LENGUAJE C

Tema 9 – Ordenamiento y búsqueda

Ordenamiento

Dados a (un arreglo) y tam (su longitud)
ordenar a (de menor a mayor o al revés)

```
Por ejemplo:
int a[] = {81, 22, 35, 18, 67, 5, 99, 74, 16, 40};
int tam = 10;
```

Ordenamiento

Aunque los lenguajes de programación modernos incluyen funciones de biblioteca para ordenar arreglos, a veces es necesario utilizar alguno de los muchos algoritmos de ordenamiento que se conocen.

Existen dos criterios opuestos a la hora de elegir uno de esos algoritmos: los algoritmos más sencillos (fáciles de comprender y de programar) suelen ser los más lentos, mientras que los más rápidos por lo general son los más complejos.

Entre los algoritmos de ordenamiento más conocidos pueden mencionarse:

- Ordenamiento por burbujeo
- Ordenamiento por selección
- Ordenamiento por inserción
- Ordenamiento Shell (por su creador: Donald L. Shell)
- Ordenamiento rápido (Quicksort)

Ordenamiento por burbujeo

```
int tam = 10;
int a[] = {81, 22, 35, 18, 67, 5, 99, 74, 16, 40};

int i, j;
for (i = tam-1; i > 0; i--)
  for (j = 0; j < i; j++)
    if (a[j] > a[j+1]) {
      int aux = a[j];
      a[j] = a[j+1];
      a[j+1] = aux;
    }
```

Ordenamiento por selección

```
int tam = 10;
int a[] = {81, 22, 35, 18, 67, 5, 99, 74, 16, 40};

int i, j;
for (i = 0; i < tam-1; i++) {
  int posMin = i;
  for (j = i+1; j < tam; j++)
    if (a[j] < a[posMin]) posMin = j;
  if (a[posMin] < a[i]) {
    int aux = a[i];
    a[i] = a[posMin];
    a[posMin] = aux;
  }
}
```

Ordenamiento por inserción

```
int tam = 10;
int a[] = {81, 22, 35, 18, 67, 5, 99, 74, 16, 40};

int i, j;
for (i = 1; i < tam; i++) {
  int aux = a[i];
  for (j = i-1; j >= 0 && a[j] > aux; j--)
    a[j+1] = a[j];
  a[j+1] = aux;
}
```

Ordenamiento Shell

```
int tam = 10;
int a[] = {81, 22, 35, 18, 67, 5, 99, 74, 16, 40};

int i, j, sep;
sep = tam/2;
while (sep > 0) {
  for (i = sep; i < tam; i++) {
    j = i;
    int aux = a[i];
    while (j >= sep && a[j-sep] > aux) {
      a[j] = a[j-sep];
      j = j-sep;
    }
    a[j] = aux;
  }
  sep = sep/2;
}
```

Ordenamiento rápido (Quicksort)

1. Se elije un elemento del arreglo como pivote.
2. Se particiona el arreglo en dos subarreglos: uno contiene aquellos elementos menores que el pivote, el otro contiene los mayores.
3. Se ordenan los subarreglos.
4. Se obtiene el arreglo ordenado uniendo de nuevo los subconjuntos ordenados y el pivote.

SU IMPLEMENTACIÓN ES RECURSIVA

Ordenamiento rápido (Quicksort)

La función qsort de C (en *stdlib.h*) realiza este ordenamiento:

```
int tam = 10;
int a[] = {81, 22, 35, 18, 67, 5, 99, 74, 16, 40};

qsort (a, tam, sizeof(a[0]), comparar);
```

Para poder utilizar qsort, es necesario implementar la función comparar:

```
int comparar(const void *a, const void *b){
  if (*(int *)a < *(int *)b) return -1;
  else if (*(int *)a > *(int *)b) return 1;
       else return 0;
}
```

Búsqueda

Dados a (un arreglo), tam (su longitud) y

d (dato a buscar), obtener p (posición de d en a)

Por ejemplo:

```
int a[] = {81, 22, 35, 18, 67, 5, 99, 74, 16, 40};
int tam = 10;
int d = 18;
```

Búsqueda secuencial

```
int tam = 10;
int a[] = {81, 22, 35, 18, 67, 5, 99, 74, 16, 40};
int d = 18;

int p = 0;
while (p < tam && a[p] != d) p++;
if (p == tam) p = -1;
```

Búsqueda binaria

```
int tam = 10;
int a[] = {5, 16, 18, 22, 35, 40, 67, 74, 81, 99};
int d = 18;

int izq = 0;
int der = tam - 1;
int p = (izq + der) / 2;
while (izq <= der) {
  if (d < a[p]) der = p - 1;
  else izq = p + 1;
  p = (izq + der) / 2;
}
if (d != a[p]) p = -1;
```

REQUIERE QUE LOS DATOS ESTÉN ORDENADOS

BIBLIOGRAFÍA & LICENCIA

- Textos tomados, corregidos y modificados de diferentes páginas de Internet, tutoriales y documentos, entre los que destaco el libro: *C/C++ Curso de programación*, 2da Ed, Javier Ceballos, Alfaomega Ra-Ma.
-
- Autor:

Matías E. García

Prof. & Tec. en Informática Aplicada

www.profmatiasgarcia.com.ar

info@profmatiasgarcia.com.ar