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

IIC2133 – Estructuras de Datos y Algoritmos

2024 - 2

Tarea 2

Fecha de entrega código: 28 de Octubre, 23:59 Chile continental

Link a generador de repos: CLICK AQUI

Objetivos

- Implementar y aplicar árboles de búsqueda para la resolución eficiente de problemas.
- Utilizar técnicas de hashing para lograr soluciones de búsqueda y almacenamiento eficientes.
- Optimizar algoritmos de búsqueda y almacenamiento para cumplir con restricciones de complejidad temporal y espacial especificadas.

Introducción

¡Ring Ring!

Has recibido una llamada inusual. Debido a tus habilidades excepcionales como programador, una agencia secreta ha decidido contactarte. Te ofrecen la oportunidad de unirte a su equipo y utilizar tus conocimientos para llevar a cabo una serie de misiones especiales. ¿Aceptarás el desafío y pondrás tus talentos al servicio de una causa secreta?

Parte 1: Operación P.E.G.A.S.U.S.

Como agente de la Elite Penguin Force (EPF), tienes acceso a una vasta red de información secreta. En esta misión, se te encomienda la tarea de desarrollar un software especializado, P.E.G.A.S.U.S. (Penguin Elite Geospatial Analysis & Surveillance Unified System), para monitorear la ubicación y las características de los pingüinos en la isla.

Con P.E.G.A.S.U.S., podrás realizar distintos tipos de consultas para analizar la distribución y el estado de los pingüinos, y deberás asegurarte de que los resultados sean eficientes y precisos. Tu misión es crítica para la seguridad y el control de la isla, y el éxito de la EPF depende de la calidad de tus algoritmos.

Problema

Primero recibirás una línea con un entero N, indicando la cantidad de pingüinos. Luego recibirás N líneas que representarán pingüinos, en donde cada línea tendrá los 5 números separados por un espacio que representarán las siguientes características:

- `id`: Identificador del pingüino.
- `ejeX`: La posición en el eje X del plano.
- `ejeY`: La posición en el eje Y del plano.
- `edad`: Edad del pingüino.
- `peso`: Peso del pingüino.

Luego recibirás un número E, seguido de E líneas con comandos, que representarán los eventos solicitados.

1.1 - Eventos con Árboles Simples

`SEARCH-BY-AGE {age}`

Retorna los ids de todos los pingüinos cuya edad sea igual a `age`. El resultado debe ordenarse por id.

output

```
SEARCH-BY-AGE {age}
{id_1} - {id_2} - ... - {id_n}
```

`SEARCH-BY-WEIGHT-RANGE {minWeight} {maxWeight}`

Retorna todos los pingüinos cuyo peso se encuentre entre `minWeight` y `maxWeight` (inclusive), ordenados de menor a mayor. En caso de empate, ordenar por id.

output

```
SEARCH-BY-WEIGHT-RANGE {minWeight} {maxWeight}
{id_1} - {id_2} - ... - {id_n}
```

1.2 - Eventos con Árboles Complejos

Tip: Te recomendamos investigar y usar árboles como Range-tree o KD-tree para estas consultas.

`IN-RECTANGLE {x1} {y1} {x2} {y2}`

Retorna todos los pingüinos que estén dentro del rectángulo definido por los vértices opuestos (x_1, y_1) y (x_2, y_2), incluyendo los límites del rectángulo. El resultado debe ordenarse primero por el valor de x y, en caso de empate, por el valor de y. Siempre $x_1 < x_2$ y $y_1 < y_2$.

output

```
IN-RECTANGLE {x1} {y1} {x2} {y2}
{id_1} - {id_2} - ... - {id_n}
```

`IN-CIRCLE {x} {y} {r}`

Retorna todos los pingüinos que estén dentro del círculo con centro en (x, y) radio r, incluyendo los que se encuentren exactamente en el perímetro del círculo. El resultado debe ordenarse primero por el valor de `x`, y, en caso de empate, por el valor de `y`.

output

```
IN-CIRCLE {x} {y} {r}
{id_1} - {id_2} - ... - {id_n}
```

Consideraciones Generales

- Los pingüinos no cambiarán su posición ni atributos.
- Todos los valores de coordenadas, edad y peso son números enteros positivos y pueden ser contenidos dentro de un `int`.
- No habrá más de un pingüino en la misma posición.
- No se entregará una complejidad esperada, sin embargo se debe cumplir con los tiempos de ejecución mencionados en la sección de Evaluación.

