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

IIC2133 — Estructuras de Datos y Algoritmos
2023 - 1

Tarea 3

Fecha de entrega código: 22 de Junio del 2023 23:59 UTC-4
Link a generador de repos: Generar repo base

Objetivos

- Aproximas soluciones de problemas NP-Hard
- Aplicar técnicas algorítmicas sobre Grafos
- Desarrollar estrategias algorítmicas codiciosas y de programación dinámica

Material Útil (Cápsulas)

- Repositorio Talleres (Cápsulas)
- DFS y Grafos en C
- Heaps y Colas de prioridad en C
- Cápsula Programación Dinámica en C

Parte 1: Backpack

Introducción: Súper spy backpack

El gran regreso de Tow-Mate, luego de la crítica situación que sufrió en radiador Spring cuando intentaron quitarle su bello color oxidado, se da cuenta de que no puede seguir en el negocio del espionaje, es por esto, que llama a Finn McMissile, su jefecito, el cual le indica que no puede salirse del espionaje sin antes completar su última misión temeraria. Esta misión consiste en infiltrarte en una organización criminal que planea sabotear el Grand Prix Mundial, la competición más prestigiosa del mundo.

Para ello, mate deberá viajar por diferentes países, con un grupo de espías, incluyéndote. De esta manera, irá recolectando ciertos artículos, pero repartiendo cargas entre todos los agentes. Es por esto, que tú como un brillante programador propones abordar la misión con estrategias algorítmicas como programación dinámica o codiciosas, y así resolver el problema de la mochila, en donde se debe elegir los objetos más útiles para cada situación sin sobrepasar el peso máximo que puedes llevar. Por ejemplo, en Tokio tendrás que escoger entre una cámara espía, un GPS, un lanzagranadas o un paracaídas; en París entre una pistola láser, un disfraz, un detector de mentiras o un sombrero; y en Londres entre unas gafas de visión nocturna, un micrófono, una llave inglesa o una taza de té.

Figura 1: Mate hablando con Jefasos

Problema: Backpack

Entonces, para completar la misión tendrás que determinar la forma de repartir una cantidad de ítems de distinto peso en la mínima cantidad de agentes, con una capacidad establecida.

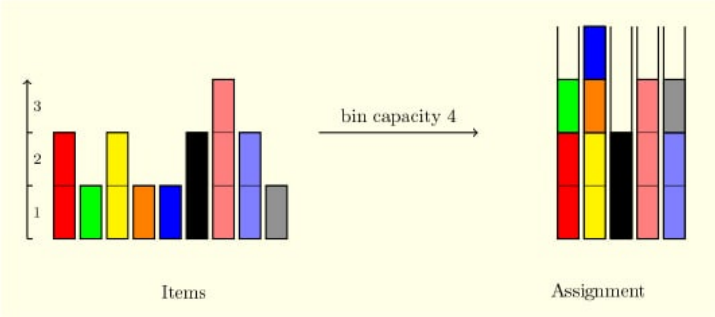

Figura 2: Ilustración del problema

Por ejemplo, en la figura 2 se muestra que existen 9 ítems de distinto peso, donde cada agente es capaz de llevar un peso de 4 como máximo. Un sub-óptimo de este problema consiste en repartir los ítems a 5 agentes.

Indicaciones generales

Para resolver este problema debes usar programación dinámica o programación codiciosa. En específico, el método de programación dinámica consiste en dividir el problema en subproblemas más pequeños y resolverlos de forma óptima, aprovechando las soluciones previas. Así, vas calculando el valor máximo que puedes obtener con cada cantidad de peso disponible y cada conjunto de objetos que encuentras. Luego, eliges el objeto que te aporte más valor sin exceder el peso restante en tu mochila. Repites este proceso hasta que completas tu equipamiento para cada misión.

Aclaraciones

- La capacidad de todos los espías es la misma
- El peso de cada ítem siempre es menor o igual a la capacidad del agente
- El problema siempre poseerá solución

Ejecución

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

```
./backpakMate input.txt output.txt
```

Input

El input está estructurado como sigue:

- Una línea con el valor C que indica la capacidad de los agentes.
- Una línea con el valor N que indica la cantidad de ítems.
- N líneas donde cada una indica el peso de cada ítem, donde deberás asignarle un id único desde 0 a $N-1$

Por ejemplo, el siguiente input sería válido (los # son *comentarios*)

```
4 # capacidad de los agentes
9 # cantidad de ítems
2
1
2
1
1
2
3
2
1
```

Output

Tu output deberá consistir de una línea que indique la cantidad (Q) de agentes que soluciona el problema óptimo, seguido de Q líneas indicando los id de los ítems que lleva cada uno.

