



PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2133 — Estructuras de Datos y Algoritmos

2021 - 2

Tarea 2

Fecha de entrega código e informe: Jueves 4 de Noviembre del 2021.

Información General

Esta tarea tiene por objetivo el aplicar capacidades de análisis respecto a problemas de backtracking y tablas de hash. En lo particular la tarea se divide en 2 problemas independientes entre sí, que serán evaluados de forma independiente (En la sección de I/O se explica más a detalle como funcionará esto). Como normas generales, gran parte de la dificultad de la tarea está centrada en descubrir y desarrollar buenas funciones de hash y buenas heurísticas. Es por esto que se recomienda que comiencen la tarea realizando su documento de diseño, modelando bien el problema, y no directamente en el código.

Parte 1 - Tablas de Hash (50%)

Objetivos

- Modelar un problema y desarrollar una solución eficiente usando funciones de hash.
- Analizar distintas técnicas de hash en la eficiencia de una solución.
- Documentar el diseño de la solución de un problema algorítmico.

Introducción

Estás sentado frente a Petter Potter mientras comparten una DCCerveza de Hidromiel celebrando tu gran trabajo filtrando imágenes mágicas cuando se les acerca Ram Weasley, y les cuenta la noticia. ¡Las criaturas infectadas rondan el departamento! ¿Que el mundo mágico de DCCWarts es muy sobrenatural? Por si la magia no era suficiente, zombies han llegado a DCCWarts. Así es, pero como no pueden distinguir entre infectados y personas normales, Peter Potter (sabiendo que eres una persona letrada en C y que pasaste Biocel para Criaturas con honores) te pide ayuda para poder encontrar mutaciones en las hebras de ADN y así poder deshacerse de todas las criaturas infectadas.



Figura 1: Zombies invaden DCCWarts

Problema

Primero, se te entregará un string de texto (ADN zombie como $hebra_1$). Esto es con la forma $ABC A \dots$ (Notar que el ADN zombie, posee 26 posibles bases nitrogenadas $[A \dots Z]$ y se comporta diferente al ADN de las criaturas no infectadas).

Luego se entrega una secuencia de bases nitrogenadas $hebra_2$ con la forma $ZDEZ \dots$ tal que $longitud(hebra_1) \geq longitud(hebra_2)$. Tu misión es determinar si dicho fragmento se encuentra en la hebra original, y en caso de estarlo, determinar el índice de inicio y de término **del primer match que encuentren**.

Sin embargo, la hebra a encontrar no es exactamente igual a la hebra original, si no que lo que nos interesa es la estructura de esta. Por ejemplo:

$$hebra_1 = AABCAABC C D E$$

$$hebra_2 = Z D D A Z$$

Como se pueden fijar, la letra Z no aparece en ningún lado dentro de $hebra_1$, lo que nos interesa es la estructura y el **orden** de la $hebra$.

Por ejemplo, si usáramos otra notación, todas las siguientes $hebra_2$ se considerarían equivalentes:

$$hebra_2 = Z D D A Z = A B B R A = D K K L D = \dots$$

Es decir, nos interesa la **posición** de cada carácter de la $hebra$ respecto a los demás caracteres de esta. En primer ejemplo, el programa ha de ser capaz de identificar que se puede *mappear* desde la $hebra_1$ hacia la $hebra_2$ realizando $Z = C$, $D = A$ y $A = B$. Algo así:

$$\begin{array}{cccccccccccc} A & A & B & C & A & A & B & C & C & D & E \\ Z & D & D & A & Z & & & & & & \end{array}$$

Por lo tanto retornaría que la hebra buscada se encuentra entre las posiciones 3 y 7.

Condiciones del Problema

La $hebra_1$ original será entregada como un texto extenso, sin embargo cada problema no será solo una consulta, si no que pueden llegar a ser cientos (Entre 100 a 999), cada una con su $hebra_2$ correspondiente. Es por esto que se recomienda profundamente modelar el problema utilizando la intuición de tablas de hash. Para que esto sea correcto y para lograr el tiempo de ejecución, se solicita que en tu modelación consideres una función de hash que cumpla con las características de *incrementalidad*.

