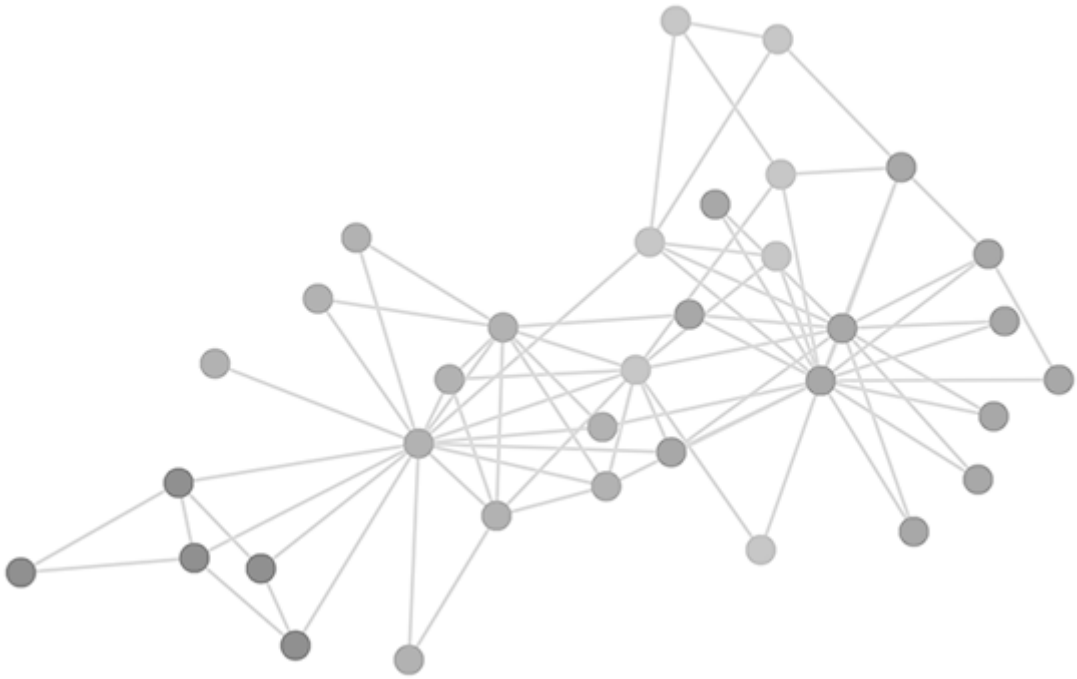
Reflexión Actividad 4.3 - Actividad Integral de Grafos (Evidencia Competencia)

Programación de estructuras de datos y algoritmos fundamentales

(Grupo 12)

Alumnos:
Marisol Rodríguez Mejía A01640086
Luis Enrique Lemus Martínez A01639698

Profesor:
Dr. Eduardo Arturo Rodríguez Tello



Marisol Rodríguez Mejía A01640086
Luis Enrique Lemus Martínez A01639698

Reflexión

Para esta actividad, tuvimos que usar algo nuevo a comparación de las anteriores, que son los grafos. En matemáticas y ciencias de la computación, un grafo es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permiten representar relaciones entre elementos de un conjunto. Típicamente, un grafo se representa gráficamente como un conjunto de puntos (vértices o nodos) unidos por líneas (aristas o arcos).
El tema de la teoría de grafos tuvo sus inicios en problemas matemáticos recreativos, pero se ha convertido en un área importante de investigación matemática, con aplicaciones en química, investigación de operaciones, ciencias sociales e informática.
Cuando dos vértices cualesquiera están unidos por más de una arista, la gráfica se llama multigraph. Un grafo sin bucles y con como máximo una arista entre dos vértices cualesquiera se denomina grafo simple. Cuando cada vértice está conectado por una arista a todos los demás vértices, se llama grafo completo. También se puede asignar una dirección a cada nodo para producir lo que se conoce como grafo dirigido.
Un número importante asociado con cada vértice es su grado, que se define como el número de nodos que entran o salen de él (Carlson, s.f.).
Tuvimos que implementar nuevas clases a la actividad para el uso de grafos y una adaptación de las pasadas para que pudiera funcionar de manera exitosa.
Fue un proceso complicado adaptarlo a grafos y reestructurar el código, porque en sí la forma de la bitácora cambió, pero una vez con esto listo, implementar lo demás fue relativamente fácil.

Para pasar de tener un archivo de texto a un grafo con todas las IPs y sus grados, leíamos el archivo y lo almacenamos en una lista de adyacencia. Una lista de adyacencia representa un grafo como un arreglo de listas enlazadas. El índice del arreglo representa un vértice y cada elemento en su lista vinculada representa los otros vértices que forman un borde con el vértice, por lo tanto, el número de elementos de la lista de adyacencia es igual al número de nodos del grafo. Dicha lista de adyacencia nos ayudó a manejar nuestro grafo durante el desarrollo de la actividad integral.

Marisol Rodríguez Mejía A01640086
Luis Enrique Lemus Martínez A01639698

Para ordenar nuestra lista de adyacencia de menor a mayor dirección IP, utilizamos heapsort, el cual es un algoritmo de ordenamiento no recursivo, no estable, con complejidad computacional, el cual podríamos decir es uno de los más eficientes que hemos visto.
En éste repetidamente elegimos el elemento más grande y lo movemos al final del vector, convirtiendo la lista de adyacencia en un Max-Heap para acelerar el proceso. Tiene una complejidad temporal linearítmica en todos sus casos. Para ordenar con heapsort, examinamos cada elemento del grafo y lo movemos hacia abajo hasta que sea más grande que los elementos anteriores a éste.

