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

IIC2133 — Estructuras de Datos y Algoritmos
2020 - 2

Tarea 2

Fecha de entrega código e informe: Domingo 25 de Octubre a las 08:00 (AM)

Objetivos

- Modelar un problema y encontrar 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: En busca de las memorias perdidas

Habiendo resuelto el sináptico problema de las colisiones [1], Guido se ha concentrado en estudiar el *plot device* por excelencia, la amnesia. Sin embargo, antes de poder *recordar* sus clases de *storytelling* [2], debe realizar un viaje de `self.descubrimiento`.

Tras volver de un profundo (y caro) viaje espiritual por una reconstrucción virtual del Sudeste Galáctico, se ha dado cuenta que lo de estructurar recuerdos en una nebulosa mental no es tan buena idea como Carl Jung supuso.[3]

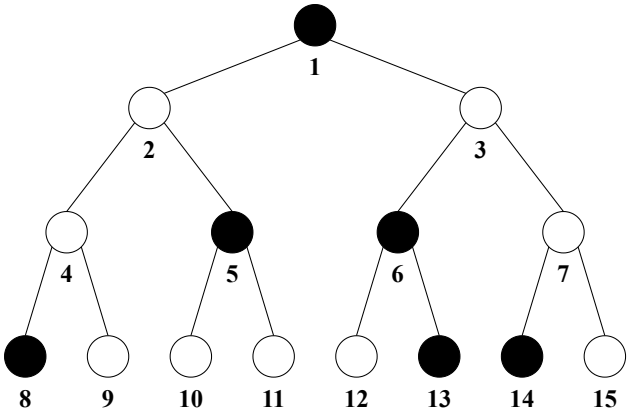
Varias propuestas vinieron rápidamente a su digital mente, imágenes de grandes sistemas de archivos[4], palacios mentales y bibliotecas Borgeanas fueron consideradas. Sin embargo, inspirado por una extraña fascinación con los árboles, decidió organizar sus preciadas memorias en una estructura de datos afín.

Por esto, ha decidido contactar a su experto en árboles favorito, por lo que debes utilizar tus conocimientos en Psycho-C para ayudarlo a recuperar sus recuerdos.

[1] Y aceptado que nunca podrá cumplir su sueño de diseñar un buen lenguaje de programación
[2] O entender por qué sigue viendo `SEGMENTATION FAULT` en sus pesadillas
[3] Poner referencia chistosa aquí
[4] Entre ellos no estaba FAT32

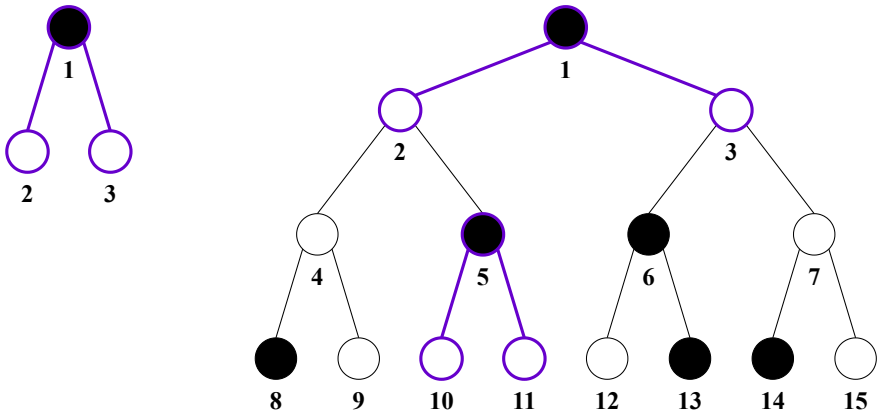
Problema: Memory tree

Los recuerdos de Guido están representados como un árbol binario completo, es decir, un árbol binario cuyas hojas se encuentran todas a la misma profundidad. Denotaremos este árbol con la letra Σ, y su altura con la letra H. Por definición de árbol completo, el número de nodos de Σ es $n = 2^H - 1$. Estos nodos de Σ tienen asignado un color, el cual puede ser blanco o negro, los cuales representan recuerdos positivos o negativos respectivamente. A continuación, un ejemplo:



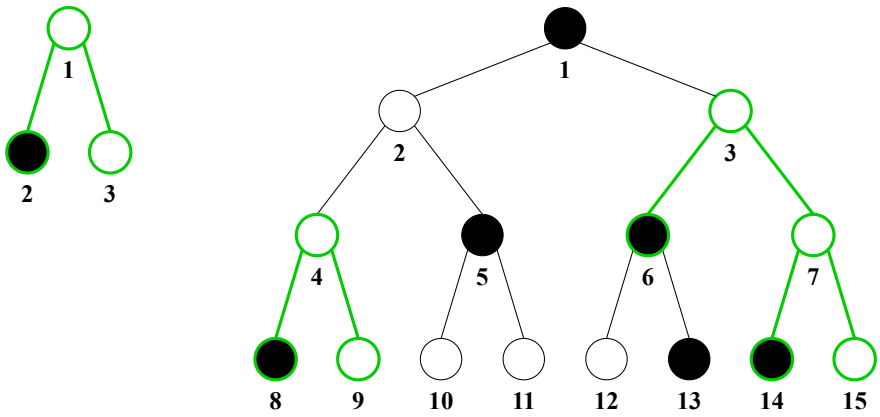
Σ con $H = 4$. Los nodos están enumerados de 1 a n. Llamamos a este número el ID del nodo.

El problema a resolver consiste en encontrar dónde se localizan en Σ las ocurrencias de otro árbol binario completo, al que llamaremos t, de altura h ($1 \leq h \leq H$). Definimos como ocurrencia de t en Σ a algún subárbol de Σ, de altura h, cuyos nodos coinciden en color con los de t. A continuación un ejemplo:



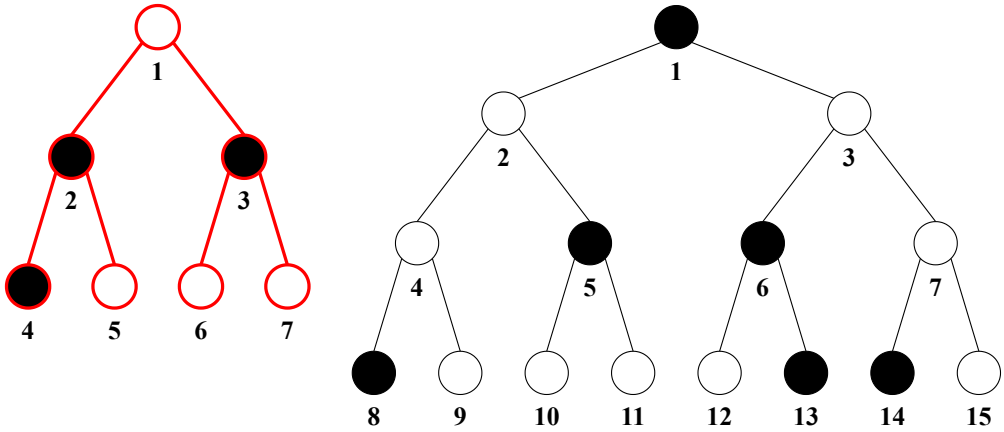
A la izquierda el árbol t a buscar, con $h = 2$. A la derecha, sus ocurrencias en t

Denotamos las ocurrencias en Σ de un subárbol t por el ID del nodo de Σ que coincide con la raíz de t. En el ejemplo anterior, las ocurrencias de t en Σ serían entonces los índices 1 y 5. Es posible que las ocurrencias de t se traslapen, como en el siguiente ejemplo:



Las ocurrencias de t en Σ serían los ID 3, 4 y 7. Notar que los subárboles 3 y 7 comparten un nodo.

Cabe destacar que t no necesariamente aparece en Σ, como en el siguiente ejemplo.



En este caso, decimos que la ocurrencia de t en Σ es el ID -1, el cual no existe.

