Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2133 - Estructuras de Datos y Algoritmos

# Tarea 3

5 de noviembre de 2018

2º semestre 2018 - Profesor Yadran Eterovic Solano

Luis Enrique Chodiman Herrera - 15635279

---

## 1. Análisis

### 1.1. Distribución de colisiones

A continuación se presenta el gráfico de colisiones para el estado final de la tabla de hash en la ejecución del test 1 de tamaño 4x4. Se estableció que la cantidad de *slots* iniciales de la tabla de hash sería de 256.

Cabe mencionar que dada la gran cantidad de *slots* en la tabla, los gráficos fueron hecho con *excel*.

Este gráfico se obtuvo poniendo un contador para cada nodo de la tabla y aumentándolo cada vez que se quería insertar algo en esa posición.

Finalmente, al terminar la ejecución se genera un archivo *csv* que permite graficar los datos.

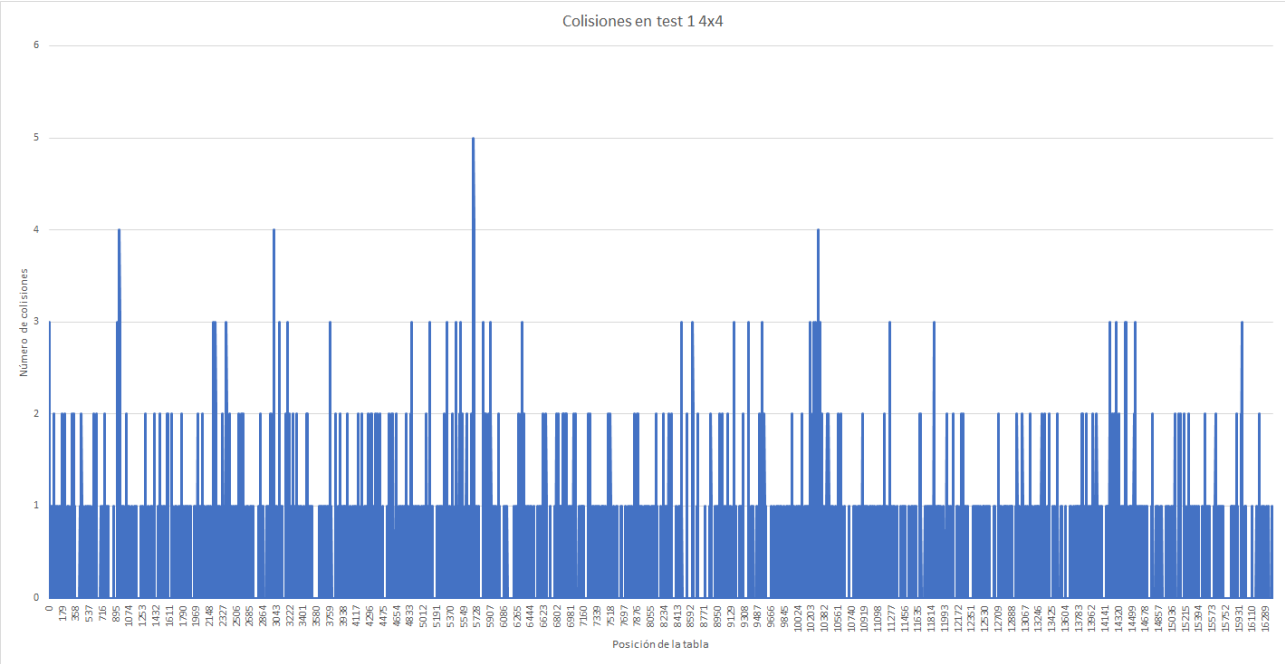


Figura 1: Colisiones en test 1 4x4, sondeo lineal

A partir de la figura [1] se observa que la función de hash distribuye relativamente uniforme. Esto se debe porque la función utilizada como hash (*crc32*) por construcción genera la menor cantidad de colisiones posibles. Cabe mencionar que esta función no tiene como objetivo *hashear* datos, sino que verificar integridad de *strings*, es por esto que la cantidad de colisiones es baja.

