][j] = tab2[1][j-1] + tab
        [0][j - 1];

    for (int i = 2; i < 6; i++)
    for (int j = 2; j < 6; j++)
            tab2[i][j] = tab2[i][j - 1] + tab2[i - 1][j] + tab[i
                - 1][j - 1] - tab2[i - 1][j - 1];
    //ejemplo: en matriz de 5*5 buscar acumulado de 2,2 hasta
        5,5
    //tab2[5][5] - tab2[2][5] - tab2[5][2] + tab2[2][2]
}
```

1.2. disjoint set union find

Construccion O(n)

```
struct union_find{
  int padre[100];
  int rango[100];
  vector<int> grupo[100];

  void iniciar(int n){
    for (int i = 0; i < n; i++)
    {
      padre[i] = i;
      rango[i] = 0;
      grupo[i].clear();
      grupo[i].push_back(i);
    }
  }

  int raiz(int x){
    if(padre[x] == x) return x;
    return raiz(padre[x]);
  }

  void unir(int x, int y){
    x = raiz(x);
    y = raiz(y);
    if(x == y) return;

    if(rango[x] > rango[y]){
      padre[y] = x;
      grupo[x].insert(grupo[x].begin(), grupo[y].begin(), grupo[
          y].end());
      grupo[y].clear();
      return;
```

```
    }

    padre[x] = y;
    grupo[y].insert(grupo[y].begin(), grupo[x].begin(), grupo[x
        ].end());
        grupo[x].clear();
    if(rango[y] == rango[x]) rango[y]++;
  }

  bool MismoGrupo(int x, int y){
    return raiz(x) == raiz(y);
  }


  void grupo_n(int n){
      cout << "elementos en el grupo de " << n << endl;
      n = raiz(n);
      for(int i = 0; i < grupo[n].size(); i++) cout << grupo[n][
          i] << " ";
      cout << endl;
  }
};
```

1.3. union find con compresion de caminos

```
struct union_find{
  int padre[100];
  int rango[100];

void iniciar(int n){
  for (int i = 0; i < n; i++) {
    padre[i] = i;
    rango[i] = 0;
  }
}

  int raiz(int x){
      if(x == padre[x] ) return x;
      else return padre[x] = raiz(padre[x]);
  }

void unir(int x, int y){
  x = raiz(x);
  y = raiz(y);
  if(x == y) return;

  if(rango[x] > rango[y]){
    padre[y] = x;
    return;
  }

  padre[x] = y;
  if(rango[y] == rango[x]) rango[y]++;
}

bool MismoGrupo(int x, int y){
  return raiz(x) == raiz(y);
```

```
  }
};
```

1.4. segment tree

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

```
struct segment_tree{
    vi st, A;
    int n;

    int mov_izq(int index){
        return index << 1;
    }

    int mov_der(int index){
        return (index << 1) + 1;
    }

    void construir(int pos, int izq, int der){
        if(izq == der){
            st[pos] = A[der];
            return;
        }

        construir(mov_izq(pos), izq, (izq + der) >> 1);
        construir(mov_der(pos), ((izq + der) >> 1) + 1, der);
        int aux1 = mov_izq(pos), aux2 = mov_der(pos);
        st[pos] = min(st[aux1], st[aux2]);
    }

    void iniciar(vi arr){//metodo a invocar
        A = arr;
        n = A.size();
        st.assign(n*4, 0);
        construir(1, 0, n - 1);
    }

    int rmq(int pos, int izq, int der, int i, int j){
        if(i > der || j < izq) return -1;
        if(i <= izq && j >= der) return st[pos];

        int aux1 = rmq(mov_izq(pos), izq, (izq + der) >> 1, i, j
            );
        int aux2 = rmq(mov_der(pos), ((izq + der) >> 1) + 1, der
            , i, j);
        if(aux1 == -1) return aux2;
        if(aux2 == -1) return aux1;

        return min(aux1, aux2);
    }

    int RMQ(int i, int j){//metodo a invocar
        return rmq(1, 0, n-1, i, j);
    }
```

```
    int cambiar(int pos, int izq, int der, int index, int nuevo)
        {
        if(index > der || index < izq) return st[pos];
        if(der == index && izq == index){
            A[index] =  nuevo;
            return st[pos] = nuevo;
        }

        int aux1 = cambiar(mov_izq(pos), izq, (izq + der) >> 1,
            index, nuevo);
        int aux2 = cambiar(mov_der(pos), ((izq + der) >> 1) + 1,
             der, index, nuevo);
        return st[pos] = min(aux1, aux2);
    }

    int update(int index, int num){//metodo a invocar
        return cambiar(1, 0, n-1, index, num);
    }
};
```

2. Grafos

2.1. Dijkstra

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

