Algoritmos y Estructuras de Datos Básico

Ramiro González Sánchez

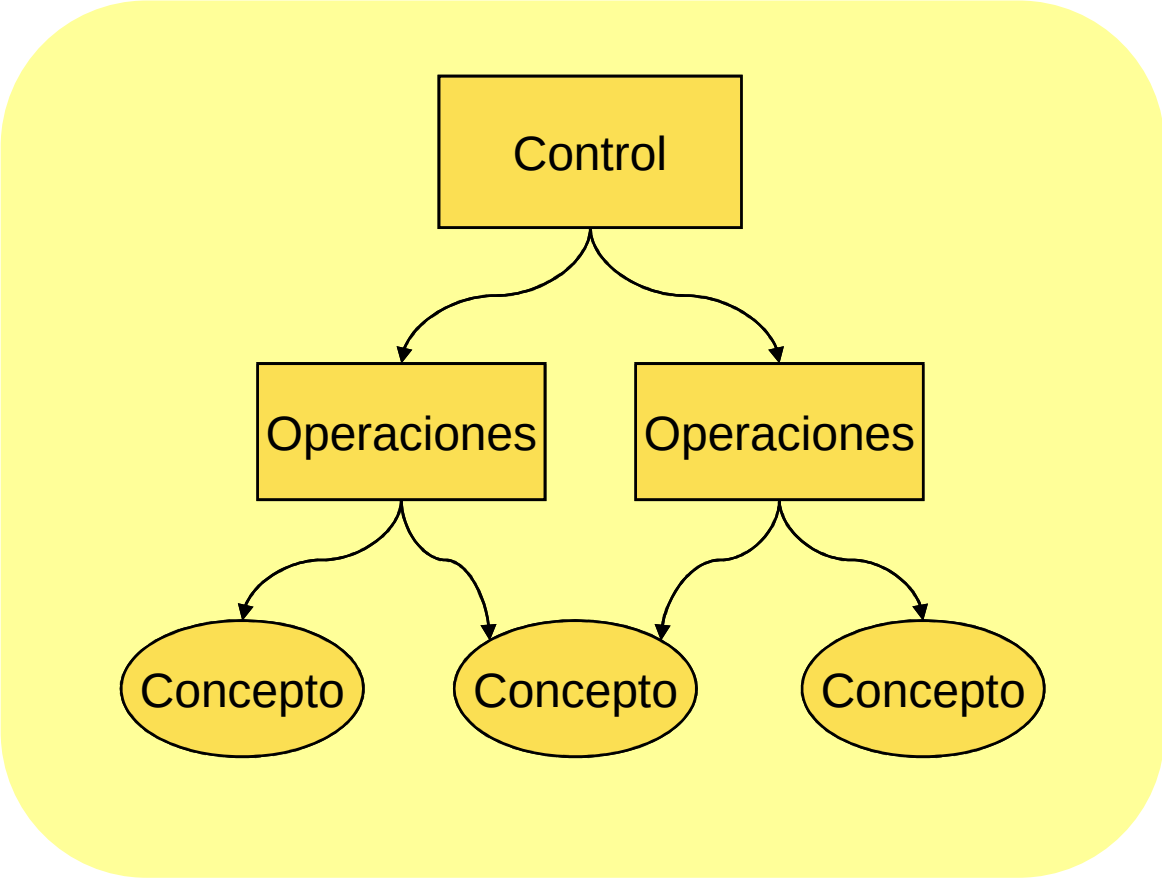
Junio del 2006

Contenido

- Estructura de un programa
- Patrones básicos de control
- Análisis de la complejidad de algoritmos
- Ordenamiento de montículo
- Ordenamiento por conteo
- Programación dinámica
- Búsquedas

Estructura de un programa

Estructura de un Programa



Patrones básicos de control

Patrones Básicos de Control

- Secuencial Simple
- Match
- Secuencial con Corte
- Secuencial con Múltiples Cortes