BONUS: +5 décimas

Investiga sobre Fractional Cascading e impleméntalo para resolver los eventos de la parte 1.2. Además, responde la pregunta extra en el informe (ver sección del informe para más detalles). Ten en cuenta que no se otorgarán bonificaciones parciales; para recibir las décimas, la implementación debe ser correcta y estar claramente explicada en el informe. Si la implementación está mal documentada, es poco clara o no cumple con la complejidad esperada, no se otorgará el bonus. Más adelante se subirá un formulario para avisar de la participación en el bonus.

Ejecución

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

```
./pegasus input output
```

donde `input` será un archivo con los eventos a simular y `output` el archivo donde se guardarán los resultados.

En caso de querer correr el programa para ver los leaks de memoria utilizando valgrind deberás utilizar el siguiente comando:

```
valgrind ./pegasus input output
```

Tu tarea será ejecutada con archivos de creciente dificultad, asignando puntaje a cada una de estas ejecuciones que tenga un output igual al esperado. A continuación detallaremos los archivos de input y output.

Input

Primero recibirás una línea con un entero N, indicando la cantidad de pingüinos. Luego recibirás N lineas que representarán pingüinos, cada linea tendrá los 5 números separados por un espacio. Luego recibirás un número E, seguido de E lineas con comandos, que representarán los eventos solicitados.

Como ejemplo tenemos el siguiente input:

input

```
5
4 2 8 4 8
6 4 6 2 3
7 2 5 9 9
0 0 3 5 5
1 1 0 7 5
4
SEARCH-BY-WEIGHT-RANGE 5 9
SEARCH-BY-AGE 7
IN-CIRCLE 2 9 1
IN-RECTANGLE 4 5 9 10
```

Output

output

```
SEARCH-BY-WEIGHT-RANGE 5 9
0 - 1 - 4 - 7
SEARCH-BY-AGE 7
1
IN-CIRCLE 2 9 1
4
IN-RECTANGLE 4 5 9 10
6
```

Parte 2: Súper Búsqueda Secreta

Dada la gran popularidad de la isla, es necesario aumentar la seguridad. Para ello, la organización de espías EPF requiere de un sistema que les permita buscar información relevante de sus agentes en base a su identificación y codificación de permisos. De forma específica, todo pingüino agente tiene tanto un código identificador `PENGUIN ID`, como una codificación binaria de permisos, `CODE TREE`.

Figura 1: Identificación de un pingüino de la EPF

Todos los códigos de acceso del pingüino son 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.

Para facilitar la visualización de ejemplos, diremos que un nodo con valor 0 corresponde a un nodo pintado de negro y 1 corresponde a un nodo no pintado. Por lo tanto, el siguiente árbol es un ejemplo de Γ.

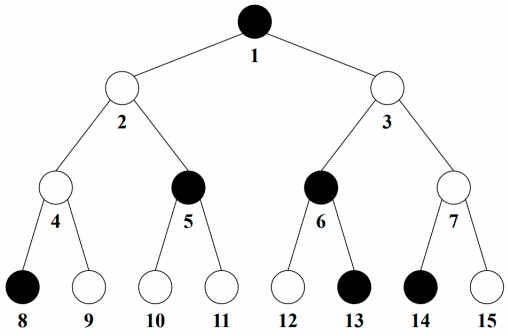

Figura 2: Γ con $h = 4$. Los nodos están numerados de 1 a n. Llamaremos a este número *el ID del nodo*. Se representa el `CODE TREE` de la figura 1

Ahora, dado Γ con altura h_Γ. Parte del problema a resolver será encontrar las ocurrencias de códigos de acceso (subtrees) $\gamma_1, \gamma_2, \ldots, \gamma_k$ en Γ, considerando que $1 \leq h_i \leq h_\Gamma \ \forall i \in \{1, \ldots, k\}$.

