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

IIC2133 – Estructuras de Datos y Algoritmos

2024 - 2

Tarea 1

Fecha de entrega código: 26 de septiembre, 23:59 Chile continental

Link a generador de repos: CLICK AQUI

Objetivos

- Aplicar heaps para resolución de problemas de prioridad.
- Usar ordenamiento lineal para soluciones eficientes.
- Optimizar algoritmos para cumplir con una complejidad.

Introducción

¡Felicidades!

¡DCClub pEDDnguin ha sido un éxito rotundo! Tanto así, que expandiste tu franquicia a un servicio de consultoría para los pobres pingüinos, ya que, si bien lograste tranquilizarlos con el orden y el manejo correcto, los problemas no paran de surgir en esta isla tan extraña *(¿estarán siendo saboteados por algún oso y un cangrejo?)*...
Al parecer, tus habilidades fueron tan sorprendentes que ahora quieren que los ayudes a manejar distintos lugares. Dada la cercanía de una de las fiestas más esperadas del año, el **18 de septiembre**, necesitan una particular ayuda con los centros turísticos. Es por esto que deberás manejar no solo uno sino múltiples lugares turísticos dentro de la isla. Los pingüinos esperan mucho de ti luego de tu gran éxito, ¡no los defraudes!

Parte 1: H(eap)piness

La isla celebra la llegada de septiembre con un gran festival *(fonda)* que ha atraído a multitudes a diferentes sectores de la ciudad, especialmente a los lugares turísticos. Para mantener el orden y asegurar el éxito del evento, es crucial contar con una guía de estadísticas en cada uno de los centros festivos. Sin embargo, dado que el festival provoca sentimientos variados entre los pingüinos, hay muchas fluctuaciones inusuales en los datos. Por lo tanto, en lugar de usar el promedio tradicional, evaluaremos el éxito del festival basándonos en las diferentes medianas de felicidad registradas en cada sector.

Problema

Tendrás que usar estructuras de datos que permitan, de forma eficiente, la variación en las multitudes de pingüinos y, a la vez, la obtención de estadísticas de los sectores para asegurar el éxito del evento.

Entidades

Lugar turístico: Pueden existir una cantidad fija S. los identificamos en base a sus índices desde 0 a $S-1$, que tienen una capacidad máxima de ocupación de pingüinos K_s.

Pingüino: Cada pingüino posee un `ID` único y además presenta una `FELICIDAD` estrictamente mayor a 0.

Esta parte inicial se divide en dos. En ciertos eventos notarás que se requiere una complejidad específica, respecto a esto, cuando se hable sobre n para indicar la complejidad de tiempo, nos referimos a n = cantidad de pingüinos dentro del sector. Por otro lado, cuando se hable de s se indica la complejidad de tiempo con s = cantidad de sectores.

1.1 - Eventos

Se te entregarán E eventos, que pueden ser:

`ENTER {sector_id} {penguin_id} {happiness}`

Este evento indica que entra el pingüino con id `penguin_id` con un estado de animo de `happiness` al lugar turístico `sector_id` (índice del sector). **Este evento no requiere de ningún output**.

Importante: Se requiere que esta inserción tenga una complejidad no mayor a $O(log(n) + log(s))$[1].

`HAPPINESS-BUFF {sector_id} {penguin_id} {increase}`

Este evento indica que el pingüino con `penguin_id`, ubicado en el sector `sector_id` (índice del sector), incrementa su felicidad actual en `increase`.

El resultado esperado debe mostrar la felicidad actualizada del pingüino:

```
output
1 Penguin {penguin_id} happiness updated to {updated_happiness}
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $O(n + log(s))$.

`FESTIVAL`

Este evento muestra el estado general del festival, proporcionando estadísticas de todos los sectores en el siguiente formato:

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

```
output
Festival Stats:
    Sector {sector_id1} has {total_penguins_inside} penguin[s]:
        {median_hapiness} {happiest_penguin_id} {saddest_penguin_id}
    Sector {sector_id2} has {total_penguins_inside} penguin[s]:
        {median_hapiness} {happiest_penguin_id} {saddest_penguin_id}
    ....
```

El orden de impresión de los sectores debe ser por `id`. En caso de empate en las estadísticas de los pingüinos, se debe mostrar el pingüino con menor `id` (o equivalentemente por orden de llegada).

Importante: Se requiere que esta operación tenga una complejidad no mayor a $O(s)$, siendo s la cantidad de sectores.

1.2 - Eventos

`LEAVE`

Este evento indica que se retirará el pingüino con la **menor felicidad** del sector que tenga la mediana de felicidad más baja.

En caso de que en el mismo sector haya más de un pingüino con la menor felicidad, se retirará el pingüino que haya llegado primero (es decir, el de menor `id`). En caso de empate de sectores con la mediana de felicidad más baja se utiliza el de índice menor.

El resultado esperado que debe devolver este evento es el siguiente:

```
output
Penguin {penguin_id} leaving sector {sector_id}
```

Importante: Se requiere que esta operación tenga una *O(log(n) + log(s))* complejidad no mayor a

`MEDIANS`

Este evento proporciona una visión general del festival, mostrando los sectores con la mejor y peor mediana en el siguiente formato:

```
output
Medians resume:
    {best_median_sector_id} v/s {worst_median_sector_id}
    {best_median} - {worst_median}
```

Si la mejor mediana y la peor mediana pertenecen al mismo sector, se considera que el sector con menor `ID` es el mejor, y el sector con mayor `ID` es el peor. Si solo hay un sector, se considera como ambos.

Importante: Esta operación debe tener una complejidad de tiempo de $O(1)$.

Consideraciones

- Cuando hay una cantidad par de datos, la mediana es el promedio de los dos números medios (usando división parte entera). Mientras que cuando la cantidad de datos es impar, la mediana es el número medio.
- Los casos de empates se deciden mediante el orden de llegada del pingüino, el cual coincide con su ID.
- Cuando un sector se encuentre a su capacidad máxima no recibirás ningún input de `ENTER` hasta que exista un espacio disponible nuevamente.
- Como recomendación de tus ayudantes, inicia modelando el problema para solamente un sector y luego puedes ampliarlo a varios sectores.

Ejecución

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

```
./happy 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 ./happy 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 número `S` que indica la cantidad de sectores a considerar, seguido `S` líneas indicando la capacidad máxima de pingüinos del sector s, con el siguiente formato K_s.