```
typedef pair<int, int> ii;//peso, nodo
typedef vector<ii> vii;
typedef vector<vii> vvii;
typedef vector<int> vi;

vi dijkstra(vvii &grafo, int nodo, int tam){
    vi dis(tam + 1, inf);
    priority_queue<ii> cola;
    cola.push(ii(-0, nodo));
    int peso, aux;
    ii par, par2;

    while(cola.size()){
        par = cola.top();
        cola.pop();
        peso = -par.first;
        nodo = par.second;

        if(dis[nodo] <= peso) continue;
        dis[nodo] = peso;

        for(int i = 0; i < grafo[nodo].size(); i++){
            par2 = grafo[nodo][i];
            aux = dis[nodo] + par2.first;
            if(dis[par2.second] > aux){
                cola.push(ii(-aux, par2.second));
            }
        }
    }

    return dis;
```

```
}
```

2.2. Bellman-Ford

Ruta minima con pesos negativos O(n^2)

```
typedef pair<int, int> ii;
typedef pair<int, ii> iii;//(peso, nodo padre, nodo hijo)
typedef vector<int> vi;

#define mpdii(a, b, c) iii(a, ii(b, c))
#define inf 1000000000

vector<iii> grafo; //lista de incidencia
int padre[10];//opcional

bool BellmanFord(vector<iii> &lista, int nodos, int inicio,
    vector<int> &dis){

  for(int i = 0; i < nodos; i++){
    dis[i] = inf;
    padre[i] = i;
  }
  dis[inicio] = 0;
  int aux;

  for (int i = 0; i < nodos; i++)
    for (int j = 0; j < lista.size(); j++)
    {
      aux = dis[lista[j].second.first] + lista[j].first;
      if (dis[lista[j].second.second] > aux)
      {
        dis[lista[j].second.second] = aux;
        padre[lista[j].second.second] = lista[j].second.first;
      }
    }

    for(int j = 0; j < nodos; j++){
      aux = dis[lista[j].second.first] + lista[j].first;
      if(dis[lista[j].second.second] > aux)
                return false;//existe ciclo!!!
    }
  return true;
}
```

2.3. Floyd Warshall

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

```
#define inf 1000
using namespace std;
vector<vector<int>> matriz(10, vector<int>(10, inf));

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

```
typedef pair<int, int> ii;
typedef pair<int, ii> piii;//peso, origen y destino
#define mpiii(a, b, c) piii(a, ii(b, c))
//ejemplo de insertar:
//grafo.push_back(mpiii(7, 0, 1))

vector<piii> grafo;//lista de incidencia
union_find arbol;

int kruskal(vector<piii> lista, int nodos, union_find &uf){
  sort(lista.begin(), lista.end());
  uf.iniciar(nodos);
  int acum = 0, ejes = 0, n = nodos - 1;

  for (int i = 0; i < lista.size(); i++)
  {
    if (!uf.MismoGrupo(lista[i].second.first, lista[i].second.
        second))
    {
      ejes++;
      uf.unir(lista[i].second.first, lista[i].second.second);
      acum += lista[i].first;
      if(ejes == n) return acum;
    }

  }
  return -1;
}
```

2.6. Topological sort

O(m + n)

```
vi res;//guarda la respuesta
vi ent;//cantidad de aristas entrantes de cada nodo

bool topological_sort(vii &lis, int tam){
    res.clear();
    queue<int> s;
    for(int i = 1; i <= tam; i++){
        if(!ent[i]) s.push(i);
    }

    int n, m;
    while(s.size()){
        n = s.front();
        s.pop();
        res.push_back(n);

        for(int i = 0; i < lis[n].size(); i++){
            m = lis[n][i];
            ent[m]--;
            if(!ent[m]) s.push(m);
        }
```

```
    }
    if(res.size() != tam) return false;//contiene ciclo!!!
    else return true;
}
```

3. Matematicas

3.1. MCD y MCM

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

```
int mcd(int a, int b){//algoritmo de euclides
    return  a? mcd(b %a, a): b;
}

int mcm(int a, int b) {
  return a*b/mcd(a,b);
}
```

3.2. Exponenciacion binaria

O(log n)

```
typedef long long int lli;

lli exp_bin (lli a, lli n) {
  lli res = 1;
  while (n) {
    if (n & 1) res *= a;
    a *= a;
    n >>= 1;
  }
  return res;
}
```

3.3. Multiplicacion modular

Encuentra (a*b) mod 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.

```
typedef long long int lli;//metodo normal
lli mulmod (lli a, lli b, lli c) {
  lli x = 0, y = a%c;
  while (b > 0){
    if (b % 2 == 1) x = (x+y) % c;
    y = (y*2) % c;
    b /= 2;
  }
  return x % c;
}

typedef __int128 bi; //metodo con __int128
lli mulmod_2(bi a, bi b, bi c){
    return (lli) ((a*b) % c);
}

```

```cpp
int main(){
    lli a, b, c;
    cin >> a >> b >> c;
    cout << mulmod_2((bi) a, (bi) b, (bi) c) << endl;
    return 0;
}
```

3.4. Exponenciacion modular

Encuentra (a^b) mod c, se nesecita implementar previamente multiplicacion modular.

