Técnicas Sort & Find
Ordenamiento y Búsqueda sobre vectores

Procesamiento de Ordenación y Búsqueda de información.

Los computadores se emplean frecuentemente para almacenar y recuperar grandes volúmenes de datos. Con su velocidad y facilidad de acceso, los computadores aventajan a otros medios de almacenamiento como el papel y las microfichas.

A medida que la información se almacena en una computadora, la recuperación, ordenación y búsqueda de esa información se convierte en una tarea principal de las computadoras.

Es importante estudiar la forma en que los computadores pueden almacenar los datos, de modo que su recuperación (búsqueda) sea rápida. Para lograr esto, y puesto que usualmente los usuarios requieren que los datos recuperados cuenten con algún orden particular, también es importante estudiar algoritmos para ordenar los datos almacenados.

En todos los ejemplos se supondrá que existe un vector que contiene N datos. Para el caso de la búsqueda, el problema consiste en averiguar si un determinado dato está o no en el vector, y si es así, queremos saber su posición.

Para el caso del ordenamiento, el problema consiste en ordenar el vector en forma ascendente (de menor a mayor). Si se quisiera trabajar con ordenamiento descendente los cambios serían mínimos.

En esta sección estudiaremos varios algoritmos de búsqueda y ordenamiento que difieren entre sí en cuanto a su complejidad y eficiencia.

Sort & Find:

La **Ordenación** (clasificación*)* es un proceso de organización de datos en algún orden o secuencia especifica (ascendente o descendente), para datos numéricos, o alfabéticamente, para datos de caracteres.

La **Búsqueda** (searching*)* de información un proceso que está relacionado con las tablas de consultas (lookup), esta tablas almacenan listas de datos en parejas. Ej. Las parejas de datos en los diccionarios (Termino: definición), listas de precios, de evaluaciones académicas de alumnos, índices de temas.

Ordenación.

La ordenación (Clasificación), consiste en organizar un conjunto de datos en algún orden dado, ascendente o descendente de datos numéricos, o bien en orden alfabético directo o inverso.

Entre las estructuras de datos que requieren ordenación están las listas numéricas, listas de clientes, agendas telefónicas, etc.

En suma: el proceso de ordenación organiza objetos en orden (orden numérico para los números y alfabéticos para los caracteres) en forma ascendente – valor predeterminado – o descendente.

Los métodos de ordenación se dividen en dos categorías.

- Ordenación de vectores y tablas (arrays).
- Ordenación de archivos.

La ordenación de arrays se denomina también Ordenación interna, ya que se lleva a cabo en la memoria RAM.

La ordenación de archivos se realiza sobre soportes de almacenamiento externo, discos cintas, etc., y por ello se le denomina también ordenación externa. Son más lentos pero útiles en el tratamiento de información masiva.

Ordenación interna: Clasificación de los valores de un vector según el orden ascendente o descendente, en memoria central: (muy rápida)

Ordenación externa: Clasificación de los registros de un archivo situado en una soporte externo: (menos rápida).

Ordenamiento por Intercambio: (Método de la Burbuja)

Algoritmo. Proceso iterativo que realiza el ordenamiento utilizando comparación de par de elementos contiguos e intercambio de variables utilizando una variable auxiliar. En cada iteración se analiza cual elemento es mayor y utilizando la variable auxiliar se realiza el cambio de orden si es necesario.

```
iteracion:
      comparacion:
      si( ítem_a > ítem_b )entonces
            intercambio:
                  aux <- ítem_a
                  ítem_a <- ítem_b
                  ítem_b <- aux
      fin_si
```

Sea el vector de 9 ítems: v[9]

	1	2	3	4	5	6	7	8	9
Punto de partida:	**[8]**	**[7]**	**[9]**	**[6]**	**[4]**	**[5]**	**[2]**	**[3]**	**[1]**

PASOS:

```
Iteración 1:
                    1ra.   Comparación
                    [8]    [7]    [9]    [6]    [4]    [5]    [2]    [3]    [1]
                    si([8] > [7])entonces
                           intercambio: [7]    [8]
                    fin_si
                           2da.   comparación.
                    [7]    [8]    [9]    [6]    [4]    [5]    [2]    [3]    [1]
                    si([8] > [9])entonces
                           intercambio:
                    fin_si
                                  3ra.   comparación.
                    [7]    [8]    [9]    [6]    [4]    [5]    [2]    [3]    [1]
                    si([9] > [6])entonces
                           intercambio: [6]    [9]
                    fin_si
                                         4ta.   comparación.
                    [7]    [8]    [6]    [9]    [4]    [5]    [2]    [3]    [1]
                    si([9] > [4])entonces
                           intercambio: [4]    [9]
                    fin_si
                                               5ta.   comparación.
                    [7]    [8]    [6]    [4]    [9]    [5]    [2]    [3]    [1]
                    si([9] > [5])entonces
                           intercambio: [5]    [9]
                    fin_si
                                                      6ta.   comparación.
                    [7]    [8]    [6]    [4]    [5]    [9]    [2]    [3]    [1]
                    si([9] > [2])entonces
                           intercambio: [2]    [9]
                    fin_si
                                                             7ma.   comparación.
                    [7]    [8]    [6]    [4]    [5]    [2]    [9]    [3]    [1]
                    si([9] > [3])entonces
                           intercambio: [3]    [9]
                    fin_si
                                                                    8va.   comparación.
                    [7]    [8]    [6]    [4]    [5]    [2]    [3]    [9]    [1]
                    si([9] > [1])entonces
                           intercambio: [1]    [9]
                    fin_si
                                                                    resultado:
                    [7]    [8]    [6]    [4]    [5]    [2]    [3]    [1]    [9]
```