Secuencial Simple

```
ProcesoSecuencialSimple {

    prologo();

    datos = leeArchivo();
    while (!EOF) {
       procesa(datos);
       datos = leeArchivo();
    }

    epilogo();

}
```

Ejercicio

- **Convierte el Proceso Secuencial Simple para procesar un Arreglo de datos.**

  (En lugar de tomar los datos de un archivo, ahora los tomará de un arreglo)

Match

```
ProcesoMatch {
        prologo();
        datosM = leeArchivoMaestro();
        datosD = leeArchivoDetalle();
        while (!EOFM && !EOFD) {
                if (datosM.llave == datosD.llave) {
                                procesaMaestroConDetalle(datosM, datosD);
                                datosM = leeArchivoMaestro();
                                datosD = leeArchivoDetalle();
                } else if (datosM.llave > datosD.llave) {
                                procesaDetalleSinMaestro(datosD);
                                datosD = leeArchivoDetalle();
                } else {
                                procesaMaestroSinDetalle(datosM);
                                datosM = leeArchivoMaestro();
                }
        }

        while (!EOFM) {
                procesaMaestroSinDetalle(datosM);
                datosM = leeArchivoMaestro();
        }
        while (!EOFD) {
                procesaDetalleSinMaestro(datosD);
                datosD = leeArchivoDetalle();
        }

        epilogo();
}
```

Ejercicio

- **Convierte el Proceso Match a 2 Arreglos.**

Secuencial con Corte

```
ProcesoSecuencialConCorte {
    prologo();

    datos = leeArchivo();
    while (!EOF) {
       llaveActual = datos.llave;
       prologoLote();
       while (!EOF && (datos.llave == llaveActual)) {
         procesa(datos);
         datos = leeArchivo();
       }
       epilogoLote();
    }

    epilogo();
}
```

Secuencial con Múltiples Cortes

```
ProcesoSecuencialConMultiplesCortes {
    prologo();

    datos = leeArchivo();
    while (!EOF) {
        llaveActualN1 = datos.llaveN1;
        prologoLoteN1();
        while (!EOF && (datos.llaveN1 == llaveActualN1)) {

          llaveActualN2 = datos.llaveN2;
          prologoLoteN2()
          while (!EOF && (datos.llaveN1 == llaveActualN1)
                      && (datos.llaveN2 == llaveActualN2)) {
                      procesa(datos);
                      datos = leeArchivo();
          }
          epilogoLoteN2();
        }
        epilogoLoteN1();
    }

    epilogo();
}
```

Ejercicio

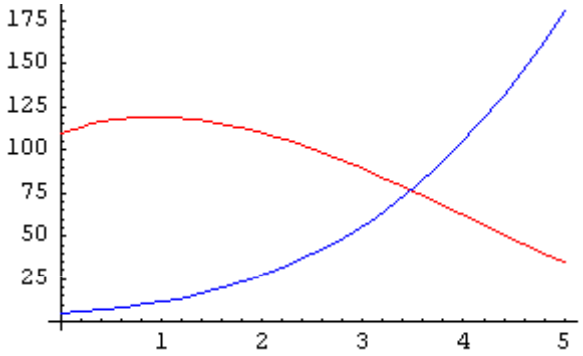
- **Implementa un programa que lea una entrada de datos ordenada con los siguientes campos:**
  - **#Sucursal**
  - **Codigo Categoria**
  - **Clave Producto**
  - **# piezas vendidas**
  - **# precio de venta**

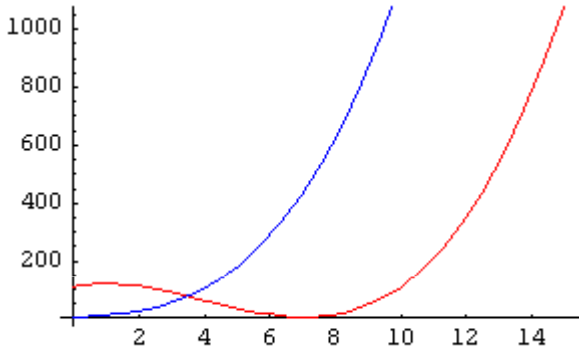
- **El programa debe generar un reporte que contenga:**
  - **Por cada Categoria de Productos**
    - El # total de piezas vendidas en la categoría
    - El importe total de venta de la categoría
    - El precio promedio de la categoría
  - **Por cada Sucursal**
    - El # total de piezas vendidas
    - El importe total de venta de la sucursal
    - El precio promedio de la categoría