Por ejemplo, un output válido sería el siguiente (los # son *comentarios*)

```
5 # cantidad de agentes
0 1 # el agente lleva el ítem de id 0 y id 1
2 3 4
5
6
7 8
```

Evaluación Parcial

Al ser un problema de resolver un problema NP-Hard. Es válido encontrar suboptimos en pro de mejorar el tiempo obtenido. Para esto existe la siguiente forma de evaluación por test

- 7.0 Si el resultado es menor o igual al subóptimo propuesto

- 5.0 Si el resultado está sobre el subóptimo propuesto. Pero no más del 20 %
- 4.0 Si el resultado está sobre el subóptimo propuesto. Pero no más del 30 %
- 1.0 en Otro caso

Código Base

No habrá código base como tal, solo se entregará un makefile y un archivo main.c para esta parte. Es tu deber encargarte de la lectura del archivo y de procesar los inputs entregados.

Parte 2 - Cobertura mínima

Introducción

Una vez armada la mochila de Mate, llega la hora de enfrentar a los enemigos en Grand Prix Mundial. En el momento que mate se dispone a llegar a la competencia recibe una llamada de Finn el que le dice que, el famoso Chick Hicks, se volvió rebelde dado que el mundo de las carreras lo ha decepcionado, y por ende, ha tomado a todos los autos en la competencia como rehenes, y amenaza con hacerlos explotar a todos si es que mate se demora más de 1 hora en traerle una pizza de peperoni del little Cars, ya que, villano que come, villano buena onda.
Para esto, Mate debe seguir la ruta específica por todas las pizzerías little Cars y así asegurarse de encontrar haber comprado la mejor pizza para Chick.

Figura 3: Chicks Malveke

Problema - Cobertura Mínima

Para el problema, debes encontrar la ruta perfecta para que el mate, pueda recorrer todas las pizzerías **Hint**. Las ubicaciones de estas serán nodos.
Para completar este objetivo tendrás que determinar la forma de generar la conexión de costo mínimo entre todas las *ubicaciones* y luego de esto disminuir la cantidad excesiva de conexiones por ubicación asegurándote de que mantengas la propiedad de costo mínimo de tus conexiones. Para realizar lo anterior tendrás que obtener un MST y luego equilibrar la cantidad de aristas de los nodos del MST.

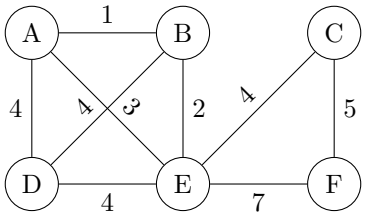

Figura 2: Grafo original

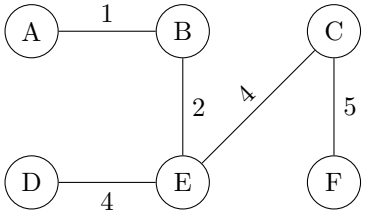

Figura 3: MST

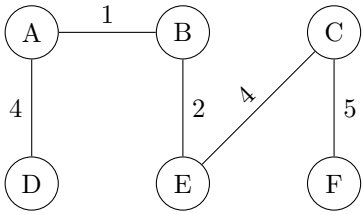

Figura 4: MST de menor conexión

Por ejemplo, en la figura 2 se muestra el grafo original. Luego, tanto la figura 3 como la figura 4 muestran MST's distintos para el mismo grafo. Sin embargo, se observa que la **cantidad de aristas del nodo** E de la figura 4 es menor y, por tanto, se prefiere el MST de la figura 4. Nota que para efectos ilustrativos el grafo original NO es completo, pero el grafo que tú considerarás sí lo será.

Acercamiento sugerido

A continuación se muestra una serie de pasos sugeridos para afrontar el problema

- Algoritmo para construir un MST
- Algoritmo para minimizar el máximo de aristas salientes desde algún nodo.

Aclaraciones

- La distancia entre los nodos está definida por la distancia manhattan. Es decir, sea $d(a, b)$ la distancia y $a, b \in \mathbb{N} \times \mathbb{N}$, entonces:

$$d(a, b) \;=\; |(a_1 - b_1)| + |(a_2 - b_2)|$$

Ejecución

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

```
./matest input.txt output.txt
```

Input/Output

Archivo de Input

El input está estructurado como sigue:

- Una línea con el valor N que indica el número de nodos
- N líneas donde cada una indica la ubicación de cada nodo

Por ejemplo, El siguiente input sería válido

```
6
1 5
2 5
7 7
1 1
3 3
7 3
```

Que indica que existen **6** nodos, ubicados en las posiciones $(1, 5), (2, 5), (7, 7), \ldots$

Archivo de Output

Tu output deberá consistir de una línea que indique el costo del MST, seguido de $N-1$ líneas que corresponden a las aristas del MST en formato a_1 a_2 b_1 b_2, que indica que existe una arista entre el punto a y b. Por ejemplo, un output válido para MST sería el siguiente

```
16
1 5 2 5
2 5 3 3
1 1 3 3
7 7 7 3
3 3 7 3
```

Podemos notar que el nodo $(3, 3)$ tiene más aristas que los demás nodos. Un output válido para el MST mejorado en este caso sería el siguiente

```
16
1 5 2 5
2 5 3 3
1 5 1 1
7 7 7 3
3 3 7 3
```

