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

IIC2133 – Estructuras de Datos y Algoritmos

2025 - 1

Tarea 2

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

Link a generador de repos: CLICK AQUÍ

Objetivos

- Implementar y utilizar árboles binarios de búsqueda (ABB) para realizar búsquedas eficientes de datos.
- Investigar y comprender estructuras de datos que permiten realizar consultas eficientes sobre rangos.

Introducción

¡Otro día de verano en Danville! Phineas y Ferb han creado el Súper Mapa Dimensional de Danville, un sistema que localiza cualquier cosa en la ciudad instantáneamente. Pero el Dr. Doofenshmirtz ha disparado su nuevo Desordena-inador, causando que toda la información del mapa se desorganice en estructuras de datos caóticas.

Perry el Ornitorrinco, como Agente P, ha descubierto los planes, pero necesita ayuda urgente. El Mayor Monograma te ha reclutado como programador de la Agencia O.S.A.

'¡Agente Programador!' exclama Monograma. 'Doofenshmirtz ha transformado toda la información de Danville en complejas estructuras de árboles. Necesitamos tus conocimientos en algoritmos para restaurar el orden.'

Tu misión: implementar estructuras de datos y algoritmos para salvar Danville, enfrentando los extraños efectos del Desordena-inador en cada desafío.

Parte 1: Pelea de casas en árboles

En un despliegue de ingenio, ¡Phineas y Ferb han creado un laberinto de casas en árboles por todo Danville! El problema es que el Desordena-inador de Doofenshmirtz ha mezclado sus ubicaciones, y tienes que volver a ordenarlas antes de que sea muy tarde. Con Candace corriendo frenéticamente hacia casa para acusarlos, el tiempo se agota. Debes implementar un algoritmo de búsqueda eficiente para mapear las casas por regiones y resolver las feroces disputas territoriales entre pandillas vecinas que luchan por la supremacía en las alturas. Cada casa tiene asignada un id único y un nombre, para reconocerla fácilmente. Tu misión es encontrar las casas de cada región según el requerimiento. ¡Phineas y Ferb dependen de tu ayuda!

Eventos

Se te entregarán N casas con los siguientes atributos:

`HOUSE {id} {year} {n_region} {x} {y}`

Donde cada atributo es: [1]

- **id** es un id único por casa y siguen orden correlativo.
- **year** es el año de la casa.
- **n_region** es el número de la región física a la que pertenece en el árbol.
- **x** e **y** son números enteros que representan las coordenadas en la que se encuentran. Además, ambas coordenadas no serán superiores a 500.

Luego, se te entregarán B búsquedas que tendrás que realizar. Los tipos de búsqueda son:

`BY-ID {id}`

Búsqueda por ID. Como el ID es único, se debe mostrar uno o ningún resultado.

`BY-YEAR {year}`

Retorna todas las casas con el número de año dado. Los resultados deben presentarse por orden de llegada.

`BY-YEAR-REGION {year} {n_region}`

Retorna todas las casas con el año y número de región dado. Los resultados deben presentarse por orden de llegada.

[1] Hint: Todos los atributos pueden ser representados por un número entero.

`IN-X-RANGE {A} {B}`

Retorna todas las casas con $A \leq x < B$. El orden de los resultados deben ascendente por el valor de x. Si dos casas tienen la misma coordenada en x entonces se debe mostrar primero por orden de llegada.

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

Retorna todas las casas que estén dentro del círculo con centro (x, y) y radio r, incluyendo el perímetro. El orden de los resultados debe ser primero por x, luego por y. Puedes asumir que no se repite la posición.

Las consultas deben tener una complejidad no mayor a $\mathcal{O}(log(N) + k)$, a excepción del evento `IN-CIRCLE`, que debe cumplir con una complejidad no mayor que $O(\sqrt{N} + k \cdot log(k))$. En ambos casos k representa la cantidad de elementos encontrados. Este requerimiento, al igual que todos, deberá coincidir con lo escrito en el informe y tu solución en código[2].

El formato de retorno de todas las consultas debe ser:

output

```
{BUSQUEDA REALIZADA}: {CANTIDAD DE RESULTADOS}
{house_1.id}
{house_2.id}
...
```

[2]Para las búsquedas en rango, te recomendamos usar KD-tree, pero no olvides que es tu deber explicar por qué en el informe!

