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

IIC2133 – Estructuras de Datos y Algoritmos

2025 - 1

Tarea 1

Fecha de entrega código: 25 de Abril, 23:59 Chile continental

Link a generador de repos: CLICK AQUÍ

Objetivos

- Aplicar heaps para resolución de problemas de prioridad.
- Implementar y comparar algoritmos de ordenamiento.
- Seleccionar un algoritmo de ordenamiento óptimo según el caso.

Introducción

La O.W.C.A. (Organization Without a Cool Acronym) enfrenta una crisis. Doofenshmirtz ha creado el Clonador-Inador, inundando la ciudad con versiones de sí mismo y desatando el caos. El Mayor Monograma, confiando en tu talento en programación, te ha encargado desarrollar un sistema que organice a la O.W.C.A. y mantenga a raya a los clones de Doofenshmirtz. Sin embargo, Doofenshmirtz, abrumado por su propio desastre, también ha recurrido a ti para poner orden entre sus clones y finalmente ejecutar su plan sin contratiempos. Ahora, ambos bandos compiten por tu ayuda. ¿Trabajarás para salvar el día o para asegurar el éxito del plan malvado?

Parte 1: Agente P, Doofenshmirtz está tramando algo, ve a por él

La O.W.C.A. está en crisis. El número de científicos malvados ha aumentado en todo el mundo, y la organización es un caos. Los agentes no saben a qué misión ir, las agencias no logran coordinarse y la seguridad está en peligro. El Mayor Monograma, al enterarse de tus habilidades en programación, te ha contratado para desarrollar un sistema que organice la O.W.C.A. a nivel mundial, asegurando que puedan enfrentar a los villanos de la manera más eficiente.

Problema

Debes implementar un sistema que permita a cada agencia identificar y asignar rápidamente a su mejor agente. Cuando surge una misión, el mejor disponible es asignado de inmediato. Además, los agentes mejoran o pierden habilidades y las agencias cambian, a lo que el sistema debe adaptarse sin afectar las operaciones. ¿Podrás asignar a los agentes adecuados en el momento preciso? Perry confía en ti... aunque, como siempre, desaparecerá misteriosamente antes de poder agradecerte.

Entidades

Agente: Cada agente posee un `ID` único y además presenta un `NIVEL` estrictamente mayor a 0.

Agencia: Cada agencia tiene un `ID` único desde 0 a $S-1$ y un máximo de agentes N.

Para efectos de la complejidad algorítmica de cada evento, considera que n es el número total de agentes en una agencia.

Primero, recibirás una línea con un entero S, indicando la cantidad máxima de agencias. Luego, recibirás una línea con un entero E que indica la cantidad de eventos a recibir. A continuación, se te entregarán los E eventos, donde cada uno puede contener más de una línea de input que deberás procesar. Puedes ver ejemplos de eventos en la sección de input.

1.1 - Eventos

`BUILD {agencia_id} {N}`

Este evento crea una nueva agencia con ID `agencia_id` y cantidad máxima de `N` agentes. La agencia se inicializa con los `N` agentes disponibles para misiones.

Se reciben `N` líneas, donde cada línea contiene el ID y el nivel de un agente. La agencia debe organizarse de modo que el agente con más alto `nivel` sea el líder. En caso de que haya dos agentes con el mismo nivel, se elegirá como líder al que tenga menor ID.

input

```
BUILD {agencia_id} {N}
{nivel_1}
{nivel_2}
...
...
{nivel_N}
```

El resultado esperado debe mostrar el siguiente output:

```
output
Agencia {agencia_id}
    # Agentes: {N}
    Lider: {agente_id}[{nivel}]
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $\mathcal{O}(n \cdot log(n))$[1].

`MISION {agencia_id}`

Este evento indica que la agencia `agencia_id` debe asignar al agente líder para una nueva misión. Se debe asignar al agente con mayor nivel presente. Una vez asignado, el agente no estará disponible para más misiones hasta que regrese.

El resultado esperado debe mostrar el siguiente output:

```
output
Agente {agente_id} asignado a una mision
```

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

`STATUS`

Este evento muestra el lider actual de cada agencia.

El resultado esperado debe mostrar el siguiente output:

```
output
Agencia {0}
    Lider: {agente_id}[{nivel}]
Agencia {1}
    Lider: {agente_id}[{nivel}]
...
Agencia {S-1}
    Lider: {agente_id}[{nivel}]
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $O(S)$

1.2 - Eventos

`RETURN {agencia_id} {agente_id} {nivel}`

Este evento indica que el agente con ID `agente_id` y `nivel`, se une a la agencia con ID `agencia_id` y está disponible para hacer misiones.

El resultado esperado debe mostrar el siguiente output:

```
output
Agente {agente_id} disponible en {agencia_id}
```

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

`UPDATE_LEVEL {agencia_id} {agente_id} {delta}`

Este evento indica que el agente con ID `agente_id` de la agencia con ID `agencia_id` ha cambiado su nivel en `delta`.

Si `delta` es un número positivo, el nivel del agente aumentará en `delta`, mientras que si `delta` es negativo, el nivel del agente disminuirá en esa cantidad.

Si tras la actualización el nivel del agente es menor que cero, su nivel final se dejara en `0`.

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

El resultado esperado debe mostrar el siguiente output:

```
output
1 Agente {agente_id} ha cambiado su nivel a {nivel_actualizado}
```

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

`MERGE {agencia_1_id} {agencia_2_id}`

Este evento indica que las agencias con ID `agencia_1_id` y `agencia_2_id` se unen en una nueva agencia, conservando el ID de `agencia_1_id`. La cantidad máxima de agentes de la nueva agencia será la suma del máximo de ambas agencias, y todos los agentes disponibles pasarán a ser parte de la nueva agencia. El líder de la nueva agencia será el agente con el nivel más alto.

El resultado esperado debe mostrar el siguiente output:

```
output
1 Agencia {agencia_1_id} y {agencia_2_id} unidas
2 Agencia {agencia_1_id}
3     # Agentes: {N}
4     Lider: {agente_id}[{nivel}]
```

Importante: Se requiere que esta operación tenga una complejidad no mayor a $O((n+m) \cdot log(n+m))$, siendo n la cantidad máxima de agentes de `agencia_1_id` y m la de `agencia_2_id`

Ejecución

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

```
./agency 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 ./agency 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 `S` que indica la cantidad máxima de agencias. Luego, se entrega `E` que corresponde al número de eventos a recibir, y las líneas correspondientes a cada uno.

Como ejemplo tenemos el siguiente input:

input