Primera iteración: Expresado en Pseudocodigo

```
//iteracion
desde(j ← 1 hasta n-1)hacer
       //comparacion
       si(v[j] > v[j+1])entonces
              intercambio(v[j], v[j+1])
       fin_si
fin_desde
```

Siendo el proceso intercambiar entre si los valores de los elementos: v[i], v[i+1] consecutivos, la siguiente secuencia de acciones, apoyados en una variable auxiliar.

Intercambiar: Expresado en Pseudocodigo

```
//intercambiar
       aux  ← v[j]
       v[j] ← v[j+1]
       v[j+1] ← aux
```

Siguientes PASOS: Hasta terminar el ordenamiento del vector

```
Iteración 2:
            comparación: true
                  intercambio:
                                                        resultado:
            [7]   [6]   [4]   [5]   [2]   [3]   [1]   [8]   [9]

Iteración 3:
            comparación: true
                  intercambio:
                                                        resultado:
            [6]   [4]   [5]   [2]   [3]   [1]   [7]   [8]   [9]

Iteración 4:
            comparación: true
                  intercambio:
                                                        resultado:
            [4]   [5]   [2]   [3]   [1]   [6]   [7]   [8]   [9]

Iteración 5:
            comparación: true
                  intercambio:
                                                        resultado:
            [4]   [2]   [3]   [1]   [5]   [6]   [7]   [8]   [9]

Iteración 6:
            comparación: true
                  intercambio:
                                                        resultado:
            [2]   [3]   [1]   [4]   [5]   [6]   [7]   [8]   [9]

Iteración 7:
            comparación: true
                  intercambio:
                                                        resultado:
            [2]   [1]   [3]   [4]   [5]   [6]   [7]   [8]   [9]

Iteración 8:
            comparación: true
                  intercambio:
                                                        resultado:
            [1]   [2]   [3]   [4]   [5]   [6]   [7]   [8]   [9]
```

Resumen: Estados sucesivos (Método de la burbuja)

	Comparaciones: (n-1)								
Iteraciones: (n-1)	[7]	[8]	[6]	[4]	[5]	[2]	[3]	[1]	[9]
	[7]	[6]	[4]	[5]	[2]	[3]	[1]	[8]	[9]
	[6]	[4]	[5]	[2]	[3]	[1]	[7]	[8]	[9]
	[4]	[5]	[2]	[3]	[1]	[6]	[7]	[8]	[9]
	[4]	[2]	[3]	[1]	[5]	[6]	[7]	[8]	[9]
	[2]	[3]	[1]	[4]	[5]	[6]	[7]	[8]	[9]
	[2]	[1]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Proceso de ordenamiento expresados en Pseudocodigo

Metodo 1. n-1 Iteraciones y n-1 comparaciones

```
Algoritmo.  Ordenamiento Por Intercambio de vectores
var
      entero: i, j, aux
inicio
      desde(i ← 1 hasta n-1)hacer           // iteraciones n-1
            desde(j ← 1 hasta n-1)hacer     // comparaciones n-1
                  si(v[j] > v[j+1])entonces
                        //intercambio
                        aux ← v[j]
                        v[j] ← v[j+1]
                        v[j+1] ← aux
                  fin_si
            fin_desde
      fin_desde
fin
```

Método 2. Mejora en la velocidad de ejecución del algoritmo.

Obsérvese que en la primera pasada o iteración del algoritmo, el número de comparaciones: n-1 mueve el mayor valor del vector a la última posición de orden: v[n] , por consiguiente, en la siguiente pasada o iteración, no es necesario comparar v[n-1] y v[n], es decir que en la 2da iteración el límite superior del bucle puede ser n -2, y así sucesivamente en cada iteración.

