Tecnológico de Monterrey

Programación de Estructuras de Datos y Algoritmos Fundamentales

Act 2.2 - Actividad Integral Estructura de Datos Lineales

Equipo 3

Paolo Antonio Pires Cano A01749355

Gabriel Edid Harari A01782146

Dr. Vicente Cubells

09 de Octubre del 2023

Para obtener las respuestas a las siguientes preguntas se utilizaron fragmentos del código y respuestas obtenidas de la Actividad 1.3. Todos los datos con los que se trabajó fueron cargados en dos ADT diferentes: en un vector y en un stack. Sus datos se llenaron a través de la búsqueda de la dirección IP Origen y Destino en el vector de Registros.

Nota: Ambos miembros del equipo participamos de forma igual y se trabajaron todos los puntos de manera coordinada y colaborativa.

Caso de Prueba: Se utilizará como caso de prueba para la red interna el valor de **72**.

1. **¿Qué dirección ip estás usando?**

   Para esto lo primero es pedirle al usuario los últimos valores de la IP interna que están el rango de 1 a 150, ese valor se guarda en una variable llamada ipInterna (en el main). Si es que el valor no está dentro del rango se vuelve a repetir la pregunta. Una vez hecho eso creamos un objeto llamado Conexiones de tipo ConexionesComputadora en donde metemos la `direccionIP` (de la actividad 1.3) y lo concatenamos con la `ipInterna` (convertida a string) y de nombre le ponemos `Conexion1`. Después se llenan las Conexiones con el vector registros.

   Ahora sí ya contestando la pregunta, simplemente se hace un cout de la `direccionIP` concatenada con la ipInterna. El resultado que da es que la IP que se utilizara es 172.21.8.72.

2. **¿Cuál fue la dirección IP de la última conexión que recibió esta computadora? ¿Es interna o externa?**

   Sabemos que estamos hablando de las conexiones entrantes ya que se pide la última dirección IP que recibió la computadora. Como `conexionesEntrantes` es una pila, agarramos el valor de hasta arriba con la función `top()` y se obtiene su

información con `getInfo()`. Se hace un cout de la información y después se verifica si los primeros tres octetos de `IpOrigen` e `IpDestino` coinciden o no. Si coinciden se imprime que la conexión es interna y si no coinciden se imprime que la conexión es externa. Finalmente el resultado que se da es que la ultima conexión es: 13-8-2020 - 13:13:3 - 172.21.8.34 - - - angela.reto.com - 172.21.8.72 - - - - y esta es Interna.

3. **¿Cuántas conexiones entrantes tiene esta computadora?**

   La forma en la que se llena un objeto de tipo `ConexionesComputadora` es con un for loop yendo por todos los elementos de nuestro vector registros (los objetos son de tipo `Registro`) y checando que primero que su atributo `IpOrigen` sea igual al de la computadora que estamos utilizando, si se cumple esto se agrega al vector de conexionesSalientes. Si es que no se cumple entonces checa que el atributo `IpDestino` sea igual al IP de la computadora que estamos utilizando y si se cumple lo agrega nuestra Pila de `conexionesEntrantes`.

   Para checar la cantidad de conexiones entrantes simplemente utilizamos la función `size()` la cual nos regresa el tamaño y de esto se hace finalmente un cout. El Total de Conexiones Entrantes es de 2.

4. **¿Cuántas conexiones salientes tiene esta computadora?**

   Como mencionamos anteriormente una vez que ya están llenas las conexiones, para checar el total de conexiones salientes simplemente utilizamos `size()` para que nos regrese el tamaño de nuestro vector. Creamos una función para poder llamarlo en el archivo de Actividad2.2 y listo. El resultado cambia con respecto a cada ejecución y red interna dada. El Total de Conexiones Salientes es de 149.

5. **¿Tiene esta computadora 3 conexiones seguidas a un mismo sitio web?**

   Lo primero que hicimos fue definir que el IP de la computadora en estos casos iba a estar en el `IpOrigen`, porque la conexión fue saliente de nuestra computadora y entrando a una página web. Después de esto se creó un for loop que va por todos los elementos del vector registros excepto por los últimos dos, esto es debido a que se compara cada registro con los siguientes dos, y si vamos por todos los elementos, los últimos dos no tienen dos siguientes con los que se puedan comparar. Después hicimos un if comparando si esos tres registros tienen la misma dirección `IpOrigen`, si es que se cumple ahora se comparan los `PuertoDestino` (los cuales serían los puertos de página web 443 y 80), primero checa 80 y si no se cumple se va a 443 . Si cumple con todo eso (solamente uno de los puertos debe de estar cumplido) ya se tendrían tres conexiones seguidas a un sitio. Si es que el "i" del loop llega a `registros.size()-2`, entonces indicamos que no hay tres conexiones seguidas usando un cout. Con el caso de prueba nos da que si hay tres visitas seguidas a sitios Web por el puerto 443 con el IP: 172.21.8.72. Cabe recalcar que utilizando un loop se verificó que este caso de prueba es el único en el que hay tres conexiones a alguno de los dos puertos en cuestión.

