Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Tarea 3

Hashing perfecto

Integrante: Samuel Chávez

Profesores: Benjamín Bustos

Auxiliar: Máximo Flores
Sergio Rojas

Ayudante: Claudio Gaete
Martina Mora
Matías Rivera

Fecha de realización: 20 de Diciembre de 2024
Fecha de entrega: 3 de Diciembre de 2025
Santiago, Chile

Índice

1. Introducción

En este informe se aborda la implementación y evaluación del algoritmo de hashing perfecto, una técnica que permite almacenar conjuntos de valores enteros sin colisiones en las búsquedas. Para cumplir con los requerimientos del problema, se parametrizan las condiciones de selección de funciones de hash y tamaños de tablas utilizando los parámetros c y k, que influyen directamente en el tamaño de las tablas y la distribución de los elementos.

El objetivo principal de este trabajo es analizar el impacto conjunto de estos parámetros sobre el tiempo de construcción y el espacio requerido por la estructura, explorando su interacción en diversos escenarios experimentales. Se plantea como hipótesis que c y k no tienen efectos independientes, ya que ambos son escalares que, de manera matemática.

Adicionalmente se postula una métrica sencilla para la evaluación de construcción de tablas

2. Desarrollo

2.1. Método Experimental

El experimento utiliza multithreading para ejecutando los experimentos de manera independiente que construyen una tabla, La selección de funciones de hash se realiza iterativamente hasta cumplir con las restricciones dadas por los parámetros c y k, asegurando que el espacio requerido se mantenga dentro de los límites establecidos.

Los experimentos realizados se dividen en cuatro fases principales:

1. **Prueba base con parámetros fijos:**
   - En esta fase se fija $c = 1$ y $k = 4$, construyo 100 tablas para tamaños en el rango $n \in [10^3 ... 10^7]$ equidistantes inclusivo.
   - Se mide el tiempo promedio de construcción y el tamaño final de las tablas para validar la linealidad esperada en el análisis teórico.

2. **Variación de k con n y c fijos:**
   - En este caso, se fija $n = 10^6$ y $c = 1$, mientras que k varío en 100 valores en el rango $k \in [1, 8]$ equidistantes inclusivo.
   - Se evalúa el tradeoff entre el tiempo de construcción y el tamaño promedio de las tablas finales, buscando observar cómo el ajuste de c afecta la eficiencia.

3. **Variación de c con n y k fijos:**
   - Aquí se fija $n = 10^6$ y $k = 4$, mientras que c varío en 100 valores en el rango $k \in [1, 4]$ equidistantes inclusivo.
   - Se analizan los mismos indicadores que en el experimento anterior para observar si c afecta de manera similar al ajuste de k.

4. **Sondeo combinado variando c, k y n:**
   - En esta fase, se tomaron puntos, 5 en $n \in [10^3 ... 10^7]$, 25 $k \in [1, 8]$, y 25 en $c \in [1, 4]$ equidistantes e inclusivos se combinan para generar una nube de puntos tamaño 3125.
   - Se busca determinar cómo la interacción entre estos parámetros afecta el rendimiento, ofreciendo recomendaciones sobre combinaciones óptimas en términos de tiempo y espacio.

También se formulo una métrica dado una cantidad de datos fija como la siguiente útil para el ultimo experimento

$$\text{score\_n} = \sqrt{t_{c,k}^2 + m_{c,k}^2}, \forall c, k \tag{1}$$

Que se explica como, dado un n se normalizan los tiempos y espacio en esa muestra para el n escogido, y se asigna el puntaje como la distancia euclida de ambos elementos calculados

Cada experimento registra el tiempo de construcción promedio, el tamaño total de la tabla y otras métricas relevantes. Los resultados serán analizados para confirmar o refutar la hipótesis planteada.

2.2. Implementación

La implementación del algoritmo de hashing perfecto fue realizada en **C++20**, haciendo uso de conceptos modernos como punteros inteligentes y multithreading via pool thread para optimizar el rendimiento. La estructura principal está compuesta por varias clases que se dividen en módulos bien definidos:

1. **Clase `UHash`:**
   - Esta clase define una función de hashing universal $h(x) = a \cdot x + b \bmod p$, donde a y b son constantes aleatorias y p número primo $2^{61} - 1$.

- El operador `()` permite evaluar $h(x)$ sobre un valor x, garantizando resultados positivos mediante un ajuste modular.

2. **Clase `PerfectHashTable`:**
   - Es la estructura principal, que gestiona tanto la tabla de primer nivel como las tablas hash perfectas de segundo nivel.
   - Utiliza un enfoque jerárquico: primero distribuye los elementos entre las celdas principales utilizando una función hash h. Luego, crea una tabla hash perfecta de segundo nivel para cada celda, cuyo tamaño se determina según $c \cdot b_i^2$, donde b_i es la cantidad de elementos asignados a la celda i.
   - Registra métricas clave, como el tiempo de construcción total, el tiempo por celda, y el espacio utilizado.
   - **Rechaza** la creación de tablas en casos donde teóricamente es imposible construir la tabla es decir
     1. $c < 1$ en este caso una tabla segundaría sin elemento puede terminar con tamaño 0.
     2. $k < c$ en este caso el algoritmo en la tabla principal no terminara, pues la suma de elementos en caso perfecto es n.
     3. $\frac{k}{c} < 2$ se considero en experimentos independientes que el hashing bajo ~ 1.9 era muy complejo de conseguir en tiempo. tomando una muestra de $10k$ funciones de hashing se tuvo min 1.873 con el promedio mucho mayor de lo que se esperaría, consideremos este es un compromiso valido pues difícilmente puede tener un uso practico al ser muy inflexibles.