Ejecución

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

```
./houses 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 ./houses 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 comenzará con el entero `N` que indica la cantidad casas, y luego las `N` casas con todos sus atributos. Luego, se entrega el entero `B` que indica la cantidad de búsquedas a realizar, y las `B` líneas correspondientes a cada una.

Como ejemplo tenemos el siguiente input:

input

```
12
HOUSE 0 2022 3 10 15
HOUSE 1 2021 1 5 7
HOUSE 2 2022 1 8 3
HOUSE 3 2020 2 15 10
HOUSE 4 2021 2 12 8
HOUSE 5 2022 1 5 10
HOUSE 6 2021 3 9 9
HOUSE 7 2022 3 11 12
HOUSE 8 2020 1 3 5
HOUSE 9 2021 1 6 6
HOUSE 10 2022 2 20 20
HOUSE 11 2021 2 14 13
4
BY-ID 4
BY-YEAR-REGION 2022 1
IN-X-RANGE 5 12
IN-CIRCLE 10 10 5
```

Output

output

```
BY-ID 4: 1
    4
BY-YEAR-REGION 2022 1: 2
    2
    5
IN-X-RANGE 5 12: 6
    1
    5
    9
    2
    6
    7
IN-CIRCLE 10 10 5: 7
    5
    6
    0
    7
    4
    11
    3
```

Parte 2: Organicemos el verano

Phineas y Ferb, como es habitual, han decidido aprovechar al máximo su tiempo libre creando un invento distinto cada día. Gracias a una de sus más recientes creaciones, pueden vivir versiones alternativas del verano, acumulando así una enorme cantidad de días dedicados exclusivamente a la invención. En cada uno de estos días registrados, se planifica un nuevo proyecto, cada uno con su propia medida de dificultad.

A medida que se desarrollan estas actividades, comienzan a surgir imprevistos y nuevas necesidades: comparaciones entre distintas etapas del verano, ajustes a la dificultad de algunos inventos, e incluso intervenciones externas que alteran tramos completos del calendario. Para mantener el control sobre su agenda de invenciones y responder eficientemente a distintos eventos, necesitarán tu ayuda en el manejo de estos datos.

Problema

Phineas y Ferb quieren anotar y mantener actualizada la dificultad que tendrá hacer un invento dado por cada día del verano. La idea es que almacenes los eventos en alguna estructura adecuada, de modo que puedas cumplir las consultas de forma eficiente.

2.1 - Consultas

Se te entregarán N inventos, uno por cada día. Cada invento tiene los siguientes atributos:

`INVENTION {id} {dificultad}`

Donde cada atributo es:

- **id:** Entero positivo, correlativo y único para cada invento. Además, coincide con el día asociado al invento.
- **Dificultad:** Entero positivo.

Cada invento es realizado en un día distinto, que se determina por orden de llegada: el primer evento se realizó el día 1, el segundo evento el día 2, y así sucesivamente hasta el invento N que se realizó el día N.
Importante: No puede haber más de un invento por día, y todos los inventos se reciben al inicio del programa.

Luego, se te entregarán Q consultas que tendrás que realizar. Los tipos de consultas son:

`RANGE-EASIEST {primer_dia} {ultimo_dia}`

Quieres saber cuál es el invento más fácil programado entre un rango de días. Para ello recibes dos enteros, indicando el primer día y último día del rango en que queremos realizar la búsqueda. Debes retornar el invento con menor dificultad en el rango `[primer_dia, ultimo_dia]`.

El resultado esperado debe mostrar el siguiente output:

```
output
1 Mas facil entre {primer_dia} y {segundo_dia}: {id_invento}
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $\mathcal{O}(log(N))$[3].

[3]Este requerimiento, al igual que todos, deberá coincidir con lo escrito en el informe y tu solución en c

`RANGE-HARDEST {primer_dia} {ultimo_dia}`

Quieres saber cuál es el invento más difícil programado entre un rango de días. Para ello recibes dos enteros, indicando el primer día y último día del rango en que queremos realizar la búsqueda. Debes retornar el invento con mayor dificultad en el rango `[primer_dia, ultimo_dia]`.

El resultado esperado debe mostrar el siguiente output:

output

```
1 Mas dificil entre {primer_dia} y {segundo_dia}: {id_invento}
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $\mathcal{O}(log(N))$.

`RANGE-SUM {primer_dia} {ultimo_dia}`

Necesitas prepararte mentalmente para el desafío que implica llevar a cabo todos estos inventos, por lo que quieres saber qué tan difícil será esta tarea entre cierto rango de días. Para ello recibes dos enteros, indicando el primer día y último día del intervalo que consideraremos. Debes retornar la suma de dificultades entre todos los inventos programados para los días en el intervalo `[primer_dia, ultimo_dia]`.

El resultado esperado debe mostrar el siguiente output:

output

```
1 Dificultad total entre {primer_dia} y {segundo_dia}: {id_invento}
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $\mathcal{O}(log(N))$.