Entonces, después de cada pasada se puede decrementar en uno el número de comparaciones(límite superior del bucle), pues estos ya están ordenados!. El algoritmo seria:

```
Algoritmo.  Ordenamiento Por Intercambio Mejorado
var
      entero: i, j, aux
inicio
      desde(i ← 1 hasta n-1)hacer           // iteraciones n-1
            desde(j ← 1 hasta n-i)hacer     // comparaciones n-i
                  si(v[j] > v[j+1])entonces
                        //intercambio
                        aux ← v[j]
                        v[j] ← v[j+1]
                        v[j+1] ← aux
                  fin_si
            fin_desde
      fin_desde
fin
```

```
Algoritmo.  Burbuja-01
tipo
      array[1..9] de entero: vector
var
      vector: v
      entero: i, j, aux
inicio
      //operación, lectura de vector
      desde(i <- 1 hasta 9)hacer
            leer(v[i])
      fin_desde

      //ordenación del vector
      desde(i ← 1 hasta n-1)hacer           // iteraciones n-1
            desde(j ← 1 hasta n-1)hacer     // comparaciones n-1
                  si(v[j] > v[j+1])entonces
                        //intercambio
                        aux ← v[j]
                        v[j] ← v[j+1]
                        v[j+1] ← aux
                  fin_si
            fin_desde
      fin_desde

      //operación de escritura del vector ordenado
      desde(i <- 1 hasta 9)hacer
            escribir(v[i])
      fin_desde

fin
```

Implementación. Ordenamiento Método 1 en código C++

```
/*************************************************************************
 * Función que ordena el vector utilizando el método de intercambio de valor*
 * para lo cual se utiliza la variable auxiliar aux                       *
 *************************************************************************/

void ordenarPorIntercambio(int v[], int n){
//var
      int aux;
      int i,j;
//inicio
            for(i=0; i < n-1; i++){ //hacer          //n-1 pasadas
                  for(j=0; j < n-1; j++){ //hacer    //n-1 comparaciones
                        if(v[j] > v[j+1]){ //entonces
                              //intercambio
                              aux = v[j];
                              v[j]= v[j+1];
                              v[j+1]= aux;
                        }//fin_si
                  }//fin_desde
            }//fin_desde
}//fin_procedimiento
```

Material de Trabajo:

Programa ejemplo que usa la función

Archivo de entrada

Ejemplo de ejecución

Búsqueda (searching).

La búsqueda de información, al igual que la ordenación, es otra operación muy frecuente en el tratamiento de la información. La búsqueda es una actividad cotidiana: Búsqueda de palabras en el diccionario, nombres en la guía telefónica, ubicación de libros en la lista de bibliotecas.

Búsqueda Lineal

Algoritmo:

Recorrer el vector de inicio a fin, comparando el dato buscado con cada elemento del arreglo.

Implementación 1:

Mediante una función que implemente el algoritmo descrito. La función retornará la posición en que se encontró el dato, o bien la cantidad de elementos en el vector en el caso en que no se encuentre (esto sería un subíndice inválido).

```
/*****************************************************************************
 * Funcion que realiza una busqueda lineal de un valor entero en un vector de enteros.
 * La funcion recibe como parametros el vector, la cantidad de elementos que hay en el
 * vector (N) y el valor entero que se busca.
 * Si se encuentra el valor entonces se devuelve la posicion de este, sino se devuelve
 * la cantidad de elementos del vector.
 *****************************************************************************/

int busquedaLineal (int Numeros[], int N, int NumeroBuscado) {
//var
        /* Inicializar el indice para recorrer el vector de numeros */
        int i = 0;
//inicio
        /* Recorrer el vector mientras no se llegue a su fin y no
        se encuentre el numero buscado */
        while ((i < N) && (NumeroBuscado != Numeros[i])){ //hacer
                i++;
        }//fin_mientras
        /* Retornar la posicion del numero o la cantidad de elementos */
        return(i);
}//fin_funcion
```

Implementación 2: La función retornará la posición en que se encontró el dato, caso contrario el indicador -1 (posición -1 indica que no se encontró)

```
/*****************************************************************************
 * Si se encuentra el valor entonces se devuelve la posicion de este, sino
 * se devuelve -1 indicador de: no se encuentra.
 *****************************************************************************/

int busquedaLineal(int n[], int rango, int valorBuscado) {
//var
        int i, pos;      //contador e indicador de posicion si se encuentra
//inicio
        pos =-1;         //*indicador no se encuentra
        i = 0;
        /* Recorrer el vector mientras no llegue a su fin */
        while ( i < rango ){//hacer
              if( valorBuscado == n[i]){//comparar valor buscado
                  pos = i;        //indice o posicion encontrada
                  i= rango-1;     //abandonar bucle
              }//fin_si
               i++;
        }//fin_mientras

        /* Retornar la posicion del elemento o indicador de ausencia de elementos */
        return(pos);
}//fin_funcion
```

Material de Trabajo:

- Programa ejemplo que usa la función
- Archivo de entrada
- Ejemplo de ejecución