```
1
5
BUILD 0 5
5
2
0
1
4
STATUS
MISION 0
MISION 0
STATUS
```

Output

output

```
Agencia 0
    # Agentes: 5
    Lider: 0[5]
Agencia 0
    Lider: 0[5]
Agente 0 asignado a una mision
Agente 4 asignado a una mision
Agencia 0
    Lider: 1[2]
```

Parte 2: El Orden-Inador: Algoritmos para la Dominación Malévola

En un intento por dominar el Área Limítrofe, el malvado Dr. Heinz Doofenshmirtz ha creado el Clonador-Inador, una máquina capaz de generar múltiples versiones de sí mismo. Su plan consiste en distribuir estos clones estratégicamente por toda la ciudad para maximizar su presencia malévola. Sin embargo, cada clon tiene características particulares que lo hacen más apto para ciertos entornos. Un clon con afición por la comida rápida sería ideal para centros comerciales, mientras que otro con miedo a las alturas resultaría ineficaz en parques de atracciones. Para optimizar su plan, Doofenshmirtz necesita un algoritmo eficiente que organice a sus clones según sus habilidades y los asigne a ubicaciones específicas. La velocidad es crucial, pues Perry el Ornitorrinco podría aparecer en cualquier momento y arruinar todo el plan.

Problema

Tendrás que ordenar a los clones del Dr. Doofenshmirtz en las distintas ubicaciones del área limítrofe, utilizando algoritmos eficientes que aseguren rapidez, y así el éxito del plan malévolo. Deberás lidiar con grandes hordas de clones, asignándolos a todos a distintas ubicaciones, y a su vez manejar reasignaciones de lugar, manteniendo el orden en el registro del Dr. Doof.

Entidades

Clones: Cada uno tiene `(id)`, `(paciencia)`, `(destreza)` y `(liderazgo)`. Los id de estos van de 0 a N-1.

Lugar: Los lugares permiten la entrada de los clones. Cada uno tiene (`id`) y comparten la capacidad máxima (`c`). Los id de estos van de 0 a L-1.

Eventos

Para efectos de la complejidad algorítmica de cada evento, considera que `N` es el número total de clones.

`ENTER {id_lugar} {cantidad_entrante}`

Se recibe id `{id_lugar}`, un entero m que indica la cantidad entrante de clones, y luego m lineas con los datos de cada clon. Debe guardar los clones en el lugar con id `{id_lugar}`. El lugar siempre se encontrará inicialmente vacío.

input

```
ENTER {id_lugar} {cantidad_entrante}
{id_1 paciencia_1 destreza_manual_1 capacidad_liderazgo_1}
...
{id_m paciencia_m destreza_manual_m capacidad_liderazgo_m}
```

output

```
Entraron {cantidad_entrante} clones al lugar {id_lugar}.
```

`MASSIVE {id_lugar} {atributo_orden} {orden}`

Ordena el lugar en su totalidad según el atributo entregado y el orden señalado, el que puede ser `creciente` o `decreciente`

output

```
Lugar {id_lugar} en orden {orden} segun {atributo}:
    {id_clon_1}
    ...
    {id_clon_n}
```

Importante: Se requiere que esta operación tenga una complejidad promedio de tiempo no mayor a $O(clog(c))$ [2]. Además, debe tener complejidad espacial no mayor a $O(log(c))$ en el peor caso. c corresponde a la capacidad del lugar.

`K-ENTER {id_lugar} {cantidad_entrante} {k}`

Se recibe un id `{id_lugar}`, un entero m que indica la cantidad entrante de clones, y luego m lineas con los datos de cada clon. Debe guardar los clones en el lugar con id `{id_lugar}`. Los clones entrantes se encuentran `k-ordenados` según el parámetro `k`. El lugar siempre se encontrará inicialmente vacío.

input

```
ENTER {id_lugar} {cantidad_entrante} {k}
{id_1 paciencia_1 destreza_manual_1 capacidad_liderazgo_1}
...
{id_m paciencia_m destreza_manual_m capacidad_liderazgo_m}
```

output

```
Entraron {cantidad_entrante} clones al lugar {id_lugar}.
```

`K-MASSIVE {id_lugar} {atributo_orden} {orden}`

Ordena el lugar de id `{id_lugar}` en su totalidad según el atributo entregado y el orden señalado, el que puede `creciente` o `decreciente`. En este caso, el lugar se encuentra k-ordenado, según el parámetro `k` entregado en `K-ENTER`. Este evento ocurrirá siempre luego de un K-ENTER. Entrega el mismo output que el evento MASSIVE.

Importante: Se requiere que esta operación tenga una complejidad de tiempo no mayor a *O(clog(k))* y de memoria *O(k)* donde c es la capacidad del lugar.

`LIGHT {id_lugar} {cantidad_entrante}`

Considera que el lugar ya tiene `m` elementos ordenados con anterioridad, y llega una cantidad pequeña `v` de clones. Deberás reordenar el lugar según el atributo por el que ya esté ordenado, manteniendo el orden creciente o descreciente según sea el caso. Además, siempre se cumplirá que $v < m$ y $v \leq 10$.

input

