Universidad Centroamericana "José Simeón Cañas" (UCA)

Facultad: Ingeniería y arquitectura

Materia: Programación de estructuras dinámicas

Catedrática: Ronaldo Armando Canizales Turcios

Sección: 02

Ciclo: 02/2019

Actividad: Exchange sort

Fecha de entrega: viernes 22 de noviembre del 2019

Integrantes/Carnet:

Ricardo Adrián Aparicio Lemus 00032219

Fernando Daniel González Batarsé 00014419

Diego Alessandro Rodríguez Villalta 00101519

INTRODUCCIÓN

“Tus problemas no te definen, te definen tus soluciones”, en la vida laboral se manifestaran demasiados problemas al momento de programar, así que es necesario conocer las diferentes formas en las que se pueden solucionar, dentro de los problemas más comunes, se encuentra el problema del ordenamiento de datos, este se puede abordar con distintos algoritmos como lo pueden ser el Bubble Sort, el Insertion Sort, el Merge Sort, el Quick Sort, el Radix Sort, el Shell Sort, los Exchange Sorts, etc.

A continuación se demostrará cómo funcionan 3 algoritmos de ordenamiento siendo estos el counting sort, distribution sort y odd-even transportation sort, detallando plenamente el funcionamiento de cada uno de estos. Realizando una problemática la cual solucionaremos aplicando los tres métodos de ordenamiento, demostrando los diferentes códigos (de cada método) que se utilizarán para solventar el dilema.

Los códigos utilizados nos ayudarán a ver las características que cada método posee, pues como uno puede ser más eficiente el otro podría ser más práctico y brindarnos el dato que deseamos, lo antes dicho se demostrará dando los puntos que tiene cada método, manifestando las ventajas y desventajas que poseen tanto al momento de aplicarlo al código como en otros problemas.

A parte de manifestar cómo se aplican los métodos, se explicará en qué situación saldría conveniente cada uno de estos métodos, frente a otros que podrían ser menos prácticos al momento de obtener la eficiencia del dato deseado.

EXCHANGE SORT

Es un algoritmo el cual compara elementos adyacentes y los mueve a su posición correcta intercambiandolos, basándose en si un elemento es menor que otro. Por ejemplo, mientras estemos ordenando ascendentemente, tendremos que intercambiar elementos si el de la izquierda es mayor que el de la derecha, tendremos que repetir este proceso de comparación adyacente e intercambios, hasta que el arreglo está ordenado.

En esta demostración de los exchange sorts se presentarán tres tipos de ordenamientos que utilizan el algoritmo Exchange sort:

- Counting sort
- Distribution sort
- Odd-even transportation sort

COUNTING SORT

Es un algoritmo de ordenamiento el cual posee complejidad $O(n^2)$ en donde n es el número de elementos introducidos en el arreglo.

Funcionamiento

A [7]

6	2	3	4	3	1	8

Se crea un nuevo arreglo del mismo tamaño.

B [7]

Se cuenta los números menores a A[i] y se colocan en B[i], en caso exista un elemento de A[i] o más que se repitan se contarán también pero sólo los que se encuentran en una posición mayor.

B

5	1	3	4	2	0	6

Se crea un nuevo arreglo del mismo tamaño.

C[7]

Se colocan los elementos de A en C según b[i] de la siguiente manera c[b[i]] = a[i], haciendo que el arreglo quede ordenado.

C

1	2	3	3	4	6	8

VENTAJAS	DESVENTAJAS
Se pueden ordenar cadenas de texto y caracteres con este algoritmo	Confuso de entender cómo funciona el posicionamiento.
Se pueden utilizar números negativos	Ineficiente para ordenar grandes cantidades de datos
Fácil de codificar	No está muy bien documentado en internet.