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

3.5. Algoritmo extendido de euclides

Encuentra dos numeros x e y tal que: MCD(a, b) = ax + by

```cpp
int gcd_ex (int a, int b, int & x, int & y) {
  if (a == 0) {
    x = 0; y = 1;
    return b;
  }
  int x1, y1;
  int d = gcd_ex (b%a, a, x1, y1);
  x = y1 - (b / a) * x1;
  y = x1;
  return d;
}

int main(){
    int n, m, x, y, res;

    while(cin >> n >> m){
        res = gcd_ex(n, m, x, y);
        cout << "gcd = " << res << ", x = " << x << ", y = " <<
            y << endl;
    }
}
```

3.6. Inverso multiplicativo modular

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

```cpp
void inverso(int a, int m){
    int x, y;

    int g = gcd_ex (a, m, x, y);
    if (g != 1)
        cout << "no solution";
    else {
```

```
        x = (x % m + m) % m;
        cout << x << endl;
    }
}
```

3.7. Phi de euler

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

```
int phi (int n) {
  int result = n;
  for (int i=2; i*i<=n; ++i)
    if (n % i == 0) {
      while (n % i == 0)
        n /= i;
      result -= result / i;
    }
  if (n > 1)
    result -= result / n;
  return result;
}
```

3.8. Rho de pollard

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

```
bool es_primo_prob (ll n, int a) {
  if (n == a) return true;
  ll s = 0,d = n-1;
  while (d % 2 == 0) s++,d/=2;

  ll x = expmod(a,d,n);
  if ((x == 1) || (x+1 == n)) return true;

  for(int i = 0; i < s-1; i++){
    x = mulmod(x, x, n);
    if (x == 1) return false;
    if (x+1 == n) return true;
  }
  return false;
}

bool rabin (ll n){ //devuelve true si n es primo
  if (n == 1) return false;
  const int ar[] = {2,3,5,7,11,13,17,19,23};
  for(int j = 0; j < 9; j++)
    if (!es_primo_prob(n,ar[j]))
      return false;
  return true;
}

ll rho(ll n){
    if( (n & 1) == 0 ) return 2;
    ll x = 2 , y = 2 , d = 1;
    ll c = rand() % n + 1;
```

```
    while( d == 1 ){
        x = (mulmod( x , x , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        y = (mulmod( y , y , n ) + c)%n;
        if( x - y >= 0 ) d = mcd( x - y , n );
        else d = mcd( y - x , n );
    }
    return d==n? rho(n):d;
}

map<ll, ll> prim;

void factRho (ll n){
  if (n == 1) return;
  if (rabin(n)){
    prim[n]++;//se agrega el primo n a la solucion
    return;
  }
  ll factor = rho(n);
  factRho(factor);
  factRho(n/factor);
}

int main(){
    ll n;
    while(scanf("%lld", &n), n > 0){
        prim.clear();
        factRho(n);

        for(map<ll, ll>::iterator it = prim.begin(); it != prim.
            end(); it++){
            cout << "el " << (it)->first << " aparece " << (it)
                ->second << " veces\n";
        }
    }
    return 0;
}
```

