Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

Act 2.3 – Actividad Integral estructura de datos lineales (Evidencia Competencia)

Realiza en forma individual una investigación y reflexión de la importancia y eficiencia del uso de las listas doblemente ligadas en una situación problema de esta naturaleza, generando un documento llamado “ReflexAct2.3.pdf”

La implementación de los algoritmos de ordenamiento y de búsqueda son importantes y esenciales, el uso adecuado y eficiente son útiles para poner datos en forma canónica y para generar resultados legibles por humanos.

Los procesos de búsqueda involucran recorrer un arreglo completo con el fin de encontrar algo. Lo más común es buscar el menor o mayor elemento (cuando se puede establecer un orden), o buscar el índice de un elemento determinado.

- Búsqueda Secuencial.

  Consiste en ir comparando el elemento que se busca con cada elemento del arreglo hasta cuándo se encuentra.

  - Eficiencia y Complejidad:

    Mejor caso:
    O(1): El elemento buscado está en la primera posición. Es decir, se hace una sola comparación.

    Peor caso:
    O(n): El elemento buscado está en la última posición. Necesitando igual cantidad de comparaciones que de elementos el arreglo.

    En promedio:
    O(n/2): El elemento buscado estará cerca de la mitad. Necesitando en promedio la mitad de comparaciones que de elementos.

- Búsqueda Binaria.

  Compara si el valor buscado está en la mitad inferior o mitad superior. En la que esté, subdivido nuevamente, y así sucesivamente hasta encontrar el valor.

  - Eficiencia y Complejidad:

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

Mejor caso:
O(1): El elemento buscado está en el centro. Por lo tanto, se hace una sola comparación.
Peor caso:
O(log(n)): El elemento buscado está en una esquina. Necesitando $log_2(n)$ cantidad de comparaciones.

En promedio:
O($log_2(n/2)$): Serán algo como $log_2(n/2)$.

Ordenar un conjunto de elementos de un alista es una tarea que se presenta con frecuencia en la programación. Un programa de computadora debe seguir una secuencia de instrucciones exactas para lograrlo. Esta secuencia de instrucciones se llama algoritmo. Un algoritmo de ordenamiento es un método que puede utilizarse para colocar una lista de elementos de una secuencia ordenada. La secuencia de ordenamiento está determinada por una clave. Existen varios algoritmos de ordenamiento y difieren en cuanto a su eficiencia y rendimiento.

Los algoritmos se clasifican por:

- Lugar donde se realice la ordenación:
    - Algoritmo de ordenamiento interno:
      En la memoria del ordenador.

    - Algoritmos de ordenamiento externo.
      En un lugar externo con un disco duro.

- Por el tiempo que tardan en realizar la ordenación, dadas entradas ya ordenadas o inversamente ordenadas:
    - Algoritmos de ordenación natural.
      Tarda lo mínimo posible cuando la entrarda está ordenada.

    - Algoritmos de ordenación no natural.
      Tarde lo mínimo posible cuando la entrada está inversamente ordenada.

- Por estabilidad.

- Complejidad computacional.

- Uso de memoria y otros recursos computacionales.

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	O(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

VECTORES

También para la creación de los algoritmos implementados usamos el concepto de la tokenización, para poder usarlo añadimos el vector por el encima del array ya que este nos brinda bastantes beneficios al momento de manipular información y en este caso al ordenarlo, una de las ventajas más remarcadas es el cual el array es estático, esto quiere decir que una vez inicializada en ese mismo momento el array ya tiene que estar definido (número de caracteres) y sus datos tienen que ser del mismo tipo (int, char, string) a diferencia del vector que esta es dinámica podemos ajustar su tamaño cuando queramos, sus listas aceptan cualquier tipo de variable siendo esta ultima la de mayor vitalidad debido a que como acepta cualquier dato hicimos un vector que contenga vectores strings logrando el concepto de tokenización logrando acceder a cierta información puntual para poder hacer las comparaciones para poder ordenar, otro punto importante que añade el vector son los métodos que este ya contiene (.size(), .begin(), .end(), .insert(), .delete()) haciendo que el código quede más limpio y claro al momento de visualizarlo.

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

Explicación:

Para la aplicación de nuestra búsqueda a ips, primero tuvimos que ver como estos se ordenaban. Se vio para que su ordenamiento se necesita comparar desde el primer digito hasta el ultimo digito (104.244.253:29).