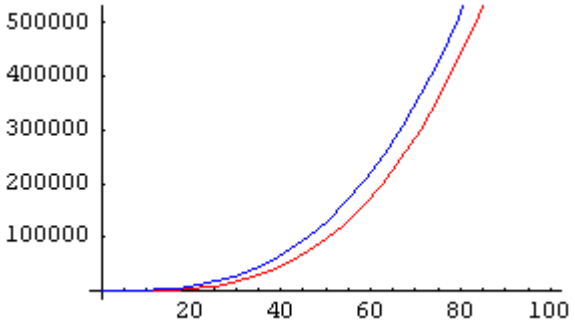
Análisis de la complejidad de algoritmos

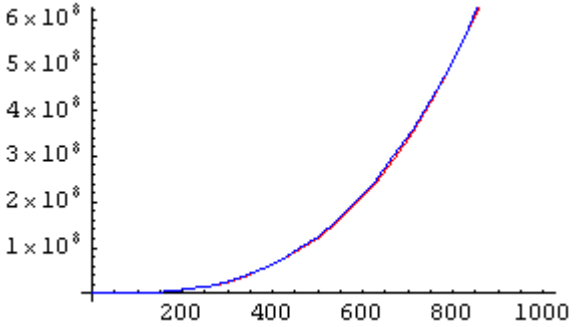
Comportamiento vs Volumen

$f(n) = n3 - 12n2 + 20n + 110$
$g(n) = n3 + n2 + 5n + 5$









Comparación de complejidades

$$O(f(n)) = O(g(n))$$

$$O(\log n) \leq O(\sqrt{n}) \leq O(n) \leq O(n \log n) \leq O(n^2) \leq O(n^3) \leq O(2^n)$$

Complejidad del Merge-Sort

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$T(n) = O(n \log n)$$

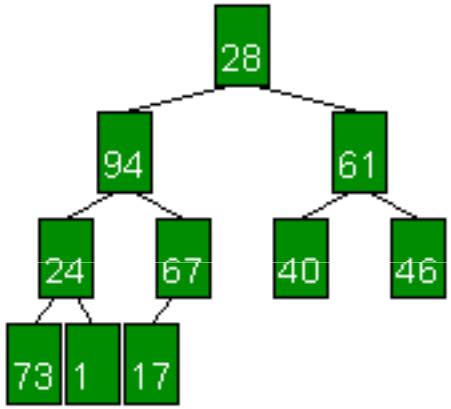
Ejercicio

- Implementa un MergeSort de enteros
- Calcula O(n) en términos de tiempo para el Algoritmo
  - ¿Se cumple la propiedad nlogn?

