Algoritmos y Estructura de Datos

Unidad 1 – Semana 6

Logro de sesión

Al finalizar la sesión, el estudiante aplica algoritmos de ordenamiento

Sesión 6 : Algoritmos Fisher-Yates Shuffle y de ordenamiento simple

Contenido:

- Introducción
- Fisher-Yates Shuffle
- Ordenamiento intercambio
- Ordenamiento burbuja
- Ordenamiento selección
- Ordenamiento inserción
- Ordenamiento Shell

Introducción

- El ordenamiento de los datos consiste en disponer un conjunto de datos (o una estructura) en algún determinado orden con respecto a alguno de sus campos.
  - Clave: Campo por el cual se ordena.

- Según donde estén almacenados los datos a ordenar, podemos decir que el ordenamiento es:
  - Interno: Arrays, listas o árbol. Típicamente en RAM.
  - Externo: En Archivos en discos.

Algoritmo Fisher-Yates Shuffle

- Similar al proceso de escuchar pistas de música aleatoriamente en un reproductor de audio.
- Este algoritmo de Fisher-Yates tiene una trayectoria bastante larga. Fue presentado por los británicos Ronald A. Fisher (estadístico y biólogo) y Frank Yates (también estadístico) en su obra Statistical tables for biological, agricultural and medical research, publicada por primera vez en 1938.

Algoritmo Fisher-Yates Shuffle

- Produce una uniforme permutación aleatoria de un arreglo de entrada en un tiempo O(n).
- Los principales pasos son:
  - Tenemos una matriz de N elementos, A [N].
    1. Se itera la matriz desde el último elemento.
    2. Si está en el i-ésimo elemento desde el final, genera un número aleatorio j, en el rango [0, i].
    3. Se intercambia A [i] y A [j].
    4. Continúe hasta llegar al elemento en el índice 1, es decir, A [1].

Algoritmo Fisher-Yates Shuffle

- Implementando

```
1  #include <iostream>
2
3  using namespace std;
4
5  void main() {
6      char* barajas[] = {"A","F","C","X" ,"R" };
7      int N = 5;
8
9      for (int i = N - 1; i>0; i--) {
10         int j = (rand() % (i + 1));
11         swap(barajas[j], barajas[i]);
12     }
13
14     int x = 0;
15     while (x < N) {
16         cout << barajas[x];
17         x++;
18     }
19     cin.get();
20 }
```

Ordenamiento por intercambio

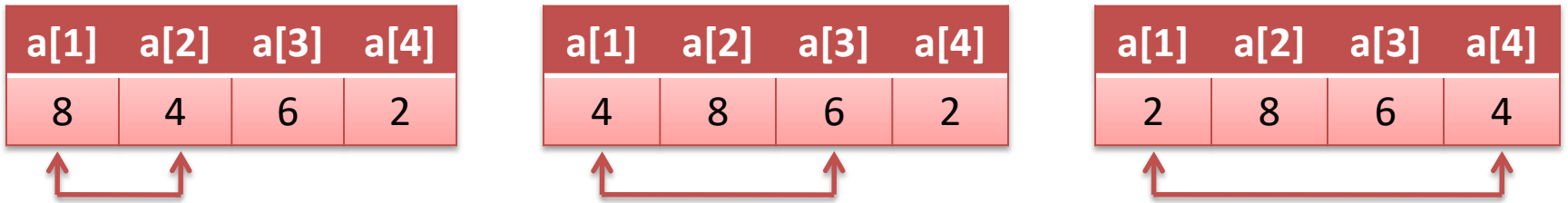
- El más sencillo de todos, conocido como burbuja. Se basa en:
  - La lectura sucesiva de la lista a ordenar,
  - Comparando el elemento inferior de la lista con todos los restantes
  - Efectuando el Intercambio de posiciones cuando el orden resultante no sea correcto.

- Siendo n la cantidad de elementos, Realizará al menos n–1 pasadas

Ordenamiento Intercambio

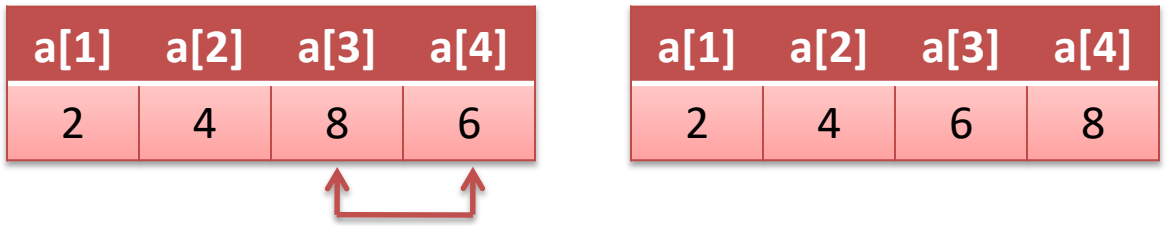
- Ejemplo

Pasada 1: Se compara a[1] con todos, así primero se cambia a[1] con a[2] pues a[1] > a[2] y debe ser Ascendente, es decir a[1]<a[2] …y por último a[1] con a[4]