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. Denotamos las ocurrencias en Γ de un subárbol γ por el *ID del nodo* de Γ que coincide con la raíz de γ.

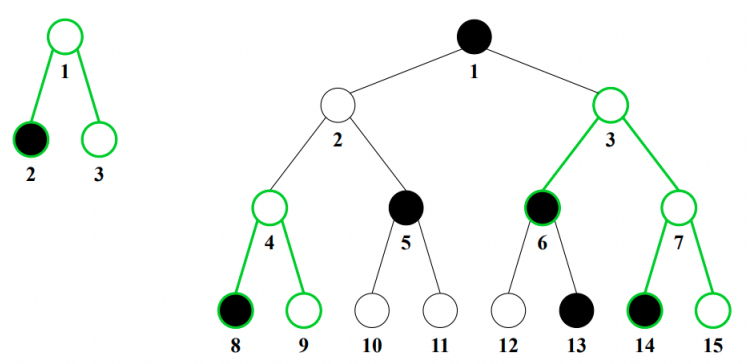

Figura 3: Las ocurrencias de γ en Γ serian los ID 3, 4 y 7. Notar que 3 y 7 se traslapan

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

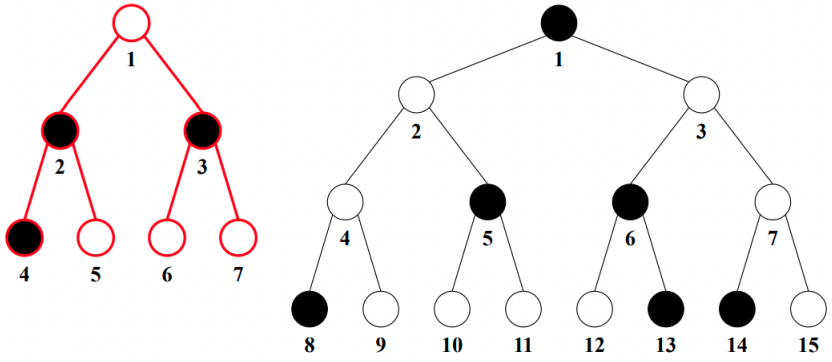

Figura 4: En este caso diremos que la ocurrencia no existe

Problema

Se te entregará una cantidad indeterminada de lineas, que tendrán 2 int separados por un espacio, un `penguin_id` y `code_tree`, luego se te entregará una linea con `END` indicando que dejarás de recibir pingüinos. Luego recibirás un número K indicando la cantidad de eventos a recibir y K lineas de eventos.

Eventos

`FIND {penguin_id}`

En este evento tendrás que entregar el árbol `code_tree` del pingüino con id `penguin_id`

output

```
Penguin {penguin_id}: {code_tree}
```

Complejidad esperada: $O(1)$ promedio. **Tip: haz uso de tablas de hash**[1].

[1] Ten en consideración los criterios de factor de carga, tipo de direccionamiento, entre otros.

`VALIDATE {penguin_id} {access_code_SubTree}`

Se te entregara id del pingüino y un código de acceso a buscar *(subtree)*. Deberás entregar todas las ocurrencias (los índices de nodo cabeza) en orden donde aparece este subárbol en el árbol del pingüino dado. En caso de que el pingüino exista, pero no tenga el permiso solicitado, es decir, no contiene el subárbol de la query, debes entregar lo siguiente:

output

```
ACCESS DENIED AGENT {penguin_id}
```

Por último, en caso de que el pingüino efectivamente tenga al menos una ocurrencia del permiso, el output esperado es retornar de `id` del pingüino junto al nivel más alto[2] de ocurrencia, representando el rango del agente, y finalmente los índices de ocurrencia de forma creciente, entregando el siguiente output:

output

```
GRANTED ACCESS AGENT {penguin_id}
    MAX LEVEL: {max_level}
    PERMISSIONS: {node_id_1} ... {node_id_k}
```

