Documento de Diseño Tarea 2

Luca Santino D'Aloy .

Rut: 23.346.588-7

Parte 1:

Para esta parte de la tarea, usar tablas de hash es una buena opción porque se intenta buscar un match entre un string dado y un subtring de una hebra entregada, esto requiere buscar todos los substrings de la hebra que sean del mismo tamaño y compararlos con el string de la consulta, entonces intuitivamente, tener los substrings de cada tamaño guardados puede ayudar mucho al problema, ya que si no estuviesen guardados, habría que hacer mas loops y la complejidad del código sería muy grande.

La tabla de hash funcionaría de la siguiente manera: Primero se guardan substrings de los tamaños necesitados según la consulta, entonces el tamaño sería el índice, y luego para cada tamaño se crea otra tabla de hash donde se guarda una característica que permite encontrar de manera rápida un match. En este caso se guardaría la cantidad de letras distintas en el substring de la hebra principal, entonces cuando estoy buscando un match en una consulta, los dos requisitos mínimos es que sea un string del mismo tamaño y con la misma cantidad de letras distintas, y eso último lo haría la función de hash, se le entrega un string y lo guarda en la tabla de hash de el respectivo largo, con la cantidad de letras distintas como índice.

Si la tabla fuese uniforme, el match se encontraría de manera instantánea, ya que no tendría que recorrer mas de una opción al entrar en los índices de una tabla de hash, entonces las soluciones se encontrarían de manera instantánea.

El probing podría ocurrir cuando hay una hebra madre particular con muy pocas letras distintas, donde se generan muchas colisiones, en este caso la función de hash no sería muy útil, aunque igual encontraría solución más rápido que hacerlo a fuerza bruta.

La complejidad de tiempo sería O(N) en el caso promedio.

Parte 2

Backtracking es buena opción porque está hecho para problemas donde la única solución es buscar todos los casos posibles, y este es uno de esos casos. Este problema se resuelve con prueba y error, y backtracking es una forma para que el programa recorra de manera más inteligente y puede llegar a una solución antes que con fuerza bruta.

El motivo por el que backtracking es mejor, es porque a la fuerza bruta se tiene que rellenar todo el tablero y al final ver si está correcto, con backtracking, si se realiza un movimiento

que invalida la solución, no sigue por ese camino, sino que vuelve para atrás y prueba otro, entonces al final recorre muchas menos posibilidades.

La complejidad sin backtracking es de O(8^{N^2}), y esta sigue siendo la complejidad con backtracking en el peor de los casos, sin embargo, en el caso promedio con backtracking, la complejidad se reduce a O(N).

Las podas usadas fueron las siguientes:

En primer lugar revisar si el movimiento que se quiere hacer es válido con dos condicionales, uno revisa si el movimiento está dentro del tablero y se quiere mover a un cuadrado vacio y además, que el movimiento que se quiere hacer no venga predeterminado en el input. Y el otro condicional revisa si se quiere hacer un movimiento a un lugar donde está destinado a hacerse ese movimiento según el input, y si no se cumple ninguna de esas dos condiciones, se rechaza el movimiento que se quiere hacer. Esto permite descartar muchas posibilidades. Y por último, se revisa que si se completa una fila o columna, esta sume 260, y si no lo hace, entonces el algoritmo vuelve hacia atrás. De esta forma se revisan todas las condiciones necesarias y si alguna no se cumple, se descarta inmediatamente ese camino.