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

Informe Tarea 2

27 de mayo de 2024
Monserrat Díaz
21626545

Motivación

Esta tarea tiene como principal motivación la aplicación de técnicas un poco más avanzadas para resolver problemas complejos. Estos problemas están diseñados para que se mejore la comprensión de conceptos claves como el hashing y el backtracking, que son fundamentales en la informática e incluso en otras áreas como la gestión de datos, la bioinformática o la logística. También es muy importante comprender que la eficiencia en la resolución de estos problemas también es crucial en aplicaciones reales donde la rapidez es determinante.

Informe

Parte 1: Rolling DNA Codes
Esta parte de la tarea consistía en encontrar posiciones de secuencias de ADN dentro de una secuencia mayor utilizando algoritmos como el de Rabin-Karp. Para resolverlo, se implementó el algoritmo R-K que utiliza hashing rodante para comparar las secuencias de manera eficiente. La complejidad de este algoritmo es $O(N \times Q)$, ya que recalcula el hash en cada paso de la ventana deslizante. En nuestro caso, ya que se involucra una variable M que corresponde al largo de las Q cadenas que debíamos buscar en la secuencia de largo N, la complejidad es un poco distinta. En el caso particular del código realizado se tiene que la complejidad total para una consulta sería $O(N \times M)$, por lo que la complejidad total del programa sería $O(N \times M \times Q)$. Sé que el enunciado pedía complejidad $O(N \times Q)$, pero realmente no supe cómo implementarlo para esa complejidad.

Usar rolling hash permite recalcular el hash eficientemente sin necesidad de reescanning completo de la subcadena, lo que optimiza el uso de memoria y tiempo.

Parte 2: Rompecabezas Cósmico de Entregas
Este problema consisitía en optimizar el espacio de almacenamiento de una nave, representado mediante una matriz. La idea era colocar paquetes de manera que no se superpongan y se utilice el espacio disponible. La estrategia utilizada fue backtracking de manera recursiva para explorar todas las posibles posiciones de los paquetes dentro de la matriz. A pesar de que el backtracking puede ser costoso en términos de tiempo, nos permitía asegurar que se encontrara una solución válida.

En cada nivel de la recursión se intenta colocar el paquete actual en todas las posiciones válidas dentro de la matriz. Al encontrar una posición válida, coloca el paquete y procede a intentar colocar el siguiente paquete. Si en algún punto debaja de ser posible colocar un paquete, se deshace ese movimiento y se prueba con la siguiente posición.

Para manejar la memoria de forma eficiente, se utiliza una estructura de datos para representar la bodega y los paquetes. Así que cada vez que se coloca o se retira un paquete, se actualiza la matriz de la bodega. Por lo mismo, es crucial liberar la memoria asignada a las matrices dinámicamente al finalizar el proceso para evitar fugas de memoria.

Parte 3: Patrones de duplicación de Bender
Esta parte final de la tarea consistía en identificar subárboles específicos dentro de un árbol

binario perfecto dado, donde cada nodo estaba representado por un caracter. Para optimizar la búsqueda, se utiliza comparación directa de subárboles combinada con el cálculo de alturas. El árbol principal y los subárboles se representan como cadenas de caracteres y se realiza una búsqueda exhaustiva para encontrar todas las ocurrencias de los subárboles en el árbol principal.

A grandes rasgos se calcula, en primer lugar, la altura del árbol binario perfecto. Luego se realiza una búsqueda en amplitud (BSF) para comparar cada nodo del árbol principal con la raíz del subárbol. Para cada coincidencia potencial, se verifica recursivamente si todos los nodos hijos coinciden con la estructura del subárbol y finalmente se registran todas las posiciones donde se encuentra coincidencia completa del subárbol.

Para evitar colisiones y asegurar que los resultados estén ordenados se utiliza comparación directa en lugar de hashing para asegurar que no haya "falsos positivos"debido a las colisiones de hash. Y los resultados se almacenan en una estructura de datos ordenada lo que genera que no haya problemas al imprimirlos en orden creciente.

Finalmente, como se utiliza una comparación directa de subárboles combinada con el cálculo de alturas para optimizar la búsqueda. La complejidad de la solución es $O(N \times K)$, donde N es el tamaño del subárbol y K el número de consultas.

Es muy importante para esta parte que se manejen correctamente los límites del árbol para evitar accesos fuera de rango y garantizar una gestión eficiente de la memoria durante la búsqueda.

Conclusión

Con la realización de este informe se presenta la implementación y explicación de los tres problemas descritos en el enunciado de esta tarea. Mediante su realización se pudo aplicar y profundizar en diferentes algoritmos, demostrando la importancia de la eficiencia en la resolución de problemas. En un futuro, se podrían explorar mejoras adicionales, como quizás técnicas más avanzadas de hashing o la optimización de backtracking con algoritmos codiciosos por ejemplo, lo que permitiría mejorar más la eficiencia de las soluciones.

PD: No sé si la misma persona que revisa el informe revisa el código, pero si ese es el caso, con fecha de hoy 27/05 me acabo de enterar de que habían cambiado los tests de la parte 2 del enunciado... Tengo otras entregas para el nuevo plazo que dieron así que no podré mejorar esa parte :(Sólo por si se puede tener en consideración ya que lo lógico sería que los tests no cambien tan radicalmente (pasé de tener todos los tests buenos a tener 4/11 buenos... Si no se puede hacer nada, será nomás porque no alcanzo a cambiar mi lógica u_u)

PD2: En la parte 3, los tests corren en menos de 10 segundos a excepción de la parte hard, en ese caso, al utilizar el otro compilador (-O3) se reduce considerablemente el tiempo.

Referencias

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
- Rabin, M. O., & Karp, R. M. (1987). *Efficient randomized pattern-matching algorithms.* IBM Journal of Research and Development.
- `https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-12-181`
- `https://www.scmr.com/article/logistics_data_management_focus_on_the_foundation`
- `https://www.transmetrics.ai/blog/data-management-in-logistics/`
- Paper de Rabin-Karp linkeado en el enunciado de la tarea.
- Documentación del curso IIC2133 – Estructuras de Datos y Algoritmos, Pontificia Universidad Católica de Chile.