Universidad Rafael Landívar
Facultad de Ingeniería
Ingeniería en Informática y Sistemas
Laboratorio de Programación, Sección: 02
Mgtr. Moisés Alonso

Algoritmos de ordenamiento

Carlos Enrique Laparra Robledo

1031120

Guatemala, 19 de septiembre de 2020

Algoritmos de ordenamiento

> Debido a que las estructuras de datos son utilizadas para almacenar información, para poder recuperar esa información de manera eficiente es deseable que aquella esté ordenada. Existen varios métodos para ordenar las diferentes estructuras de datos básicas.(Anónimo, sf).

Los métodos de ordenamiento no suelen ser utilizados de manera muy recurrente, a veces se utilizan una única vez. Algunas veces hay métodos simples que se pueden utilizar con una cantidad no tan grande de datos, mientras hay otros métodos más complejos pero que son más eficientes en tiempo de ejecución.

Para conocer de los algoritmos de ordenamiento es importante conocer los siguientes conceptos:

- Clave: La parte de un registro por la cual se ordena la lista. Por ejemplo, una lista de registros con campos nombre, dirección y teléfono se puede ordenar alfabéticamente de acuerdo a la clave nombre. En este caso los campos dirección y teléfono no se toman en cuenta en el ordenamiento.
- Criterio de ordenamiento (o de comparación): el criterio que se utiliza para asignar valores a los registros con base en una o más claves.
- Registro: Un grupo de datos que forman la lista. Pueden ser datos atómicos (enteros, caracteres, reales, etc.) o grupos de ellos, que en C equivalen a las estructuras.
- Estabilidad: cómo se comporta con registros que tienen claves iguales. Algunos algoritmos mantienen el orden relativo entre éstos y otros no.
- Tiempo de ejecución: la complejidad del algoritmo, que no tiene que ver con dificultad, sino con rendimiento. Es una función independiente de la implementación.

Requerimientos de memoria: El algoritmo puede necesitar memoria adicional para realizar su labor. En general es preferible que no sea así, pero es común en la programación tener que sacrificar memoria por rendimiento.

(C++ Con Clase, s.f.)

Quicksort es un algoritmo de ordenación considerado entre los más rápidos y eficientes. Fue diseñado en los años 60s por C. A. R. Hoare un científico en computación.

El algoritmo usa la técnica divide y vencerás que básicamente se basa en dividir un problema en subproblemas y luego juntar las respuestas de estos subproblemas para obtener la solución al problema central.

(Yabar, 2009)

Ejemplo de código de QuickSort

```
#include <stdlib.h>
#include <stdio.h>
#define MAX 20

 void intercambio(int vector[],int i, int j);
 void quicksort(int vector[],int inf, int sup);
 void imprimeVector(int vector[],int elem);

 int main()
 {
 int vector[MAX]={0};
 int numElem,i;
 clrscr();
 printf("\nCuantos elementos quieres en el arreglo? ");
 scanf("%d",&numElem);
 for(i=0;i<numElem;i++)
 {
 printf("\nDato: ");
 scanf("%d",&vector[i]);
 }

 quicksort(vector,0,numElem-1);
 imprimeVector(vector,numElem);
 getch();
 return 0;
 }

 void intercambio(int vector[],int i, int j)
 {
 int aux;
 aux=vector[i];
 vector[i]=vector[j];
 vector[j]=aux;
 }


 void quicksort(int vector[],int inf, int sup)
 {
```

```
int i;
int j;
int x;
i=inf;
j=sup;
x=vector[(i+j)/2]; //obteniendo el pivote
while(i<=j)
{
while(vector[i]<x)    //obteniendo elemento mayor de lado izquierdo del  pivote para intercambiar
i++;
while(vector[j]>x)   // obteniendo elemento menor de lado derecho del pivote a intercambiar
j--;
if(i<=j)            //intercambiando cuando se sigan presentando valores menores ala derecha y
mayores ala izquierda
{
intercambio(vector,i,j);
i++;
j--;
}
}
if(inf<j)
quicksort(vector,inf,j);
if(i<sup)
quicksort(vector,i,sup);
}

void imprimeVector(int vector[],int elem)
{
int i;
printf("\nVector ordenado con quicksort\n");
for(i=0;i<elem;i++)
{
printf("[%d] ",vector[i]);
}
}
```

(Carre, 2014)

Al igual que QuickSort, MergeSort es un algoritmo de dividir y conquistar. Divide la matriz de una entrada en dos mitades, se llama así mismo para las dos mitades y luego fusiona las dos mitades ordenadas. La función merge() se usa para fusionar dos mitades.

(GeeksForGeeks, 2020)

```
If r > l

     1. Find the middle point to divide the array into two halves:
             middle m = (l+r)/2
     2. Call mergeSort for first half:
             Call mergeSort(arr, l, m)
     3. Call mergeSort for second half:
             Call mergeSort(arr, m+1, r)
     4. Merge the two halves sorted in step 2 and 3:
             Call merge(arr, l, m, r)
```

(GeeksForGeeks, 2020)

Bibliografía

Algoritmos de ordenamiento. (s.f.). Obtenido de http://ict.udlap.mx/people/ingrid/Clases/IS211/Ordenar.html

C++ Con Clase. (s.f.). Obtenido de Algoritmos de ordenamiento: http://c.conclase.net/orden/

Carre, E. L. (22 de Noviembre de 2014). *Correo uabc*. Obtenido de QuickSort código en C: https://sites.google.com/a/uabc.edu.mx/quicksort2014-2/home/codigo

GeeksForGeeks. (6 de Julio de 2020). Obtenido de Merge Sort: https://www.geeksforgeeks.org/merge-sort/

Yabar, R. (19 de Julio de 2009). *WordPress*. Obtenido de QuickSort en C++: https://ronnyml.wordpress.com/2009/07/19/quicksort-en-c/#:~:text=Quicksort%20en%20C%2B%2B,los%20m%C3%A1s%20r%C3%A1pidos%20y%20eficientes.&text=El%20algoritmo%20usa%20la%20t%C3%A9cnica,la%20soluci%C3%B3n%20al%20problema%20central.