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

IIC2133 — Estructuras de Datos y Algoritmos
2022 - 1

Tarea 3

Fecha de entrega código: 24 de Junio del 2022

Objetivos

- Modelar un problema utilizando grafos.
- Diseñar algoritmos basados en estrategias codiciosas.
- Diseñar algoritmos basados en el esquema de programación dinámica.
- Encontrar soluciones para problemas NP-hard.

Misión 1:

Luego de detectar y lograr atrapar a los infiltrados, has descubierto que el próximo plan de Stuart Little es escapar por las alcantarillas (que resulta ser un grafo completo). Para esto se te han asignado dos misiones, la primera misión se trata de reducirle la movilidad al fugitivo y poder atraparlo más fácilmente, por lo que se te ocurrió quedarte con una cantidad reducida de alcantarillas teniendo en cuenta que no debes cortar el flujo de agua en ninguna intersección o generarás problemas en sectores de la ciudad, Stuart Little descubrirá el plan y decidirá escapar por otros medios.

Figura 1: Stuart Little y sus secuaces intentando escapar

Luego de implementar esta parte de la misión, te has dado cuenta de que hay algunas intersecciones con muchas alcantarillas, generando algunos problemas que a la larga ponen en peligro el plan, por lo que deberás encontrar la manera de solucionar esto.

1. Problema: Rata de dos patas... ¿Dónde estás?

Para completar la misión 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.

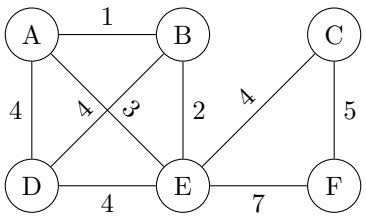

Figura 2: Grafo original

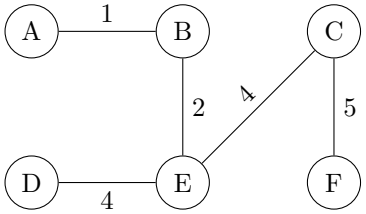

Figura 3: MST

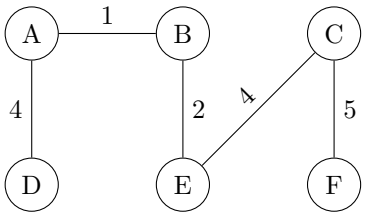

Figura 4: MST de menor conexión

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.

1.1. Indicaciones generales

A continuación se muestra una serie de pasos que tu programa en C deberá contener:

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

1.2. 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:
$$d(a, b) \;=\; |(a_1 - b_1)| + |(a_2 - b_2)|$$
- No puedes usar librerías que te faciliten la implementación ni el diseño de tus algoritmos. Por ejemplo, no puedes utilizar librerías que contengan sorting. Ademas, no podras usar qsort que viene en las librerias estandar.

1.3. Ejecución

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

```
./sewerrat input.txt output.txt
```

1.4. Evaluación

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

- 7.0 Si el resultado es mejor o igual a la pauta
- 6.0 Si el resultado es a lo mas un 20 % inferior a la pauta
- 5.5 Si el resultado es a lo mas un 35 % inferior a la pauta
- 4.5 Si el resultado solo resuelve el MST
- 1.0 Si el resultado no es un MST

1.5. 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
```

1.6. 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
```

1.7. 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.

Misión 2:

La segunda parte de la misión consiste en preparar a los agentes que participarán de la persecución. Para esto buscamos disponer de la menor cantidad de agentes con la finalidad de no entorpecer el plan. Sin embargo, queremos además llevar todas las herramientas que disponemos y un agente no puede llevar todo.

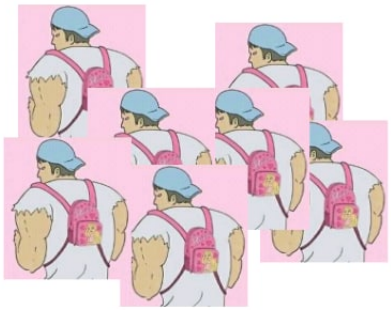

Figura 4: Grupo de agentes en camino a la persecución.

