Reflexión Actividad 4.3 (Evidencia)

Santiago Reyes Moran - A01639030

Lucas Wong Mang - A01639032

23/11/2021

Importancia y eficiencia de los Grafos

Los grafos son una estructura de datos fundamental en el mundo de la programación; esta se utiliza para representar relaciones entre distintos elementos. Esta estructura de datos está compuesta por nodos (vértices) y conexiones (bordes o edges), cada nodo tiene una lista de conexiones, la cual indica a cuáles otros nodos está conectado ese nodo. Para la actividad que se deseaba realizar se implementa un grafo no ponderado, lo cual significa que las conexiones entre nodos no tienen "peso". En un grafo ponderado, las conexiones entre nodos sí tienen peso, representando de alguna forma una "distancia" entre los nodos; es en este tipo de grafo donde se pueden utilizar algoritmos de pathfinding como el algoritmo de Dijkstra.

Los grafos son sumamente importantes debido a su representación de relaciones entre elementos, lo cual es algo que se utiliza ampliamente hoy en día. Los grafos se pueden utilizar para desarrollar plataformas como Facebook, en donde un nodo sería un usuario y sus conexiones serían sus amigos. La amplia implementación de esta estructura de datos en la industria moderna hace que sea indispensable conocer su importancia y comprender cómo utilizarlos.

Algoritmos utilizados y su complejidad

Push:

Este método es el que se utiliza para agregar elementos al Max Heap de objetos tipo registroGrafo, en donde se almacena la IP junto con su grado (cantidad de conexiones o adyacencias). En un vector normal, la complejidad temporal es constante O(1), sin embargo, como tenemos que asegurar que las propiedades básicas de un Heap no son violadas y corregir el heap si sí lo son, entonces la complejidad temporal se vuelve O(log(n)). Sin embargo, si la inserción inicial se hace sin violar las propiedades del Heap entonces sería el mejor caso O(1). Si todos los push se hacen de forma correcta, la raíz (el primer elemento del vector) será el elemento con mayor prioridad.

Pop:

El Pop tiene la misma complejidad temporal que el Push O(log(n)), lo cual se debe a que después de borrar el elemento deseado se debe de utilizar el método de heapify; este tiene una complejidad temporal de O(log(n)). Heapify se debe de implementar para asegurar que las propiedades del heap no se violen y por lo tanto el elemento con mayor prioridad quedará como la raíz (el primer elemento del vector) después del pop.

Unordered_map:

En un grafo se suele utilizar algún tipo de lista o vector para almacenar las conexiones entre los nodos del grafo, lo cual se llevó a cabo en clase, pero para esta actividad decidimos integrar una estructura de datos distinta: el Unordered_map. Debido a que estamos leyendo las direcciones IP como strings, es sumamente conveniente poder acceder a la lista de adyacencia de una IP con una referencia simple al string de esa IP. En un vector no se puede acceder a indices como strings (lista[“0.0.0.0”] sería inválido), por lo cual se tendría que llevar a cabo una función de hashing y utilizar métodos más complejos para llegar al mismo resultado.

En un Unordered_map utilizamos un par llave-valor, en nuestra implementación la llave es la IP (origen) y el valor es un vector de objetos tipo registro (representando el registro de la IP destino, la cual incluye la IP, el mes, el dia, la hora y el error). Llevar la lista de adyacencia a cabo de la forma propuesta hace que su complejidad de inserción y búsqueda sea constante O(1) y simplifica significativamente la implementación de un grafo para esta actividad específica.

Conclusión

Los grafos son estructuras de datos sumamente importantes en la industria, por lo cual saber implementarlos de distintas maneras es fundamental para el desarrollo de un profesionista. En esta actividad es de gran importancia implementar grafos de forma apropiada para poder determinar el grado de cada IP de forma eficiente y ordenada; esto nos llevó a comprender el uso de esta estructura de datos de una mejor manera, lo cual sin duda nos ayudará a implementar grafos en un futuro de nuestra vida profesional.

Referencias:

- *Heap implementation: Push: Pop: Code*. TECH DOSE. (2021, January 26). Retrieved November 6, 2021, from https://techdose.co.in/heap-implementation-push-pop-code/.ç
- UNORDERED_MAP in C++ STL. GeeksforGeeks. (2021, September 17). Retrieved November 23, 2021, from https://www.geeksforgeeks.org/unordered_map-in-cpp-stl/#:~:text=The%20time%20complexity%20of%20map,O(1)%20on%20average.&text=A%20lot%20of%20functions%20are%20available%20which%20work%20on%20unordered_map.
- freeCodeCamp.org. (2016, December 1). *A gentle introduction to data structures: How graphs work*. freeCodeCamp.org. Retrieved November 23, 2021, from https://www.freecodecamp.org/news/a-gentle-introduction-to-data-structures-how-graphs-work-a223d9ef8837/#:~:text=Graphs%20are%20a%20powerful%20and,types%20of%20data%20(nodes).