El primer parte fue acceder a la información de las ips al vector, vimos que este es vector[i][3]:

Aug 28 23:07:49 897.53.984.6:6710 Failed password for root

En la función “ipsConvertidos()” recorremos todos los vectores y accedimos a la información de las ips, con ayuda de dos vectores separamos las ips por numero sin tener modificarlos en esa columna ya que en caso de modificarlos sería muy difícil volver a añadirle los puntos y el dos puntos. Con el primer vector fue crear una copia de la columna de las ips por cada iteración y separarlos por puntos. Ejemplo: [897, 53, 984, 6:6710] y con el segundo vector le siempre le fuimos añadiendo el elemento tres del primer vector pero con el mismo método de separación mediante la búsqueda de un char especial, al final obteniendo solamente dos elementos en el vector dos pero con los números separados [6, 670], ya teniendo todos los números guardados en variables el siguiente paso fue añadir todos esos números con los vectores que queremos ordenar mediante el método .begin(), luego se implemente el BubbleSort para el ordenamiento y al final borramos las primeras 5 posiciones de todos los vectores que era ahí donde todos los números de las ips separadas estaban.

ipsConvertidos():

```
//Hacemos una concatenación de strings de las horas, minutos y segundos, lo guradamos en el vector en donde estaban las horas.
void functions::ipsConvertidos(vector<vector<string>> &lista)
{
  vector<string> vectorTemp, vectorTemp2;
  string suma,suma1,suma2,suma3,suma4;
  for (int i = 0; i < lista.size(); i++)
  {
    vectorTemp = dividir(lista[i][3], '.');
    vectorTemp2 = dividir(vectorTemp[3],':');
    suma = (vectorTemp[0]);
    suma1 = (vectorTemp[1]);
    suma2 = (vectorTemp[2]);
    suma3 = (vectorTemp2[0]);
    suma4 = (vectorTemp2[1]);
    lista[i].insert(lista[i].begin(),suma4);
    lista[i].insert(lista[i].begin(),suma3);
    lista[i].insert(lista[i].begin(),suma2);
    lista[i].insert(lista[i].begin(),suma1);
    lista[i].insert(lista[i].begin(),suma);
  }
}
```

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)

BubbleSort():

```
//Función Bubblesort para ordenar los datos de los vectores de menor a mayor.
void functions::ordenBurbuja(vector<vector<string>> &a)
{
  int n = a.size();
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < n - i - 1; j++)
    {
      if (stoll(a[j][0]+a[j][1]+a[j][2]+a[j][3]+a[j][4]) > stoll(a[j + 1][0]+a[j + 1][1]+a[j + 1][2]+a[j + 1][3]+a[j + 1][4]))
      {
        vector<string> tmp_value = a[j];
        a[j] = a[j + 1];
        a[j + 1] = tmp_value;
      }
    }
  }
}
```

ipsDesconvertidos():

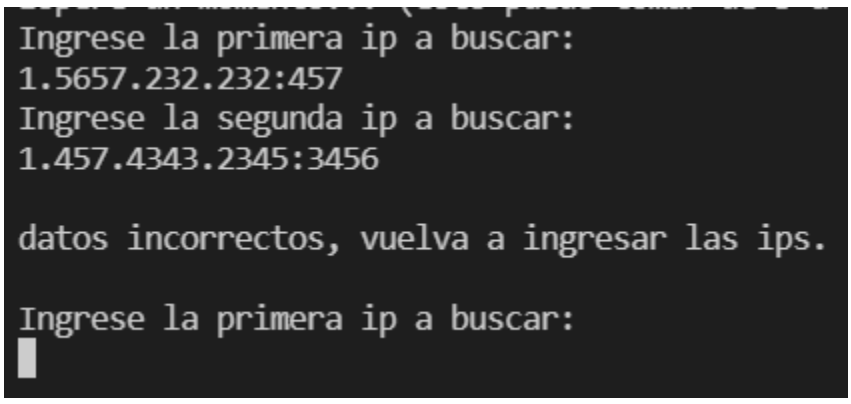
```
void functions::ipsDesconvertidos(vector<vector<string>> &a){
  for(int i = 0; i < a.size(); i++){
    a[i].erase(a[i].begin());
    a[i].erase(a[i].begin());
    a[i].erase(a[i].begin());
    a[i].erase(a[i].begin());
    a[i].erase(a[i].begin());
  }
}
```