Código implementado en el proyecto

```
165.    void count_sort(estudiante arreglo[], int n){
166.        clock_t tiempo = clock();
167.        int aux=0;
168.        int b[n];
169.        estudiante c[n];
170.        for(int i=0;i<n;i++){
171.            for(int j=0;j<n;j++){
172.                if(arreglo[j].calificacion<arreglo[i].calificacion and i!=j){
173.                    aux++;
174.                }
175.                   else if(arreglo[j].calificacion==arreglo[i].calificacion and i<j){
176.                    aux++;
177.                   }
178.            }
179.            b[i]=aux;
180.            aux=0;
181.        }
182.        for(int i=0;i<n;i++){
183.            c[b[i]].numlista=arreglo[i].numlista;
184.            c[b[i]].nombre=arreglo[i].nombre;
185.            c[b[i]].apellido=arreglo[i].apellido;
186.            c[b[i]].carne=arreglo[i].carne;
187.            c[b[i]].calificacion=arreglo[i].calificacion;
188.        }
189.        tiempo=clock()-tiempo;
190.        cout<<"Tiempo: "<<(float)tiempo/CLOCKS_PER_SEC<<endl<<endl;
191.        mostrar(c,n);
192.    }
```

DISTRIBUTION SORT

Es un algoritmo de ordenamiento el cual posee complejidad O(n+k), en donde n es el número de elementos introducidos en el arreglo y k es el rango

Funcionamiento

A [7]

6	2	3	4	3	1	8

Se crea un nuevo arreglo de tamaño: rango(mayor - menor)+1.

B[8]

1	2	3	4	5	6	7	8

A cada b[i] de B se le asigna una posición de respaldo, estas posiciones van a ir desde el menor hasta el mayor número. Se almacena en b[i] el número de veces que se encuentra esta posición de respaldo en A como elemento.

B

1_1	1_2	2_3	1_4	0_5	1_6	0_7	1_8

Se suma b[i] con b[i-1] desde i=1 hasta que termina el arreglo.

B

1_1	2_2	4_3	5_4	5_5	6_6	6_7	7_8

Se crea un nuevo arreglo del mismo tamaño que A.

C[7]

Se colocan los elementos de A en C según el elemento de B (que se encuentre en la posición de respaldo que sea igual al elemento de A) - 1, Después de asignar un número siempre se le va a restar uno al elemento de B con el que se trabajó.

C

1	2	3	3	4	6	8

VENTAJAS	DESVENTAJAS
Es mucho más eficiente que muchos algoritmos al tener una gran cantidad de números a ordenar y un rango pequeño.	Si el rango de número a utilizar es muy grande, no es eficiente.
Se pueden ordenar números negativos.	No se puede utilizar con cadenas de texto o caracteres.
Es frecuentemente utilizado como una sub-rutina frente a otro ordenamiento como radix sort.	Es muy ineficiente y más complicado con números decimales ya que se tendría que trabajar con matrices

Código implementado en el proyecto

```
127.    void distribution_sort(estudiante arreglo[],int n){
128.        clock_t tiempo = clock();
129.        estudiante arreglonuevo[n];
130.        int mayor=arreglo[0].numlista;
131.        int menor=arreglo[0].numlista;
132.        int rango=0;
133.        for(int i=0;i<n;i++){
134.            if(arreglo[i].numlista>mayor){
135.                mayor=arreglo[i].numlista;
136.            }
137.            else if(arreglo[i].numlista<menor){
138.                menor=arreglo[i].numlista;
139.            }
140.        }
141.        rango=mayor-menor+1;
142.        int arregloaux[rango];
143.        for(int i=0;i<rango;i++){
144.            arregloaux[i]=0;
145.        }
146.        for(int i=0;i<n;i++){
147.            arregloaux[arreglo[i].numlista-menor]++;
148.        }
149.        for(int i=1;i<rango;i++){
150.            arregloaux[i]+=arregloaux[i-1];
151.        }
152.        for(int i=0;i<n;i++){
153.            arreglonuevo[arregloaux[arreglo[i].numlista-menor]-1].numlista=arreglo[i].numlista;
154.            arreglonuevo[arregloaux[arreglo[i].numlista-menor]-1].nombre=arreglo[i].nombre;
155.            arreglonuevo[arregloaux[arreglo[i].numlista-menor]-1].apellido=arreglo[i].apellido;
156.            arreglonuevo[arregloaux[arreglo[i].numlista-menor]-1].carne=arreglo[i].carne;
157.            arreglonuevo[arregloaux[arreglo[i].numlista-menor]-1].calificacion=arreglo[i].calificacion;
158.            arregloaux[arreglo[i].numlista-menor]--;
159.        }
160.        tiempo=clock()-tiempo;
161.        cout<<"Tiempo: "<<(float)tiempo/CLOCKS_PER_SEC<<endl<<endl;
162.        mostrar(arreglonuevo,n);
163.    }
```

