Axel Adair Chávez Ochoa

22110167

18/02/2024

Estructura de datos y algoritmia

Método de ordenamiento burbuja (Bubble sort)

El método de ordenación por burbuja se basa en comparaciones sucesivas de dos elementos consecutivos y realizar un intercambio entre los elementos hasta que queden ordenados.

Supongamos el vector mostrado en la siguiente tabla, para realizar la ordenación se han de seguir estos pasos:

- Se comparan los dos primeros elementos, si el segundo es superior al primero, se dejan tal como están, pero si el primero es el más grande, se intercambian los elementos.
- A continuación, se compara el segundo elemento con el tercero aplicando los mismos criterios del paso anterior.
- De esta forma se repite la operación de comparación con todos los elementos que forman el vector. Cuando se alcance el último elemento se ha encontrado el elemento que tiene el valor más elevado y este ha quedado situado al final del vector.
- Para encontrar el segundo elemento se repite la operación de ordenación de todos los elementos del vector excepto con el último.
- Reiterando el proceso descrito, el vector quedará ordenado cuando solo se comparen los dos primeros elementos.

Primera ordenación.

Ejemplo de ordenación por burbuja para encontrar el primer elemento.

	Vector	1ª Comparación	2ª Comparación	3ª Comparación	4ª Comparación	5ª Comparación
V[1]	3	**3**	3	3	3	3
V[2]	34	**34**	**1**	1	1	1
V[3]	1	1	**34**	**34**	34	34
V[4]	53	53	53	**53**	**15**	15
V[5]	15	15	15	15	**53**	**6**
V[6]	6	6	6	6	6	**53**

El proceso de ordenación expresado en pseudocódigo estructurado es:

```
desde j ← 1 hasta n-1 hacer
        si elemento[j] > elemento[j+1] entonces
                intercambiar(elemento[j], elemento[j+1])
        fin_si
fin_desde
```

Algoritmo de burbuja.

Generalizando el proceso de ordenación se obtiene el siguiente pseudocódigo estructurado para el algoritmo de burbuja.

i -Variable ordenaciones
j -Variable comparaciones
n - Número de elementos del vector (en C representa el índice mayor)

```
algoritmo Burbuja

inicio
// Ordenaciones
desde i ← 1 hasta n-1 hacer
        // Comparaciones
        desde j ← 1 hasta n-i hacer
                si elemento[j] > elemento[j+1] entonces
                        // Intercambiar los elementos
                        aux ← V[j]
                        V[j] ← V[j+1]
                        V[j+1] ← aux
                fin_si
        fin_desde
fin_desde
fin
```

Ejemplo en C:

```
int v[]={3, 34, 1, 53, 15, 6};
int j,i, aux;

// Ordenación
for(i=0; i<5; i++){
        // Comparaciones
        for(j=0; j<5-i; j++){
        // Intercambiar los elementos
        if(v[j] > v[j+1]){
                aux=v[j];
                v[j]=v[j+1];
                v[j+1]=aux;
                }
        }
}
```

El ejemplo en C++ Builder:

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <iostream.h>
#include <conio.h>
#pragma hdrstop

//---------------------------------------------------------------------------

#pragma argsused
int main(int argc, char* argv[]){

// Variables
int v[]={3, 34, 1, 53, 15, 6};
int j, aux;
int i = 0;
bool ord = false;

// Visualizar el vector
for(int n=0; n<6; n++){
   cout << v[n] << " ";
}

// Ordenaciones
while(!ord){
        // Comparaciones
        ord=true;
        for(j=0;j<5-i;j++){
                if(v[j]>v[j+1]){
                        // Intercambiar los elementos
                        aux=v[j];
                        v[j]=v[j+1];
                        v[j+1]=aux;
                        ord = false;    // Indicador de vector ordenado
                }
        }
        i++;
}

