Instituto Tecnológico y de Estudios Superiores de Monterrey

Arturo Sanchez Rodriguez - A01275427

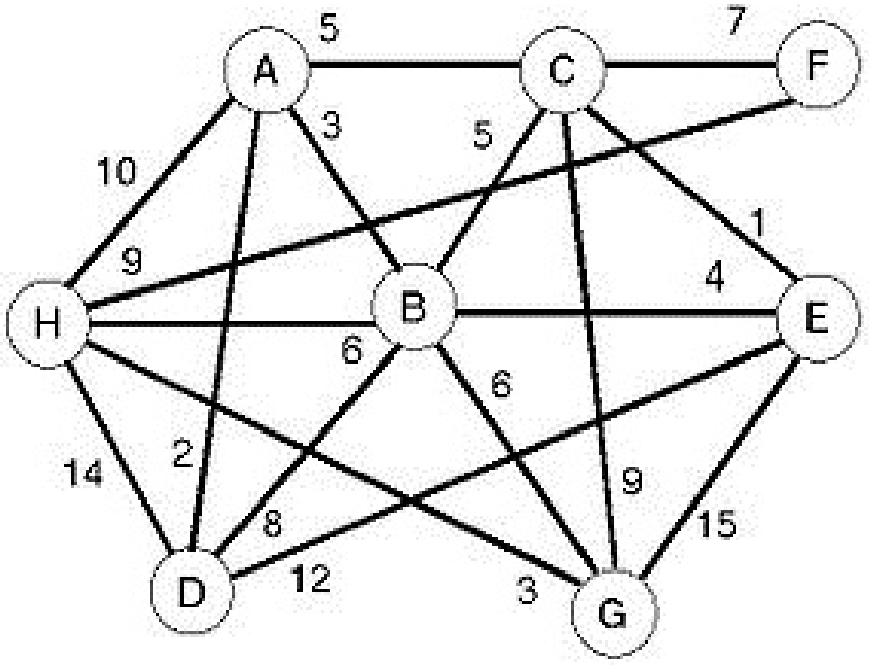
Programación de estructura de datos y algoritmos fundamentales (Gpo 850)

ReflexActIntegradora

01 de Junio del 2023

Para abordar este trabajo, se propone utilizar grafos como una estructura de datos eficiente para representar la red de direcciones IP y resolver los problemas planteados. Un grafo consiste en un conjunto de nodos (vértices) y conexiones (aristas) entre ellos. La lista de adyacencias se utiliza para representar el grafo en este caso. Cada nodo del grafo corresponderá a una dirección IP de origen, y las adyacencias se generarán en función de las conexiones encontradas en el archivo de entrada "bitacoraGrafos.txt".

En programación de estructuras de datos y algoritmos, un grafo es una estructura de datos que consiste en un conjunto de nodos (también llamados vértices) conectados entre sí mediante arcos (también llamados bordes). Los grafos se utilizan para representar relaciones entre objetos y se aplican en una amplia gama de problemas, desde redes sociales hasta algoritmos de rutas y planificación.



https://www.bibliadelprogramador.com/2018/06/implementado-algoritmos-de-grafos.html

En C + +, los grafos se pueden implementar de varias formas. Una de las formas más comunes es mediante el uso de listas de adyacencia. En esta representación, cada nodo del grafo se representa como un objeto o una estructura que contiene una lista de los nodos a los que está conectado.

Para calcular el grado de salida de cada nodo, se recorre la lista de adyacencias y se cuenta el número de IPs adyacentes a cada IP de origen. Luego, se almacena esta información en el archivo "grados_ips.txt" en forma de pares (IP, grado de salida). Para determinar las 5 IPs con mayor grado de salida, se puede utilizar una estructura de datos como un montículo (heap) para mantener un seguimiento de las IPs con los mayores grados de salida. Se insertan los pares (IP, grado de salida) en el montículo y se extraen los 5 mayores elementos, que luego se almacenan en el archivo "mayores_grados_ips.txt" en orden descendente.

Si ya se ha implementado una clase Heap en C + +, se puede utilizar para este propósito. En caso contrario, se debe implementar dicha clase antes de continuar con la aplicación. Una vez que se tienen las 5 IPs con mayor grado de salida, se debe encontrar en qué dirección IP presumiblemente se encuentra el boot master. La dirección IP correspondiente al boot master será aquella que tenga la mayor cantidad de conexiones salientes (mayor grado de salida). Esto se puede obtener directamente de la lista generada anteriormente.

El siguiente paso es encontrar el camino más corto entre el boot master y el resto de los nodos (IPs) del grafo. Para esto, se puede utilizar un algoritmo de búsqueda de caminos más cortos, como el algoritmo de Dijkstra. El boot master se considerará como el nodo de origen, y se calculará la distancia mínima desde él a cada uno de los demás nodos. Luego, se almacenará esta información en el archivo "distancia_bootmaster.txt" en forma de pares (IP, distancia). Finalmente, para determinar la dirección IP que presumiblemente requiere menos esfuerzo para que el boot master la ataque, se debe buscar el nodo (IP) con la distancia mínima al boot master. Esto se puede obtener a partir de la lista generada anteriormente. La IP correspondiente a la distancia mínima representará la dirección IP que presumiblemente requiere menos esfuerzo para que el boot master la ataque.

Es importante tener en cuenta la complejidad computacional de los métodos implementados. La construcción de la lista de adyacencias requiere recorrer el archivo de entrada, por lo que su complejidad es lineal en función del tamaño del archivo. El cálculo del grado de salida de cada nodo también tiene una complejidad lineal, ya que implica recorrer la lista de adyacencias. La búsqueda de las 5 IPs con mayor grado de salida utilizando un heap tiene una complejidad logarítmica en función del número total de nodos. El algoritmo de Dijkstra utilizado para encontrar los caminos más cortos tiene una complejidad de O((V + E) log V),

donde V es el número de nodos y E es el número de aristas. Por último, la búsqueda de la IP con la distancia mínima también tiene una complejidad lineal.

En resumen, el informe propone el uso de grafos como una estructura de datos eficiente para abordar un problema relacionado con direcciones IP. Se utiliza una representación de grafo mediante listas de adyacencia para almacenar y analizar las conexiones entre las direcciones IP. El informe sugiere calcular el grado de salida de cada nodo y encontrar las 5 direcciones IP con mayor grado de salida. Para ello, se propone utilizar una estructura de datos como un montículo (heap) para mantener un seguimiento de las IPs con los mayores grados de salida.

Además, se plantea identificar la dirección IP correspondiente al boot master, que se define como la que tiene la mayor cantidad de conexiones salientes. Luego, se propone encontrar el camino más corto entre el boot master y el resto de los nodos utilizando el algoritmo de Dijkstra. Se destaca la importancia de tener en cuenta la complejidad computacional de los métodos implementados y se proporciona información sobre la complejidad de cada uno de ellos. En general, el informe presenta una estrategia clara para abordar el problema utilizando grafos y proporciona recomendaciones sobre cómo implementar los algoritmos necesarios en C + +.

Referencias. -

- GraphEverywhere, E. (2021). Grafos | Qué son, tipos, orden y herramientas de visualización. *GraphEverywhere*. https://www.grapheverywhere.com/grafos-que-son-tipos-orden-y-herramientas-de-visualizacion/
- *La utilidad y aplicación de los Grafos y Sistemas de Información Geográfica*. (s. f.). Delfino.cr. https://delfino.cr/2023/01/la-utilidad-y-aplicacion-de-los-grafos-y-sistemas-de-informacion-geografica