```
LIGHT {id_lugar}
{id_1 paciencia_1 destreza_manual_1 capacidad_liderazgo_1}
...
{id_v paciencia_v destreza_manual_v capacidad_liderazgo_v}
```

output

```
Entraron {v} clones al lugar {id_lugar}. Lugar reordenado:
    {id_clon_1}
    ...
    {id_clon_n}
```

Importante: Se requiere que esta operación tenga una complejidad tiempo no mayor a $O(c)$ en el peor caso, donde c es la capacidad del lugar. Además, debe tener complejidad de memoria $O(1)$.

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

`TOTAL-ORDER {atributo_orden} {orden}` Deberás juntar y ordenar todos los clones según `{atributo_orden}` en un solo arreglo. Cada lugar se encontrará previamente ordenado según el mismo atributo y en el mismo orden. Todos los lugares tendrán al menos 1 clon.

Importante: Se requiere que esta operación tenga una complejidad de tiempo no mayor a *O(Nlog(L))* y de memoria *O(N+L)*.

output

```
Orden para todos:
    {id_clon_1}
    ...
    {id_clon_N}
```

`BACK-TO-ROOMS {atributo_orden} {orden}` Este evento ocurre justo después después de TOTAL-ORDER. Suponiendo que hay L lugares y N clones, los primeros $\lfloor \frac{N}{L} \rfloor$ se van ordenados según el atributo `{atributo_orden}` y en orden `{orden}` al lugar 0, luego los siguientes $\lfloor \frac{N}{L} \rfloor$ al lugar 1, y así sucesivamente. Si es que existen clones que sobran[3], serán eliminados.

output

```
Vuelven al lugar 0:
    {clon_i}
    ...
    {clon_j}
    ...
Vuelven al lugar M:
    {clon_k}
    ...
    {clon_l}
```

De ser el caso que existan clones sobrantes, el mensaje también deberá decir:

output

```
Los siguientes clones seran eliminados:
    {clon_p}
    ...
    {clon_v}
```

`LEAVE-PLACE {id_lugar}` El lugar con id `{id_lugar}` se vacía por completo.

output

```
Todos los clones del lugar {id_lugar} han sido eliminados.
```

Consideraciones Generales

- Nunca van a intentar entrar más clones de los que el lugar puede recibir en ese momento
- El evento `LIGHT` siempre sucederá en un lugar previamente ordenado.
- Todos los valores entregados son enteros.
- Todo caso de desempate se resuelve dando prioridad al clon de menor id (ya sea en orden creciente o decreciente).

[3] Como consecuencia de la función piso, podrían no repartirse todos los clones existentes.

Ejecución

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

```
./clones 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 ./clones 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 `L` que corresponde al número de lugares. Luego, se entrega el entero `c` que indica la capacidad máxima de los lugares. Seguido, se entrega el número `E` de eventos y la líneas correspondientes a cada uno.

input

```
3
10
3
ENTER 0 4
0 3 5 7
1 2 4 5
2 4 7 9
3 3 5 7
MASSIVE 0 paciencia creciente
LIGHT 0 2
6 2 4 5
7 1 7 2
```

Output

output

```
Entraron 4 clones al lugar 0.
Lugar 0 en orden creciente segun paciencia:
    1
    0
    3
    2
Entraron 2 clones al lugar 0. Lugar reordenado:
    7
    1
    6
    0
    3
    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. 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. Algoritmos implementados: Para los eventos `MASSIVE`, `K-MASSIVE`, `LIGHT` y `TOTAL-ORDER` explica el algoritmo de ordenamiento elegido y justifica cómo cumple con la complejidad solicitada.
  2. Supón que tienes que implementar `TOTAL-ORDER`, pero los lugares no están previamente ordenados. ¿Qué algoritmo usarías para resolver el evento? ¿Cómo afectaría a la complejidad?
- 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 tres 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	20 % Tests Parte 1	5 % Sin leaks de memoria
10 % Parte 2	50 % 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 25 de Abril!). 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`