Deberás escribir un programa en **C** que, dada la descripción de Σ, sea capaz de buscar las ocurrencias en Σ de N árboles t. Para esta labor, se recomienda **fuertemente** la utilización de hash; es tu deber identificar una función adecuada al problema, pudiendo codificar eficientemente los datos entregados. También deberás implementar una *Hash Table* apropiada, ajustando correctamente los parámetros asociados a esta, tales como factor de carga, direccionamiento, función de *resize*, función de *probing*, entre otros.

Ejecución

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

```
./remember input output
```

Donde `input` es la ruta a un archivo con el árbol inicial y una serie de consultas, y `output` es la ruta al archivo de output. En esta ocasión, se les entregará el repositorio de la tarea pero este no contará con código base más allá de la `Makefile` y el archivo `main.c`. Puedes hacer uso del código de lectura de archivo de tareas pasadas.

Input

El input está estructurado como sigue:

- Una linea conteniendo la descripción de Σ. Esto es, un número n, correspondiente a la cantidad de nodos de Σ, seguido de los colores de los n nodos, ordenados por índice. Negro = 0, Blanco = 1.
- Una linea conteniendo el número N de consultas.
- N líneas cada una a un árbol t a consultar. Este árbol está descrito en la misma manera que Σ en la primera línea.

Por ejemplo, el input para el Σ de ejemplo y las 3 consultas que hicimos:

```
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, seguida por los elementos de este. Luego se indica que existirán tres consultas; dos árboles con $h = 2$ y un árbol con $h = 3$.

Output

Tu output deberá consistir de N líneas, cada una con los ID de las ocurrencias del árbol t correspondiente en el input. Estos ID se imprimen ordenados de menor a mayor. Por ejemplo, para el input anterior, que tiene 3 consultas, se obtiene el siguiente resultado:

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

Evaluación

La nota de tu tarea se separa en 2 partes: nota de código y una nota especial asociada a un documento de diseño. A continuación se detalla cada uno de estas partes.

Documento de diseño: 50 % de la nota

Deberás escribir un documento de diseño **simple** donde expliques cómo resolver el problema usando *hashing*. Se recomienda **fuertemente** escribir este informe antes de escribir el código en **C**. En particular, se espera que:

- Expliques cómo usar un diccionario para resolver este problema de manera eficiente.
- Diseñes una función de hash uniforme para un árbol σ de altura h, que pueda calcularse de manera incremental.
- Analices la distribución de tu función de hash.
- Justifiques la elección de los parámetros de la tabla, tales como factor de carga, direccionamiento, función de *resize*, función de *probing*, entre otros.

Código: 50 % de la nota

Tu código será evaluado con tests de dificultad creciente. Estos tests están separados en 3 categorías:

- Easy: Para debugear manualmente. No serán evaluados.
- Normal: Usados para testear la correctitud de tu implementación. Serán evaluados.
- Hard: Usados para testear la eficiencia de tu implementación. Serán evaluados.

Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos. De lo contrario, recibirás 0 puntos en ese test.

Entrega

Código: GIT - Repositorio asignado. Se entrega a más tardar el día de entrega a las 8:00 AM 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 8:00 AM hora de Chile continental.

Bonus

Los siguientes bonus solo aplican si tu nota correspondiente es mayor o igual a 4.

Manejo de memoria perfecto: +5 % a la nota de código

Recibirás este bonus si `valgrind` reporta que tu programa no tiene ni leaks ni errores de memoria en los tests que logres resolver.

Diseño impecable: +5 % a la nota de diseño

Recibirás este bonus, a criterio del corrector, si tu documento de diseño está especialmente bien estructurado y contiene un análisis sólido. Para este bonus el diseño y análisis de tu función de hash es de especial importancia.

Hash goes brrr: Sonic Racing (+X décimas a la nota final de la tarea)

Se otorgará un bonus de hasta 10 décimas (a la nota final de la tarea) a las 20 soluciones que corran más rápido. 10 décimas a los dos primeros lugares, 9 al tercer y cuarto puesto, etc.

Para participar en esta carrera, tu programa deberá resolver el test `Lunatic`, de elevada dificultad, correctamente y en menos de 10 segundos. Este test no será considerado dentro de la evaluación, solo permite la entrada a la gran carrera.