Ejecución

Para esta parte del problema, se entregará un archivo de input con la hebra original, seguido con una lista de consultas de la forma:

Archivo Input

```
T XXXXXXXX..  
M  
N XXXXXXXX..  
N XXXXXXXX..  
N XXXXXXXX..  
...  
...
```

En la primera línea, se te entregará un entero T que indica el número de caracteres de la hebra original, seguido por los caracteres de esta. Esto seguido por una línea con el entero M el número de consultas a realizar. Hacia abajo estarán las M consultas, con el formato que se indica arriba, donde N es el número de caracteres de la hebra, seguido por sus caracteres.

Archivo Output

El archivo output de tu programa debe poseer la siguiente estructura:

```
START END
```

```
START END
```

```
...
```

Cada línea corresponde a una consulta, donde se indica la posición de inicio y la de término del primer *match* que encuentren con respecto a dicha consulta. Un ejemplo más claro es el siguiente:

```
2 3  
4 15  
53 60
```

Esto indicaría que la primera consulta se encuentra entre los índices 2 y 3, la segunda entre 4 y 15 y así sucesivamente.

Documento de Diseño

Además del código, se solicita un documento de diseño (las ponderaciones de cada parte son informadas al final) donde como mínimo se espera lo siguiente:

- Explicar por qué utilizar una *Tabla Hash* es buena opción.
- Presentar una *función de hash* que solucione el problema.
- Demostrar que la solución es incremental.
- Explicar qué beneficios traería que la *función de hash* fuera uniforme.
- Comentar condiciones y/o elementos de *resizing*, *probing*, u otros según corresponda.

Recordar que es un documento de diseño, no necesariamente ha de ser relacionado a tu implementación en C, si no que puede ser basado en modelaciones teóricas.

Parte 2 - Bakantracking (50%)

Objetivos

- Modelar un problema y encontrar una solución usando backtracking.
- Documentar y justificar las podas usadas en el backtracking.

Introducción

Además de eliminar a todas las criaturas infectadas, Ram te recuerda que deben detener la maldición que causaba todo esto. Por lo que se dirigen al bosque perdido hacia la fuente de infección. Camino al bosque se encuentran con un obstáculo: un gigante tablero de ajedrez mágico, pero con solo una pieza, un caballo. Además de ciertos números escritos en cada celda del tablero, encuentran una nota en el suelo que dice "*Solo cruzarán aquellos que resuelvan mi puzzle...*", seguido de 3 condiciones:

- El número en el tablero indica el n -ésimo movimiento del caballo.
- Pueden moverse solo en movimientos de caballo y deben recorrer todo el tablero.
- Cada fila y columna ha de sumar lo mismo.

Problema

El problema consiste en usar backtracking para arreglar un tablero de ajedrez mágico. Un tablero de ajedrez mágico es un tablero de ajedrez clásico (de 8 por 8) el cual posee 2 condiciones:

- Todo cuadrado está etiquetado con un número entre 1 y 64 tal que el n -ésimo cuadrado está a una movida de caballo¹ del $n+1$ -ésimo cuadrado
- Ademas de esto toda fila y columna suma lo mismo.

1	48	31	50	33	16	63	18	260
30	51	46	3	62	19	14	35	260
47	2	49	32	15	34	17	64	260
52	29	4	45	20	61	36	13	260
5	44	25	56	9	40	21	60	260
28	53	8	41	24	57	12	37	260
43	6	55	26	39	10	59	22	260
54	27	42	7	58	23	38	11	260
260	260	260	260	260	260	260	260	

Ejecución

Input

Tu tarea tiene que aceptar como input un tablero de ajedrez mágico que perdió algunas de sus etiquetas (por suerte el 1 no se perdió). Si un cuadrado tiene un 0 significa que perdió su etiqueta. Se muestra un ejemplo de input:

¹<https://en.wikipedia.org/wiki/Chess#Movement>

```
1 48 0 0 33 0 63 18
30 51 0 3 0 0 0 0
0 0 0 0 15 0 0 0
0 0 0 45 0 0 36 0
0 0 25 0 9 0 21 60
0 0 0 0 24 57 12 0
0 6 0 0 39 0 0 0
54 0 42 0 0 0 0 0
```

Output

El output tiene que ser el tablero de ajedrez mágico arreglado, por ejemplo:

```
1 48 31 50 33 16 63 18
30 51 46 3 62 19 14 35
47 2 49 32 15 34 17 64
52 29 4 45 20 61 36 13
5 44 25 56 9 40 21 60
28 53 8 41 24 57 12 37
43 6 55 26 39 10 59 22
54 27 42 7 58 23 38 11
```

Documento de Diseño

La sección de informe de esta parte de la tarea debe contener como mínimo:

- ¿Por qué backtracking es buena opción para resolver el problema?
- Comparación complejidades con y sin backtracking.
- Presentar y justificar las podas utilizadas.

Código Base y Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un ejecutable de nombre `pottermagic` que se ejecuta con el siguiente comando:

```
./pottermagic <A/B> <input> <output>
```

Donde A/B es la señal si es un problema de Hash (A) o de backtracking (B) respectivamente. Seguido por el input en formato `.txt` y el output en el mismo formato.

Código Base

Como código base, se entrega una sección que detecta qué tipo de problema se resolverá, la lectura de archivos ha de ser implementada por ti.

Recomendación de tus ayudantes

Las tareas requieren de mucha dedicación de tiempo generalmente, por lo que desde ya te recomendamos distribuir tu tiempo considerando los plazos definidos. Así mismo, te recomendamos fuertemente que antes de empezar a programar tu tarea, leas el enunciado y te dediques a entender de manera profunda lo que te pedimos. Una vez que hayas comprendido el enunciado, dedica el tiempo que sea necesario para la planificación y modelación de tu solución, para posteriormente poder programar de manera eficiente. Estos son consejos de tus ayudantes que te pueden ayudar a pasar el ramo :)

Uso de memoria

Parte de los objetivos de esta tarea, es que implementen en la práctica el *trade-off* entre memoria y tiempo, es por esto que independiente del test, tendrán como máximo 1.2 GB de memoria RAM disponible. Pueden revisar la memoria que utiliza su programa con el comando `htop`. Además, el servidor avisará en caso de superar el máximo permitido.

Eficiencia

Se solicita que el programa (tanto para la Parte 1 como la Parte 2), sea eficiente. No podrá tomar más de 10 segundos `user + sys`, pueden utilizar `time` seguido de el comando de ejecución de su programa para revisarlo.

Análisis

Deberás escribir un informe de análisis² (documento de diseño para ambas partes) donde se cubran los siguientes puntos:

- Los requisitos mínimos del documento de diseño de cada una de las secciones.
- Un análisis de complejidad tanto de tiempo como de espacio por cada problema de forma teórica (no explicar parte por parte del flujo).

Evaluación

La nota de tu tarea se descompone como se detalla a continuación:

- 60% a la nota de tu código: que retorne el output correcto. Para esto, tu programa será ejecutado con archivos de input de creciente dificultad, los cuales tendrán que ser ejecutados en menos de 10 segundos cada uno.
- 40% a la nota del documento de diseño.

Nimbus 2000? Faster than Snape confronted with shampoo (+X decimas a la nota de código)

Se darán décimas a las tareas más rápidas de la sección de hash y backtracking. Se tomarán las 15 mejores tareas y se asignarán desde 10 a 1 décimas a estas.

²en un máximo de 6 planas (imágenes no cuentan)

No Segfault allowed (+10% a la nota de código)

Por no tener leaks de memoria y errores, se asignará un 10% adicional a la nota de código, dividido en 5% por leaks y 5% por errores.