Casos prueba:

	CASOS DE PRUEBA ACT 1.3				
No.	Precondiciones	Resumen	Pasos	Resultado Esperado	Resultado obtenido
1	Tener el código y la bitácora en la misma dirección.	El usuario introduce ips invalidas.	1. Iniciar el programa. 2. Introducir ips que no se encuentran en la bitácora.	Una vez insertada las ips el programa deberá preguntarle de nuevo por	El programa entra en el bucle preguntándole las ips hasta que sean correctas.

				las ips hasta que sean válidas.	
2	Tener el código y la bitácora en la misma dirección.	El usuario solamente introduce una ip válida y la otra no.	1. Correr el programa. 2. Introducir las ips, pero una fecha que aparezca en la bitácora y otra que no esté dentro de la bitácora.	Cuando termine de introducir las ips el programa entra en el bucle while hasta que introduzca ambas ips válidas.	El programa entra en el bucle preguntándole las ips hasta que sean correctas.
3	Tener el código y la bitácora en la misma dirección.	El usuario introduce una ip inicial mayor a la final de búsqueda	1. Correr el programa 2. Introducir la ip de inicio búsqueda 3. Introducir la ip final de búsqueda 4. Rectificar que la ip de inicio sea mayor a la final.	Entrar en el bucle marcando por fechas invalidas y volviendo a preguntar.	El programa no vuelve a preguntar y termina acabando al final sin imprimir nada
4	Tener el código y la bitácora en la misma dirección	El usuario introduce ambas ips de forma correcta.	1. Correr el programa 2. Introducir las ips (la de inicio y final) 3. Verificar que ambas ips están insertadas correctamente.	El código imprime los datos entre las ips seleccionadas y crea un archivo .txt donde se encuentran los datos	Se imprime en consola los datos y se crea el archivo donde se muestran los datos seleccionados.

Caso prueba 1:



Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)
Caso prueba 2:

```
Ingrese la primera ip a buscar:
118.15.416.57:4486
Ingrese la segunda ip a buscar:
1.457.4343.2345:3456

datos incorrectos, vuelva a ingresar las ips.

Ingrese la primera ip a buscar:
```

Caso prueba 3:

```
Ingrese la primera ip a buscar:
70.24.277.19:5800
Ingrese la segunda ip a buscar:
70.17.769.73:4749

PS C:\TC1031\Act2.3>
```

Caso prueba 4:

```
Ingrese la primera ip a buscar:
70.16.722.45:5886
Ingrese la segunda ip a buscar:
84.88.502.53:6452
```

```
Jun 05 22:54:18 701.8.251.81:5291 Failed password for admin
Jul 22 12:47:44 702.1.133.77:4963 Illegal user
Oct 12 07:49:08 702.13.320.1:6036 Failed password for admin
Jul 22 01:15:44 702.23.498.5:5350 Illegal user
Jun 08 22:37:29 702.26.74.28:6558 Failed password for illegal user test

Jun 02 15:49:57 702.26.95.51:6673 Failed password for admin
Jun 02 20:57:01 702.42.44.29:6115 Failed password for illegal user test

Jul 13 17:31:24 70.24.277.19:5800 Failed password for illegal user test

Oct 08 09:54:32 703.26.552.6:4413 Failed password for illegal user gues
t
Sep 20 17:27:23 703.38.825.5:4132 Failed password for root
Sep 11 23:46:16 704.2.445.62:6257 Failed password for illegal user test

Jul 21 22:23:00 70.45.707.43:6103 Failed password for illegal user test

Aug 26 17:47:48 70.45.969.36:6195 Failed password for illegal user gues
t
Jul 22 09:04:34 70.47.252.28:5273 Failed password for admin
Sep 21 01:06:43 704.7.717.75:4832 Failed password for admin
Aug 15 08:03:28 705.18.26.98:4451 Failed password for illegal user test

Sep 04 10:58:48 705.38.738.3:5760 Failed password for admin
Jun 19 06:24:26 705.47.779.4:6753 Failed password for admin
Aug 20 12:17:29 70.59.316.14:4012 Failed password for admin
Jun 26 10:16:56 706.24.940.7:6874 Illegal user
Jul 03 07:23:14 706.5.221.91:4451 Failed password for admin
Sep 16 14:21:30 706.68.250.8:6902 Failed password for illegal user test

Sep 01 09:04:00 706.7.935.28:5321 Failed password for illegal user gues
t
Jul 22 14:28:14 706.8.882.21:6525 Failed password for illegal user test

Jun 15 05:34:35 70.73.861.73:6461 Failed password for admin
Aug 05 11:37:01 707.43.198.4:4655 Illegal user
Jul 23 11:25:35 707.56.20.38:4993 Failed password for illegal user gues
t
Sep 14 19:19:53 707.62.68.73:5384 Failed password for root
Sep 25 14:16:22 70.77.679.44:5100 Failed password for illegal user gues
t
```