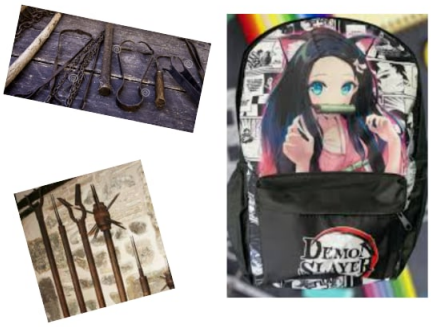

Figura 5: Debes llenar la mochila de cada agente con esas herramientas

2. Problema: La batalla final

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.

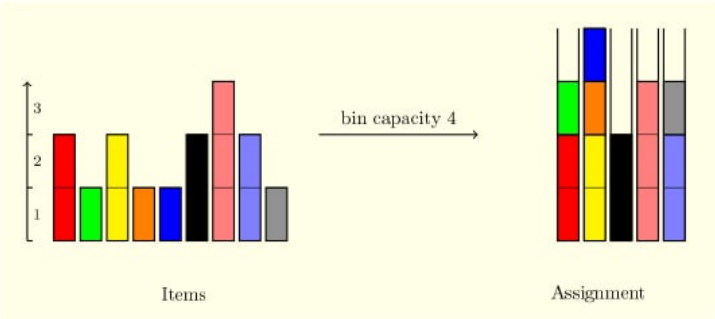

Figura 6: Ilustración del problema

Por ejemplo, en la figura 6 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.

2.1. Indicaciones generales

Para resolver este problema debes usar programación dinámica.

2.2. Aclaraciones

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

2.3. Ejecución

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

```
./agentbin input.txt output.txt
```

2.4. Evaluación

La evaluacion es por test y las notas por cada test se desglosan como:

- 7.0 Si el resultado es menor o igual al subóptimo propuesto.
- 5.0 Si el resultado está por sobre el sub-óptimo propuesto, pero solo se aleja un 20 % de este.
- 4.0 Si el resultado está por sobre el súb-óptimo propuesto, pero solo se aleja hasta en un 30 % de este.
- 1.0 En otro caso

2.5. 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
```

2.6. 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
```

2.7. 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.

Consideración Importante

Tu repositorio será creado de la siguiente forma

```
src\
    sewerrat\
    agentbin\
```

Donde las carpetas sewerrat y agentbin encontrarás los main.c para trabajar los problemas de manera individual. Además al hacer make se generarán 2 ejecutables

- sewerrat: Ejecutable del Problema 1 (tu solución)
- agentbin: Ejecutable del Problema 2 (tu solución)

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 implementen en la práctica el *trade-off* entre memoria y tiempo, es por esto que independiente del test, tendrán como máximo `1.2 GB` de memoria `RAM` disponible. Pueden revisar la memoria que utiliza su programa con el comando `htop`. Además, el servidor avisará en caso de superar el máximo permitido.

Eficiencia

Se solicita que el programa sea eficiente. No podrá tomar más de 10 segundos `user + sys`, pueden utilizar `time` seguido del comando de ejecución de su programa para revisarlo.

Evaluación

No hay informe, solo se evaluará tu código en C. La nota de tu tarea se descompone como se detalla a continuación:

- El output de tu programa sea correcto y eficiente. Para esto el puntaje se divide en partes iguales para los tests Easy, Medium y Hard.
    - 60 % a la nota para el Problema 1
    - 40 % a la nota para el Problema 2

Además, para esta tarea no existe un output único, depende de su implementación, por lo que se corroborrá con un script.

Bonus

Los siguientes bonus solo aplican si tu nota correspondiente es mayor o igual a 4.

Manejo de memoria perfecto: +5 % a la nota de código

Recibirás este bonus si `valgrind` reporta que tu programa no tiene ni leaks ni errores de memoria en los tests que logres resolver.

Entrega

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

Figura 7: El coord feliz de haber capturado a Stuart Little y a [REDACTED]