



## Recorrido Mínimo Uno a Todos - Soluciones

Compilado: 7 de junio de 2024

### 1. Policías

#### Problema

La nueva reglamentación de una ciudad establece que toda esquina debe estar a lo sumo a 5 cuadras de una estación de policía. Dada la lista de esquinas  $\{v_1, \dots, v_n\}$  de la ciudad, la lista  $\{p_1, \dots, p_k\}$  de esquinas donde hay policías<sup>a</sup>, y la lista  $E$  de calles, debemos indicar si la normativa se cumple, y en caso contrario cuáles esquinas son las que quedan “desprotegidas”.







<sup>a</sup>O sea, asumimos que la policía siempre se ubica en una esquina.

#### 1.1. Resolución



Primero, modelemos el problema con un grafo  $G$ , donde  $V(G) = \{v_1, \dots, v_n\}$  (las esquinas) y  $E(G) = E$  (las calles). Traduzcamos el problema a este grafo: necesitamos encontrar los vértices que estén a distancia mayor a 5 de algún vértice del conjunto  $P = \{p_1, \dots, p_k\}$ .

Podríamos probar primero haciendo un BFS desde cada uno de los vértices  $v$  de  $V(G)$ , encontrando para cada uno la distancia mínima a un vértice de  $P$ . Si la mínima distancia de  $v$  a un vértice de  $P$  es mayor a 5, sabemos que esa esquina está desprotegida. Esto tendría complejidad  $O((n + m) * n)$ , siendo  $n = |V(G)|$  y  $m = |E(G)|$ .

Pero podemos hacer mejor que esto. En vez de tomar las distancias desde cada uno de los vértices, tomamos las distancias desde las estaciones de policía. Y una cosa a notar es la siguiente: no nos importa a qué estación de policía están más próximas las esquinas, sino solamente si tienen una a menos de 6 cuadras. Si pudiésemos de alguna manera correr BFS desde cada estación de policía al mismo tiempo, estaríamos marcando todos los vértices a distancia 1 de cualquier estación, después a distancia 2, etc.

Bueno, esto sí se puede hacer: agregamos un vértice “fantasma”  que sea adyacente a todos los vértices con estación de policía, y empezamos el BFS desde ahí. Así, vamos a marcar todos los vértices con la distancia a . Como todo camino desde  tiene como segunda posición una estación de policía, cada camino de distancia  $d$  entre  y otro vértice  $v$  contiene un camino de distancia  $d - 1$  entre una estación de policía y  $v$ . Por lo tanto,  $v$  está a distancia como máximo  $d - 1$  de alguna estación de policía. Por otro lado, un vértice  $v$  está a distancia  $d - 1$  de una estación de policía sólo si está a distancia  $d$  de  (¿Por qué?). Luego, restarle uno a las distancias de cada vértice a  resulta en las distancias a alguna estación de policía.

Resumiendo entonces:

1. Definimos el grafo  $G = (\{v_1, \dots, v_n\}, E)$  como lista de adyacencias. Complejidad:  $O(n + m)$ .
2. Definimos el grafo  $G'$  como el resultado de agregar un vértice  a  $G$  adyacente a todos los vértices  $\{p_1, \dots, p_k\}$ . Complejidad:  $O(k)$ .
3. Corremos BFS en  $G'$  desde , guardando las distancias a cada vértice. Complejidad:  $O(n + m)$ .
4. Restamos 1 a todas las distancias. Complejidad:  $O(n)$ .



5. Si no hay vértices con distancia mayor a 5, decimos que la normativa se cumple. De lo contrario, enumeramos los vértices con distancia mayor a 5. Complejidad:  $O(n)$ .

Complejidad total:  $O(n + m + k) = O(n + m)$ .

## 2. Martín y los Mares

### Problema

Martín es un conocido comerciante marítimo. Planea cruzar el Mar Tortuñoso empezando desde la isla *Anastasia*, parando en algunas islas para comerciar y descansar, y llegando a la isla *Betuna*. Sabe que la mayor amenaza a su barco no son los piratas: son los percebes que se enganchan y pudren la madera. También sabe qué camino entre las piedras debe usar para pasar de una isla a otra, y tiene tan estudiadas las rutas que sabe exactamente cuántos percebes van a engancharse a su barco en cada ruta.

En algunas de esas rutas, sin embargo, ocurre lo contrario: tortugas marinas vienen y se comen un porcentaje de los percebes que tenga el barco. Lamentablemente, Martín sabe que es muy peligroso pasar por dos de estas rutas, ya que las tortugas a veces muerden de más. Por lo tanto, quiere pasar por máximo una sola de las rutas con tortugas.

Martín cursó TDA pero hace muuucho tiempo, y entonces les pregunta a ustedes cuál es la mínima cantidad de percebes con la que puede alcanzar el otro lado del Mar Tortuñoso.