Posterior a ello se entrega `E` que corresponde al número de eventos a recibir, y las `E` líneas correspondientes a cada uno.

Como ejemplo tenemos el siguiente input:

input

```
2 # tendremos dos lugares turisticos
10 # capacidad de lugar_0 sera 10
20 # capacidad de lugar_1 sera 20
9 # eventos
ENTER 0 0 99
ENTER 0 1 28
ENTER 1 2 10
ENTER 1 3 10
ENTER 1 4 20
LEAVE
FESTIVAL
HAPPINESS-BUFF 1 3 10
MEDIANS
```

Output

output

```
    Penguin 2 leaving sector 1
Festival Stats:
    Sector 0 has 2 penguin[s]:
        63 0 1
    Sector 1 has 2 penguin[s]:
        15 4 3
Penguin 3 happiness updated to 20
Medians resume:
    0 v/s 1
    63 v/s 20
```

Parte 2: O(n)rdenamiento

Durante el festival en la isla, las atracciones turísticas son tan populares que se forman largas colas de pingüinos. Dado el espacio limitado y la alta demanda, es crucial gestionar la cola de manera eficiente. Algunos pingüinos pueden abandonar la fila, mientras que otros se unen. Para mantener el orden, es necesario organizar la cola según atributos como edad o membresía.

Problema

Tú tarea es desarrollar un sistema que permita ordenar a los pingüinos según el atributo requerido. Debes ser capaz de ordenarlos según se te pida cumpliendo con una **complejidad de** $O(nk)$ como máximo, donde n es la cantidad de pingüinos y k es el dominio del atributo por el cual se ordena.

Entidades

Cola de espera: Existe solamente una cola de espera.

Pingüino: Cada pingüino tiene una edad (`age`) asociada, que oscila entre 0 y 99 años, y un atributo *booleano* de membresía (`membership`) que puede ser 1 (si la posee) o 0 (si no la posee).

Eventos

`ENTER {id} {age} {membership}`

Este evento registra a un pingüino entrando a la cola de espera, donde se entregara la edad (`age`) del pingüino y su membresía (`membership`) respectivamente.

output

```
Penguin {id} with age {age} has entered
```

`LEAVE {id}`

Este evento registra la salida de un pingüino, siempre se entregará una id válida.

output

```
Penguin {id} has left
```

`SORT-BY {type} {order}`

Este evento indicara que deberas ordenar la cola de espera dado un tipo (`type`) que podrá ser edad (`age`) o membresía (`membership`). Por otro lado se entregará un orden (`order`) el cúal tomará los valores de `'asc'` o `'des'`. Que corresponde a un orden[2] ascendente o descendente respectivamente.

output

```
Penguins sorted by {type} in {order} order:
ID_1, ID_2, ID_3, ..., ID_n
```

`DOUBLE-SORT-BY {type_1} {order_1} {type_2} {order_2}`

Deberás implementar una función similar a `SORT-BY`, pero con dos atributos y órdenes respectivos. Primero, ordena por el primer atributo, y luego aplica el segundo orden sobre los elementos que compartan el mismo valor en el primer atributo.

output

```
Penguins sorted by {type_1}, {type_2} in {order_1}, {order_2} order:
ID_1, ID_2, ID_3, ..., ID_n
```

[2]El orden ascendente o descendiente en membresía será en relación al valor booleano que tome

Ejecución

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

```
./order 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 ./order 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 número `E` que corresponde al número de eventos a recibir, y las `E` líneas correspondientes a cada uno.

input

```
12
ENTER 3 21 1
ENTER 5 34 1
ENTER 8 81 0
ENTER 13 15 1
ENTER 21 18 0
ENTER 34 27 0
ENTER 55 19 1
ENTER 89 44 0
SORT-BY age des
LEAVE 8
DOUBLE-SORT-BY membership age des asc
LEAVE 55
```

Output

output

```
Penguin 3 with age 21 has entered
Penguin 5 with age 34 has entered
Penguin 8 with age 82 has entered
Penguin 13 with age 15 has entered
Penguin 21 with age 18 has entered
Penguin 34 with age 27 has entered
Penguin 55 with age 19 has entered
Penguin 89 with age 44 has entered
Penguins sorted by age in des order:
8 89 5 34 3 55 21 13
Penguin 8 has left
Penguins sorted by membership, age in des, asc order:
13 55 3 5 21 34 89 8
Penguin 55 has left
```

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. Explicación General: Proporciona una breve descripción de cómo abordaste el problema, incluyendo las estructuras de datos que utilizaste y su justificación.
  2. Complejidad: Justifica cómo tu solución cumple con la complejidad solicitada, desglosando cada operación clave.
- Parte 2
  1. Explicación General: Resume la lógica y algoritmos detrás de tu implementación del problema de ordenamiento.
  2. Complejidad: Explica cómo se cumple la complejidad $O(nk)$ para cada tipo de ordenamiento solicitado.
- 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.

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	50 % Tests Parte 1	5 % Sin leaks de memoria
10 % Parte 2	20 % 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[3]. 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 (Recuerda que la entrega es el 26 de Septiembre!). 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.

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