IIC2133 - T03

Leonardo Olivares

5/11/18

1 Número de Colisiones en la Tabla de Hash

Para esta actividad, debido a la gran cantidad de estados posibles para cada tablero, se debía contar con una estructura capaz de identificar si un estado ya había sido analizado. La complejidad de esta operación, para lograr encontrar la solución en un tiempo corto, debia tener una complejidad constante O(1).

La estructura utilizada para lograr realizar estas verificaciones fue un diccionario, a partir de una Tabla de Hash que almacenaba nodos con información de los estados visitados.

Sin embargo, para que la Tabla de Hash lograra verificar en un tiempo constante, fue necesario utilizar una función de hash ideal, que no produjera tantas colisiones.

Las colisiones ocurrren, en esta actividad, cuando dos estados distintos retornan un mismo valor de hash, que luego los mapea a un mismo espacio en memoria. Debido a la colisión se hace necesario revisar la información almacenada, para verificar que efectivamente es el estado que se busca. Si no era el mismo estado, se revisa el espacio de memoria adyacente (direccionamiento abierto).

Si en una tabla se produce una gran cantidad de colisiones, se requerirá revisar varios espacios de memoria, lo que ocasionará un mayor tiempo de búsqueda.

A continuación, se muestra un gráfico que indica la cantidad de veces que ocurrió una colisión en cada posición de la tabla de hash.

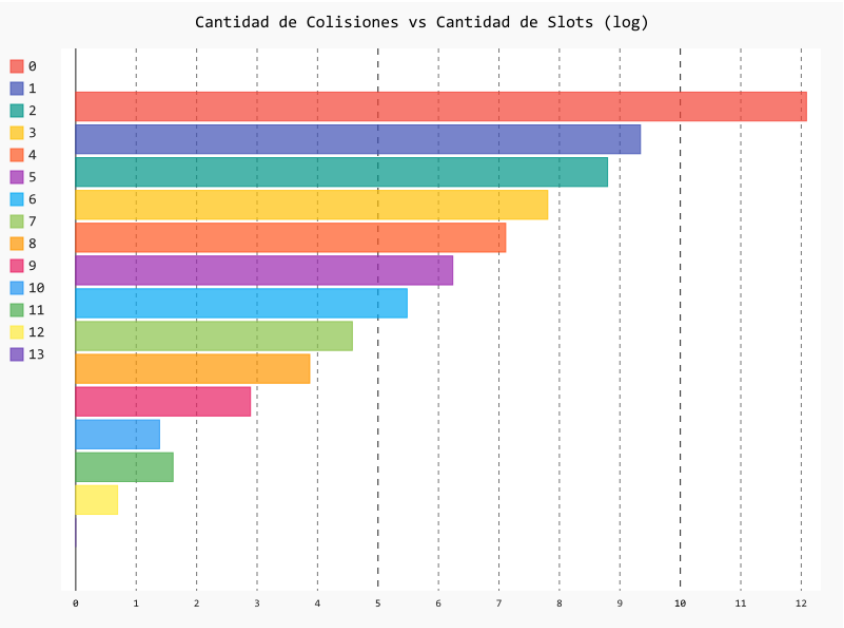

Figure 1: Cantidad de colisiones vs Cantidad de Slots (escala logarítmica)