`UPDATE {dia} {nuevo_valor}`

Te das cuenta de que estimaste mal la dificultad del invento del día `{dia}`. Por lo tanto, debes cambiar la dificultad de ese invento por el valor `{nuevo_valor}`.

El resultado esperado debe mostrar el siguiente output:

output

```
1 Invento {id_invento} actualizado.
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $\mathcal{O}(log(N))$.

`RANGE-UPDATE {primer_dia} {ultimo_dia} {valor}`

Metiste la pata y registraste mal la dificultad de todos los días en el intervalo `[primer_dia, ultimo_dia]`. Para corregir este error, deberás actualizar la dificultad de los inventos en este rango sumando el valor `{valor}` (que puede ser positivo o negativo).

El resultado esperado debe mostrar el siguiente output:

output

```
1 Rango {primer_dia} - {ultimo_dia} actualizado.
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $\mathcal{O}(N)$.

Ejecución

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

```
./invention 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 ./invention input output
```

Consideraciones

- En caso de empate en alguna consulta, deberás retornar el invento de menor id.
- Puedes asumir que para todas las consultas se entregarán días válidos.
- Las dificultades siempre serán positivas.
- Se recomienda fuertemente investigar sobre árboles que cumplan la complejidad pedida.

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 comenzará con el entero `N`, que corresponde al número de inventos. Luego, se entregará la información de cada uno de los inventos, con el formato indicado anteriormente. Posterior a eso, se entregará el entero `Q`, que representa el número de consultas a pedir. A continuación, recibirás cada una.

input

```
4
1 1
2 1
3 3
4 4
5
RANGE-EASIEST 1 3
RANGE-HARDEST 1 4
UPDATE 2 10
RANGE-UPDATE 1 3 7
RANGE-HARDEST 1 4
```

Output

output

```
Mas facil entre 1 y 3: 1
Mas dificil entre 1 y 4: 4
Invento 2 actualizado.
Rango 1 - 3 actualizado.
Mas dificil entre 1 y 4: 1
```

Explicación

Primero se recibe N, los inventos y Q. Luego, para la primera consulta, se tiene que entre los días 1 y 3 el invento de menor dificultad es 1 y 2, pero por el criterio de desempate, se retorna 1. En el caso del más difícil, se retorna 4. Luego, se actualiza el invento 2, quedando en dificultad de 10. A continuación, se le debe sumar 7 unidades a cada invento entre 1 y 3, ambos inclusive. Así, la dificultad de cada día, en orden correlativo, es 8, 17, 10. Finalmente, el evento más difícil entre 1 y 4 es el que corresponde al evento del día 2.

Informe

Además del código de la tarea, se debe redactar un **breve** informe que explique cómo se pueden implementar algunas estructuras y algoritmos del enunciado. Les recomendamos **comenzar la tarea escribiendo este informe**, ya que pensar y armar un esquema antes de pasar al código ayuda a estructurarse al momento de programar y adelantar posibles errores que su solución pueda tener.

Este informe se debe entregar en un archivo PDF escrito usando LaTeX junto a la tarea (por lo cual tienen la misma fecha de entrega), con nombre `informe.pdf` en la raíz del repositorio. En este se debe explicar al menos los siguientes puntos:

- Parte 1
  1. Implementación de la base de datos: Debes explicar cómo almacenas los datos entregados. Si usas un ABB (o más) debes justificar tu elección. Puedes apoyarte de las ventajas y desventajas del árbol.
  2. Complejidad: Debes explicar cómo abordaste el problema de consultar los datos almacenados eficientemente. Es decir, explicar por qué las búsquedas cumplen con la complejidad solicitada.
- Parte 2
  1. Explica la/s estructura/s utilizada/s, indica su complejidad de construcción y cómo permite cumplir con la complejidad solicitada en los eventos `RANGE-SUM` y `UPDATE`.
  2. Supón que quieres responder a la consulta `RANGE-UPDATE` en tiempo *O(log(N))*, manteniendo el tiempo de respuesta para las consultas previas. ¿Cómo lo resolverías? Explica las modificaciones necesarias y justifica por qué funciona.
- Bibliografía: Citar código de fuentes externas.

Cómo citar código: 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 tres páginas (sin incluir bibliografía), informes ilegibles, o generados con inteligencia artificial. Además, en caso de no cumplir con el formato de entrega indicado previamente, habrá un descuento de 10 décimas sobre la nota del 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	40 % Tests Parte 1	5 % Sin leaks de memoria
10 % Parte 2	30 % 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.

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 (Recuerda que la entrega es el 28 de Mayo!). 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`