## 1.2. Función de hash

La función *crc32* calcula una crc (*cyclic redundancy checksum*) de 32 bits para un *string* y es usada generalmente para verificar integridad de datos.

Además, dado el gráfico, se observa que la cantidad de colisiones es relativamente uniforme.

Esto se puede comprobar en la siguiente tabla, en donde se muestra la frecuencia con la que ocurren cierta cantidad de colisiones para el mismo gráfico anterior.

| N colisiones | Frecuencia |
|---|---|
| 0 | 14636 |
| 1 | 1374 |
| 2 | 304 |
| 3 | 60 |
| 4 | 9 |
| 5 | 1 |

Cuadro 1: Frecuencia del número de colisiones, sondeo lineal

Se aprecia que la mayor parte de la tabla (0,893 %) se mantuvo sin colisiones, por lo que se cumple que la función distribuye minimizando la cantidad de estas.

## 1.3. Número de estados y *rehashing*

A continuación se presentan estadísticas para el número de estados explorados, el número de veces que se tuvo que hacer *rehashing* y el tiempo total que se ocupó en *rehashing*.

El número de estados se obtuvo agregando todos los estados a un archivo de texto y contando su número. Cabe mencionar que estos sólo se agregaban si no habían sido agregados antes, por lo que son todos diferentes.

La cantidad de veces que se tuvo que hacer *rehashing* se obtuvo poniendo un contador a la tabla de hash que se actualiza cada vez que se llama a la función para hacer el *rehashing*.

El tiempo invertido en esto último se calcula poniendo una variable *clock_t* al comienzo y otra al final de la función de *rehashing* y se suma la diferencia a un contador.

Para los test de 3x3:

| Test | N estados | N rehashing | Tiempo rehashing[s] |
|---|---|---|---|
| 0 | 172 | 1 | 0 |
| 1 | 12549 | 7 | 0 |
| 2 | 263926 | 12 | 0.156 |
| 3 | 262938 | 12 | 0.125 |
| 4 | 242668 | 11 | 0.063 |
| 5 | 326647 | 12 | 0.172 |

Cuadro 2: Análisis de estados y rehashing tests 3x3

Para los test de 4x4:

| Test | N estados | N rehashing | Tiempo rehashing[s] |
|---|---|---|---|
| 0 | 202 | 1 | 0 |
| 1 | 6634 | 6 | 0 |
| 2 | 170085 | 11 | 0.094 |
| 3 | 350677 | 12 | 0.172 |
| 4 | 1239894 | 14 | 0.734 |
| 5 | 6119025 | 16 | 3 |

Cuadro 3: Análisis de estados y rehashing tests 4x4

En relación a los cuadros 2 y 3, se observa que a medida que aumenta el número de estados que se deben explorar, aumenta también la cantidad de *rehashings* de manera que la tabla de *hash* pueda almacenar dichos estados. Debido a que se debe hacer *rehashing*, el tiempo invertido en *rehashing* va a aumentar dependiendo de la cantidad de datos que se deben *rehashear*.

Este tiempo usado en *rehashing* será lineal respecto a la cantidad total de datos *rehasheados*, y esta cantidad es, a su vez, lineal respecto a la cantidad de *rehashing* totales y la cantidad de estados explorados.

En conclusión, el tiempo total que toma *rehashear* viene dado por: $O(N * R)$, donde $N$ es el número de estados y $R$ el número de *rehashing* realizados.

## 1.4. Sondeo lineal v/s cuadrático

A continuación se presenta un gráfico similar al de la sección 1, pero implementando sondeo cuadrático para la resolución de colisiones. Cabe mencionar que en la figura 1 se utilizó sondeo lineal.

Para este sondeo cuadrático se escogieron las constantes $c_1 = 0$, $c_2 = 1$.

Al igual que en la figura 1 se utilizó el test 1 de tamaño 4x4. En ambos casos se parte con una tabla de tamaño inicial 256 *slots*.

