Agustín Pumarejo Ontañón, A01028997
Adriana Abella Kuri, A01329591

Reporte de primer avance del reto

Diagrama de ADT (Clase: Record)

Encabezado: Record
Descripción de los datos:

- Fecha: fecha de la conexión
- Hora: hora de la conexión
- IP fuente: dirección IP del usuario
- Puerto fuente: puerto del usuario
- Nombre fuente: nombre del usuario
- IP destino: dirección IP del servicio
- Puerto destino: puerto del servicio
- Nombre destino: nombre del servicio

Lista de operaciones:

- printRecord()
    - Entrada: ninguna
    - Precondiciones: se utiliza dentro de la función countRecords() de la clase RecordManager
    - Proceso: accede a las variables del objeto
    - Salida: no devuelve ningún valor pero imprime un string con todas las variables del objeto
    - Poscondiciones: ninguna

Diagrama de ADT (Clase: RecordManager)

Encabezado: RecordManager
Descripción de los datos:

- Entries: un vector de objetos de tipo Record

- Order: el criterio bajo el que se encuentra ordenado el vector entries, es un string con el nombre de una variable de la clase Record

Lista de operaciones:

- printRecord()
    - Entrada: string file
    - Precondiciones: se utiliza dentro de la función countRecords() de la clase RecordManager
    - Proceso: accede a las variables del objeto
    - Salida: no devuelve ningún valor pero imprime un string con todas las variables del objeto
    - Poscondiciones: ninguna
- read()
    - Entrada: el archivo que se pretende leer
    - Precondiciones: se llama en el main() pasando como argumento el documento csv
    - Proceso: separa el archivo por sus líneas, siendo cada línea un Record, y después esas líneas por sus comas, siendo cada parte entre las comas un atributo de Record. Los almacena en el vector entries.
    - Salida: no devuelve ningún valor
    - Poscondiciones: ninguna
- sort()
    - Entrada: vector a ordenar y criterio de ordenamiento
    - Precondiciones: se llama antes de búsquedas bajo cierto criterio para asegurarse de que funcionen
    - Proceso: cambia la variable order al nuevo criterio de ordenamiento
    - Salida: no devuelve ningún valor
    - Poscondiciones: llama a la función sortAux()
- sortAux()
    - Entrada: vector, límite inferior, límite superior, variable de ordenamiento
    - Precondiciones: se llama dentro de la función sort() o dentro de sortAux()
    - Proceso: llama a la función partition() y a sí misma
    - Salida: no regresa ningún valor

    - Poscondiciones: después de un determinado número de veces el vector debe quedar ordenado
- partition()
    - Entrada: un vector, límite inferior, límite superior, variable de ordenamiento
    - Precondiciones: se llama dentro de la función sortAux()
    - Proceso: usa un pivote y dos índices para intercambiar valores dentro del vector y ordenarlo
    - Salida: no regresa ningún valor
    - Poscondiciones: el vector queda ordenado
- intercambiar()
    - Entrada: un vector y dos índices de posiciones
    - Precondiciones: se llama dentro de la función partition() para intercambiar valores en un vector
    - Proceso: mueve el valor en el primer índice al segundo índice y viceversa
    - Salida: no devuelve ningún valor
    - Poscondiciones: ninguna
- binarySearch()
    - Entrada: recibe el valor que se quiere buscar dentro del vector
    - Precondiciones: el valor debe coincidir con el criterio de ordenamiento (variable order) para funcionar
    - Proceso: compara el valor que se quiere encontrar con el valor medio del vector, y si el valor a buscar es mayor se busca en la primera mitad del vector, si el valor a buscar es menor se busca en la segunda mitad, y así sucesivamente hasta que se encuentra el valor o se concluye que el vector no lo contiene
    - Salida: si el valor no fue encontrado regresa un -1, si fue encontrado regresa su posición en el vector
    - Poscondiciones: ninguna
- comparator()
    - Entrada: dos récords y un criterio de comparación
    - Precondiciones: ninguna
    - Proceso: determina el atributo que se quiere comparar y llama la función comparatorAux() con los argumentos adecuados

    - Salida: devuelve el valor de la función comparatorAux()
    - Poscondiciones: ninguna
- compareStrVsR()
    - Entrada: un string, un récord y un criterio de comparación
    - Precondiciones: ninguna
    - Proceso: llama a la función comparatorAux con los argumentos adecuados para la comparación
    - Salida: devuelve el valor de la función comparatorAux()
    - Poscondiciones: ninguna
- comparatorAux()
    - Entrada: dos strings a comparar
    - Precondiciones: es llamada en la función comparator() o en la función compareStrVsR()
    - Proceso: evalúa si los strings son iguales o si uno es mayor que el otro
    - Salida: de ser iguales devuelve cero, si el primero es mayor devuelve -1, y si el segundo es mayor devuelve 1
    - Poscondiciones: ninguna
- countRecords()
    - Entrada: atributo que se quiere contar dentro del vector, tipo de atributo, y valor booleano
    - Precondiciones: se llama en el main() para saber cuántos récords hay con un determinado atributo
    - Proceso: busca el atributo que se quiere contar, y después busca de qué posición a qué posición del vector se repite ese mismo valor para saber el número de récords que comparten ese atributo
    - Salida: no devuelve ningún valor
    - Poscondiciones: imprime el número de récords encontrados
- printDays()
    - Entrada: ninguna
    - Precondiciones: ninguna
    - Proceso: imprime una sola vez cada fecha presente en los datos
    - Salida: ninguna
    - Poscondiciones: ninguna

