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

IIC2133 — Estructuras de Datos y Algoritmos
2022 - 2

Tarea 3

Fecha de entrega código: 25 de Noviembre
Link a generador de repos: Ir a github

Objetivos

- Aplicar y optimizar técnicas codiciosas aplicado a grafos
- Analizar posibles tradeoff entre tiempo de ejecución y suboptimo encontrado
- Resolver problemas NP-Hard con técnicas algorítmicas

Introducción: El ultimo Jaja

Gracias a tu ayuda, Batman pudo finalmente capturar a *Riddler* y poner fin a su distorsionada forma de demostrar su intelecto. Por lo que el momento llegó, solo queda un criminal en DCCiudad Gótica que separa a Batman de su jubilación. Joker, Alias *El Jajas*, Alias *El Coringa*. El mayor genio criminal desde la invención de las galletas con pasas.

Figura 1: Imagen inedita de Joker.

Luego de hacer muchas críticas a la sociedad (vivimos en una sociedad) y muchos post "sarcásticos" en redes sociales, Joker descuido su meticulosa rutina como administrador de página de memes y cometió el error fatal de todo mod de Discord de salir de su casa. En ese momento, los sistemas avanzados de detección de Batman (que tu optimizaste en la T1) entraron en acción y avisaron sobre la ubicación de este genio criminal. En el momento dado, Batman prepara una emboscada para atrapar a Joker, pero era una trampa y Joker desaparece riendo de tus habilidades de entender estructuras de datos. Batman, quien confia mucho en ti, queda estupefacto y empezó a darse cuenta de que respiro un gas con una potente neurotoxina que

causa que olvides todos tus conocimientos del ramo estructuras de datos y algoritmos (claramente no como ustedes).

Parte A - Cobertura Minima 2.0: Batinodos

Considerando lo anterior ustedes son su única salvación[1], debes encontrar la ruta perfecta para que el antídoto, que hizo Alfred con los batimilliones, pueda llegar por todas las células del cuerpo. **Hint**. Las células son 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.

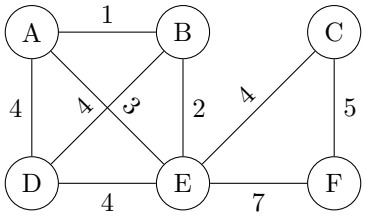

Figura 2: Grafo original

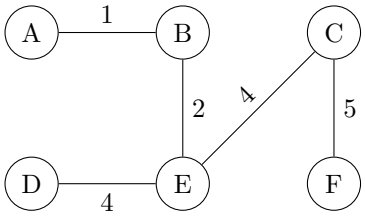

Figura 3: MST

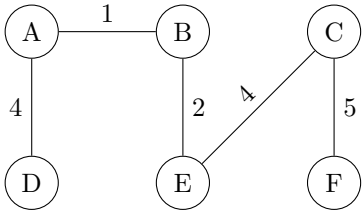

Figura 4: MST de menor conexión

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 tu considerarás sí lo será.

Acercamiento sugerido

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

- Algoritmo para realizar sorting de aristas
- Algoritmo para construir un MST
- Algoritmo para minimizar el maximo de aristas salientes desde algun nodo.

[1] Porque Yadran está de vacaciones en el universo

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 `mst` que se ejecuta con el siguiente comando:

```
./mst input.txt output.txt
```

Evaluación

La evaluación es por test y se desglosa de la siguiente forma:

- 7.0 Si el resultado es mejor o igual al subóptimo esperado
- 6.0 Si el resultado es a lo más un 20 % inferior al subóptimo esperado
- 5.5 Si el resultado es a lo más un 35 % al subóptimo esperado
- 4.0 Si el resultado solo resuelve el MST sin introducir mejoras
- 1.0 Si el resultado no es un MST