Por ejemplo, sea un pingüino con `PENGUIN ID: 200000001` y `CODE TREE: 011100101111001`, y tenemos la siguiente consulta:

input

```
VALIDATE 200000001 101
```

output

```
GRANTED ACCESS AGENT 200000001
    MAX LEVEL: 2
    PERMISSIONS: 3 4 7
```

Tip: Se recomienda fuertemente el uso de incremental hashing[3].

Complejidad esperada: No se pide complejidad, sin embargo debe cumplir con los tiempos de ejecución de la tarea.

Consideraciones generales:

- Todos los ID y CODE TREE de pingüinos serán de *datatype* int.
- Considerar que tanto para `Find` como para `VALIDATE` puedes asumir que siempre se consultará por un pingüino existente.

[2] Considerar que el primer nodo de un árbol lo consideramos como nivel 0. Además puedes calcular fácilmente el nivel de un nodo con $\lfloor \log_2(\text{nodo\_id}) \rfloor$

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

Ejecución

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

```
./secretfind input output
```

donde `input` será un archivo con los eventos a simular y `output` el archivo donde se guardarán los resultados.

En caso de querer correr el programa para ver los leaks de memoria utilizando valgrind deberás utilizar el siguiente comando:

```
valgrind ./secretfind input output
```

Tu tarea será ejecutada con archivos de creciente dificultad, asignando puntaje a cada una de estas ejecuciones que tenga un output igual al esperado. A continuación detallaremos los archivos de input y output.

Input

El archivo de input tenga un listado de pingüinos hasta un comando `END`. Luego de esto se recibirá una cantidad `K` de queries de subárboles.

input

```
200000001 01001010010
122 1101100
9000101 101011101101111
END
3
FIND 122
VALIDATE 122 000
VALIDATE 122 001
```

Output

output

```
Penguin 122: 1101100
GRANTED ACCESS AGENT 122
    MAX LEVEL: 1
    PERMISSIONS: 2
ACCESS DENIED AGENT 122
```

Informe

Debes entregar un informe breve en LaTeX junto a tu tarea, que explique cómo implementaste las estructuras y algoritmos solicitados. **El informe debe estar en un archivo PDF llamado informe.pdf en la carpeta raíz del repositorio (Descuento de 5 décimas en caso de no cumplir).** En este se debe explicar al menos los siguientes puntos:

- Parte 1
  1. ¿Qué estructuras de datos elegiste para cada consulta y por qué?
  2. ¿Cómo enfrentaste los retos de consultas geométricas, como las búsquedas en rectángulos o círculos?
  3. **BONUS (contestar en caso de aplicar al bonus, se puede usar media página adicional exclusivamente a esta pregunta):** Explique en sus propias palabras qué es *Fractional Cascading*, cómo cambia la complejidad de las operaciones al utilizar esta técnica y en qué casos resulta ventajosa su aplicación. También, describa cualquier desafío encontrado durante la implementación y cómo se abordó para mejorar la eficiencia.
- Parte 2
  1. Describe todas las tablas y funciones de hash utilizadas en tu solución. Explica como manejas el factor de carga para cada caso.
  2. Explica como implementaste/hubieras implementado incremental hashing en tu solución.
  3. Indica como manejas colisiones en caso de existir. En otro caso, justificar porque no hay colisiones.
- Bibliografía: Citar código de fuentes externas.

IMPORTANTE: Para asegurar una mejor comprensión y evaluación del informe, es **esencial** que cites el código al que haces referencia. Cada vez que menciones un fragmento de código, incluye el nombre del archivo y el número de línea correspondiente y su *hipervínculo* correspondiente. Puedes aprender cómo crear un *permalink* a tu código consultando la documentación de GitHub. Esto facilitará la revisión y garantizará que el informe sea coherente con la implementación.