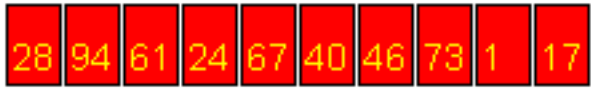
Ordenamiento de montículo

Montículo

Output



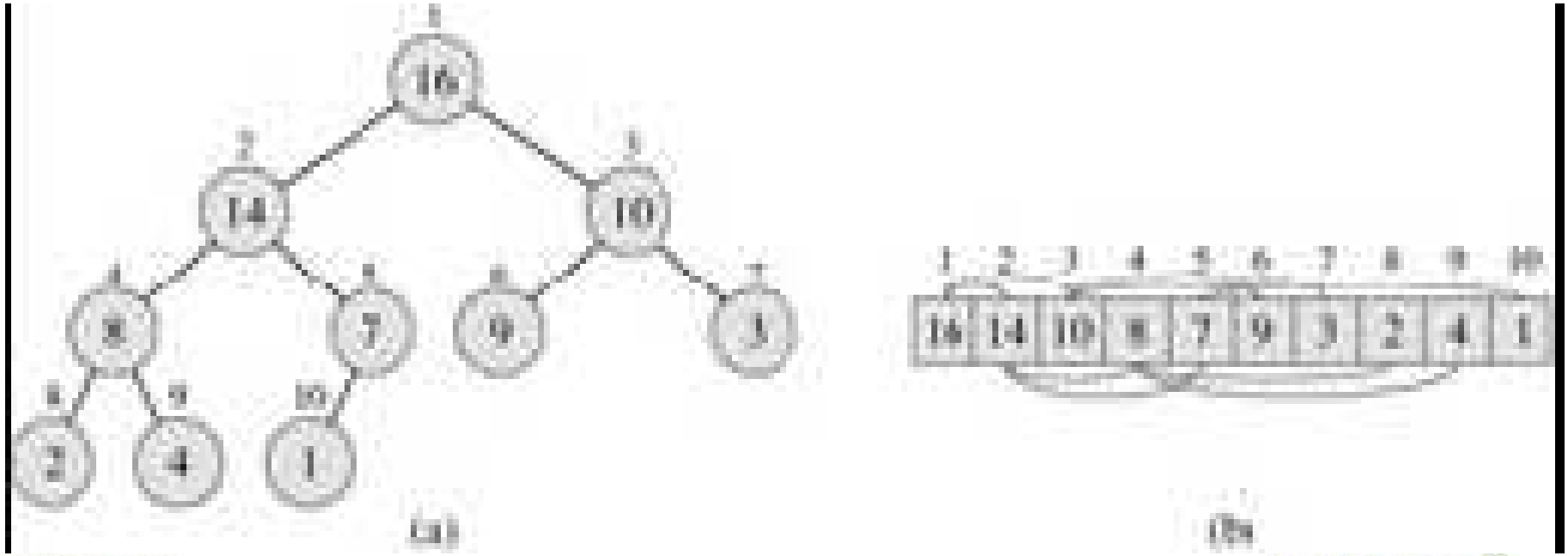
tree representation of the same heap



array representation of the heap

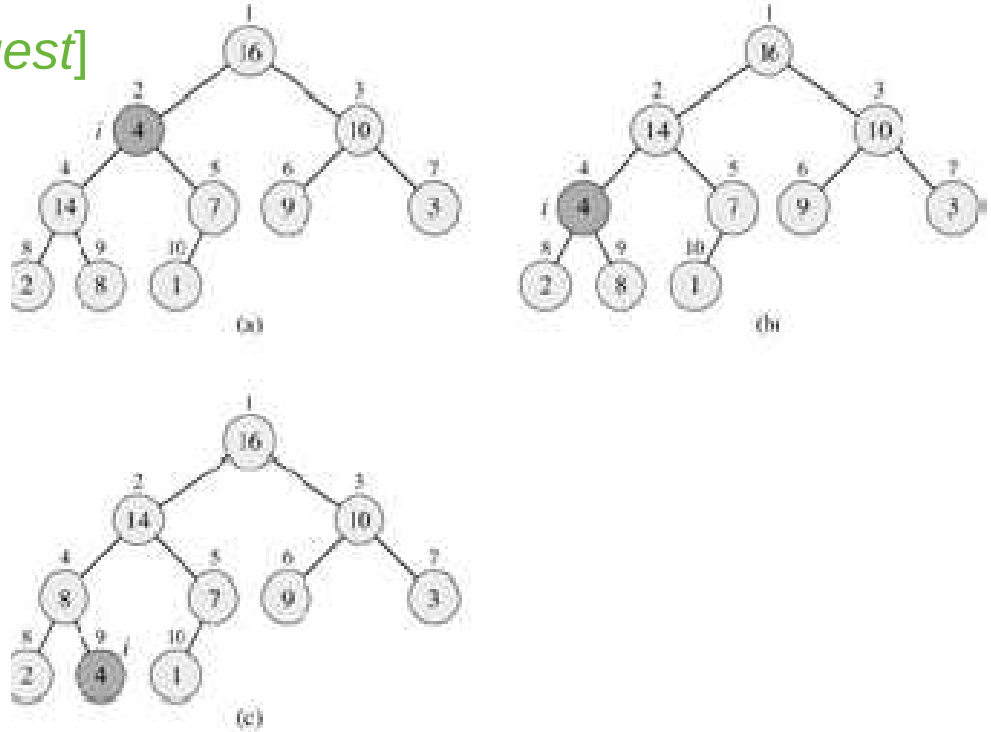
Construcción del Montículo

PARENT(*i*)
    return $\lfloor i/2 \rfloor$
LEFT(*i*)
    return $2i$
RIGHT(*i*)
