

# Índice

1. Cosas	1
1.1. Segment Tree (Guty)	1
1.2. Inverso modular Primo y PotLog (Guty)	2
1.3. Dijkstra (Guty)	2
1.4. Edmonds Karp (Guty)	2
1.5. Trie (Jonaz)	3
1.6. Geometra Basica Vectorial (Guty)	3
1.7. Criba y Factorizacion (Jonaz)	4

## BGL-UBA - Reference

### 1. Cosas

#### 1.1. Segment Tree (Guty)

```

1 // Nodo del segment tree
2 struct Nodo{
3     tint x;
4     Nodo (tint xx)
5         x = xx;
6 };
7 // Operacion del segment tree : tiene que ser ASOCIATIVA
8 Nodo op (Nodo n1, Nodo n2){
9     return Nodo(n1.x+n2.x);
10 }
11 vector<Nodo> buildSegTree (vector<Nodo> &v ){
12     // Completa el tamaño
13     tint k = 4, n = v.size();
14     while (k < 2*n)
15         k *= 2;
16     // Rellena las hojas
17     vector<Nodo> seg (k, Nodo(0));
18     forn(i,n)
19         seg[k/2+i] = v[i];
20     // Completa los padres
21     while (k > 0) {
22         seg[(k-1)/2] = op(seg[k-1],seg[k-2]);
23         k -= 2;
24     }
25     return seg;
26 }
27 // i es el indice de [0,n) en el arreglo original
28 // Nodo es lo que queremos poner ahora como hoja
29 void update(tint i, Nodo nodo,vector<Nodo> &seg){
30     tint k = seg.size()/2 + i;
31     seg[k] = nodo;
32     while (k > 0){
33         seg[k >> 1] = op(seg[k],seg[k^1]);
34         k /= 2;
35     }
36 }
37 Nodo queryAux(tint k, tint l, tint r, tint i, tint j, vector<Nodo> &seg){
38     if (i <= l && r <= j)
39         return seg[k];

```

```

40 if (r <= i or l >= j)
41     return 0; // Aca va el NEUTRO de la funcion "op"
42 Nodo a = queryAux(2*k,l,(l+r)/2,i,j,seg);
43 Nodo b = queryAux(2*k+1,(l+r)/2,r,i,j,seg);
44 return op(a,b);
45
46 }
47 // i,j son los indices del arreglo del que se hace la query
48 // la query se hace en [i,j]
49 Nodo query(tint i, tint j, vector<Nodo> &seg){
50     return queryAux(1,0,seg.size()/2,i,j,seg);
51 }
52 int main(){
53     tint n = 15;
54     vector<Nodo> v (n, Nodo(0));
55     forn(i,n)
56         v[i] = Nodo((3*(i+1)) % 7 - 9*(i-4) % 13);
57     vector<Nodo> seg = buildSegTree(v);
58     imprimirVector(v);
59     cout << query(3,11,seg).x << "\n";
60     return 0;
61 }

```

## 1.2. Inverso modular Primo y PotLog (Guty)

```

1 const tint nmod = 1000000007; // o el primo que deseamos
2
3 tint potLogMod (tint x, tint y){ // Calcula: (x^y) mod nmod
4     tint ans = 1;
5     while (y > 0){
6         if (y % 2)
7             ans = (x * ans) % nmod;
8         x = (x * x) % nmod;
9         y /= 2;
10    }
11    return ans;
12 }
13 tint invLog(tint a){ // Solo funciona si nmod es primo y devuelve un numero b tal
14     que: (a*b) = 1 mod nmod
15     return potLogMod(a,nmod-2);
16 }

```

## 1.3. Dijkstra (Guty)

```

1 const int INF = 1000000000; // Aca va una cota que funque para el problema en
2   vez de 1000000000
3 struct Arista{

```

```

3     tint v1,v2,peso;
4     Arista(tint vv1, tint vv2, tint pp){
5         v1 = vv1;
6         v2 = vv2;
7         peso = pp;
8     }
9 };
10 bool operator < (Arista a1, Arista a2){
11     return make_tuple(a1.peso,a1.v1,a1.v2) > make_tuple(a2.peso,a2.v1,a2.v2);
12 }
13 vector<tint> dijkstra (vector<vector<tint> > &ladj, vector<vector<tint> > &w,
14     tint s){ // Devuelve un vector d, tal que d[v] es la minima distancia de "s
15     " a "v"
16     tint n = ladj.size();
17     vector<tint> d (n,INF);
18     priority_queue<Arista> v;
19     d[s] = 0;
20     for(auto vecino : ladj[s])
21         v.push(Arista(s,vecino,w[s][vecino]));
22     while (!v.empty()){
23         if (d[v.top().v2] == INF or d[v.top().v1] + v.top().peso <= d[v.top().v2]){
24             Arista e = v.top();
25             v.pop();
26             d[e.v2] = d[e.v1] + e.peso;
27             for(auto vecino : ladj[e.v2])
28                 v.push(Arista(e.v2,vecino,w[e.v2][vecino]));
29         }else
30             v.pop();
31     }
32     return d;
33 }

