Principios de Programación

Algoritmos de Búsqueda y Ordenación

1. Algorimos de ordenación

Discutiremos el problema de ordenar un array de elementos. A los efectos de simplificar asumiremos que los arrays contienen solo enteros aunque obviamente estructuras más complicadas son posibles. Asumiremos también que el ordenamiento completo se puede realizar en memoria principal o sea la cantidad de elementos es relativamente pequeo (menos de un millón).

Ordenamientos que no se pueden realizar en memoria deben realizarse en disco. Se encuentran ejemplos en la bibliografía.

1.1. Preliminares

Los algoritmos que describiremos reciben como argumentos un array que pasa los elementos y un entero que representa la cantidad de elementos. Asumiremos que N (el número de elementos) es un número legal. Para alguno de los programas que veremos será conveniente utilizar un sentinela en posición 0, por lo cual nuestros array irán del 0 al N. Los datos irán del 1 al N.

Asumiremos la existencia de operadores de comparación $<$ y $>$. Llamaremos al ordenamiento que se basa en el uso de estos operadores *ordenamiento basado en comparaciones*.

1.2. Ordenamiento Burbuja

Consideremos el programa de ordenamiento **burbuja** que ordena un array de enteros en orden creciente:

```
void burbuja(int a[],int n)
{
        int i,j,temp;

        for(i=0;i<n-1;i++)
            for(j=0;j<n-1;j++)
                if (a[j]>a[j+1])
                  intercambio(&a[j],&a[j+1]);
}


void intercambio(int *a, int *b)
{
            int aux;
```

```
                aux=*a;
                *a = *b;
                *b=aux;
}
```

En cada ejecucion del for interior (en el que varia j) se coloca el elemento mayor en su lugar. Realizo esta repetición tantas veces como elementos.

1.3. Ordenamiento por inserción (Insertion Sort)

Es uno de los algoritmos más simples. Consiste en $N-1$ pasadas. En las pasadas 2 a N se cumplirá que los elementos de las posiciones 1 a P están ordenados. En la pasada P movemos el elemento P-esimo a su lugar correcto, este lugar es encontrado en las posiciones de los elementos 1 a P. El programa es el siguiente:

```
void Sort_por_insercion (int A[], int N)
{
        int j,P,tmp;
        {
            A[0]=Min_int;
            for(P=2;P<=N;P++)
                {
                 j=P;
                 tmp=A[P];
                 while(tmp < A[j-1])
                 {
                        A[j] = A[j-1];
                        j-=1;
                 }
                 A[j]=tmp;
                }
        }
}
```

La idea del programa es la siguiente: coloco un centinela en la primer posición (posición 0) por lo cual el elemento a insertar será colocado en caso de que sea el mínimo en la posición 1. Guardamos el valor del elemento a insertar en una variable auxiliar tmp. Si tmp es menor que A[j-1] su posición será anterior o igual a (j-1), luego corro el elemento de la posición (j-1) a la posición j para hacer lugar para tmp. Repito esto para todos los elementos anteriores al tmp. Cuando encuentre una posición tal qué tmp no es menor indica que los elementos anteriores están ordenados (estaban ordenados del 0 al $P-1$). Luego lo unico que resta es colocar tmp en la posición j.

1.4. Ordenamiento por Selección (Selection Sort)

La idea del selection sort es la siguiente : en la pasada i-esima seleccionamos el elemento menor entre $A[i], \ldots, A[n]$ y lo intercambiamos con el $A[i]$. Como

resultado, luego de i pasadas los menores i elementos ocuparán las posiciones $A[1], \ldots, A[i]$ y además los elementos de dichas posiciones estarán ordenados. El programa es el siguiente:

```
void Sort_por_seleccion (int A[], int N)
{
        int i,j,sel,clave_sel;
        {

            for(i=1;i<N;i++)
                {
                 sel=i;
                 clave_sel=A[i];
                 for(j=i+1;j<=N;j++)
                 {
                    if (A[j]<clave_sel)
                       { clave_sel=A[j];
                         sel=j;
                       }
                 }
                 intercambio(A[i],A[sel]);
                }
        }
}
```

2. Algoritmos de Búsqueda

Con frecuencia el programador trabajará con grandes cantidades de información almacenada en arreglos. Podría ser necesario determinar si algún arreglo contiene un valor que sea igual a cierto valor clave.

El proceso para encontrar un elemento particular en un arreglo se llama búsqueda. Estudiaremos dos técnicas de búsqueda: una técnica simple llamada busqueda lineal y una más eficiente llamada busqueda binaría.

Ambos programas se pueden implementar recursivamente o no. En este capítulo veremos la implementación no recursiva.

2.1. Búsqueda lineal

La búsqueda lineal compara los elementos del array con la clave de búsqueda hasta que encuentra el elemento o bien hasta que se determina que no se encuentra.

```
int busqueda_lineal (int A[], int clave, int n)
{
            for(int i=0;i<n;i++)
                if (A[i]==clave) return i;
```

```
        return -1;
}
```

Este método funciona bien con arreglos pequeos y con los no ordenados. En arreglos grandes u ordenados conviene aplicar la búsqueda binaria que es más eficiente.

2.2. Búsqueda binaria

Dados un entero X y un array A de n enteros que se encuentran ordenados y en memoria, encontrar un i talque $A[i] == X$ o retornar 0 si X no se encuentra en el array (consideramos los elementos del array de 1 a N.

La estrategia consiste en comparar X con el elemento del medio del array, si es igual entonces encontramos el elemento, sino, cuando X es menor que el elemento del medio aplicamos la misma estrategia al array a la izquierda del elemento del medio y si X es mayor que el elemento del medio aplicamos la misma estrategia al array a la derecha de dicho elemento.

Para simplificar el código definimos A[0]=X.

```
int busqueda_binaria(int A[],int X,int n)
{
        int izq=1,medio,der=n;

        A[0]=X;
        do
        {
            medio = (izq+der) / 2;
            if(izq > der)
              medio=0;
            else if A[medio] < X
                    izq = medio +1;
                 else der=medio-1;
        }
        while (A[medio] != X);
        return medio;
}
```