Ejemplo:

Si en la Parte 1 del informe estás explicando una función que implementa una determinada operación, la referencia al código podría ser así:

> En nuestra implementación, utilizamos una lista enlazada para manejar la estructura de datos, como se muestra en la función `insertar_elemento` (`src/estructuras.c`, línea 45). Esta elección permite...

De esta manera, aseguras que los revisores puedan localizar rápidamente el fragmento de código relevante y verificar que la explicación sea consistente con la implementación.

Nota: Si por alguna razón no lograste completar alguna parte del código mencionado en el informe, **aún puedes obtener puntaje** en la sección correspondiente. Para ello, debes describir claramente cuál era tu idea de implementación, los pasos que planeabas seguir, y explicar por qué no pudiste completarlo. Esto demuestra tu comprensión del problema y del proceso, lo cual también es valioso para la evaluación.

Finalmente, **puedes** referenciar código visto en clases sin necesidad de incluir el código o pseudocódigo en el informe.

Puedes encontrar un template en Overleaf disponible en este enlace para su uso en la tarea.

Para aprender a clonar un template, consulta la documentación de Overleaf.

No se aceptarán informes de más de dos páginas (sin incluir bibliografía), informes ilegibles, o generados con inteligencia artificial **(Descuento de 6 puntos en el informe)**.

Evaluación

La nota de tu tarea es calculada a partir de testcases de input/output, así como un informe escrito en LaTeX que explique un modelamiento sobre cómo abarcar el problema, las estructuras de datos a utilizar, etc.
La ponderación se descompone de la siguiente forma:

Informe (20 %)	**Nota entre partes (70 %)**	**Manejo de memoria (10 %)**
10 % Parte 1	35 % Tests Parte 1	5 % Sin leaks de memoria
10 % Parte 2	35 % Tests Parte 2	5 % Sin errores de memoria

Para cada test de evaluación, tu programa deberá entregar el output correcto en **menos de 5 segundos** y utilizar menos de 1 GB de RAM[4]. De lo contrario, recibirás 0 puntos en ese test.

Además, se proporciona un archivo `.devcontainer` como el ambiente estándar para esta tarea. En caso de que tu programa falle tanto en el servidor como en el ambiente definido por este `.devcontainer`, no se podrá optar a una recorrección. Es tu responsabilidad asegurarte de que tu código funcione correctamente en este entorno.

Recomendación de tus ayudantes

Estas tareas generalmente requieren de mucha dedicación, 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 detenidamente y con anticipación para que te dediques a entender de manera profunda lo que te pedimos**. Una vez hayas comprendido el enunciado, dedica el tiempo que sea necesario para la planificación, modelación y elaboración de tu informe, para posteriormente poder programar de manera más eficiente. No olvides compilar el código como se indica en la tarea y de preguntar cualquier duda o por problemas que te surjan en Discussions. Estos son consejos de tus ayudantes que te pueden ayudar a pasar el ramo.

Entrega

Código: GIT - Rama principal del repositorio asignado, que debes generar con el link dado al principio de este enunciado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Atraso: Esta tarea considera la política de atraso y cupones especificada en el repositorio del curso.

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.

Uso de IA

Se prohíbe el uso de herramientas de inteligencia artificial para la realización de esta tarea. Esto incluye, pero no se limita a, el uso de modelos de lenguaje como ChatGPT, Copilot, u otras tecnologías similares para la generación automática de código, soluciones, o cualquier parte del trabajo requerido. Nos reservamos el derecho de revisar todas las tareas entregadas con el fin de detectar el uso de inteligencia artificial en la creación de las soluciones. Las tareas en las que se determine que se ha utilizado IA serán consideradas como una infracción a la política de honestidad académica y serán tratadas como casos de copia.

[4] Puedes revisarlo con el comando `htop` u ocupando `valgrind --tool=massif`