    return $2i + 1$



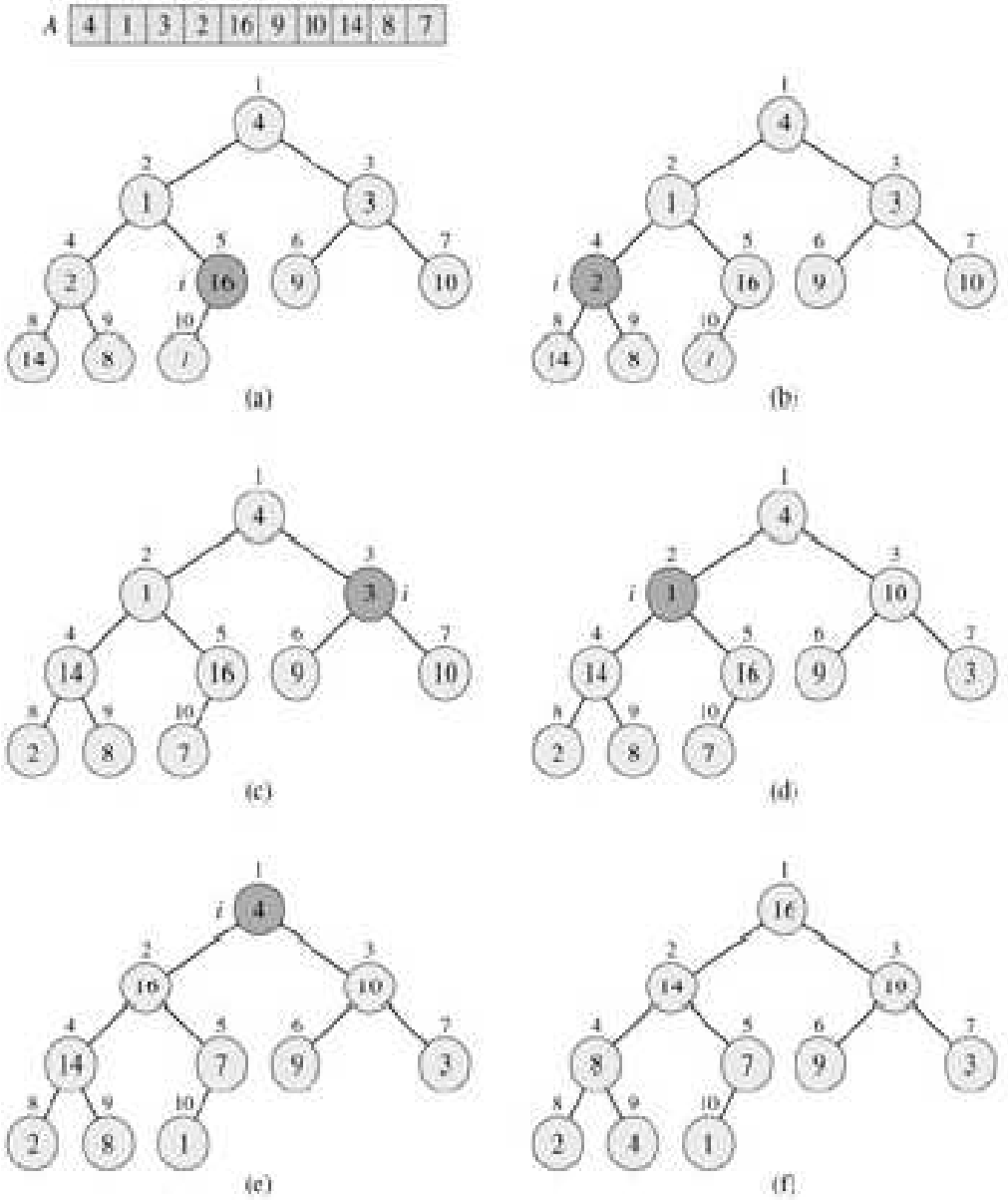
MAX-HEAPIFY

```
MAX-HEAPIFY(A, i)
1 l ← LEFT(i)
2 r ← RIGHT(i)
3 if l ≤ heap-size[A] and A[l] > A[i]
4   then largest ← l
5   else largest ← i
6 if r ≤ heap-size[A] and A[r] > A[largest]
7   then largest ← r
8 if largest ≠ i
9   then exchange A[i] ↔ A[largest]
10        MAX-HEAPIFY(A, largest)
```