ODD-EVEN TRANSPOSITION SORT

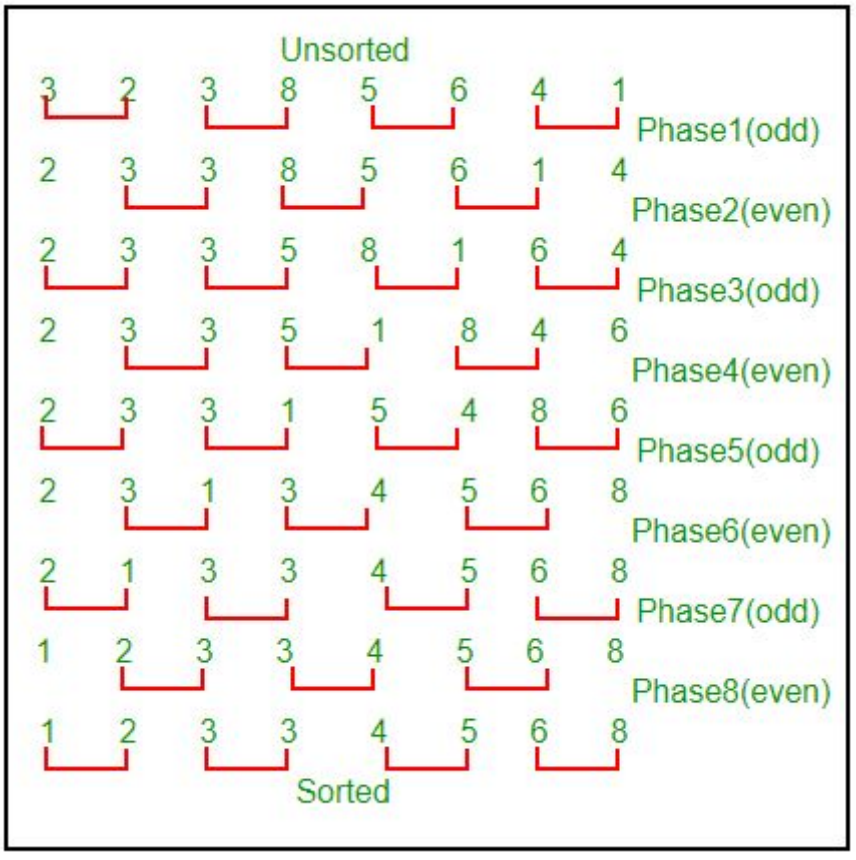
Es un algoritmo de ordenamiento el cual posee complejidad $O(n^2)$ en donde n es el número de elementos introducidos en el arreglo.

Funcionamiento

Consiste en realizar iteraciones hasta que los datos se encuentren ordenados, cada iteración se divide en dos partes:

La primera es la que se encarga de empezar las comparación con los elementos pares (odds) y hacer que se compare con el impar adyacente

La segunda es la que se responsabiliza de empezar la comparación con elementos impares (evens) y hacer que se compare con el par adyacente.