3. **Clase `innerPerfectHashTable`:**
   - Representa una tabla hash perfecta de segundo nivel, asociada a un conjunto específico de elementos.
   - Implementa una estrategia tipo Las Vegas para garantizar que no haya colisiones, generando funciones hash hasta encontrar una válida.
   - Los datos se almacenan en un vector de punteros únicos, donde cada celda contiene como máximo un elemento.

El código se realizo con diseño modular en mente y se hizo uso de funciones auxiliares como `delta` para ayudar a la claridad y mantencion del mismo, el tiempo se midio en el constructor de `PerfectHashTable` y La memoria se calculo multiplicando el tamaño del vector de las estructuras de `innerPerfectHashTable` por su tamaño, mas el tamaño del vector de elementos final en estas, también se distinguió el tamaño y tiempo de construcción de la tabla principal y secundaria.

3. Resultados

El código fuente adjunto se compiló usando gcc versión 14.2.1 y se ejecutó en ArchLinux (kernel ZEN 6.12.6) en un laptop Asus Zephyrus G14 con 40 GB @ 3600 Mhz de RAM en una configuración de 8 + 32, y procesador Ryzen 9 5900HS que contiene 8 núcleos físicos y 16 virtuales, limitado a 3000 MHz con 16 MB L3, 512 KB L2 y 32KB L1 de caché respectivamente. La última versión implementada se mantuvo corriendo por cerca de 1 hora y media.

3.1. Experimento 1

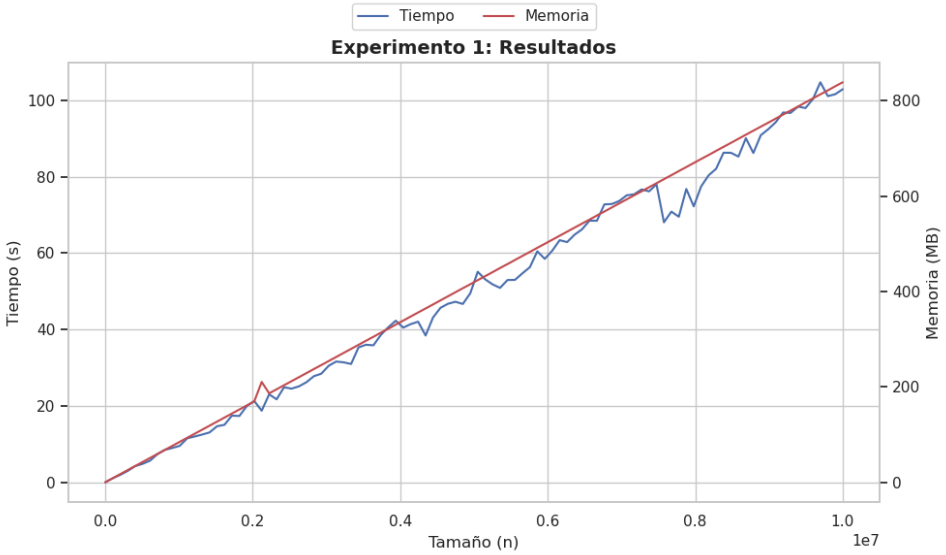

Figura 1: Resultados de la construccion de tablas con c y k fijos.

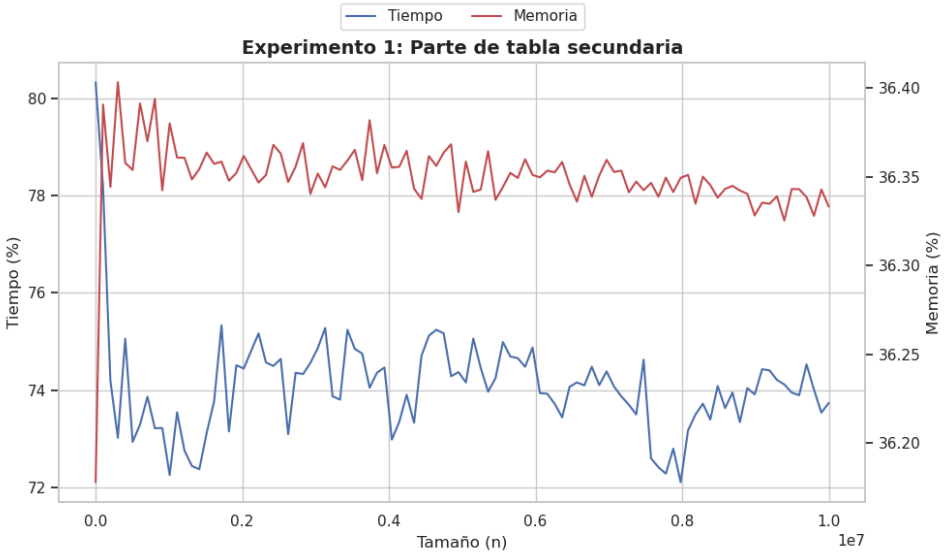

Figura 2: porcentaje responsable de tiempo y memoria de las tablas de segundo nivel en el experimento 1.

