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

IIC2133 – Estructuras de Datos y Algoritmos

2024 - 2

Tarea 3

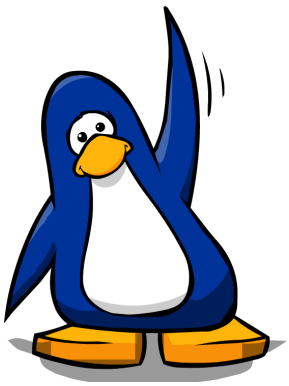
Fecha de entrega código: 25 de noviembre, 23:59 Chile continental

Link a generador de repositorios: CLICK AQUÍ

Objetivos

- Explorar el uso de algoritmos que se emplean sobre grafos.
- Emplear técnicas algorítmicas para encontrar soluciones óptimas.

Introducción



La Última Misión

La paz de la isla de Club Penguin depende de tu ingenio una última vez. Los habitantes de la isla te piden ayuda en un desafío final, donde deberás aplicar toda tu habilidad en lógica y estrategia para resolver problemas que solo un verdadero experto puede superar.
Cada problema resuelto acerca a la isla a un futuro en calma y armonía. ¿Estás listo para cumplir esta última misión y demostrar, una vez más, que eres el héroe que la isla necesita?

Problema 1: Cruzando el Iglú en Silencio

Un puffle duerme plácidamente en medio del iglú, y necesitas cruzar la habitación sin despertarlo. ¡Cuidado! Si lo despiertas, se enojará y podría lanzar llamas. Debes cruzar muy silenciosamente.

El suelo de la sala rechina, así que cada paso despierta ligeramente al puffle. Si esperas un poco antes de dar el siguiente paso, el puffle vuelve a dormir profundamente.

Para cruzar el iglú, debes dar x pasos. El puffle empieza con una profundidad de sueño p. Si la profundidad de sueño cae por debajo de un umbral w (donde $0 < w < p$), el puffle despertará.

Cada paso disminuye la profundidad del sueño en una cantidad s_i específica (donde $0 < s_i < p$), para $s_0, s_1, \ldots, s_{x-1}$. Por cada unidad de tiempo k que esperas antes de dar un paso, la profundidad del sueño del puffle aumenta en 1.

¿Cuál es el menor tiempo total de espera posible para cruzar el iglú sin despertar al puffle?

Entrada

La entrada consta de las siguientes líneas:

- Una línea con cuatro enteros x, k, w y p, que representan el número de pasos necesarios para cruzar, la cantidad de tiempo necesaria para aumentar en 1 la profundidad del sueño del puffle, el umbral mínimo de profundidad de sueño para no despertar al puffle, y el la profundidad de sueño inicial (y máxima) del puffle.
- Una segunda línea con x enteros $s_0, s_1, \ldots, s_{x-1}$, donde cada s_i representa cuánto disminuye la profundidad del sueño al dar el paso i.

Salida

- Si en algún momento la profundidad de sueño del puffle supera el valor inicial de p debido al tiempo de espera necesario para dar un paso, retorna -1.
- En caso contrario, retorna la suma acumulada del tiempo de espera necesario para cruzar completamente el iglú sin despertar al puffle.

Ejemplos

Ejemplo 1

input

```
4 1 4 5
2 3 2 1
```

output

```
-1
```

Explicación:

- Espera: Es necesario esperar, sin embargo la profundidad requerida (6) es mayor que la máxima permitida (5). No es posible cruzar sin despertar al puffle.

Ejemplo 2

```
input
3 2 3 5
1 2 1
```

Salida:

```
output
4
```

Explicación:

- Espera: No es necesario esperar p = 5.
- Paso 0: p = 4.
- Espera: Espera de 2 unidades de tiempo p = 5.
- Paso 1: p = 3.
- Espera: Espera de 2 unidades de tiempo p = 4.
- Paso 2: p = 3.

Tiempo total de espera: $2 + 2 = 4$.