VENTAJAS	DESVENTAJAS
Pseudocódigo fácil de entender y codificar	Ineficiente si se quiere ordenar por ejemplo de mayor a menor, y el menor está al principio
Funciona con strings (palabras)	Ineficiente si se quiere ordenar por ejemplo de menor a mayor, y el mayor está al principio
Fácil de implementar debido a familiaridad (se parece al ordenamiento de burbuja)	Muy ineficiente con una gran cantidad de números

Código implementado en el proyecto

```
96.     void odd_even_sort(estudiante arreglo[],int n){
97.         clock_t tiempo = clock();
98.         bool ordenado=false;
99.         while(!ordenado){
100.            ordenado=true;
101.            for(int i=0;i<n-1; i=i+2){
102.                if(arreglo[i].apellido>arreglo[i+1].apellido){
103.                    arreglo[i].nombre.swap(arreglo[i+1].nombre);
104.                    arreglo[i].apellido.swap(arreglo[i+1].apellido);
105.                    arreglo[i].carne.swap(arreglo[i+1].carne);
106.                    swap(arreglo[i].numlista,arreglo[i+1].numlista);
107.                    swap(arreglo[i].calificacion,arreglo[i+1].calificacion);
108.                    ordenado=false;
109.                }
110.            }
111.            for(int i=1;i<n-1; i=i+2){
112.                if(arreglo[i].apellido>arreglo[i+1].apellido){
113.                    arreglo[i].nombre.swap(arreglo[i+1].nombre);
114.                    arreglo[i].apellido.swap(arreglo[i+1].apellido);
115.                    arreglo[i].carne.swap(arreglo[i+1].carne);
116.                    swap(arreglo[i].numlista,arreglo[i+1].numlista);
117.                    swap(arreglo[i].calificacion,arreglo[i+1].calificacion);
118.                    ordenado=false;
119.                }
120.            }
121.        }
122.        tiempo=clock()-tiempo;
123.        cout<<"Tiempo: "<<(float)tiempo/CLOCKS_PER_SEC<<endl<<endl;
124.        mostrar(arreglo,n);
125.    }
```

CONCLUSIONES

Este tipo de algoritmo son muy efectivos con un bajo rango de números rango de números siendo son muy fáciles de implementar y entender, por lo cual, lo hace ideal, para explicar los conceptos básicos de ordenamiento, sin embargo su poca practicidad, los hace inefectivos para el uso e implementación en la vida real.

Consideramos que estos algoritmos se pueden añadir a cualquier código básico en el cual se necesite ordenar rápidamente una poca cantidad de números, ya que son fáciles de implementar y sencillos de entender, por los cuales son ideales para sacar de apuros en los que se necesite ordenar y no se conocen otros algoritmos más complejos.

A su vez tenemos que los métodos utilizados poseen características propias, es decir, que se pueden aplicar a distintos problemas. Hay distintas problemáticas en el día a día que necesitan solución mediante un tipo de ordenamiento, es por ello que los métodos explicados en el informe son convenientes; Tomando en cuenta que cada método es más efectivo que el otro dependiendo el contratiempo a resolver.

PREGUNTAS DEL MOODLE

1- ¿Cuáles son las ventajas del Distribution Sort?
Respuesta(cualquiera es válido):

- Pseudocódigo fácil de entender y codificar
- Funciona con strings (palabras)
- Fácil de implementar debido a familiaridad (se parece al ordenamiento de burbuja)

2- ¿Cuáles son las desventajas del Odd-Even Transportation Sort?
Respuesta(cualquiera es válido):

- Ineficiente si se quiere ordenar por ejemplo de mayor a menor, y el menor está al principio
- Ineficiente si se quiere ordenar por ejemplo de menor a mayor, y el mayor está al principio
- Muy ineficiente con una gran cantidad de números

3- ¿Cuáles son las ventajas del Counting Sort?

- Se pueden ordenar cadenas de texto y caracteres con este algoritmo
- Se pueden utilizar números negativos
- Fácil de codificar

Bibliografía:

- Ronaldo Canizales. [0219PED]Proyecto. Recuperado de: https://drive.google.com/drive/folders/1I8-2Ob94OqrXcpgiORPL-7MXU7oxdCBR
- Mithun kumar. Counting sort. Recuperado de: https://www.geeksforgeeks.org/counting-sort/
- Shivam Garg. Odd-Even Transportation Sort. Recuperado de: https://www.geeksforgeeks.org/odd-even-transposition-sort-brick-sort-using-pthreads/
- Ayush Sharma. Algorithms in Python: Exchange Sort. Recuperado de: https://dev.to/ayushsharma/algorithms-in-python-exchange-sorts-3kkd