3.2. Experimento 2

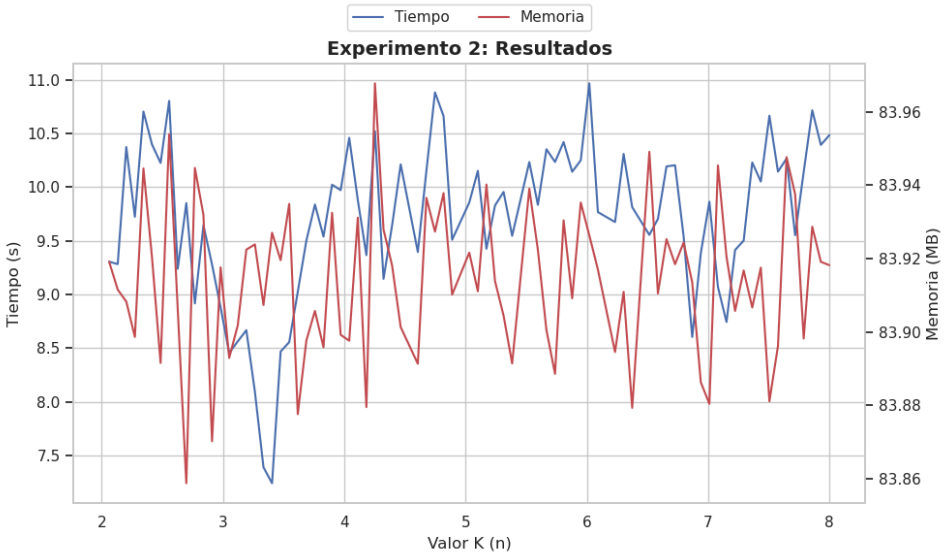

Figura 3: Resultados de la construccion de tablas con c y n fijos.

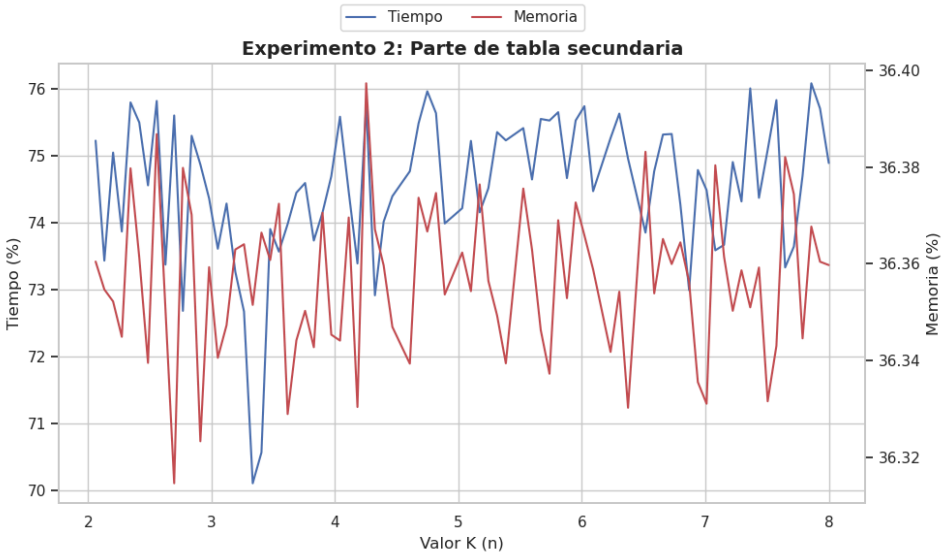

Figura 4: porcentaje responsable de tiempo y memoria de las tablas de segundo nivel en el experimento 2.

3.3. Experimento 3

Figura 5: Resultados de la construccion de tablas con k y n fijos.

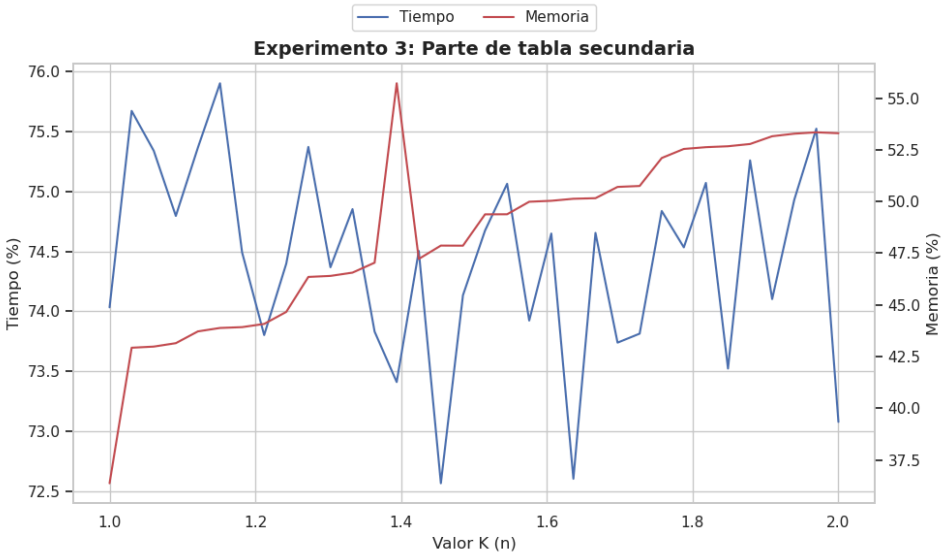

Figura 6: porcentaje responsable de tiempo y memoria de las tablas de segundo nivel en el experimento 3.

3.4. Experimento 4

3.4.1. Tiempo

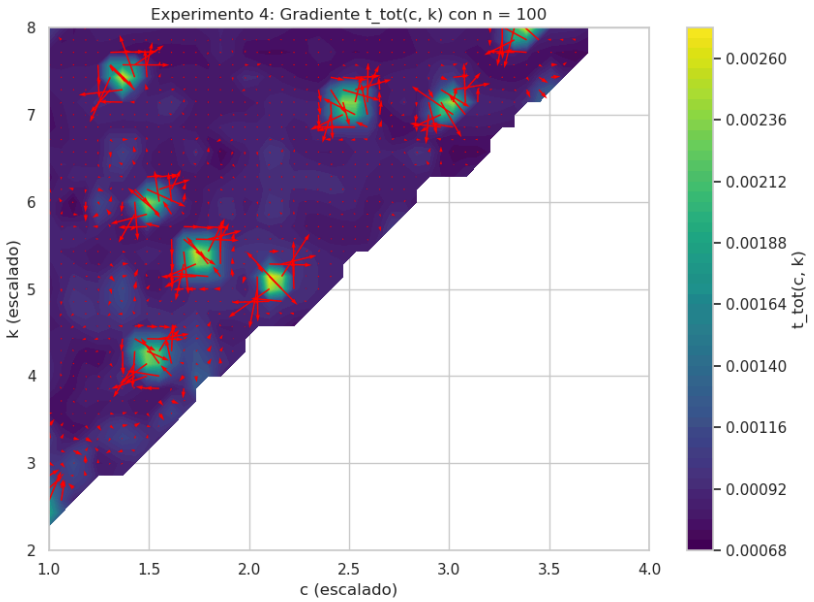

Figura 7: Resultados de tiempo sobre la construccion de tablas variando c y k, para el n 1.