Its leviosa not leviosaa (+10% a la nota de informe)

A los mejores documentos de diseño se les asignará *hasta* un 10% de nota adicional debido a su excelente calidad, estas décimas serán asignadas a criterio de cada ayudante.

Entrega

Código: GIT - Rama master del repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Informe: SIDING - En el cuestionario correspondiente, en formato PDF. Sigue las instrucciones del cuestionario. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Atraso: A lo largo del semestre tendrás 4 días de gracia, los cuales podrás utilizar en caso de que no alcances a entregar alguna tarea en el tiempo indicado. Para esto, puedes usar un máximo de 2 días en una tarea, sin importar si tienes más días disponibles, y tendrás que avisar si quieres que se revise un commit posterior a la fecha de entrega en el formulario que se mandará el día siguiente. Cabe destacar que si entregas a las 00:01 hrs perderás un día en caso de llenar el formulario, y no será revisado ese commit en caso de que decidas no contestarlo. Por otro lado, si se te acaban los 4 días y entregas una tarea atrasada, entonces tendrás la calificación mínima **sin derecho a reclamo**. Cabe recalcar que el informe solo se puede realizar en un **máximo de 6 hojas**, de pasarse no podrán objetar por posibles descuentos.

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** por el alumno y **sin apoyo de terceros**. Se espera que los alumnos mantengan altos estándares de honestidad académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.

¡Éxito! :)

Y para quienes llegaron hasta acá...

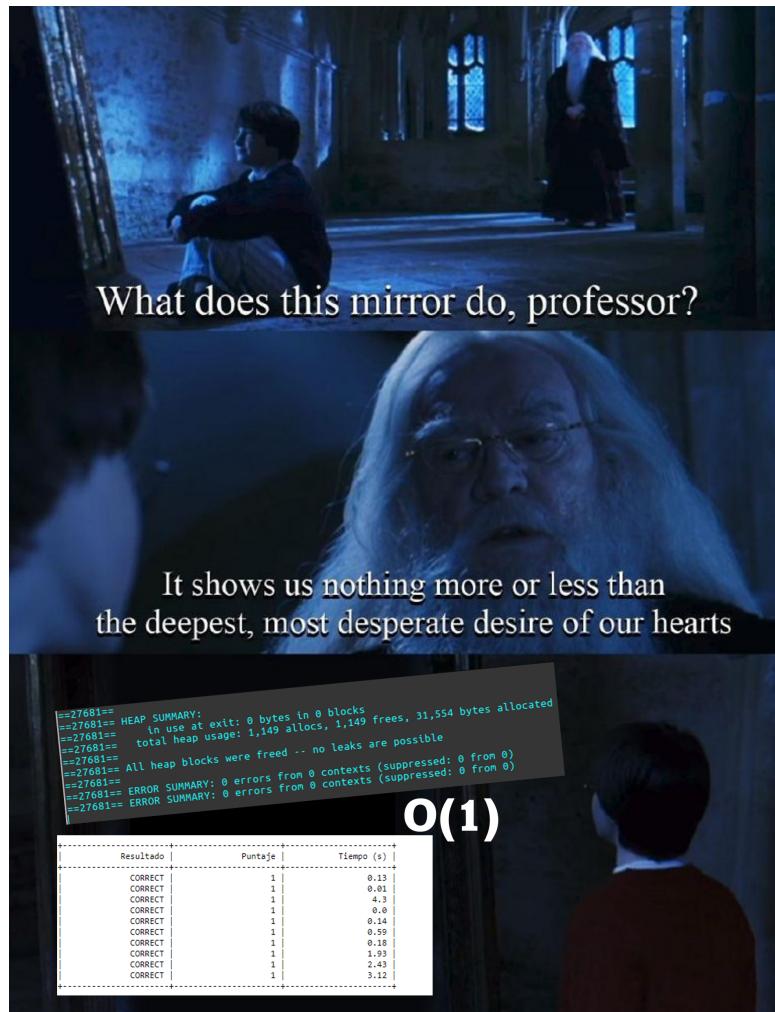


Figura 2: Meme gracioso