- printDestPort()
    - Entrada: ninguna
    - Precondiciones: ninguna
    - Proceso: imprime una sola vez cada puerto destino presente en los datos
    - Salida: ninguna
    - Poscondiciones: ninguna

Algoritmos de ordenamiento y búsqueda utilizados

Como algoritmo de ordenamiento se utilizó Quick Sort. Originalmente planeamos utilizar Merge Sort pero tuvimos segmentation faults y no logramos encontrar el error en el código así que escogimos la segunda mejor alternativa en términos de velocidad. Este algoritmo tiene una complejidad temporal de $O(n^2)$ en el peor de los casos y de $O(n * log(n))$ en el mejor de los casos. Su complejidad espacial es de $O(1)$ ya que no genera arreglos adicionales, haciéndolo mejor que merge sort en este aspecto.

Como algoritmo de búsqueda utilizamos Binary Search. Este es el mejor algoritmo de búsqueda tanto en velocidad como en complejidad espacial y por eso lo utilizamos. Binary Search, en el peor de los casos tiene una complejidad temporal de $O(log(n))$ y en el mejor de los casos $O(1)$. Su complejidad espacial es de $O(1)$.

Aportaciones

Agustín Pumarejo: Escribió el código en su computadora, sentó las bases para las clases, hizo los métodos de comparación, los métodos para imprimir los días y los ports de destino e hizo el código en el main de la pregunta 2 a la 7.

Adriana Abella: Supervisó la redacción de todo el código, escribió todos los métodos para sortear y comparar récords, escribió el método para comparar strings y los diagramas de clases.

Preguntas:

1. ¿Cuántos registros tiene tu archivo?

36949

2. ¿Cuántos récords hay del segundo día registrado? ¿Qué día es este?

Hay 3317 récords del día 11/8/2020

3. ¿Alguna de las computadoras pertenece a Jeffrey, Betty, Katherine, Scott, Benjamin, Samuel o Raymond?

Una de las computadoras le pertenece a Benjamín y ha hecho 1140 búsquedas

4. ¿Cuál es la dirección de la red interna de la compañía?

172.23.97, el número siguiente a esos 3 indica la computadora individual

5. ¿Alguna computadora se llama server.reto.com?

No

6. ¿Qué servicio de mail utilizan de todos estos: gmail.com, outlook.com, protonmail.com, freemailserver.com?

freemailserver.com y se utilizó 13429 veces

7. Considerando solamente los puertos destino ¿Qué puertos abajo del 1000 se están usando?

Todos los puertos de destino que se utilizaron son abajo del 1000 y son 135, 443, 465, 53, 67, 80, 965 y 993,

Lista los puertos e investiga qué aplicación/servicio lo utiliza generalmente.

135: El endpoint mapper de una aplicación. Sirve para redireccionar tráfico a otros puertos asignados por el RPC (Remote Procedure Call).

443: TCP (Transmission Control Protocol) se utiliza en el protocolo HTTPS de páginas web para transferir hipertexto de forma segura.

465: Este es un puerto de tipo SMTP que sirve como entrada a un servidor, se usa sobretodo con servicios de correo electrónico

53: Puede seguir el protocolo TCP o UDP y se usa principalmente en servicios DNS que sirve para traducir los nombres de los dominios.

67: Servicios del Protocolo Bootstrap o de inicio (BOOTP). Sirve para asociar estaciones de trabajo a servidores y asignar direcciones IP.

80: Protocolo de transferencia de HTTP para los servicios de WWW, para la navegación web insegura.

965: Al igual que el puerto 53 es de uso público y puede seguir el protocolo TCP además del UDP, que también es un protocolo de transferencia de información pero es menos seguro.

993: Se utiliza para utilizar IMAP (Internet Message Access Protocol) de manera cifrada. Se utiliza comúnmente para la recuperación de correos.

```
Pregunta 1:
El número de registros es: 36949
Pregunta 2:
10-8-2020
11-8-2020
12-8-2020
13-8-2020
14-8-2020
17-8-2020
18-8-2020
19-8-2020
20-8-2020
21-8-2020
Número de récords con: 11-8-2020 = 3317
Pregunta 3:
Número de récords con: jeffrey.reto.com = 0
Número de récords con: betty.reto.com = 0
Número de récords con: katherine.reto.com = 0
Número de récords con: scott.reto.com = 0
Número de récords con: benjamin.reto.com = 1140
Número de récords con: samuel.reto.com = 0
Número de récords con: raymond.reto.com = 0
Pregunta 5:
Número de récords con: server.reto.com = 0
Pregunta 6:
Número de récords con: gmail.com = 0
Número de récords con: outlook.com = 0
Número de récords con: protonmail.com = 0
Número de récords con: freemailserver.com = 13429
Pregunta 7:
-
135
443
465
53
67
80
965
993
```