Ejemplo 3

```
input
5 3 2 6
1 1 2 1 1
```

```
output
6
```

Explicación:

- Espera: No es necesario esperar p = 6.
- Paso 0: p = 5.
- Espera: No es necesario esperar p = 5.
- Paso 1: p = 4.
- Espera: No es necesario esperar p = 4.
- Paso 2: p = 2.
- Espera: Espera de 3 unidades de tiempo p = 3.
- Paso 3: p = 2.
- Espera: Espera de 3 unidades de tiempo p = 3.
- Paso 4: p = 2.

Tiempo total de espera: $3 + 3 = 6$.

Ejecución

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

```
./puffle 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 ./puffle 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.

Problema 2: Red de Entrenamiento de Card-Jitsu

Los Maestros de *Card-Jitsu* se han reunido para formar una red de entrenamiento y mejorar sus habilidades en los elementos de fuego, agua y nieve. Hay N maestros numerados del 1 al N, cada uno con conocimientos especiales en una técnica. Para fortalecerse en todos los aspectos del *Card-Jitsu*, los maestros necesitan compartir sus técnicas y aprender unos de otros. Cuando dos maestros se conectan, ambos comparten sus conocimientos. Si un tercer maestro se conecta a cualquiera de ellos, también podrá aprender todas las técnicas que ya comparten. Así, una vez que un maestro ha aprendido una técnica, cualquier otro puede obtenerla a través de la red de conexiones establecida.

Entrada

La entrada se compone de varias líneas:

- La primera línea contiene un entero N, el número de maestros.
- La segunda línea contiene un entero L, el número de posibles sesiones de entrenamiento.
- Las siguientes L líneas contienen tres enteros cada una: maestro1, maestro2 y costo. Estos enteros indican que organizar una sesión de entrenamiento entre maestro1 y maestro2 tiene un costo específico. Las conexiones son bidireccionales.

Salida

La salida debe ser un solo entero que representa el costo mínimo total necesario para que todos los maestros estén interconectados. Si no es posible conectar a todos los maestros, devuelve -1.

Ejemplos

input

```
3
3
1 2 5
1 3 6
2 3 1
```

output

```
6
```

Explicación: Podemos elegir las siguientes conexiones: maestro 1 y 2 con costo 5, maestro 2 y 3 con costo 1. De esa forma, todos los maestros están conectados con costo total de 6.

input

```
4
3
1 2 3
3 4 4
2 3 1
```

output

```
-1
```

Explicación: Aunque tenemos algunas conexiones, no hay forma de conectar a todos los maestros de Card-Jitsu, incluso si usamos todas las conexiones disponibles. Así que es imposible e imprimimos -1.

Ejecución

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

```
./cardjitsu 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 ./cardjitsu 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.

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. Describe cómo sería una solución utilizando un enfoque "*Greedy*".
  2. Investigue brevemente qué es la subestructura óptima de una solución. ¿Por qué este problema sigue la subestructura óptima? ¿Es esto útil para resolver este problema?
- Parte 2
  1. Describe y justifica brevemente tu solución, puedes apoyarte de algún algoritmo visto en clases.
  2. Explica en qué casos no es posible conectar a todos los maestros y de qué manera puedes chequearlo en código.
  3. Explica la complejidad temporal de tu solución en notación Big O.
- 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 Overleaft.

No se aceptarán informes de más de 2 planas (sin incluir bibliografía), informes ilegibles, o generados con inteligencia artificial **(Descuento de 6 puntos en el informe)**.

Evaluación

La nota de tu tarea se calculará a partir de pruebas de input/output, así como un informe en LaTeX que explique el modelamiento del problema, las estructuras de datos empleadas, y la lógica utilizada para resolverlo. La ponderación se descompone de la siguiente manera:

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[1]. De lo contrario, recibirás 0 puntos en ese test.

Además, se proporciona un archivo `.devcontainer` como el entorno estándar para esta tarea. Si tu programa falla 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. Asimismo, 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 en busca del uso de inteligencia artificial en la creación de las soluciones. Las tareas en que se identifique el uso de IA serán consideradas como una infracción a la política de honestidad académica y tendrán las consecuencias correspondientes.

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