BUILD-MAX-HEAP

BUILD-MAX-HEAP(*A*)
1 *heap-size*[*A*] ← *length*[*A*]
2 **for** *i* ← ⌊*length*[*A*]/2⌋ **downto** 1
3 **do** MAX-HEAPIFY(*A*, *i*)



Ordenamiento por Montículo (HEAPSORT)

```
HEAPSORT(A)
1 BUILD-MAX-HEAP(A)
2 for i ← length[A] downto 2
3   do exchange A[1] ↔ A[i]
4         heap-size[A] ← heap-size[A] - 1
5         MAX-HEAPIFY(A, 1)
```

Ejercicio

- **Implementar en Java el HeapSort**

Ordenamiento por conteo

Algoritmo

```
countingsort(A[], B[], k)  {
    for (i = 1 to k) {
        C[i] = 0
    }
    for (j = 1 to length(A)) {
        C[A[j]] = C[A[j]] + 1;
    }
    for (i = 1 to k) {
        C[i] = C[i] + C[i-1]
    }
    for (j = 1 to length(A)) {
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] – 1
    }
}
```

Programación dinámica

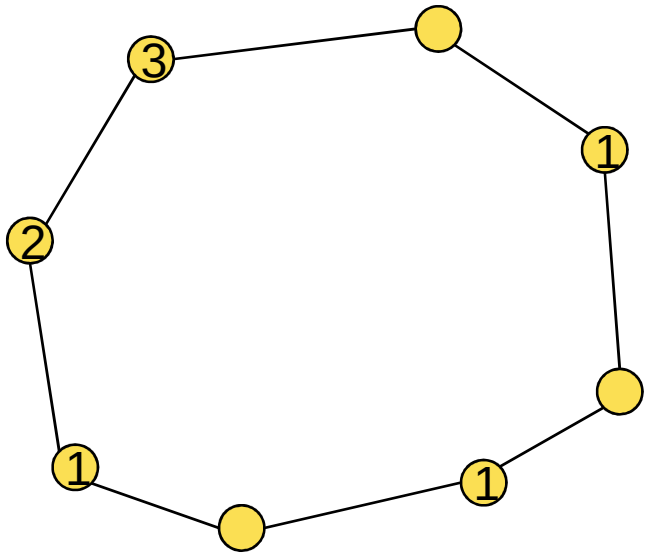
Secuencia Creciente Más Larga

10	20	2	13	8	14	3	6	9
1	2	1	2	2	3	2	3	4

La secuencia más larga es de 4 Elementos:

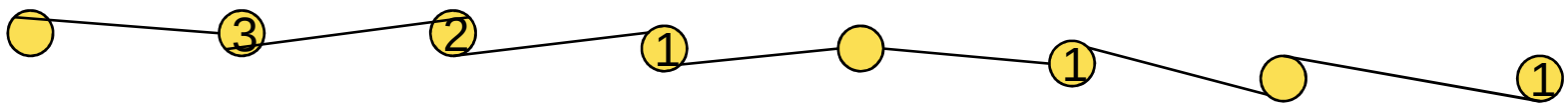
2 3 6 9

Problema de las Tortas



¿En qué sentido y a partir de qué nodo debe andar alguien para nunca quedarse sin Torta?