Evaluación

Este problema al poseer distintas válidas. Se realizará una evaluación por Test de la siguiente forma
Sea G_{max} el grado máximo de un nodo (Cantidad de aristas que llegan a un nodo)

- 0 puntos si el grafo no es un MST
- 0.7 puntos por resolver el MST sin ninguna optimización (G_{max} en el peor caso)
- 0.9 Si la solución está entre G_{max} base y G_{max} de la solución
- 1 punto si la solución es mejor o igual a la propuesta ($G_{max} >= G_{propuesto}$)

Código Base

No habrá código base como tal, solo se entregará un makefile y un archivo main.c para esta parte. Es tu deber encargarte de la lectura del archivo y de procesar los inputs entregados. Puedes utilizar código de tareas anteriores o de cápsulas.

Documento de diseño

Para facilitar futuras recorrecciones y además dar un tiempo para pensar la tarea previo a empezar a desarrollarlo. Esta tarea contiene en su ponderación un "Documento de diseño". Este documento se evalúa de forma binaria, donde obtiene todo el puntaje si está presente (Conteniendo la información mínima solicitada) y 0 en caso contrario.

Contenidos mínimos

El documento no es nada formal, solamente posee el objetivo de incentivar que la tarea sea analizada y pensada antes de ejecutarla. Es por esto que debe contener como mínimo 3 secciones

1. Análisis del enunciado: Un breve análisis del enunciado, identificando variables y dominio del problema
2. Planificación de solución: una idea a rasgos generales sobre como abordaran el problema, idealmente mencionando nombres de las funciones y structs a utilizar
3. Diagrama: Un diagrama/imagen donde indique graficamente como se relacionan las distintas funciones y structs que implementaran/implementaron (Como una función llama a otra, que contiene el struct)

Entrega documento de diseño

El documento y todos los anexos correspondientes deben ir en la carpeta `docs/` del repositorio.
Los documentos entregados deben ser solo en formatos `.pdf, .txt, .html, .md, .png, .jpg, .jpeg`

General

La idea de este documento no es que sea algo formal ni muy detallado, sino una forma que dejen constancia del proceso de pensamiento de revisar una Tarea. De esa forma los puede ayudar a detectar problemas en su código antes de llevar muchas horas programando. Además, este documento nos ayudará mucho al momento de recorregir. Debido a esto es que se le dará una prioridad mayor a aquellas tareas con documento presente. Es recomendable realizar los primeros dos puntos del documento antes de empezar a programar, ya que puede ser de gran utilidad para resolver el problema

MEMEstructurín: haz reír a les ayudantes y profes, y te damos décimas

Espacio en el que podrás crear un meme con temática dentro de una de las 3 categorías que te presentamos a continuación:

- Tu meme favorito de toda la vida con un twist propio
- chascarro de clases
- chascarro favorito de tu vida

Gracias a este cuestionario (que igual puedes considerarlo como un espacio de avance en la tarea, pues te da décimas, pero más chistoso) puedes optar hasta **3 décimas**. Es muy importante que notes que el meme **debe** ser elaborado por ti. Memes descargados de internet o provenientes de screenshots no serán considerados.
Para acceder al formulario haz click acá. Recuerda que este formulario cierra al mismo momento que la tarea

Evaluación General

La nota de tu tarea es calculada a partir de tests. Separando entre la Parte 1 y 2 de la tarea

- 10 % Documento de diseño
- 45 % Parte 1: Backpack
- 45 % Parte 2: MST

Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos y utilizar menos de 1 GB de ram[1]. De lo contrario, recibirás 0 puntos en ese test.

Recomendación de tus ayudantes

Las tareas requieren de mucha dedicación de tiempo generalmente, 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 y te dediques a entender de manera profunda lo que te pedimos. Una vez que hayas comprendido el enunciado, dedica el tiempo que sea necesario para la planificación y modelación de tu solución, para posteriormente poder programar de manera eficiente. Estos son consejos de tus ayudantes que te pueden ayudar a pasar el ramo :)

Entrega

Código: GIT - Rama master del repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Política de atraso: La política de atraso del curso considera dos criterios:

- Utilización de **cupones**[2] de atraso. Donde un cupón te proporciona 24 horas adicionales para entregar tu tarea sin penalización a tu nota
- Entrega atrasada sin cupón donde se aplica un descuento *suave*[3]. Calculada de la forma

$$N_f = min(70 - 7 \cdot d, N_o)$$

  Donde d es la cantidad de días atrasados **con un máximo de 4 días**[4], N_f la nota final y N_o la nota obtenida en la tarea

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.

[1] Puedes revisarlo con el comando htop o con el servidor
[2] Recuerda que solo tienes 2 cupones para todo el semestre
[3] Es un descuento a la nota máxima posible
[4] Luego de eso la nota máxima obtenible es un 1.0