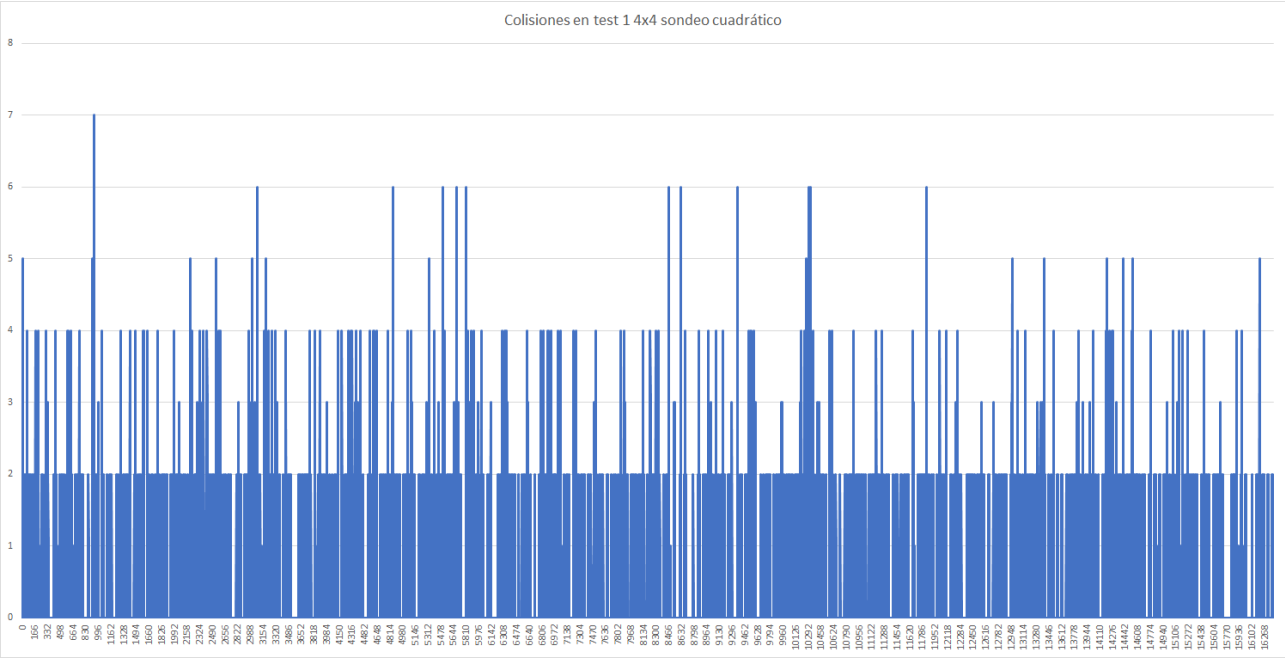


Figura 2: Colisiones en test 1 4x4, sondeo cuadrático

A partir de la figura 2, se observa que en relación a la figura 1, la que distribuye cuadráticamente lo hace de manera más equitativa, es decir, la cantidad de colisiones es mayor pero estas ya no se encuentran en un *cluster* como sucede en el sondeo lineal.

Esto se puede corroborar observando la tabla de frecuencias de colisiones para el caso cuadrático:

| N colisiones | Frecuencia |
|---|---|
| 0 | 14715 |
| 1 | 444 |
| 2 | 964 |
| 3 | 105 |
| 4 | 128 |
| 5 | 14 |
| 6 | 11 |
| 7 | 1 |

Cuadro 4: Frecuencia del número de colisiones, sondeo cuadrático

Si se calculan la cantidad de colisiones totales a partir de las tablas de frecuencia se obtiene que en el sondeo lineal hay 2203 colisiones, mientras que en el cuadrático hay 3342.

En suma, en ambos casos se ve un buen desempeño en general, sin embargo, existe un *trade-off* entre la cantidad de colisiones y qué tan bien se distribuyen dichas colisiones. Teniendo en

cuenta esto último, en este caso particular conviene usar sondeo lineal y un factor que limite el largo de un *cluster* de colisiones, de manera que no sea costosa la operación de buscar ahí.