### 2.1. Resolución

Por simplicidad, llamemos  $A$  y  $B$  a *Anastasia* y *Betuna*, respectivamente. Podríamos empezar modelando un digrafo  $D$  con vértices  $A$ ,  $B$ , y las islas intermedias, y con aristas correspondiendo a las rutas, con costo  $c(u, v)$  para los que suman percebes, y con porcentaje  $t(u, v)$  (representado como fracción) para los que sacan un porcentaje. Uno entonces pensaría quizá encontrar el camino mínimo desde  $A$  hasta  $B$  con algún algoritmo que conozcamos. Esto tiene dos problemas:

1. Los costos no son todos sumados al costo de un camino: algunos son un producto por una fracción.
2. El camino puede pasar por máximo una arista de tortugas, y no estamos restringiendo eso de ninguna manera acá.

Tenemos que pensar un poco más.

Antes de poder usar alguno de los algoritmos que conocemos tenemos que pensar qué hacemos con las aristas de tortugas, que me dividen. En principio vamos a sacarlas y después vemos cómo las agregamos nuevamente. Trabajamos sobre el digrafo  $D'$  que es igual al  $D$  sacando las aristas con tortugas.

Usando Dijkstra vamos a aplicar una técnica que se usa bastante para este tipo de problemas: en lugar de correr una sola vez el algoritmo de camino mínimo vamos a correrlo dos veces. Pero, ¿desde qué nodos? Estamos intentando llegar de un vértice  $A$  a otro  $B$  así que seguramente tengamos que



correrlo desde  $A$ . ¿Podemos correr Dijkstra desde  $B$ ? Quizás así nomás no pero si damos vuelta la orientación de las aristas del digrafo (trasponemos el digrafo) puede servir. Veámoslo de a pasitos:

Primero corremos Dijkstra desde  $A$ , por lo que obtenemos los pesos de los caminos desde  $A$  a todos los nodos del digrafo  $D'$ . Luego trasponemos el digrafo y hacemos un Dijkstra pero desde  $B$ , lo que nos da los pesos de todos a  $B$  en el digrafo  $D'$ . Pero, ¿para qué nos sirve esto? En este momento es cuando vuelven a aparecer las aristas que sacamos, ya que para cada arista  $(u, v)$  que sacamos podemos calcular cuál sería el costo mínimo de un recorrido que tiene a esa arista. La cuenta que hacemos para cada arista va a ser:  $w(A, u) * (1 - t(u, v)) + w(v, B)$  (siendo  $w(\cdot, \cdot)$  el peso del camino mínimo y  $t(\cdot, \cdot)$  el porcentaje removido en una arista tortuga). Recordemos que el costo de las aristas con tortugas es el porcentaje de percebes que comen, por eso no es solo sumar el costo de la arista.

Repasemos los pasos del algoritmo:

1. Corremos Dijkstra desde  $A$ . Complejidad:  $O(\min\{m \cdot \log(n), n^2\})$ .
2. Trasponemos el digrafo. Complejidad:  $O(m + n)$ .
3. Corremos Dijkstra desde  $B$ . Complejidad:  $O(\min\{m \cdot \log(n), n^2\})$ .
4. Para cada “arista tortuga” calculamos el peso del recorrido que la usa con  $w(A, u) * (1 - t(u, v)) + w(v, B)$  y devolvemos el camino con menor peso. Complejidad:  $O(m)$ .

Entonces la complejidad del algoritmo es  $O(\min\{m \cdot \log(n), n^2\} + n)$ .

## 2.2. Demostración de correctitud

Para la demostración queremos ver que el algoritmo devuelve la solución óptima, donde la solución óptima y lo que devuelve el algoritmo son:

**Solución:**

$$\min(\{\text{Pesos de todos los caminos que no tienen aristas tortuga}\}, \{\text{Pesos de los caminos con exactamente una arista tortuga}\}).$$

Para facilitar la notación será  $\min(P_{\text{sin} \text{ 🐢}}, P_{\text{con} \text{ 🐢}})$ .

**Algoritmo:**

$$\min(\{\text{Peso mínimo de camino sin aristas tortuga}\}, \{w(A, u) * (1 - t(u, v)) + w(v, B) \mid (u, v) \text{ es una arista tortuga}\}).$$

Para facilitar la notación será  $\min(P'_{\text{sin} \text{ 🐢}}, P'_{\text{con} \text{ 🐢}})$ .  $P'_{\text{sin} \text{ 🐢}}$  debo considerarlo para el caso que convenga pasar por 0 aristas tortuga.

Ver que  $\min(P_{\text{sin} \text{ 🐢}}) = \min(P'_{\text{sin} \text{ 🐢}})$  es trivial ya que  $\min(P_{\text{sin} \text{ 🐢}})$  es exactamente el peso mínimo de camino sin aristas tortuga, pero ¿cómo veo que  $\min(P_{\text{con} \text{ 🐢}}) = \min(P'_{\text{con} \text{ 🐢}})$ ?

