Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
2023 - 1

Tarea 2

Fecha de entrega código: 24 de Mayo del 2023 23:59 UTC-4
Link a generador de repos: Generar repo base

Objetivos

- Aplicar estrategias algorítmicas para resolver problemas NP-HARD
- Utilizar técnicas de tablas de hash para optimización de busqueda
- Utilizar backtracking para encontrar soluciones optimas

Parte 1:

Introducción: Árboles del Superespía Mate

Últimamente, Mate ha estado viendo muchas películas de espías encima de árboles (está difícil esta economía) luego de haber sido dejado en el altar por decir que su algoritmo de ordenación favorito es Bogo-Sort. Lamentablemente, para él, un día un espía enemigo estaba celoso de su destreza algorítmica y decidió secuestrar al amor de su vida: su cuenta de Netflix. Ahora, Mate se ve obligado a seguir a este rufián para recuperar su clave. Por lo tanto, ha solicitado tu ayuda, como el alumno estrella de Estructuras de Datos y Algoritmos que eres.

Durante tu misión, Mate te llama llorando porque el espía le ha enviado una foto de él viendo Netflix en el mejor árbol que ha visto Mate desde que aprendió a diferenciar entre un árbol y un cactus. Utilizando tus poderes de superespía, descubres que el árbol se encuentra en el bosque de Radiator Springs. Para ayudar a Mate, debes encontrar todos los subárboles que coincidan con el árbol donde se encuentra el espía y recuperar la clave de Netflix para que Mate pueda seguir viendo RuPaul's Drag Race y no tenga que ver las mismas películas de Marvel una y otra vez.

Figura 1: Mate sólo por amar a Bogo-Sort

Problema: Búsqueda del árbol del Super Espia Mate

Todos los árboles fueron almacenados como un árbol binario completo. Es decir, un árbol binario donde, dada su altura h, tendrá $n = 2^h - 1$ nodos. Denotaremos este árbol por Γ.

Lo particular del árbol Γ, es que solo almacenan un valor binario 1 o 0. Donde 0 indica que el nodo del árbol es negro y 1 si es que es blanco.

Para facilitar la visualización de ejemplos, diremos que un nodo con valor 0 corresponde a un nodo pintado y 1 corresponde a un nodo no pintado.

Por lo tanto, el siguiente árbol es un ejemplo de Γ.

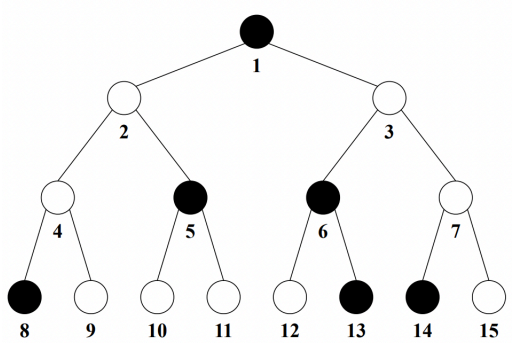

Figura 2: Γ con $h = 4$. Los nodos están numerados de 1 a n. Llamaremos a este número *el ID del nodo.*

Ahora, dado Γ con altura h_Γ. El problema a resolver será encontrar las ocurrencias de distintos árboles $\gamma_1, \gamma_2, \ldots, \gamma_k$ en Γ. Considerando qué $1 \leq h_i \leq h_\gamma \ \forall i \in \{1, \ldots, k\}$.
Definiremos como *ocurrencia* de γ_i sobre Γ a algún subárbol de Γ de altura h_i, cuyos nodos coinciden en color con los de γ. Por ejemplo

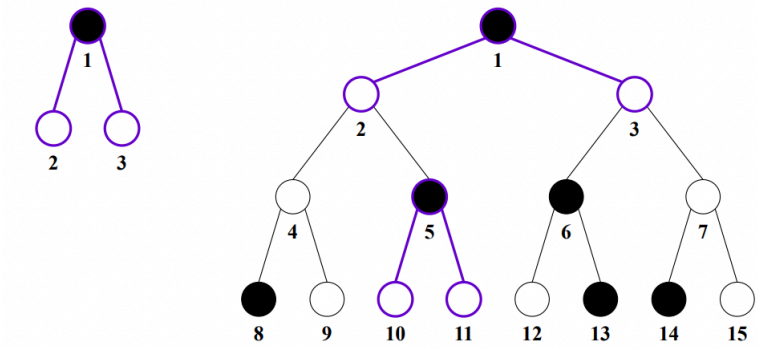

Figura 3: A la izquierda el árbol γ a buscar, con $h_\gamma = 2$. A la derecha, sus *ocurrencias* en Γ