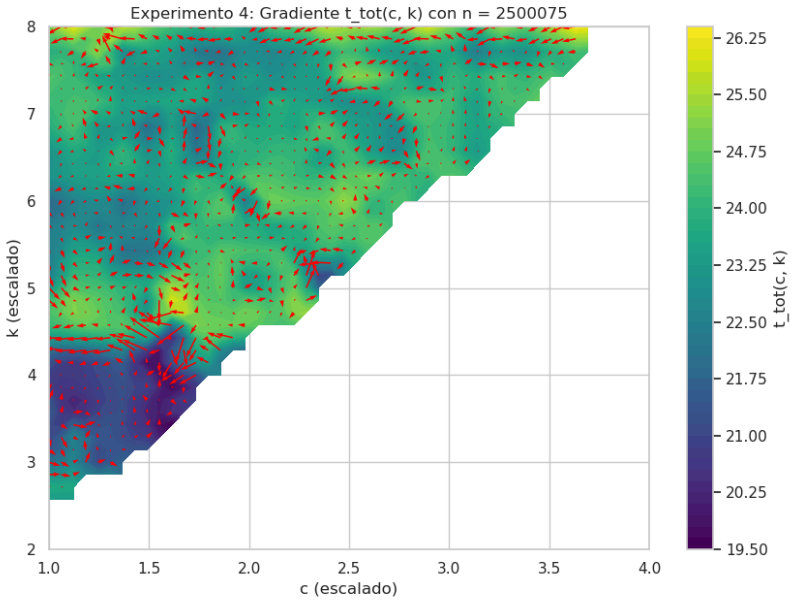

Figura 8: Resultados de tiempo sobre la construccion de tablas variando c y k, para el n 2.

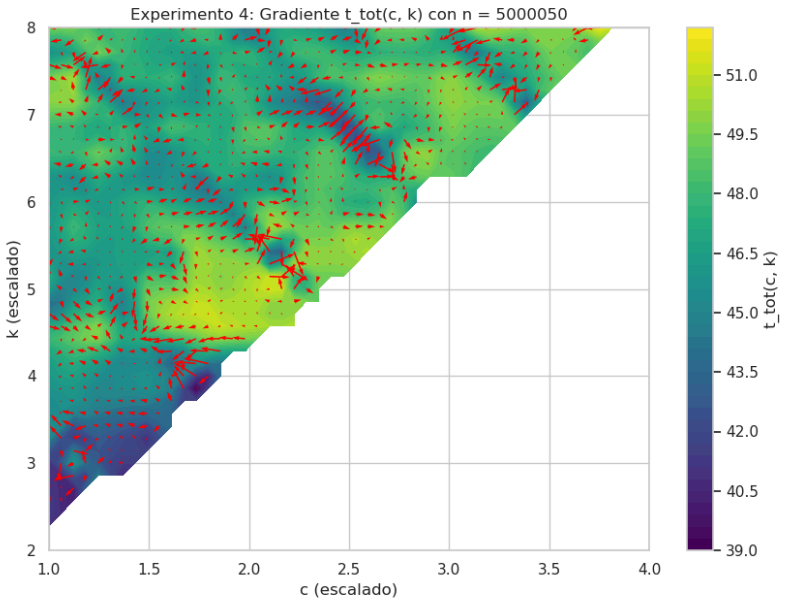

Figura 9: Resultados de tiempo sobre la construccion de tablas variando c y k, para el n 3.

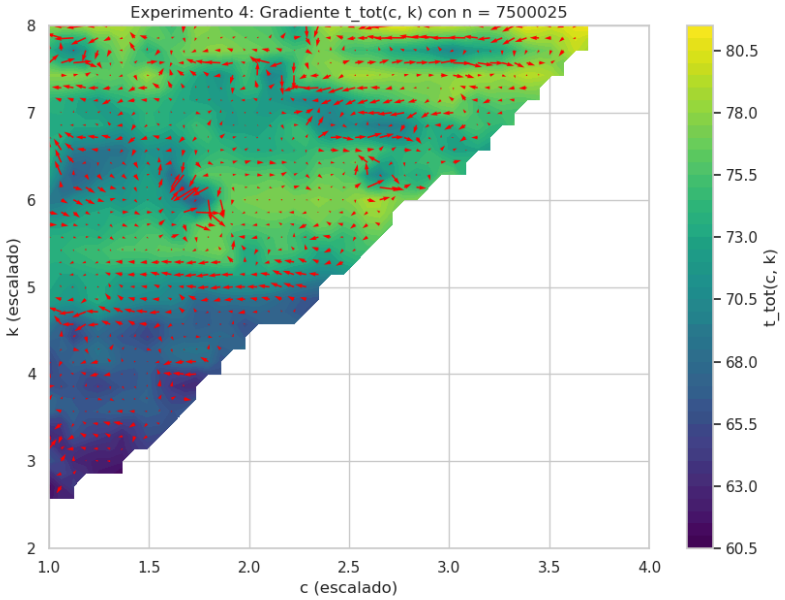

Figura 10: Resultados de tiempo sobre la construccion de tablas variando c y k, para el n 4.