// Visualizar el vector ordenado
cout << endl;
for(int n=0; n<6; n++){
   cout << v[n] << " ";
}

getch();
return 0;
}
//---------------------------------------------------------------------------
```

Método de ordenamiento por selección (Selection sort)

Consiste en encontrar el menor de todos los elementos del arreglo o vector e intercambiarlo con el que está en la primera posición. Luego el segundo más pequeño, y así sucesivamente hasta ordenarlo todo. Su implementación requiere O(n2) comparaciones e intercambios para ordenar una secuencia de elementos.

Este algoritmo mejora ligeramente el algoritmo de la burbuja. En el caso de tener que ordenar un vector de enteros, esta mejora no es muy sustancial, pero cuando hay que ordenar un vector de estructuras más complejas, la operación de intercambiar los elementos sería más costosa en este caso. Su funcionamiento se puede definir de forma general como:

- Buscar el mínimo elemento entre una posición i y el final de la lista
- Intercambiar el mínimo con el elemento de la posición i

Puede que exista algo de discrepancia en cuanto a si es o no estable este algoritmo, pero en realidad esta implementación parece ser bastante estable. Se puede verificar esto ordenando un conjunto de datos que tenga un par de ellos con la misma clave. Se vera claramente que el orden relativo entre ellos es conservado. Algunos autores no lo consideran así, pero independientemente de esto, este algoritmo tienes entre sus ventajas: Es fácil su implementación. No requiere memoria adicional. Realiza pocos intercambios. Tiene un rendimiento constante, pues existe poca diferencia entre el peor y el mejor caso. Como todos también tiene algunas desventajas: Es lento y poco eficiente cuando se usa en listas grandes o medianas. Realiza numerosas comparaciones.

Así, se puede escribir el siguiente pseudocódigo para ordenar una lista de n elementos indexados desde el 1:

para i=1 **hasta** n-1;

```
    minimo = i;
    para j=i+1 hasta n
        si lista[j] < lista[minimo] entonces
            minimo = j
        fin si
    fin para
    intercambiar(lista[i], lista[minimo])
```

Java void selecccion(int[] a) {

```
	for (int i = 0; i < a.length - 1; i++)
	{
		int min = i;
		for (int j = i + 1; j < a.length; j++)
		{
			if (a[j] < a[min])
			{
				min = j;
			}
		}
		if (i != min)
		{
			int aux= a[i];
			a[i] = a[min];
			a[min] = aux;
		}
	}}
```

Método de ordenamiento por inserción (Insertion sort)

El Insertion Sort o Ordenamiento por Inserción es un algoritmo sencillo de ordenamiento, no muy eficiente comparado con otros mucho más rápidos como Heap Sort, Quick Sort o Merge Sort, pero muy simple de comprender e implementar, por lo que puede ser muy útil en ciertos escenarios.

El algoritmo recibe un arreglo o lista de objetos comparables y retorna el mismo ordenado, para esto realiza un ciclo comenzando desde la segunda posición y terminando en N (última posición), entonces toma cada elemento del arreglo[i] y lo inserta en su lugar correspondiente de la secuencia previamente ordenada arreglo[1...i-1]. Y de esta manera elemento a elemento vamos ordenando el arreglo.

Complejidad temporal

La complejidad temporal del algoritmo descrito es O(N^2) en el peor de los casos (cuando el arreglo está ordenado en orden inverso), pero puede tener un mejor comportamiento (O(NK)) en arreglos parcialmente ordenados, donde cada elemento esté a lo sumo K posiciones de su lugar correspondiente.

Estabilidad

Se dice que un algoritmo de ordenamiento es estable cuando elementos iguales mantienen el orden relativo que llevaban en el preorden. El Insertion Sort lo mantiene por tanto es estable.

C

```c
void Insercion(int numbers[], int array_size) {
  int i, a, index;
  for (i=1; i < array_size; i++) {
    index = numbers[i];
    a = i-1;
    while (a >= 0 && numbers[a] > index) {
      numbers[a + 1] = numbers[a];
      a--;
    }
    numbers[a+1] = index;
  }
}
```

```
vector <int> InsertionSort( vector <int> arreglo ) {
    // Comenzamos desde el segundo elemento ( índice 1 )
    for( int i = 1 ; i < arreglo.size() ; i++ )
    {
        // Obtenemos el elemento pivote que vamos a
insertar
        int elemento = arreglo[i];
        int j = i;
        // Verificamos si aún no estamos en la posición que
le corresponde al elemento
        for( ; j >= 1 && arreglo[j-1] > elemento ; j-- )
            // Hacemos un corrimiento del elemento en j-1 a
j ya que este es mayor que el pivote
            arreglo[j] = arreglo[j-1];
        // Colocamos al pivote en su posición
correspondiente
        arreglo[j] = elemento;
    }
    return arreglo;
}
```