Consola

Tecnológico de Monterrey
TC1031.11: Programación de estructuras de datos y algoritmos fundamentales (GPO 11)
Edgar Cruz Vázquez (A01730577)
Archivo.txt

```
Jul 03 20:28:36 70.17.769.73:4749 Failed password for illegal user test
Jun 05 22:54:18 701.8.251.81:5291 Failed password for admin
Jul 22 12:47:44 702.1.133.77:4963 Illegal user
Oct 12 07:49:08 702.13.320.1:6036 Failed password for admin
Jul 22 01:15:44 702.23.498.5:5350 Illegal user
Jun 08 22:37:29 702.26.74.28:6558 Failed password for illegal user test
Jun 02 15:49:57 702.26.95.51:6673 Failed password for admin
Jun 02 20:57:01 702.42.44.29:6115 Failed password for illegal user test
Jul 13 17:31:24 70.24.277.19:5800 Failed password for illegal user test
Oct 08 09:54:32 703.26.552.6:4413 Failed password for illegal user guest
Sep 20 17:27:23 703.38.825.5:4132 Failed password for root
Sep 11 23:46:16 704.2.445.62:6257 Failed password for illegal user test
Jul 21 22:23:00 70.45.707.43:6103 Failed password for illegal user test
Aug 26 17:47:48 70.45.969.36:6195 Failed password for illegal user guest
Jul 22 09:04:34 70.47.252.28:5273 Failed password for admin
Sep 21 01:06:43 704.7.717.75:4832 Failed password for admin
Aug 15 08:03:28 705.18.26.98:4451 Failed password for illegal user test
Sep 04 10:58:48 705.38.738.3:5760 Failed password for admin
Jun 19 06:24:26 705.47.779.4:6753 Failed password for admin
Aug 20 12:17:29 70.59.316.14:4012 Failed password for admin
Jun 26 10:16:56 706.24.940.7:6874 Illegal user
Jul 03 07:23:14 706.5.221.91:4451 Failed password for admin
Sep 16 14:21:30 706.68.250.8:6902 Failed password for illegal user test
Sep 01 09:04:00 706.7.935.28:5321 Failed password for illegal user guest
Jul 22 14:28:14 706.8.882.21:6525 Failed password for illegal user test
Jun 15 05:34:35 70.73.861.73:6461 Failed password for admin
Aug 05 11:37:01 707.43.198.4:4655 Illegal user
Jul 23 11:25:35 707.56.20.38:4993 Failed password for illegal user guest
Sep 14 19:19:53 707.62.68.73:5384 Failed password for root
Sep 25 14:16:22 70.77.679.44:5100 Failed password for illegal user guest
Jul 12 08:38:42 70.79.617.27:6203 Failed password for admin
Aug 12 06:18:09 707.96.58.14:4281 Failed password for illegal user guest
Aug 28 02:00:26 70.80.577.14:6593 Failed password for root
Oct 21 04:02:24 70.81.570.21:4872 Failed password for illegal user test
Jun 27 19:27:18 70.82.120.55:6564 Failed password for root
Jul 14 05:54:33 708.42.692.9:5686 Illegal user
Oct 17 03:14:26 708.63.26.52:5189 Failed password for illegal user test
Jul 22 01:52:45 70.86.443.10:4316 Failed password for illegal user test
Oct 14 12:01:38 708.71.98.13:5477 Failed password for root
Aug 11 17:00:29 708.82.80.47:5152 Failed password for admin
Aug 20 14:18:14 708.85.23.76:6568 Failed password for root
Oct 04 00:02:18 84.24.817.53:4233 Illegal user
Jun 23 08:29:56 842.67.526.7:6870 Failed password for illegal user guest
Aug 07 15:47:01 843.45.11.83:5065 Failed password for root
Jul 21 14:02:07 843.47.103.6:5332 Failed password for illegal user guest
Jun 27 12:35:18 84.34.853.44:5397 Failed password for illegal user guest
Sep 19 00:21:31 843.70.502.6:6095 Failed password for illegal user test
Jun 05 18:50:17 84.39.186.18:4673 Failed password for illegal user test
Sep 19 10:45:20 84.40.371.67:4721 Failed password for illegal user guest
Oct 20 17:18:39 84.41.519.33:4963 Illegal user
Oct 07 12:26:36 844.4.903.92:5886 Failed password for root
Aug 13 20:46:31 84.49.150.92:4023 Illegal user
Jun 26 10:13:33 84.49.930.30:4821 Failed password for illegal user guest
Jul 12 05:39:43 84.51.413.84:5584 Illegal user
Sep 14 21:36:00 845.19.91.58:5918 Failed password for illegal user test
Oct 02 01:40:09 845.21.47.81:5265 Failed password for illegal user guest
Jun 30 08:30:04 845.5.305.37:6365 Failed password for illegal user guest
Sep 09 13:18:17 845.5.436.91:6981 Failed password for illegal user test
Aug 09 05:02:55 845.63.305.1:5517 Failed password for illegal user guest
Jul 15 05:14:07 845.97.79.36:6970 Failed password for illegal user guest
Oct 02 12:44:41 84.62.181.80:5715 Failed password for admin
Jun 23 22:14:41 846.6.864.92:5458 Failed password for admin
Jun 14 07:32:22 846.83.57.53:6372 Failed password for admin
Jun 27 14:40:24 846.9.571.13:6435 Failed password for illegal user test
Oct 16 21:09:35 847.29.650.7:4893 Illegal user
Oct 17 10:33:57 84.73.698.80:5127 Failed password for illegal user test
Jul 28 17:42:56 847.77.651.9:5761 Failed password for illegal user guest
Oct 15 23:44:00 847.85.966.8:4364 Failed password for illegal user test
Jul 11 00:27:19 84.81.628.59:6792 Failed password for root
Sep 02 18:24:48 848.16.35.38:5031 Failed password for root
Sep 16 01:32:17 848.33.803.9:4737 Failed password for illegal user guest
Jul 27 19:35:12 848.48.83.74:4406 Failed password for root
Aug 22 19:54:22 84.85.621.97:4696 Failed password for admin
Sep 19 02:38:18 84.87.568.62:6368 Failed password for illegal user test
```

