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

IIC2133 — Estructuras de Datos y Algoritmos
2023 - 2

Tarea 3

Fecha de entrega código: 23 de Noviembre del 2023 23:59 UTC-4
Link a generador de repos: https://classroom.github.com/a/1JCPPgg7

Objetivos

- Emplear diferentes paradigmas de diseño de algoritmos.
- Aplicar técnicas algorítmicas sobre Grafos

Material Útil

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

Figura 1: El *death*line de la T3 acercándose

Parte 1 - Fiestas y Oro

Introducción

El Gato con Botas se ha convertido en un héroe con todo lo que ha logrado durante sus aventuras. La gente está muy agradecida, a tal punto que le entregan donaciones con solo verlo. Como gran ~~empresario~~ gato, decide sacarle el máximo de esto, organizando fiestas en diferentes pueblos para obtener donaciones. Los gobernadores de los pueblos no están felices, por lo que al realizar una fiesta, no le dejarán realizar otra en donde la realizó y en los pueblos vecinos.

Figura 2: Una fiesta tranquila donde se ganó oro

Entonces, el Gato con Botas buscará entonces obtener la mayor cantidad de oro posible, considerando cuanto podría ganar en cada pueblo y que no podrá repetir la fiesta en el pueblo y sus vecinos.

Problema

Se tendrá p pueblos, donde en cada uno se podrá ganar $g_i \in G$ oro ($0 \leq i < p$). Cuando se hace una fiesta en i, el pueblo i, los dos pueblos anteriores ($i-2$ e $i-1$) y los dos siguientes ($i+1$ e $i+2$) no permitirán una fiesta adicional. Se busca obtener la mayor cantidad de oro que se podrá recaudar, dado la lista de oro G que indica cuanto se podrá ganar en cada pueblo en particular.

Ejecución

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

```
./party input.txt output.txt
```

Input

El input está estructurado como una línea que indica la cantidad de pueblos p y una línea con la lista G que indica el oro g_i que se podría ganar en cada pueblo. Por ejemplo, el siguiente input sería válido:

```
6
1 2 4 0 3 0
```

Output

Tu output deberá ser solo la cantidad máxima de oro que se podrá ganar. Por ejemplo, el input anterior daría output 5.

Parte 2 - Puntos de articulación en un grafo

Introducción: Conexiones mágicas

Gran Jack Horner, con el mapa a la estrella de los deseos, desea prepararse para ir lo más equipado para asegurar su deseo. Para eso, decide empaquetar varias herramientas mágicas para asistirlo en su meta. Dado que la magia es complicada (necesidades de guion), se tiene que todos los elementos que tendrá que llevar están conectados mágicamente. Las conexiones no necesariamente son directas, pueden pasar por otros elementos, y siempre se tendrá una secuencia de elementos que conecte todo par de elementos distintos.

Figura 3: Jack con su cartera de herramientas mágicas

Jack necesita identificar cuáles son las herramientas mágicas críticas, que son las que al removerlas (1 a la vez), al menos una conexión entre herramientas mágicas se pierde. Además, necesita saber que tan mágicamente separadas quedan las otras herramientas, para ver cuáles tendrían un peor efecto si es que llegan a fallar o perderse.

Problema - Puntos de articulación en un grafo

Llamamos **punto de articulación** en un grafo a un nodo, tal que, al extraerlo a él y a los caminos a los que este está conectado, notamos que aumenta el número de componentes conexas dentro del grafo (i.e. separa el grafo en dos o más grafos independientes). Así, su misión se enmarca en lograr **encontrar a todos los puntos de articulación** que posea el grafo entregado.

Veamos un ejemplo:

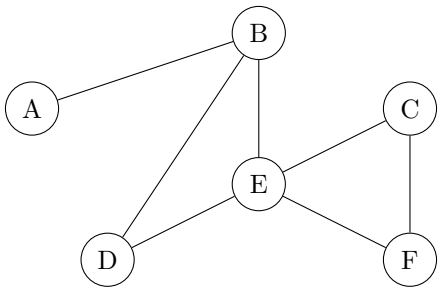

Figura 1: Ilustración del problema

En este caso, tenemos dos puntos de articulación. El primero es el nodo E. Notemos que al extraerlo logramos aumentar el número de componentes conexas:

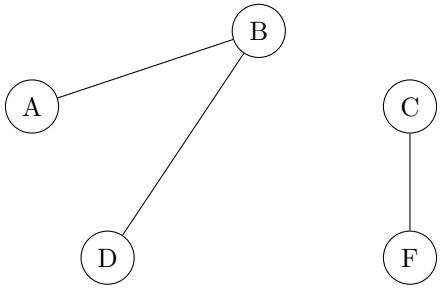

Figura 2: Extracción nodo E forma dos componentes conexas

Para encontrar el segundo punto de articulación volvemos a mirar el grafo original. Momento en el cual podemos notar el nodo buscado es el B. Lo confirmamos entonces extrayéndolo del grafo y viendo como aumenta el número de componentes respecto al original:

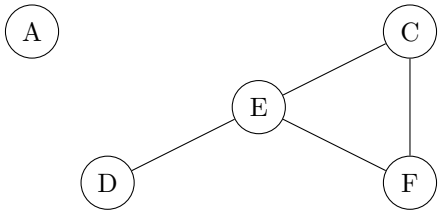

Figura 3: Extracción nodo B forma dos componentes conexas

Aclaraciones

- Al extraer un punto de articulación se pueden formar **dos o más** componentes conexas. El ejemplo es un caso particular donde en ambos casos se crearon dos.
- Siempre tienen buscar puntos de articulación en el **grafo original**. De lo contrario, pueden estar encontrando puntos de articulación nuevos, que se formaron al remover el punto de articulación previamente encontrado, que no corresponden al grafo original.
- Se te entregará un grafo no dirigido.

Indicaciones generales

Solo para que lo tengan en cuenta, este problema se puede resolver por fuerza bruta. Esto es, extraer cada nodo y luego contar el número de componentes que genera. La idea es que este no sea el acercamiento que ustedes utilicen, pues para grafos grandes su ejecución será muy costosa. De hecho, será del orden de $\mathcal{O}(V(V+E))$.

¿Cómo nos podemos aproximar a resolver el problema de buena manera entonces? Será **DFS** el algoritmo base que debes usar para esta entrega, gracias a lo cual podremos lograr una complejidad base del orden $\mathcal{O}(V+E)$, bastante mejor que el caso anterior.

HINT: Entender la idea de las aristas de retorno o BackEdges dentro del contexto de un DFS.

Ejecución

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

```
./magic input.txt output.txt
```

Input

El input está estructurado como sigue:

- Una línea con el valor N que indica el número de nodos.
- Una línea con el valor E que indica el número de aristas.
- E líneas donde cada una línea indica una arista entre 2 nodos.

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

```
7    # N nodos
8    # E aristas
1 3
1 2
4 3
2 3
4 2
2 0
4 5
4 6
```

Notemos que en el ejemplo anterior, los nodos se han identificado por número de 0 al 6 al tener $N = 7$.

Output

Tu output deberá consistir de una línea que indique la cantidad (A) de nodos de articulación encontrados, seguido de A líneas que indiquen el id del nodo de articulación encontrado y la cantidad de componentes conexas que genera.

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

```
2    # A puntos de articulación
4 3  # Remover nodo id=4 genera 3 componentes
2 2
```

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.

Parte 3 - Cobertura mínima

Introducción: Ruta entre puntos de interés

Ricitos de Oro y los tres osos pueden, con la posesión de un mapa mágico, logran identificar por donde tendrán que pasar para poder obtener la estrella de los deseos. Entre ellos, deciden que la mejor estrategia sería identificar que caminos usar para obtener la cobertura total de los puntos de interés, con la menor distancia que tendrán que conocer y resguardar. Es decir, luego de pasar lentamente la primera vez por un camino, podrán devolverse muy rápidamente dado que ya conocen como es el camino.

Figura 7: Mapa a destinos a cubrir

Para que puedan lograr esto, tienen que identificar cuáles son los caminos (o aristas) que se tendrán que considerar en su travesía, por lo que te piden a ti, el que Mejor Sabe Todo.

Problema - Cobertura Mínima

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.

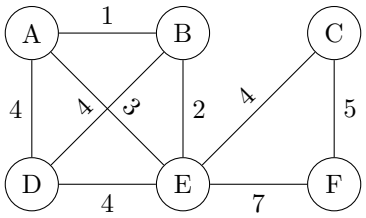

Figura 2: Grafo original

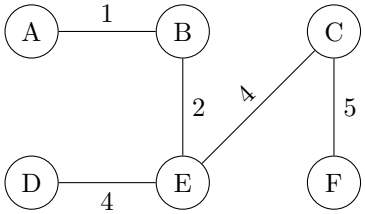

Figura 3: MST

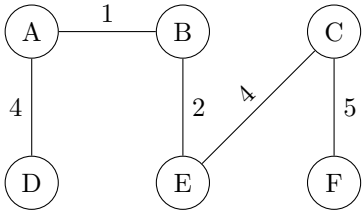

Figura 4: MST de menor conexión

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 `paths` que se ejecuta con el siguiente comando:

```
./paths 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 ($G_{max} \geq G_{propuesto}$)

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

- 10 % Documento de diseño
- 20 % Parte 1: PD
- 35 % Parte 2: Articulación
- 35 % Parte 3: Cobertura mínimca

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. Te recomendamos aprovechar la distribución del puntaje que tiene cada parte y esto para ir sumando nota de la tarea.

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