```

## 1.4. Edmonds Karp (Guty)

```

1 // Hay que tener definidas de antemano:
2 // capacidades: Una matriz que en el lugar (i,j) guarda la capacidad que une al
3   nodo i con j.
4 // ladj: Para cada nodo tiene la lista de vecinos. Notar que al comenzar el
5   codigo se agregan las aristas que faltan para la red residual
6 // flow: Se debe dar un flujo inicial de 0
7 // flowPath: Guarda en el lugar (i,j) la cantidad de flujo que efectivamente
8   pasa por la arista que une i con j. Inicialmente debe ser una matriz de
9   ceros.
10
11 const tint INF = 999999999999;
12
13 tint maxFlow (vector<vector<tint> > &capacidades, vector<vector<tint> > &ladj,

```

```

    tint qNodos, tint source, tint terminal){
10 tint flow = 0;
11 vector<vector<tint> > flowPath (qNodos, vector<tint> (qNodos,0));
12 tint capacityFound = -1;
13 forn(i, ladj.size())
14     for (auto &a : ladj[i])
15         if (capacidades[i][a] != 0)
16             ladj[a].push_back(i);
17 while (capacityFound != 0){
18     vector<tint> path (qNodos,-1);
19     // Aca empieza el bfs

20
21     path[source] = -2;
22     capacityFound = 0;
23     vector<tint> pathCapacity (qNodos,INF); // Aca va una cota para el flujo del
        problema
24     deque<tint> visit = {source};
25     while (!visit.empty()){
26         tint actual = visit.front();
27         visit.pop_front();
28         for (auto vecino : ladj[actual]){
29             if (capacidades[actual][vecino] > flowPath[actual][vecino] && path[
                vecino] == -1 ){
30                 path[vecino] = actual;
31                 pathCapacity[vecino] = min(pathCapacity[actual],capacidades[actual] [
                    vecino] - flowPath[actual][vecino]);
32                 if (vecino != terminal)
33                     visit.push_back(vecino);
34                 else{
35                     capacityFound = pathCapacity[vecino];
36                     visit.clear();
37                     break;
38                 }
39             }
40         }
41     }
42     // Aca termina el bfs
43     if (capacityFound == 0)
44         break;
45     flow += capacityFound;
46     tint v = terminal;
47     while (v != source){
48         tint u = path[v];
49         flowPath[u][v] += capacityFound;
50         flowPath[v][u] -= capacityFound;
51         v = u;

```

```

52     }
53 }
54 return flow;
55 }

```

## 1.5. Trie (Jonaz)

```

1 const int MAXN = 100000;
2 int NODS = 1;
3 struct trie {
4     map<char, int> sig;
5     char c;
6     bool final;
7 };
8 trie t[MAXN];
9 void resetNode(int i){
10     t[i].sig.clear();
11     t[i].final = false;
12 }
13 void initT(){
14     forn(i,nods){
15         resetNode(i);
16     }
17     NODS = 1;
18 }
19 void insertar(string st){
20     int pos=0;
21     forn(i,st.size()){
22         if(t[pos].sig.find(st[i]) == t[pos].sig.end()){
23             t[pos].sig[st[i]] = NODS;
24             resetNode(nods);
25             t[nods].c = st[i];
26             NODS++;
27         }
28         pos = t[pos].sig[st[i]];
29     }
30     t[pos].final = true;
31 }

```

## 1.6. Geometra Basica Vectorial (Guty)

```

1 struct Punto{
2     double x,y;
3     Punto (double xx, double yy){ x = xx; y = yy; }
4 };
5 Punto operator + (Punto p1, Punto p2){
6     return Punto(p1.x+p2.x, p1.y+p2.y);

```

```

7 }
8 Punto operator - (Punto p1, Punto p2){
9     return Punto(p1.x-p2.x, p1.y-p2.y);
10 }
11 Punto operator * (ldouble k, Punto p){
12     return Punto(k*p.x, k*p.y);
13 }
14 double operator * (Punto p1, Punto p2){
15     return p1.x*p2.x + p1.y*p2.y;
16 }
17 double norma(Punto p){
18     return sqrt(p*p);
19 }
20 double pcruz (Punto p1, Punto p2){
21     return p1.x*p2.y - p2.x*p1.y;
22 }
23 double areaTriangulo(Punto a, Punto b, Punto c){
24     return abs(pcruz(b-a,c-a))/2.0;
25 }
26 double areaParalelogramo(Punto a, Punto b, Punto c){
27     return abs(pcruz(b-a,c-a));
28 }

```

## 1.7. Criba y Factorizacion (Jonaz)

```

1  const int MAXN = 10000000; // hasta ahi se la re banca, y uno mas tambien
2  // si p[i] = 0      => i es primo
3  // si p[i] = j != 0 => j divide a i (y j primo)
4  int p[MAXN];
5  // arma la criba
6  void criba() {
7      for(int i=4; i<MAXN; i+=2) p[i]=2;
8      for(int i=3; i<MAXN; i+=2)
9          if(!p[i]) for(int j=2*i; j<MAXN; j+=i) p[j] = i;
10 }
11 // devuelve una factorizacion de N del tipo <primo,exponente>
12 map<int,int> factorizar(int N){
13     map<int,int> res;
14     while(p[N]){
15         res[p[N]]++;
16         N /= p[N];
17     }
18     res[N]++;
19     return res;
20 }
21 // devuelve vector con los factores primos que dividen a N en orden
22 vector<int> factorizarLista(int N){

```

```

23     vector<int> res;
24     while(p[N]){
25         res.push_back(p[N]);
26         N /= p[N];
27     }
28     res.push_back(N);
29     // puede omitirse si no interesa que esten ordenados
30     sort(res.begin(), res.end());
31     return res;
32 }

```