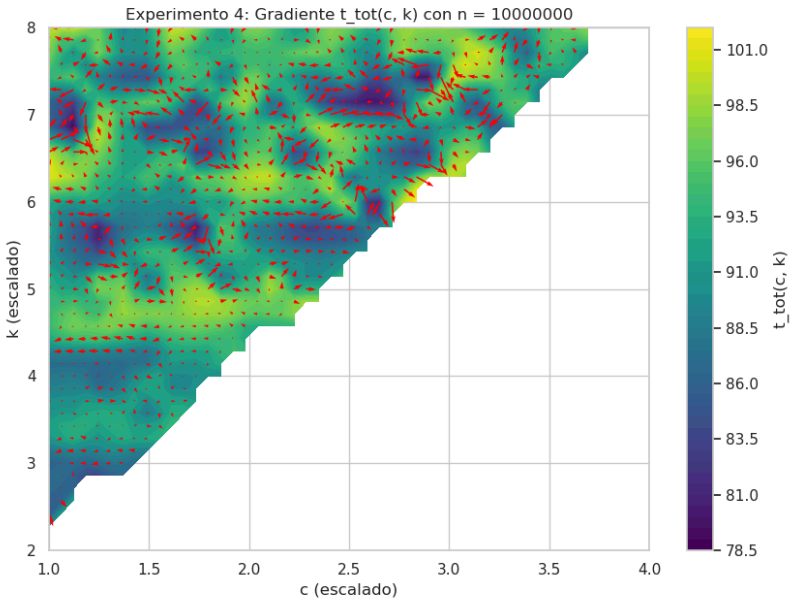

Figura 11: Resultados de tiempo sobre la construccion de tablas variando c y k, para el n 5.

3.4.2. Espacio

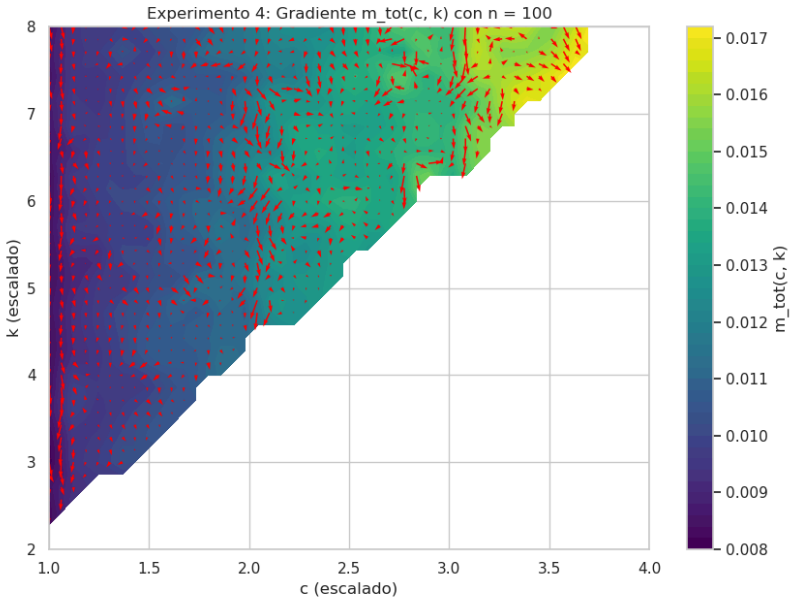

Figura 12: Resultados de espacio sobre la construccion de tablas variando c y k, para el n 1.

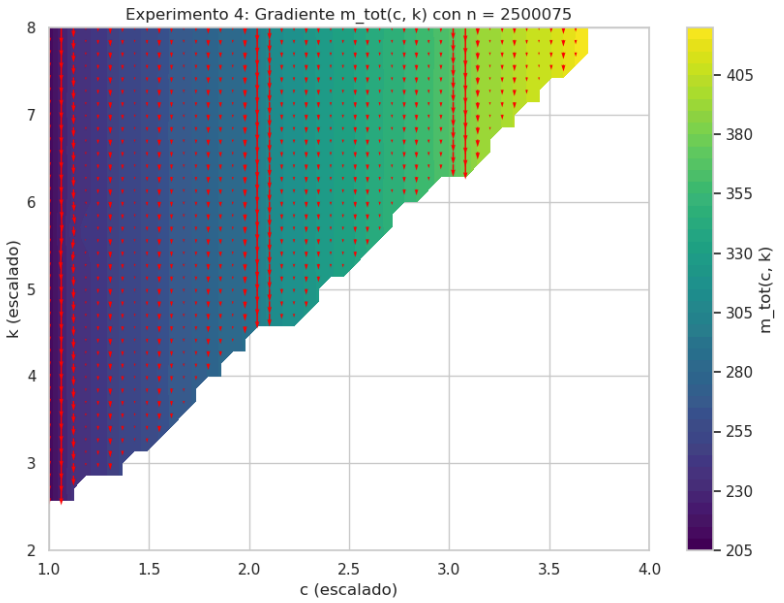

Figura 13: Resultados de espacio sobre la construccion de tablas variando c y k, para el n 2.