Investigación y Reflexión Individual Gabriel Edid:

Para este tipo de situaciones problemas es importante utilizar estructuras de datos lineales ya que estos pueden ser modificados y elegidos de acuerdo a las necesidades del problema. En este caso se eligió un stack para las conexiones Entrantes y un vector para las conexiones Salientes. Se eligió el Stack ya que existía la necesidad de obtener el último elemento que entró al ADT y con un stack esa operación tiene una complejidad constante ya que siempre

hay un apuntador al último valor que entró. Por otro lado, se eligió el vector ya que existía la necesidad de que estén ordenadas conforme fueron agregadas al ADT. Ambas estructuras de datos tienen sus funciones como size(), at(), getInfo(), top(), push_back(), etc, que facilitan las operaciones el trabajo con estas. Es importante mencionar que estas dos estructuras de datos tienen una complejidad temporal y espacial de O(n), es decir que siempre son lineales en el peor caso. Este dato es muy relevante ya que en el caso de un ciberataque de una botnet es importante que el tiempo que tarda en ejecutarse el sistema de seguridad sea en el menor tiempo posible y pueda eliminar la amenaza de forma sencilla. Sin este tipo de ADT el tiempo y eficiencia de respuesta se puede ver severamente afectado lo que podría causar daños difíciles de reparar o irreparables. También es importante conocer estas estructuras de datos como práctica general para un programador ya que siempre se debe de priorizar la creación de programas más eficientes y mejor organizados.

Investigación y Reflexión Individual Paolo Pires:

Cuando se requiere almacenar datos en una secuencia ordenada, utilizar estructuras de datos lineales es una opción perfecta. Dentro de la categoría de estructuras de datos lineales tenemos listas, arreglos, etc. Eso es como definición general, pero si nos enfocamos más a la programación, encontramos un grupo aún más grande de este tipo de estructuras de datos. En C++ tenemos lo que son los Stacks (Pilas), Queues (Colas), Linked Lists (Listas Enlazadas) y Vectores. Al utilizarlos tenemos varios tipos de ventajas y también algunas desventajas, pero todo depende de lo que queramos hacer y cómo lo queremos hacer.

Como había mencionado anteriormente el arreglo es un tipo de estructura de datos lineales, pero tiene una desventaja importante ya que la memoria en un arreglo se asigna de manera estática, lo cual quiere decir que no puede expandirse más del tamaño previamente definido para ese arreglo. ¿Entonces qué podemos utilizar para no tener este problema? Las listas enlazadas son una gran solución llena de ventajas. El primer elemento de una lista enlazada

apunta al segundo, el segundo al tercero, el tercero al cuarto y podemos seguir y seguir, así que no tenemos que definir un tamaño para este tipo de listas. También existen varias variaciones que nos podrían ayudar para diferentes casos. En clase hemos visto la lista simplemente enlazada, la lista circular simplemente enlazada, lista doblemente enlazada, etc.

De lo visto en clase sabemos que de las listas enlazadas nacen más tipos de estructuras de datos lineales, en el caso del reto nos funcionaron bastante bien los Stacks. Un "Stack" o en español una "Pila", es una variante de las listas en el cual las operaciones de inserción y eliminación ocurren en el mismo extremo. Cuando implementamos una Pila con las listas enlazadas, lo estamos haciendo dinámicamente. Dentro de la pila tenemos los nodos, un nodo representa a un elemento dentro de la pila, tenemos el elemento de más arriba que es el tope. Podemos meter (push) elementos a la pila creando un nuevo nodo y actualizando el apuntador del tope a que apunte al nuevo nodo, también podemos eliminar (pop) elementos de la pila removiendo el nodo del tope y actualizando el apuntador del tope a que apunte al siguiente elemento. También como mencioné anteriormente tenemos otro tipos de estructuras como los vectores o las colas, pero en resumen los utilizamos para poder trabajar con una base de datos que tenga una gran cantidad de elementos en donde se tengan que agregar y eliminar bastantes datos y valores, como encontrar cierta cantidad de conexiones entrantes o salientes, etc. En este caso estamos trabajando con más de 3000 datos diferentes entonces sería casi imposible hacer todo lo que se pide a mano, modificando diferentes listas, etc. También es importante escoger la estructura basada en tus necesidades, pero es muy interesante como algo tan básico como una lista, puede evolucionar a cosas mucho más complejas, cada vez entiendo más el por qué es tan importante la programación.

Referencias:

- Data Flair. (s.f.) *C++ Data Structures – Secret Behind A Successful Programmer.* Data Flair. https://data-flair.training/blogs/data-structures-in-cpp/
- tusclases. (s.f.) *La diferencia entre estructura de datos dinámicas y lineales.* Tusclases.https://www.tusclases.mx/blog/diferencia-entre-estructura-datos-dinamicas-lineales#:~:text=y%20usos%20espec%C3%ADficos.-,Las%20estructuras%20lineales%20son%20eficientes%20para%20almacenar%20y%20acceder%20a,de%20agregar%20o%20eliminar%20elementos.
- Notas de clase y presentaciones del profesor Vicente Cubells.