Luego formalizando. Denotamos las ocurrencias en Γ de un subárbol γ por el *ID del nodo* de Γ que coincide con la raíz de γ. En el ejemplo anterior, las ocurrencias de γ en Γ serían entonces los ID 1 y 5. Además, es posible que las ocurrencias de γ se traslapen, como muestra el siguiente ejemplo:

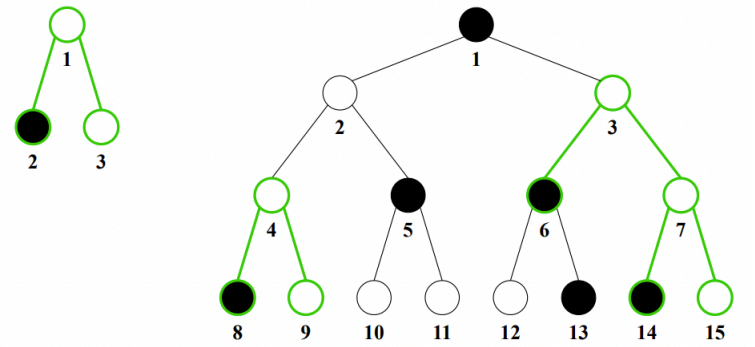

Figura 4: Las ocurrencias de γ en Γ serian los ID 3, 4 y 7. Notar que 3 y 7 se traslapan

Además, un árbol γ puede **no** aparecer en Γ. Por ejemplo:

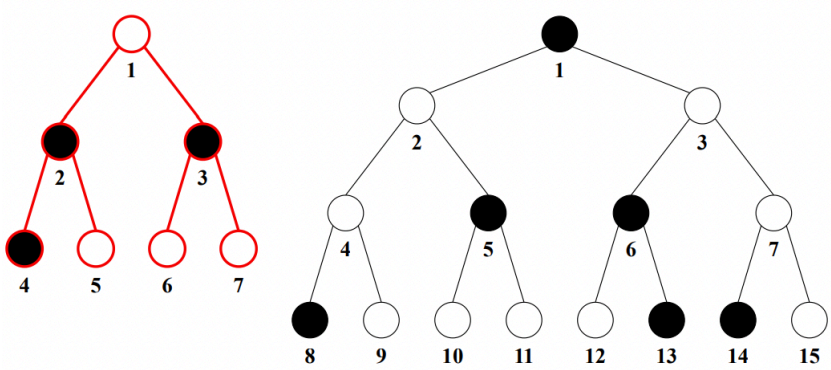

Figura 5: En este caso diremos que la ocurrencia es el ID -1, ya que no existe

Finalmente, tu tarea será escribir un programa en el lenguaje C que, dada una descripción de Γ, pueda encontrar las ocurrencias de K árboles γ. Para esto **deberás** utilizar tablas y funciones de hash. Donde es tu responsabilidad diseñar la función adecuada al problema que pueda codificar de forma eficiente los datos entregados. Se recomienda fuertemente la utilización de una función de hash incremental[1]

También tendrás que implementar una *Hash Table* apropiada, analizando los parámetros asociados a esta con el fin de disminuir la memoria utilizada y el tiempo de búsqueda. tomando en consideración los criterios de factor de carga, tipo de direccionamiento, entre otros.

[1]Significa que a partir del hash de N, sea *rapido* calcular el de N+1

Ejecución

Tu programa se debe poder compilar con el comando `make` y generar un ejecutable de nombre `mateconhuesillo`. Que se debe ejecutar con el siguiente comando

```
./mateconhuesillo input.txt output.txt
```

Donde `input` es el archivo inicial con el árbol original y una serie de consultas. Y `output` es la ruta al archivo de output. En esta tarea, el código base solo contendrá un archivo `Makefile` y un archivo `main.c` con lo básico de C, pero que no contendrá lectura de input.