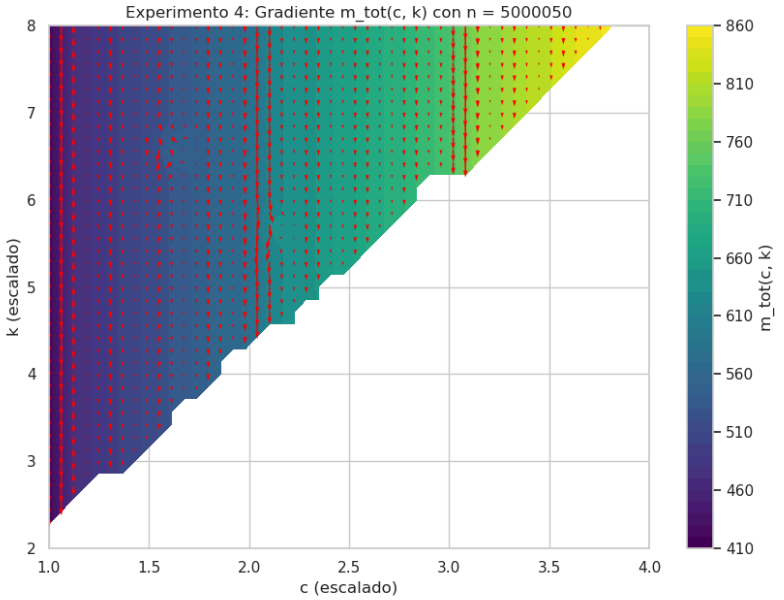

Figura 14: Resultados de espacio sobre la construccion de tablas variando c y k, para el n 3.

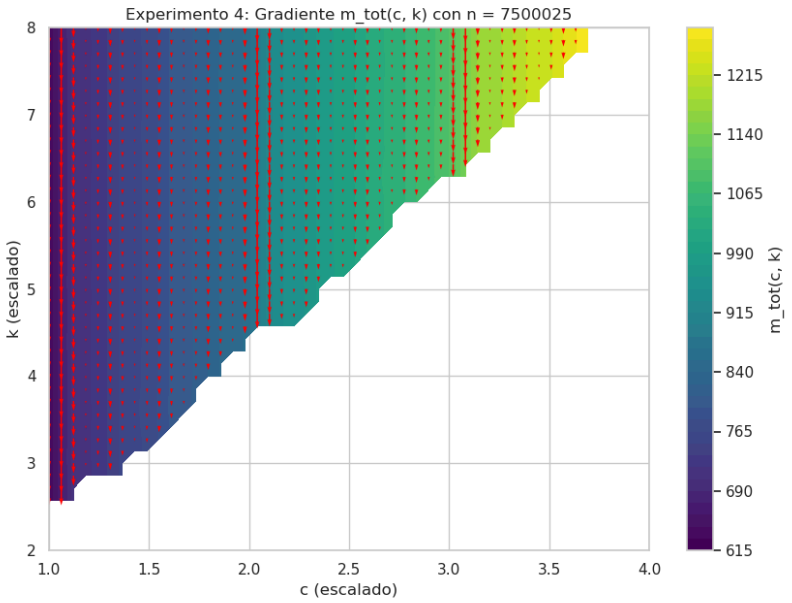

Figura 15: Resultados de espacio sobre la construccion de tablas variando c y k, para el n 4.

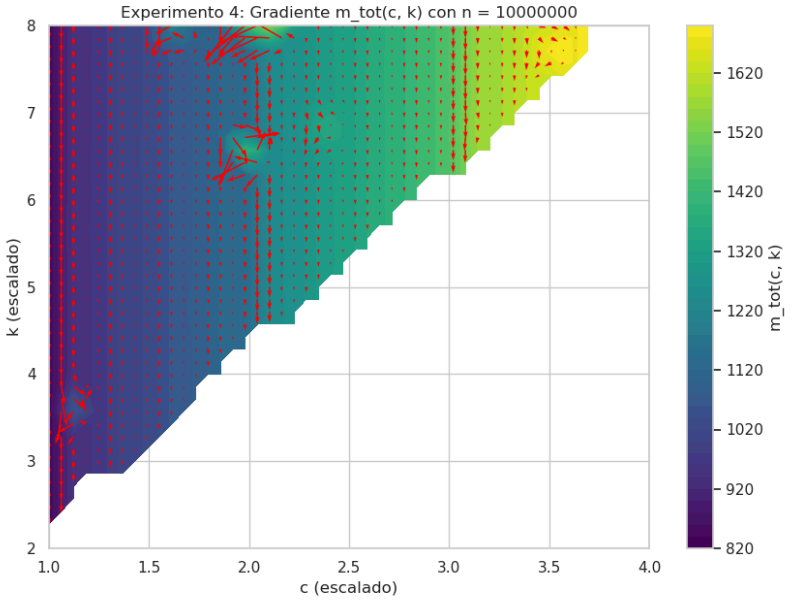

Figura 16: Resultados de espacio sobre la construccion de tablas variando c y k, para el n 5.

3.4.3. Score

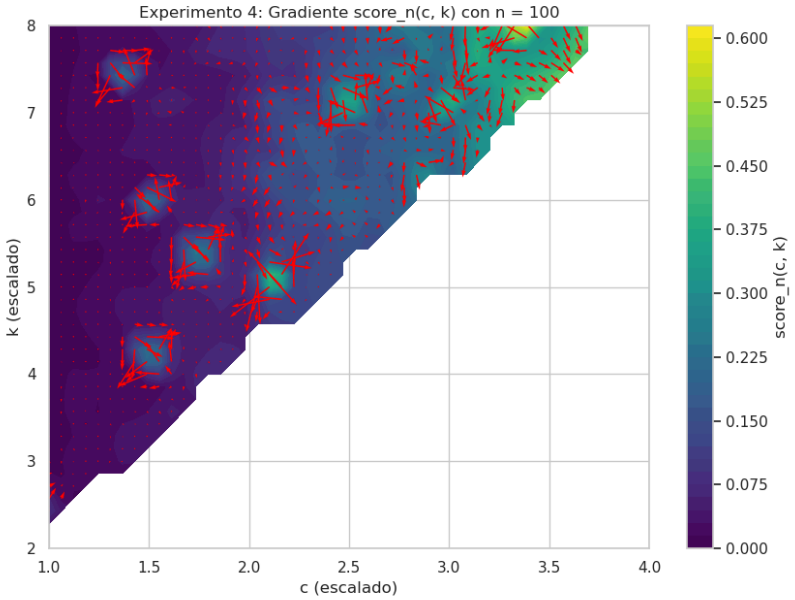

Figura 17: Resultados de score sobre la construccion de tablas variando c y k, para el n 1.

