
Reflexión Actividad 5.2 (Evidencia)

Santiago Reyes Moran - A01639030

Lucas Wong Mang - A01639032

04/12/2021

Importancia y eficiencia de las Tablas Hash

Las tablas hash, dentro del ámbito de programación, son estructuras de datos que permiten acceder en tiempo constante $O(1)$ a los valores de una tabla basándose en una función de hashing interna para la "llave" de dicha tabla para encontrar su posición dentro de la misma. Al aplicar esta función de hashing se puede acceder fácilmente al valor dentro de la tabla utilizando el índice encontrado.

Otro factor importante a considerar al implementar una tabla hash es el manejo de colisiones. Existen muchas posibles implementaciones del manejo de colisiones dentro de una tabla hash, pero para los propósitos de esta actividad se nos pidió hacer la implementación con prueba cuadrática. Se puede calcular la cantidad esperada de colisiones en una implementación con muestreo lineal utilizando la siguiente fórmula (en donde $\alpha = n/m$):

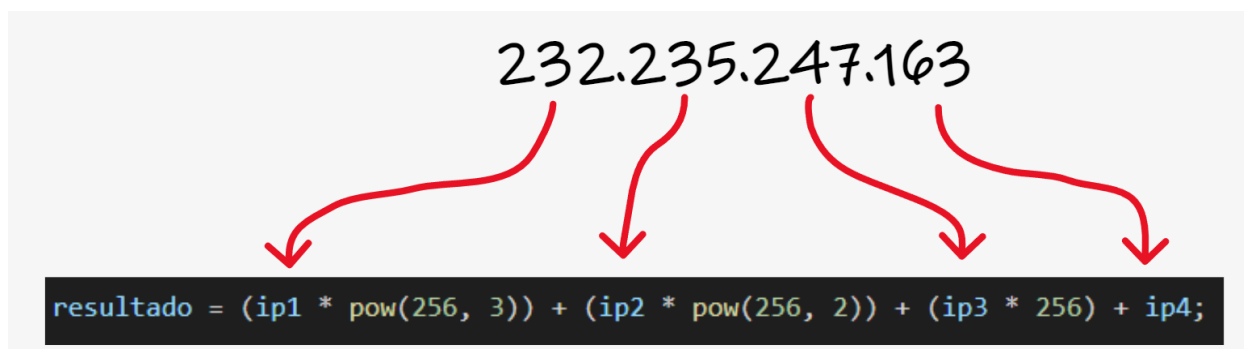
$$S = \frac{1}{2} \left(1 + \frac{1}{1 - \alpha} \right), \quad U = \frac{1}{2} \left(1 + \frac{1}{(1 - \alpha)^2} \right)$$

En esta actividad n sería 13370 (La cantidad de IPs a insertar) y m sería 13381 (El tamaño de la tabla hash). El resultado sería $S=608.72$ y $U=304.739$. Se tiene que tomar en consideración que en este caso no se utilizó muestreo lineal, por lo cual la fórmula únicamente es para ejemplificación. Adicionalmente, la cantidad de colisiones dependerá mucho del tamaño designado de la tabla hash, mientras este sea mayor, la cantidad de colisiones será menor y el tiempo de ejecución también será menor, pero la complejidad espacial será más grande.

Algoritmos utilizados y su complejidad

Función de hash:

Para poder hacer la implementación de la tabla hash se tuvo que definir una función de hashing. Para este caso utilizamos la conversión a valor numérico de una IP, en donde el resultado de este está definido por la siguiente fórmula:



The diagram illustrates the conversion of the IP address 232.235.247.163 into a numerical hash value. Red arrows point from each octet of the IP address to its corresponding term in the formula below:

$$\text{resultado} = (\text{ip1} * \text{pow}(256, 3)) + (\text{ip2} * \text{pow}(256, 2)) + (\text{ip3} * 256) + \text{ip4};$$

Al tener el valor numérico de la IP podemos utilizar módulo contra el tamaño de la tabla (%13381) para obtener la posición inicial en la cual se deberá de tratar de insertar el objeto dentro de la tabla hash. Después de eso, si es que el espacio en la posición que se buscó inicialmente está ocupado, se deberá de utilizar prueba cuadrática para encontrar la próxima posición de la tabla a buscar, lo cual se tendrá que hacer hasta encontrar una posición vacía (Si es que no está llena la tabla).

Heap Sort:

El Heapsort es el método que se utiliza para ordenar los vectores de resultado para los resúmenes de las IPs (IPs accedidas e IPs que accedieron) basado en el valor numérico de la IP. Este método únicamente se utiliza en el resumen de IPs para imprimir dichos resúmenes ordenados en base a su IP. El algoritmo hace uso de el método de Heapify, el cual tiene una complejidad temporal $O(\log(n))$, pero al estarlo llamando una cantidad n de veces, la complejidad temporal total del algoritmo de Heapsort se vuelve $O(n\log(n))$, la cual es una buena complejidad temporal para un algoritmo de ordenamiento.

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 índices 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 pares, en los cuales la primera parte del par es la dirección IP asociada y el segundo valor es un 0 o un 1 dependiendo de si la IP fue accesada por la IP origen o si la IP origen fue accesada por la IP. 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

Las tablas hash son unas de las estructuras de datos más importantes en la programación, su implementación se encuentra presente dentro de objetos como los diccionarios en python o en los mismos unordered maps de c++. El poder referenciar valores utilizando cualquier tipo de objeto como llave es algo sumamente útil en la programación. Esta actividad es sumamente relevante porque se puede aplicar un método de hashing intuitivo para las IPs y de esta manera acceder a los objetos (ResumenIP en el caso de nuestra implementación) guardados en las casillas correspondientes de la tabla hash. Sin duda tendremos en cuenta las aplicaciones que este tipo de estructura de datos puede tener en la programación ahora que, mediante esta actividad, comprendemos su utilidad.

Referencias:

- Hsu, J. (2020, March 24). What are hash tables and why are they amazing? Medium. Retrieved December 5, 2021, from <https://betterprogramming.pub/what-are-hash-tables-and-why-are-they-amazing-89cf52246f91>.
- Heap implementation: Push: Pop: Code. TECH DOSE. (2021, January 26). Retrieved November 6, 2021, from <https://techdose.co.in/heap-implementation-push-pop-code/.c>
- 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](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).