Input

El input se estructura de la siguiente forma

- Una línea que describe el árbol Γ. Esto es, un número n que indica la cantidad de nodos. Seguido por el color de los n nodos, ordenando por índice. Negro = 0, Blanco = 1
- Una línea indicando el número K de consultas
- K líneas que contienen un árbol γ_i a consultar. Este árbol se describe de la misma forma que Γ en la primera línea

Por ejemplo, para el input Γ de ejemplo, y las 3 consultas mostradas:

```
15 0 1 1 1 0 0 1 0 1 1 1 1 0 0 1
3
3 0 1 1
3 1 0 1
7 1 0 0 0 1 1 1
```

La primera línea indica que es un árbol de 15 nodos y luego la descripción del árbol. Luego se indica el número de consultas. Finalmente cada una de las consultas. Se subirán tests adicionales a los del repositorio base durante los siguientes días. Podrán encontrar el link directo desde el repositorio del curso

Output

El output debe consistir de K líneas, cada una con los ID de las ocurrencias del árbol γ correspondiente en el input. Estos ID se imprimen ordenados de menor a mayor. Por ejemplo, para el input anterior de 3 consultas, el resultado es el siguiente:

```
1 5
3 4 7
-1
```

Evaluación Parte A

La nota de esta parte, a diferencia de tareas anteriores, aceptará respuestas parcialmente correctas. Y lo evaluaremos por % de consultas respondidas correctamente. Una consulta correcta corresponde a que se reportaron correctamente **todos** los ID's respuesta de la misma. Luego, en un test donde se obtuvo X % de las consultas correctas, el puntaje es dado por:

- 0 si $X < 75\,\%$
- 0.5 si $75 \leq X < 85$
- 0.7 si $85 \leq X < 95$
- 0.8 si $95 \leq X < 1$
- 1 si $X = 1$

Parte 2 - Bakantracking (50 %)

Objetivos

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

Introducción

Ahora, ya que ayudaste a mate a recuperar su clave de netflix, el espía enemigo se enoja aún más y decide quitarle a Mate algo mucho más valioso que su netflix, su estilo.
El espía se mete a la casa de mate y lo noquea.. Unas horas después Mate despierta desorientado y amarrado en un lugar misterioso. En ese momento se enciende la luz y se da cuenta que está en el taller de ramón, y en una esquina se encuentra el espía enemigo. El enemigo quiere limpiar a mate y sacarle ese maravilloso color oxidado que le entrega belleza al mundo. Sin embargo, ramón se entera de que están ocupando su taller sin su permiso, por lo cual te llama a ti para que lo ayudes a detener este atentado al flow natural de Mate.
Camino al taller se encuentran con un obstáculo puesto por el espía: un gigante tablero de ajedrez, muy peculiar, el cual contiene ciertos números escritos en cada celda, y posee solo una pieza, un caballo. Ustedes intentar pasarlo pero las ruedas de Ramón deslizan sobre el tablero, sin embargo, existe un camino alternativo el cual solamente será liberado si resuelves el acertijo detrás del tablero. Para resolver el puzzle debes considerar las siguientes 3 condiciones necesarias:

- 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.

Figura 6: Mate limpio sin oxidación de amor

Problema

El problema consiste en usar backtracking para arreglar un tablero de ajedrez (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
- Además 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	

Código Base y Ejecución

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

```
./matechess <input><output>
```

Input y Output

Input

Tu tarea tiene que aceptar como input un tablero de ajedrez 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 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 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
```

Evaluación General

La nota de tu tarea es calculada a partir de tests. Separando entre la Parte 1 y 2 de la tarea

- 60 % Parte 1: Hashing
- 40 % Parte 2: Backtracking

Competencia (bonus)

Dado que se busca medir eficiencia, se realizara una competencia premiando así a las 5 mejores tareas (En base a su tiempo total de ejecución) con los siguientes premios:

- 1º Lugar: 5 décimas + cupón de atraso adicional
- 2º lugar: 5 décimas
- 3º lugar: 3 décimas
- 4° lugar: 2 décimas

Importante: para poder clasificar al leaderboard, se debe poseer 100 % de correctitud en los tests
Los primeros tres lugares además aparecerán en un leaderboard público para la posterioridad del curso
Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos y utilizar menos de 1 GB de ram[2]. De lo contrario, recibirás 0 puntos en ese test.

[2]Puedes revisarlo con el comando htop o con el servidor

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 :)

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.

Política de atraso: La política de atraso del curso considera dos criterios:

- Utilización de **cupones**[3] de atraso. Donde un cupón te proporciona 24 horas adicionales para entregar tu tarea sin penalización a tu nota
- Entrega atrasada sin cupón donde se aplica un descuento *suave*[4]. Calculada de la forma

$$N_f = min(70 - 7 \cdot d, N_o)$$

  Donde d es la cantidad de días atrasados **con un máximo de 4 días**[5], N_f la nota final y N_o la nota obtenida en la tarea

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.

[3] Recuerda que solo tienes 2 cupones para todo el semestre
[4] Es un descuento a la nota máxima posible
[5] Luego de eso la nota máxima obtenible es un 1.0