Pasada 2: El elemento mas pequeño esta en a[1] y se analiza la sublista restante.
Al cabo de la pasada, el segundo mas chico esta en a[2]….

Pasada 3:



Pasada i: Al cabo de la pasada i, el elemento de orden i, está en a[i]

Algoritmo: Ordenamiento Intercambio

```
Algoritmo ordIntercambio (a[], n)
        i, k, aux: enteros  /* se realizan n-1 pasadas */

        para i desde 1  hasta n-1 hacer
          para k desde  i+1  hasta  n  hacer
                si a[i] > a[k] entonces
                   aux ← a[i]
                   a[i] ← a[k];
                   a[k] ← aux ;
```

Complejidad (n)(n–1)
Del Orden $F(n)=n^2$.

Mejor Caso = $O(n^2)$
Peor Caso = $O(n^2)$

Ordenamiento Intercambio

- Implementando

```
1   #include <iostream>
2
3   using namespace std;
4
5   void ordIntercambio(int a[], int n)
6   {
7       for (int i = 0; i < n - 1; i++)
8       {
9           for (int k = i + 1; k<n; k++)
10          {
11              if (a[i] > a[k])
12              {
13                  int aux = a[i];
14                  a[i] = a[k];
15                  a[k] = aux;
16              }
17          }
18      }
19  }
```

Ordenamiento Burbuja

- Los elementos burbujean:
  - Los mas grandes, caen al fondo del arreglo (posición n)
  - Los mas chicos suben a la cima (posición 1).
- Estudia parejas de elementos Adyacentes
  - a[1] y a[2], a[2] y a[3]…a[i] y a[i+1]… a[n–1] y a[n].
  - Si a[i+1] < a[i] Entonces Los INTERCAMBIA
- Algoritmo:
  - Pasada 1: considera desde (a[1], a[2]) hasta (a[n–1], a[n]).
    - En a[n] esta el elemento mas grande.
  - Pasada 2: considera desde (a[1], a[2]) hasta (a[n–2], a[n–1]).
    - En a[n–1] esta el segundo elemento mas grande.
  - Pasada i: considera desde (a[1], a[2]) hasta (a[n–i], a[n-i+1]).
    - En a[n–i+1] esta el elemento de orden i.
  - El proceso termina con la pasada n–1
    - El elemento mas pequeño esta en a[1].

Ordenamiento Burbuja

Pasada 1:

a[1]	a[2]	a[3]	a[4]	a[5]
50	20	40	80	30

a[1]	a[2]	a[3]	a[4]	a[5]
20	50	40	80	30

a[1]	a[2]	a[3]	a[4]	a[5]
20	40	50	80	30

a[1]	a[2]	a[3]	a[4]	a[5]
20	40	50	80	30

a[1]	a[2]	a[3]	a[4]	a[5]
20	40	50	30	**80**

Pasada 2:

a[1]	a[2]	a[3]	a[4]	a[5]
20	40	50	30	**80**

a[1]	a[2]	a[3]	a[4]	a[5]
20	40	50	30	**80**

a[1]	a[2]	a[3]	a[4]	a[5]
20	40	30	**50**	**80**

Pasada 3:

a[1]	a[2]	a[3]	a[4]	a[5]
20	40	30	**50**	**80**

a[1]	a[2]	a[3]	a[4]	a[5]
20	40	30	**50**	**80**

a[1]	a[2]	a[3]	a[4]	a[5]
20	30	40	50	80

Algoritmo: Ordenamiento Burbuja

```
Algoritmo ordBurbuja (a[], n)
    i, k, aux: enteros

    Repetir con i desde 1 hasta n-1
        Repetir con k desde 1 hasta n-i
            si (a[k] > a[k+1]) entonces
                aux = a[k]
                a[k] = a[k+1]
                a[k+1] = aux
```