Algoritmo de Optimización



Nodo	0	1	2	3	4	5	6	7
Valor en Nodo	0	3	2	1	0	1	0	1
Entra	0	-1	1	2	2	1	1	0
Sale	0	2	3	3	2	2	1	1

¡Nodo más débil!

Búsquedas

Búsqueda Binaria

```
int bbin(int dato,int *v,int n){
    int inicio,final,medio;

    inicio = 0;
    final = n-1;
    while (inicio <= final) {
        medio = (inicio+final)/2;
        if (dato == v[medio])
            return (medio);
        else if (dato > v[medio])
            inicio = medio+1;
        else
            final = medio-1;
    }
    return -1;
}
```

Búsqueda en Anchura

```
BFS(grafo G, nodo s, nodo d) {  //s – fuente, d - destino
    // recorremos todos los vértices del grafo inicializándolos a NO_VISITADO,
    // distancia INFINITA y padre de cada nodo NULL
    for u in V[G] do {
      estado[u] = NO_VISITADO;
      distancia[u] = INFINITO; /* distancia infinita si el nodo no es alcanzable */
      padre[u] = NULL;
    }
    estado[s] = VISITADO;
    distancia[s] = 0;
    Encolar(Q, s);
   u = extraer(Q);
    while (Q != 0) && (u != d) do {
      // extraemos el nodo u de la cola Q y exploramos todos sus nodos adyacentes
      for  v in adyacencia[u]  do {
        if estado[v] == NO_VISTADO then {
            estado[v] = VISITADO;
            distancia[v] = distancia[u] + 1;
            padre[v] = u;
            Encolar(Q, v);
        }
      }
     u = extraer(Q);
    }
    if (u == d) return u;
  }
```

Búsqueda en Profundidad

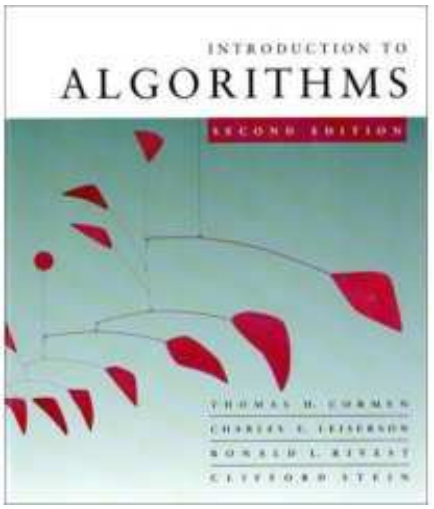
```
DFS(grafo G, nodo s, nodo d) //s – fuente, d - destino
 for each vertice u in V[G]do
        estado[u]=NO_VISITADO
        padre[u]= NULL
 tiempo=0
 DFS-Visitar(s, d)
```

```
DFS-Visitar(nodo u, nodo d)
 estado[u]=VISITADO
 tiempo=tiempo+1
 d[u]=tiempo
 for each v in Vecinos[u] do
        if estado[v]=NO_VISITADO then
                padre[v]=u
                if (v == d) return;
                DFS-Visitar(v)
 estado[u]=TERMINADO
 tiempo=tiempo+1
 f[u]=tiempo
```

Referencias

- **Wikibooks**
  - http://en.wikibooks.org/wiki/Algorithms

- **Introduction to Algorithms, Second Edition (Hardcover)**
  - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein



- **Artificial Intelligence: A Modern Approach (2nd Edition) (Hardcover)**
  - Stuart J. Russell, Peter Norvig