Reflexión:

En esta evidencia logre desarrollar las competencias requeridas para esta evidencia y realización este trabajo, durante el procedimiento se tuvo que aplicar habilidades y conocimientos aprendidos durante la clase ya que si bien fue muy parecido a la primera evidencia no era la misma, se tuvieron que hacer ajustes de tal forma para que se cambiara el tipo de búsqueda requerida usando algoritmos de ordenamiento y búsqueda, usando en la parte de ordenamiento el BubbleSort que visto anteriormente su complejidad de tiempo es O(n^2) la cual la hace ineficiente pero con una complejidad espacial de O(1) el mejor de todos, para la búsqueda utilizamos la secuencial en la cual su peor caso es O(n), en el código para obtener los resultados requeridos por hacer nuevas operaciones y aun cambiando varias funciones con parámetros de referencia para evitar copias de información que ocuparan memoria en nuestro equipo, esta tardo en un aproximado de 5-8 min lo cual es demasiado si se quisiera aplicar para problemas de la vida real volviéndose esta inútil para su resolución, en la parte de búsqueda usamos la búsqueda secuencial implementándose debido a su simplicidad teniendo este un promedio O(log(n/2) mejor caso O(1) y peor caso O(log(n)), este tipo de búsqueda es de las más simples y fáciles de implementar debido a que solamente recorre todos los datos hasta llegar al dato requerido el problema radica si son demasiados líneas de información y lo que se quiere encontrar está en los últimos lugares convirtiéndose en su peor caso el cual es O(n) siendo n el mismo número de iteraciones que de líneas de información, eso lo convierte en un método de búsqueda ineficiente para ese caso se requeriría la búsqueda binaria haciendo la búsqueda con mucho menos pasos, también logre ver la versatilidad que pueden llegar a tener los vectores al momento de la manipulación de información y su implemento para en este caso, su ordenación.