Mejor Caso = $O(n^2)$
Caso Prom. = $O(n^2)$
Peor Caso = $O(n^2)$

Algoritmo: Ordenamiento Burbuja

```
Algoritmo ordBurbujaModificado (a[], n)
        i, k, aux: enteros
        ordenado: boolean

        Repetir con i desde 1 hasta n-1
                ordenado = true
                Repetir con k desde 1 hasta n-i
                        si (a[k] > a[k+1]) entonces
                                aux = a[k]
                                a[k] = a[k+1]
                                a[k+1] = aux
                                ordenado = false
                si (ordenado) break
```

Mejor Caso = $O(n)$
Caso Prom. = $O(n^2)$
Peor Caso = $O(n^2)$

Algoritmo: Ordenamiento Burbuja

- Implementando

```
5   void ordBurbujaModificado(int a[], int n)
6   {
7       bool ordenado;
8       for (int i = 0; i < n - 1; i++)
9       {
10          ordenado = true;
11          for (int j = 0; j < n - (i + 1); j++)
12          {
13              if (a[j] > a[j + 1])
14              {
15                  int aux = a[j];
16                  a[j] = a[j + 1];
17                  a[j + 1] = aux;
18                  ordenado = false;
19              }
20          }
21          if (ordenado) break;
22      }
23  }
```

Ordenamiento Selección

- Realiza sucesivas pasadas que
  - Buscan el elemento más pequeño de la lista a y lo escribe al frente de la lista a[1].
  - Considera las posiciones restantes, a[2]…a[n]
  - Finaliza cuando ya no hay Posiciones Restantes.

- En la pasada i
  - Está Ordenado: desde a[1] hasta a[i–1].
  - Está Desordenado: Desde a[i] hasta a[n].

- El proceso continua n–1 vueltas.

Ordenamiento Selección

Lista Original:

a[1]	a[2]	a[3]	a[4]
51	21	39	80

Pasada 1: Lista entre 1 y 4. Selecciona el menor (21) y lo pasa al a[1]

a[1]	a[2]	a[3]	a[4]
21	51	39	80

Pasada 2: Lista entre 2 y 4. Selecciona el menor (39) y lo pasa al a[2]

a[1]	a[2]	a[3]	a[4]
21	39	51	80

Pasada 3: Lista entre 3 y 4. No selecciona nada.

a[1]	a[2]	a[3]	a[4]
21	39	51	80

Algoritmo: Ordenamiento Selección

```
Algortimo ordSeleccion (a[], n)
        menor, k, j: enteros;
        Repetir con i desde 1 hasta n-1
        k = i      /* comienzo de la exploración en índice i */
        menor = a[i]
        Repetir con j desde  i+1 hasta  n
                si  a[j] < menor entonces
                        menor = a[j]
                        k = j
                a[k] = a[i]
                a[i] = menor
```

Complejidad (n (n–1))/2
Del Orden $F(n)=n^2$.

Mejor Caso = $O(n^2)$
Caso Prom. = $O(n^2)$
Peor Caso = $O(n^2)$

Algoritmo: Ordenamiento Selección

- Implementando

```
4
5   void ordSeleccion(int a[], int n)
6   {
7       int k, menor;
8       for (int i = 0; i < n - 1; i++)
9       {
10          k = i;
11          menor = a[i];
12          for (int j = i + 1; j<n; j++)
13          {
14              if (a[j] < menor)
15              {
16                  menor = a[j];
17                  k = j;
18              }
19          }
20          a[k] = a[i];
21          a[i] = menor;
22      }
23  }
24
```

Ordenamiento Inserción

- Similar al proceso de ordenar tarjetas en un tarjetero por orden alfabético:
  - Consiste en insertar un elemento en su posición correcta, dentro de una lista que ya está Ordenada.

- Algoritmo:
  - El 1er elemento a[1] se lo considera ordenado.
  - Se inserta a[2] en la posición correcta, delante o detrás del a[1], según sea mayor o menor.
  - Por cada bucle i (desde i = 2 hasta n) se explora la sublista a[1]..a[i–1] buscando la posición correcta de inserción del elemento a[i].
  - Al dejar vacía la posición a[i] se impone un desplazamiento de todo el vector, desde el lugar de inserción.