3.9. BigInteger c++

```
#include <bits/stdc++.h>

using namespace std;
typedef vector<int> vi;

struct biginteger{
    vi num;

    void iniciar(string c){
        num.clear();
    int tam = c.length();
    for(int i = tam - 1; i > -1; i--) num.push_back(c[i] - '0');
    quitar_zeros_izq();
  }

  void iniciar(int c){
    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{
```

```
                    aux += res.num[i + j] + (num[i] * y[j]);
                    res.num[i + j] = aux % 10;
                    aux /= 10;
                }
            }
            if(i != l){
                aux += res.num[i];
                res.num[i] = aux % 10;
                aux /= 10;
            }
        }

        while(aux){
            res.num.push_back(aux % 10);
            aux /= 10;
        }
        res.quitar_zeros_izq();
        return res;
    }

    biginteger resta(biginteger b2){//asumimos que b2 es menor
        vi x = b2.num;
        biginteger res;
        res.num.assign(num.begin(), num.end());
        int i;

        for(i = 0; i < x.size(); i++){
            if(x[i] > res.num[i]){
                res.num[i] += 10;
                res.num[i + 1]--;
            }
            res.num[i] -= x[i];
        }
        while(res.num[i] < 0){
            res.num[i++] += 10;
            res.num[i]--;
        }
        res.quitar_zeros_izq();
        return res;
    }

    int comparar(biginteger b){//1 mayor, 0 igual, -1 menor
        if(num.size() > b.num.size()) return 1;
        else
            if(num.size() < b.num.size()) return -1;
            else{
                for(int i = num.size() - 1; i > -1; i--){
                    if(num[i] > b.num[i]) return 1;
                    else if(num[i] < b.num[i]) return -1;
                }
            }
        return 0;
    }
};
typedef biginteger bigint;

bigint operator+(bigint &x, bigint &y){return x.suma(y);}
bigint operator-(bigint &x, bigint &y){return x.resta(y);}
```

```
bigint operator*(bigint &x, bigint &y){return x.multiplicar(y);}

bigint operator+=(bigint &x, bigint &y){return x = x.suma(y);}
bigint operator-=(bigint &x, bigint &y){return x = x.resta(y);}
bigint operator*=(bigint &x, bigint &y){return x = x.multiplicar
    (y);}

bool operator<(bigint &x, bigint &y){return x.comparar(y) ==
    -1;}
bool operator>(bigint &x, bigint &y){return x.comparar(y) == 1;}
bool operator==(biginteger &x, biginteger &y){ return x.comparar
    (y) == 0;}
bool operator<=(bigint &x, bigint &y){
    int q = x.comparar(y);
    return q == -1 || q == 0;
}
bool operator>=(bigint &x, bigint &y){
    int q = x.comparar(y);
    return q == 0 || q == 1;
}

int main(){
    string n, m;
    biginteger b1, b2;

    while(cin >> n >> m){
        b1.iniciar(n);
        b2.iniciar(m);
        b1 = b1 * b2;
        b1.imprimir();
    }
}
```

4. Otros

4.1. Busqueda binaria

O(log n)

```
int f(int a, int b){
    return ar[a] > b;
}

int busqueda_binaria(int min, int max, int v){
    int epsilon = 1, med = 0;

    while(max-min > epsilon){
        med = (max+min)/2;
        if(f(med,v))
            max = med;
        else
            min = med;
    }
    return min;
}
```

5. Programacion dinamica

5.1. Subconjuntos de un conjunto

$O(2^n)$

```
void mask(int n, int ar[]){
    int l = 1 << n;

    for(int i = 0; i < l; i++){
        for(int j = 0; j < n; j++){
            if(i & (1 << j)){
                printf("%d ", ar[j]);
            }
        }
        printf("\n");
    }
}
```

5.2. Problema de la mochila

```
vi ben;//beneficio
vi cos;//costo
int knapsack(int cap, vi &cos, vi &ben, int n) {
    int dp[n+1][cap+1];

    for(int i = 0; i <= n; i++){
        for(int j = 0; j <= cap; j++){
            if(i == 0 || j == 0) dp[i][j] = 0;
            else if(cos[i - 1] <= j)
                dp[i][j] = max(ben[i - 1] + dp[i - 1][j - cos[i
                    - 1]], dp[i - 1][j]);
            else
                dp[i][j] = dp[i - 1][j];
        }
    }
    return dp[n][cap];
}
```

5.3. Longest Increment Subsecuence

Subsecuencia creciente mas larga
O(n log n)

```
int LIS(int arr[]){
    int res = 0;
    vector<int> vec(8, 1);

    for(int i = 0; i < 8; i++){
        for(int j = i + 1; j < 8; j++)
            if(arr[i] < arr[j]) vec[j]=max(vec[j], vec[i]+1);
        res = max(res, vec[i]);
    }

    return res;
}
```

5.4. Max Range Sum

O(n)

```
int main(){
    int n, num, res, aux;

    while(scanf("%d", &n), n){
        res = aux = 0;
        for(int i = 0; i < n; i++){
            scanf("%d", &num);
            aux += num;
            res = max(aux, res);
            if(aux < 0) aux = 0;
        }

        if(res > 0) printf("MRS␣=␣%d\n", res);
        else printf("negativo.\n");
    }
    return 0;
}
```

5.5. Subset Sum

```
bool dp[5][50];

void pre(vi &num){
    memset(dp, false, sizeof(dp));
    //for(int i = 0; i < num.size())

    for(int i = 0; i < num.size(); i++){
        if(i) for(int j = 1; j < 50; j++)
            if(dp[i - 1][j]) dp[i][j + num[i]] = true;

        dp[i][num[i]] = true;
    }
}
```