Para la búsqueda de las IPs utilizamos el algoritmo de búsqueda binaria. Tiene una complejidad de O(log n), donde dicha complejidad se debe a que divide a la mitad el conjunto de entrada en cada iteración. Es más rápido que muchos otros tipos de algoritmos de búsqueda (Jindal, 2021).
El pseudocódigo del algoritmo es el siguiente:

$\boldsymbol{busquedaBinaria(A, n, k)}$
Input: Un arreglo A ordenado de tamaño n, una llave de búsqueda k
Output: El índice del primer elemento en A igual a k o -1 si no se encuentra
$l \leftarrow 0$
$r \leftarrow n - 1$
while $l \leq r$ **do**
    $m \leftarrow l + (r - l)/2;$
    if $k == A[m]$ **then**
        return $m;$
    else if $k < A[m]$ **then**
        $r \leftarrow m - 1;$
    else
        $l \leftarrow m - 1;$
    end
end
return -1;

Cabe destacar el Max-Heap que usamos en esta actividad, el cual es una estructura de datos especializada basada en árboles que es esencialmente un árbol casi completo que satisface la propiedad del montón: en un montón máximo, para

cualquier nodo C dado, si P es un nodo padre de C, entonces la clave (el valor) de P es mayor o igual que la clave de C. (colaboradores Wikipedia, 2021).
El Max-Heap nos permitió ir ordenando nuestras direcciones IP dependendiendo del número de adyancencias (o grado) de cada una. Para esto, fuimos insertando cada vértice del grafo en el Max-Heap con el método push. Su complejidad puede ser calculada considerando un escenario en el cual insertamos un nuevo nodo en el heap, y la prioridad del padre del nodo es mayor que la del nodo recién insertado. En tal caso, no necesitamos hacer nada y no se necesita ningún cambio en el Max-Heap ya que cumple con las reglas necesarias; por ello, la complejidad sería de O(1). Sin embargo, en otros casos no es así; en el peor caso, como esta estructura es un árbol completo y balanceado, el nuevo nodo insertado tiene que ser intercambiado en cada nivel desde el último hasta la raíz para mantener las propiedades de los heap. Al tener que intercambiar en cada nivel del árbol y sabiendo que tenemos una altura de O(logn), recorrer toda la estructura hace que nuestro algoritmo push cuente con una complejidad temporal de O(logn).
Por otra parte, para obtener las 5 IPs con más IPs adyacentes, utilizamos el método de pop. Este método se encarga de eliminar y retornar la raíz del Max-Heap, es decir, el elemento de mayor prioridad. Su complejidad es de O(logn) debido a que para obtener la raíz tarda O(1), pero, para eliminar dicha raíz, hay un proceso de reemplazo desde la raíz hasta la parte inferior del montón, ese recorrido del árbol toma O(logn) tiempo.

Por último, obervamos el uso efectivo de los grafos al resolver el problema y pudimos notar la eficiencia que tienen, por ello, fue enriquecedor implementarlos.
En una situación problema de esta naturaleza, los grafos nos pueden ser de gran ayuda. Lo anterior debido a que son una estructura que nos permite marcar relaciones entre diferentes elementos, en este caso, las direcciones IP. Capturar las adyacencias a cada IP nos permitió concluir cuál era el boot master.

En conclusión, los grafos se pueden usar para modelar muchos problemas del mundo real. Al realizar la actividad, nos dimos cuenta de que este tipo de estructura de datos hace que sea más sencillo trabajar con cantidades grandes de datos.

De igual manera, los grafos nos parecen una herramienta visual muy eficaz porque tienen la capacidad de presentar la información de forma rápida, sencilla y entendible, revelando tendencias en nuestros datos.
Al principió fue difícil y tardado adaptarlo con grafos pero ya que lo logramos, lo demás fue más rápido. Y el trabajo en equipo fue algo que nos ayudó mucho, ya que nos complementamos muy bien y así terminamos el trabajo más eficientemente.

Marisol Rodríguez Mejía A01640086
Luis Enrique Lemus Martínez A01639698

Referencias:

Carlson, S. (s. f.). graph theory | Problems & Applications. Recuperado 23 de noviembre de 2021, de https://www.britannica.com/topic/graph-theory

colaboradores de Wikipedia. (2021, agosto 4). Grafo. Recuperado 22 de noviembre de 2021, de https://es.wikipedia.org/wiki/Grafo

GeeksforGeeks. (2021, 15 septiembre). HeapSort. Recuperado de https://www.geeksforgeeks.org/heap-sort/

Jindal, H. (2021, 11 marzo). Búsqueda binaria. Recuperado de https://www.delftstack.com/es/tutorial/algorithm/binary-search/#:%7E:text=Complejidad%20del%20algoritmo%20de%20b%C3%BAsqueda%20binaria,-Complejidad%20del%20tiempo&text=Cuando%20realizamos%20la%20b%C3%BAsqueda%20binaria,a%20la%20mitad%20cada%20vez.&text=Este%20resultado%20de%20esta%20recurrencia,orden%20de%20O(logn)%20.

Wikipedia contributors. (2021b, noviembre 5). Heap (data structure). Recuperado de https://en.wikipedia.org/wiki/Heap_(data_structure)