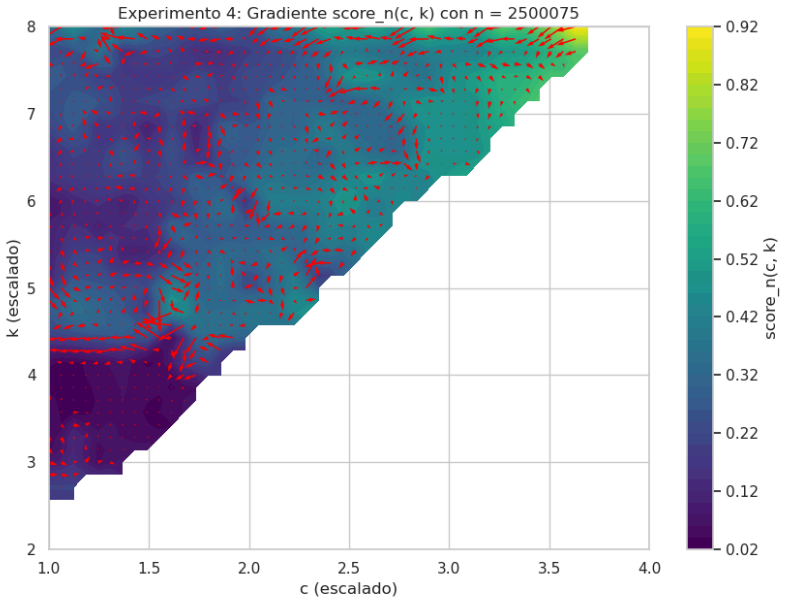

Figura 18: Resultados de score sobre la construccion de tablas variando c y k, para el n 2.

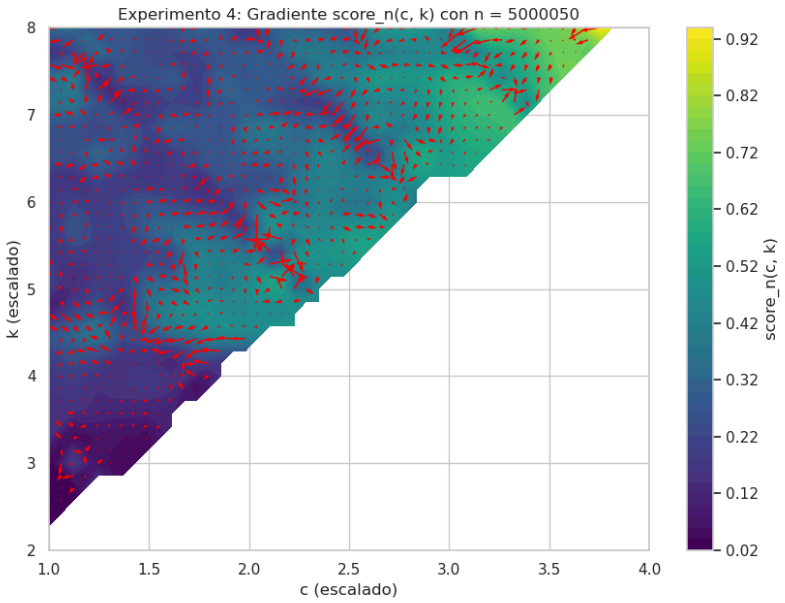

Figura 19: Resultados de score sobre la construccion de tablas variando c y k, para el n 3.

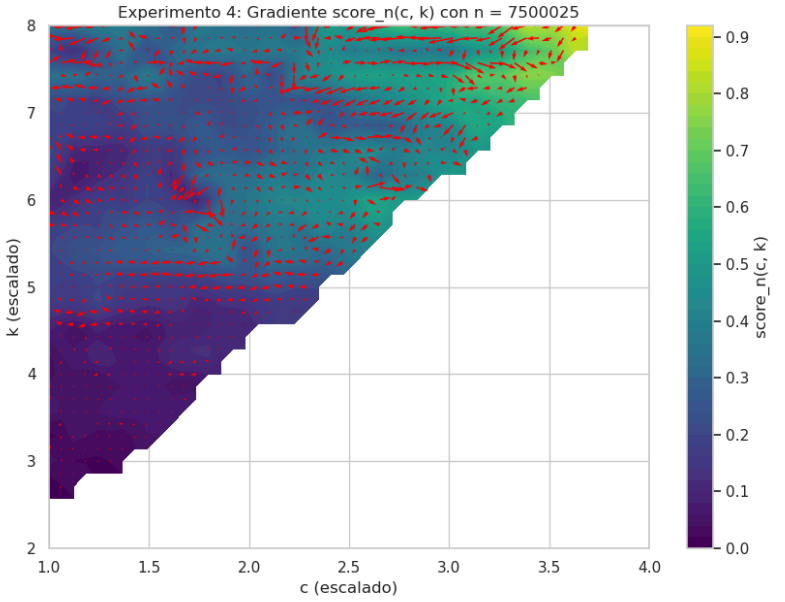

Figura 20: Resultados de score sobre la construccion de tablas variando c y k, para el n 4.