Ordenamiento Inserción

Lista Original:

a[1]	a[2]	a[3]	a[4]
51	21	10	15

Pasada 1: Comenzamos en a[2] desplazando a la derecha todos los valores que sean mayores a 21 (a[1])

a[1]	a[2]	a[3]	a[4]
21	51	10	15

Pasada 2: El 10 lo mueve hasta a[1]

a[1]	a[2]	a[3]	a[4]
10	21	51	15

Pasada 3: El 15 lo mueve hasta a[2].

a[1]	a[2]	a[3]	a[4]
10	15	21	51

Ordenamiento Inserción

```
Algoritmo ordInsercion (a[], n)
        i, j, aux : enteros
   Repetir con i desde 2 hasta n
        aux = a[i]
        k = i-1
        Repetir mientras (k >= 1) y (aux < a[k])
                a[k+1] = a[k]
                k = k-1
        a[k+1] = aux
```

Complejidad (n (n–1))/2
Del Orden $F(n)=n^2$.

Mejor Caso = O(n)
Caso Prom. = O(n+d)
Peor Caso = $O(n^2)$

d es la cantidad de inversiones

Ordenamiento Inserción

- Implementando

```
1  #include <iostream>
2
3  using namespace std;
4
5  void ordInsercion(int a[], int n)
6  {
7      int aux, k;
8      for (int i = 1; i<n; i++)
9      {
10         aux = a[i];
11         k = i - 1;
12         while (k >= 0 && aux < a[k])
13         {
14             a[k + 1] = a[k];
15             k--;
16         }
17         a[k + 1] = aux;
18     }
19 }
```

Ordenamiento Shell

- Este algoritmo mejora el ordenamiento por inserción al dividir la lista original en varias sublistas más pequeñas, cada una de las cuales se ordena mediante un ordenamiento por inserción. La manera única en que se eligen estas sublistas es la clave del ordenamiento Shell. En lugar de dividir la lista en sublistas de ítems contiguos, el ordenamiento Shell usa un intervalo i, a veces denominado brecha, para crear una sublista eligiendo todos los ítems que están separados por i ítems.

Ordenamiento Shell

- Algoritmo
  - Dividir lista original en n/2 grupos de 2 elementos
  - Intervalo entre los elementos: n/2.
  - Clarificar cada grupo por separado.
  - Si No Están Ordenados Entonces CAMBIARLOS.
  - Dividir lista original en n/4 grupos de 4 elementos.
  - Intervalo entre los elementos: n/4.
  - Continuar sucesivamente hasta que el intervalo==1.

Ordenamiento Shell

Lista Original n=6.
Intervalo Inicial: n/2=6/2=3
    Intervalos Siguientes=IntervaloAnterior/2
Se compara a[i] con a[i+Intervalo]
    Si No Están Ordenados Entonces CAMBIARLOS

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
6	1	5	2	3	4	0

Paso	Intervalo	Parejas que Intercambian por estar desordenados	La Lista Queda
1	3	(6,2)= 2, 1, 5,6, 3, 4, 0 (5,4)= 2, 1, 4,6, 3,5, 0 (6;0)=2, 1, 4,0, 3,5, 6	2, 1, 4,0, 3,5, 6
2	3	(2, 0)=0, 1, 4,2, 3,5, 6	0, 1, 4,2, 3,5, 6
3	3	Ninguno	0, 1, 4,2, 3,5, 6
4	3/2=1	(4, 2)=0, 1, 2,4, 3,5, 6 (4, 3)= 0, 1, 2,3,4,5, 6	0, 1, 2,3, 4,5, 6
5	1	Ninguno	Lista Ordenada

Ordenamiento Shell

- Implementando

```
5   void ordShell(int a[], int n)
6   {
7       int i, j, k, intervalo = n / 2;
8       int temp;
9       while (intervalo > 0)
10      {
11          for (i = intervalo; i <= n; i++)
12          {
13              j = i - intervalo;
14              while (j >= 0)
15              {
16                  k = j + intervalo; //queda k=i;
17                  if (a[j] <= a[k]) j = -1; /*termina el bucle, par ordenado */
18                  else {
19                      temp = a[j];
20                      a[j] = a[k];
21                      a[k] = temp;
22                      j -= intervalo;
23                  }
24              }
25          }
26          intervalo = intervalo / 2;
27      }
28  }
```

El 1er while: $Log_2 n$

El for: n

$F(n)=n*Log_2(n)$

Ejercicios

- Ordenar la secuencia 8,1,4,1,5,9,2,6,5 usando:
  - Ordenamiento Inserción
  - Ordenamiento Shell para intervalos {1,3,5}

Referencias

- Sedgewick, R., et. al. (2011) Algorithms, Fourth Edition. Pearson.
- Cormen, H., et. al. (2009) Introduction to Algorithms, MIT Press.
- Allen, Mark (2014) Data Structures and Algorithms Analysis in C++, Fourth Edition. Pearson.

EXIGETE INNOVA