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
```

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.

B) It's Morbin Time

Luego de eliminar la toxina del cuerpo de Batman, es momento de volver a la ofensiva. Lamentablemente, Joker se atrincheró en una fábrica con cientos de rehenes y bombas con temporizador conectadas a cada rehén. Sin embargo, es un juego retorcido, este temporizador no está configurado en un tiempo en específico, sino que tiene exactamente la cantidad mínima de pasos que Batman tiene que tomar para llegar a dicha fábrica (más el tiempo de desactivar las bombas). Por lo que nuevamente, como el fiel amigo de Batman, te encargas de modelar un programa que encuentre el camino más corto desde la posición de Batman hasta la fábrica donde Joker está atrincherado.

Problema

Para resolver la situación, deberás implementar algún algoritmo que te permita encontrar el costo del camino más corto entre dos nodos de un grafo no direccional[2]. En el siguiente ejemplo se muestra un grafo donde se busca el camino más corto entre el nodo A y el nodo C.

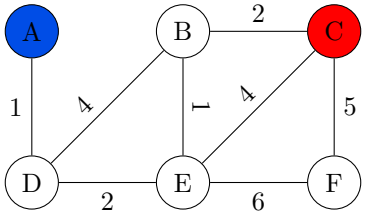

Figura 2: Grafo de ejemplo

Se observa que el camino más corto de solución es:

$$A \xrightarrow{1} D \xrightarrow{2} E \xrightarrow{1} B \xrightarrow{2} C$$

Que posee un costo de **6**. Es importante notar que este no es el camino con menos nodos, ya que dicho camino sería el:

$$A \xrightarrow{1} D \xrightarrow{2} E \xrightarrow{4} C$$

Que aún siendo el camino con menos nodos, su costo total es de **7**.

s **Nota**: Para este problema se recomienda utilizar el algoritmo de *Djikstra* aunque igualmente puedes utilizar el algoritmo que prefieras

Aclaraciones

- Un grafo podría tener dos caminos diferentes que tengan exactamente el mismo costo. Para consideraciones de la tarea da lo mismo el que elijan, ya que se pide solo entregar el costo total.
- Un grafo podría presentar ciclos, por lo que es importante detectarlos y evitar caer en ciclos infinitos.

[2]Si A está conectado con B, entonces B está conectado con A

Ejecución

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

```
./greedy input.txt output.txt
```

Evaluación

La evaluación es por test y se desglosa de la siguiente forma:

- 7.0 Si el resultado encontrado corresponde al esperado
- 5.5 Si el resultado encontrado es a lo más un 10 % superior al esperado
- 4.5 Si el resultado encontrado es a lo más un 20 % superior al esperado
- 1.0 En otro caso

Input/Output

Archivo de Input

El input está estructurado como sigue:

- Una línea con el valor V que indica el número de nodos numerados de 0 a $V-1$
- Una línea con el valor E que indica el número de vértices
- Una tupla XY de ids, que indican el nodo de partida y el nodo de término para el camino
- E líneas donde cada una indica una tupla `IdNodoI IdNodoJ X` que indica que el nodo de id I esta conectado al nodo de id J con una arista de peso X

Por ejemplo, El siguiente input sería válido

```
4
0 3
5
1 3 4
0 2 1
1 2 3
0 3 6
1 0 8
```

Aclaraciones del input

- Los nodos no necesitan tener posición, ya que solo importan los costos de viajar entre cada par
- Los Ids de nodos van numerados desde 0 a $V-1$
- Todos los costos serán enteros positivos

Restricciones del input

$$\begin{aligned} 2 \leq \quad & V \quad \leq 10^9 \\ 1 \leq \quad & E \quad \leq \frac{V(V-1)}{2} \end{aligned}$$

Archivo de Output

El output deberá ser solo un número que indica el costo mínimo encontrado para el camino. En un formato de entero. Por ejemplo:

```
5
```

Para evitar problemas asociados a un input muy corto se recomienda utilizar la función `fprintf`

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.

Cápsulas y Consideraciones

Para esta tarea se recomienda revisar las cápsulas del curso asociadas a grafos y una próxima a ser publicada de heaps binarios

- Para ambas partes de la tarea es recomendable ver la cápsula de DFS y Grafos
- Para la sección de camino más corto se subirá una cápsula sobre Heap Binario y Colas de prioridad (Se enviará anuncio cuando esté publicada)

Evaluación

La nota de la tarea se desglosa como:

- 50 % Nota parte MST.
- 50 % Nota parte Camino mas corto.

Importante: 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[3]. 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 :)

Uso de memoria

Parte de los objetivos de esta tarea es que trabajen solicitando y liberando memoria manualmente. Para evaluar esto, usaremos *valgrind*. Se recomienda fuertemente ver los videos de este repositorio.

Para asegurarte que no tengas errores de memoria debes correr tu programa con:
`valgrind ./mst input.txt output.txt`

y el output debe contener
"`All heap blocks were freed -- no leaks are possible`" y
"`ERROR SUMMARY: 0 errors from 0 contexts`"

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:

[3]Puedes revisarlo con el comando htop o con el servidor

- Utilización de **cupones**[4] de atraso. Donde un cupón te proporciona 24 horas hábiles adicionales para entregar tu tarea sin penalización a tu nota
- Entrega atrasada sin cupón donde se aplica un descuento *suave*[5]. Calculada de la forma

$$N_f = min(70 - 0{,}7 \cdot d^{1{,}3}, N_o)$$

Donde d es la cantidad de días atrasados, 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.

[4] Recuerda que solo tienes 2 cupones para todo el semestre
[5] Es un descuento a la nota máxima posible