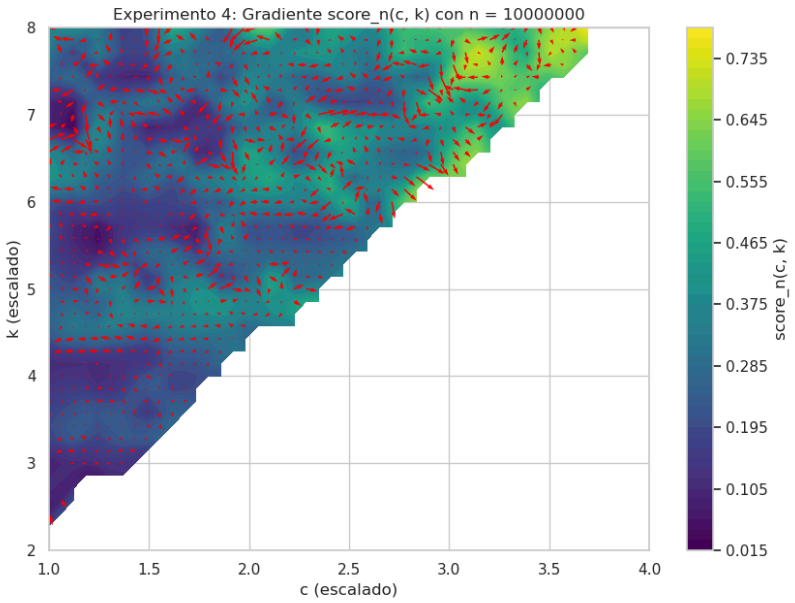

Figura 21: Resultados de score sobre la construccion de tablas variando c y k, para el n 5.

4. Análisis de resultados

1. **Conclusiones sobre los experimentos 1;**

- Se logra ver la linealidad teórica que predice su análisis
- Se tiene gran parte del tiempo en construcción de las tablas de segundo nivel
- Se tiene relativamente poco espacio, asignado en las tablas de segundo nivel

Encontramos que el tercer hallazgo, es bastante sorprendente, no obstante creemos que esto puede ser debido a como se realizo el calculo de la memoria en la estructura.

2. **Analizando los resultados del experimento 2;**

- En general encontramos que la variación de K fue
- Las variaciones en tiempo y memoria son relativamente pequeñas, lo atribuimos a la naturaleza del aleatoria de la estructura.

Podemos sacar en conclusión que K al solo controlar la restricción de creación de la primera tabla, siendo una "tolerancia" tiene poco efecto en el tiempo y complejidad espacial final.

3. **Revisando lo obtenido en el experimento 3;**

- Los datos son significativamente menos erráticos
- La relación en espacio en memoria es fuerte, indicando que un mayor c aumenta la memoria de la tabla final.
- Para c menores, existe una pequeña correlación indirecta con el tiempo

Nos sorprendió la gran relevancia aparente que tiene c sobre el experimento en especial con la memoria, no obstante, no es realmente sorprendente tomando en cuenta que tiene un impacto directo en la alocacion de memoria en las tablas segundarías

4. **Ponderando los resultados del experimento 4;**

Cabe destacar, que se intento hacer una nube de datos con los resultados, usando el gradiente entre las diferencias de tiempo y memoria, no obstante, sus resultados eran confusos y no realmente útiles, por lo que se opto a lo presentado en este informe, fijándonos en los niveles de un n especifico

- 1. **Tiempo**
  - Podemos encontrar una relación multinivel, exceptuando para $n = 100$ de un mayor tiempo para combinaciones de c, k mayores, con especial cuidado a mayores k dado un c
- 2. **Memoria**
  - En memoria hubo una clara tendencia en todos los niveles marcada, se tiene mayor uso de memoria a mayor c esto es indicativo directo, pues c es escalar directo de la memoria en tabla segundaría, en k no se logra tener efecto aparente
- 3. **Score**
  - El score, como medida a encontrar el mejor intercambio entre memoria y tiempo (menor es mejor) indico, de forma multinivel y clara que las combinaciones de c, k menores claramente son mejores.

5. Conclusión

En conclusión, los experimentos realizados sobre el algoritmo de hashing perfecto permitieron confirmar parcialmente la hipótesis planteada. Como se esperaba, k no presentan efectos independientes en el rendimiento de la estructura, aunque se pudo observar que c tiene un impacto más pronunciado en el uso de memoria, k actúa principalmente como una tolerancia que afecta la viabilidad de la construcción, pero no su eficiencia general.

Por otro lado, se observó que para configuraciones donde c y k son bajos, el tiempo y el uso de memoria se mantienen más equilibrados, lo que confirma que estos parámetros deben ser ajustados de manera conjunta para obtener un desempeño óptimo. Además, el diseño jerárquico de la estructura, particularmente la construcción de tablas de segundo nivel, contribuye de manera significativa al tiempo de construcción y al uso de espacio, lo que resalta su relevancia dentro del proceso, dado estas observaciones en conjunto con la puntuación podemos afirmar un rango de $c = 1, k = 3$ es bastante bueno en el intercambio de tiempo y espacio.

Estos resultados sugieren que, si bien la implementación es robusta dentro de los rangos evaluados, existe un compromiso entre los recursos utilizados y los valores de c y k, y justifica la necesidad de considerar en mayor detalle estos parámetros según el contexto de aplicación. Adicionalmente, la métrica de puntaje propuesta demostró ser útil para identificar configuraciones más eficientes en términos de tiempo y memoria.

Una mejora futura para este experimento sería evaluar el rendimiento del algoritmo con un mayor rango de valores de n, lo que permitiría explorar su escalabilidad en contextos más exigentes. También sería interesante analizar su comportamiento frente a distribuciones de datos más diversas y complejas, así como estudiar estrategias para optimizar la construcción de tablas de segundo nivel, reduciendo su impacto en los recursos generales.