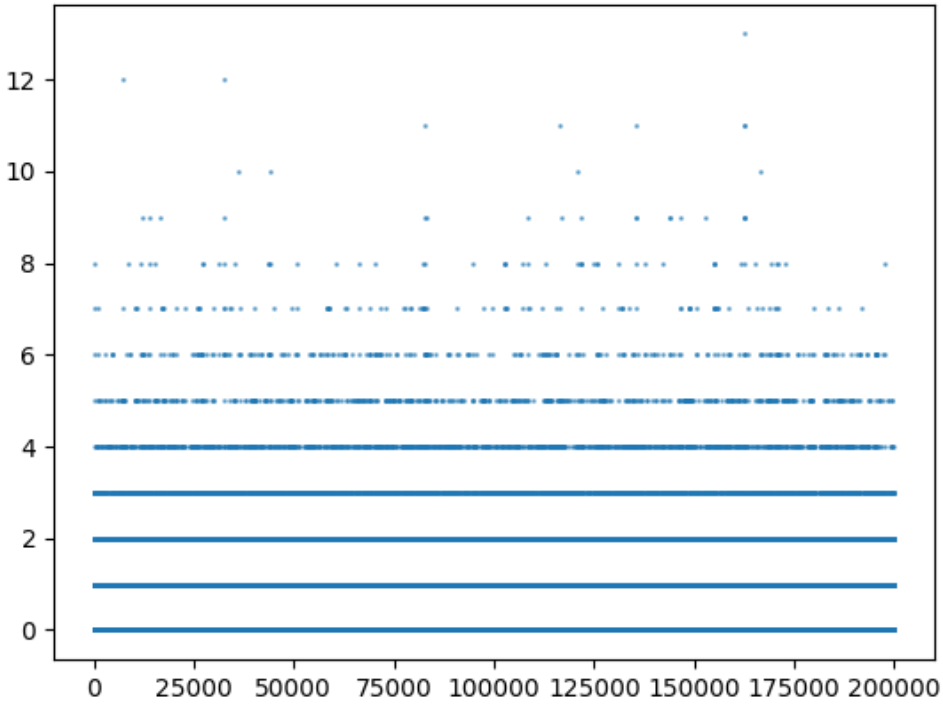

Figure 2: Cantidad de colisiones por Slot (4x4/test2.txt)

2 Justificación de la Función de Hash

La función utilizada para hashear los estados del tablero, es conocida como Zobrist, y es utilizada con frecuencia para hashear distintos tipos de tableros, como ajedrez.

La ventaja de Zobrist se basa en su capacidad de utilizar numeros al azar para cada casillero, y a partir de ellos, en conjunto con operaciones de bits, encontrar un hash para el tablero. Mientras mayor sea la cantidad de números al azar que se generen, menor es la probabilidad de que ocurran colisiones en la tabla de hash. Además, Zobrist permite realizar un proceso de hashing incremental, que evita realizar el calculo del hash completo, y solo considerando las celdas que cambian.

A partir de esta función de hash y los gráficos anteriores, se puede concluir que Zobrist es logra de manera eficiente, evitar que el número de colisiones aumente. Se puede observa en el primer gráfico, que para el test 4x4/test2.txt la mayor cantidad de colisiones que ocurrieron en una sola casilla fue de 13. Esto es una cantidad razonable de colisiones considerando que la capacidad de la tabla era de 200.000 casillas y estaba ocupada con un total de 67038 estados. Por otro lado, considerando la escala logarítmica, en la mayor parte de las casillas no ocurrieron colisiones.

Del mismo modo, en el segundo gráfico se puede observar una descripción general de todos las casillas de la tabla de hash. Los resultados concuerdan con el primer gráfico, siendo que la mayoría de las casillas tienen 0 o 1 colisiones. Luego, las colisiones van reduciendo considerablemente.

3 Número de Estados y Rehashing

Para todos los tests, la tabla de hash tuvo una capacidad máxima inicial de 100.000 posiciones, y un carga máxima de 0.5. Es decir, el proceso de rehashing ocurre cada vez que la tabla ocupa la mitad de sus posiciones, y se cambia por una con el doble de capacidad.

A continuación se muestra una tabla con registros de cada uno de los tests, en donde se muestra la cantidad de estados que fueron revisados por el algoritmo, y la cantidad de rehashings que se necesitaron, bajo las condiciones mencionadas anteriormente.

Test	**Estados Creados**	**Cantidad de Rehash**	**Tiempo Total en Rehash (seg)**
3x3/test0.txt	140	0	0
3x3/test1.txt	5209	0	0
3x3/test2.txt	158957	2	0.033196
3x3/test3.txt	177497	2	0.032695
3x3/test4.txt	175579	2	0.034865
3x3/test5.txt	258691	3	0.075504
4x4/test0.txt	144	0	0
4x4/test1.txt	2560	0	0
4x4/test2.txt	67038	1	0.012508
4x4/test3.txt	290852	3	0.097669
4x4/test4.txt	830445	5	0.422867
4x4/test5.txt	1961207	6	0.91288

Se puede observar que el tiempo total utilizado en procesos de rehashing, incrementa con el número de estados que son necesarios revisar para llegar a la solución final. Esto ocurre debido a que, al realizar un rehash de la tabla, es necesario visitar cada bucket de la tabla, verificar si existe un estado en dicha posición y aplicar un nuevo hash que lo mapee a la siguiente tabla.