Lo vamos a probar viendo que  $\min(P'_{\text{con} \text{ 🐢}}) \geq \min(P_{\text{con} \text{ 🐢}})$  y que  $\min(P'_{\text{con} \text{ 🐢}}) \leq \min(P_{\text{con} \text{ 🐢}})$ :

- $\min(P'_{\text{con} \text{ 🐢}}) \geq \min(P_{\text{con} \text{ 🐢}})$ : como en  $P'_{\text{con} \text{ 🐢}}$  tenemos los pesos de los caminos con una arista tortuga tal que los costos de  $A$  a  $u$  y de  $v$  a  $B$  son mínimas, sabemos que estará incluido en los



pesos de todos los caminos que tienen exactamente 1 arista tortuga (aunque no tengamos los mínimos) entonces  $P'_{\text{con}} \subseteq P_{\text{con}}$ .

Además, sabemos que  $P'_{\text{con}} \subseteq P_{\text{con}} \implies \min(P'_{\text{con}}) \geq \min(P_{\text{con}})$  pues como todo elemento en  $P'_{\text{con}}$  está en  $P_{\text{con}}$ , el mínimo de  $P_{\text{con}}$  será al menos tan chico como el mínimo de  $P'_{\text{con}}$  que tiene menos elementos, con lo que probamos que  $\min(P'_{\text{con}}) \geq \min(P_{\text{con}})$ .

- $\min(P'_{\text{con}}) \leq \min(P_{\text{con}})$ : tomemos una arista tortuga  $u \rightarrow v$ . Sabemos que  $w(A, u)$  es mínimo y  $w(v, B)$  también, por lo que  $w(A, u)$  es menor o igual al peso de cualquier camino de  $A$  a  $u$  que no use aristas tortuga, y  $w(v, B)$  es menor o igual al peso de cualquier camino de  $v$  a  $B$  que no use aristas tortuga. Sabemos que cualquier costo de un camino  $P$  entre  $A$  y  $B$  que use sólo la arista tortuga  $u \rightarrow v$  se puede calcular como

$$(\text{costo del subcamino de } P \text{ de } A \text{ a } u) * (1 - t(u, v)) + (\text{costo del subcamino de } P \text{ de } v \text{ a } B). \quad (1)$$

La ecuación (1) medio que sale por definición<sup>1</sup>. Veamos que el costo de  $P$  es mayor o igual a  $w(A, u) * (1 - t(u, v)) + w(v, B)$ . Supongamos que es menor. Llamemos  $w_P(A, u)$  al costo del subcamino de  $P$  de  $A$  a  $u$ , y  $w_P(v, B)$  al costo del subcamino de  $P$  de  $v$  a  $B$ . Entonces:

$$\begin{aligned} w_P(A, u) * (1 - t(u, v)) + w_P(v, B) &< w(A, u) * (1 - t(u, v)) + w(v, B) \\ &\Updownarrow \\ w_P(A, u) * (1 - t(u, v)) &< w(A, u) * (1 - t(u, v)) + w(v, B) - w_P(v, B) \\ &\leq w(A, u) * (1 - t(u, v)) \quad (w(v, B) \leq w_P(v, B)) \end{aligned}$$

Llegamos a que  $w_P(A, u) * (1 - t(u, v)) < w(A, u) * (1 - t(u, v))$ . Como  $w_P(A, u) \geq w(A, u) \geq 0$ , esto es un **ABSORDO!** que vino de suponer que  $P$  tenía menor costo que  $w(A, u) * (1 - t(u, v)) + w(v, B)$ .

Ahora, el camino con costo  $\min(P_{\text{con}})$  pasa necesariamente por una arista tortuga  $u \rightarrow v$ , y tendrá costo mayor o igual a  $w(A, u) * (1 - t(u, v)) + w(v, B)$ . Este costo pertenece a  $P'_{\text{con}}$ , y por lo tanto  $\min(P'_{\text{con}})$  será menor o igual a este costo. Finalmente,  $\min(P'_{\text{con}}) \leq \min(P_{\text{con}})$ .

$\implies$  Solución = algoritmo.

### 3. Manuel y los Monstruos

#### Problema

Manuel estaba aburrido en la clase de TDA, así que se puso a programar un juego. Quiere hacer que el jugador pase por distintos mundos pasando por portales unidireccionales, matando los monstruos de los mundos por los que pasa. Cada portal  $i$  por el que logra pasar le da  $p_i \in \mathbb{R}$  puntos. El objetivo del juego es llegar desde el mundo  $m_1$  hasta el mundo  $m_n$  con la mayor cantidad de puntos posible.

A Manu le preocupa que el juego esté balanceado, así que quiere saber cuál es la máxima

<sup>1</sup>Ejercicio: demostrar la ecuación 1 por inducción en la cantidad de aristas del camino.



cantidad de puntos que puede generar un jugador dada la cantidad de mundos, los portales que hay, y cuántos puntos da cada portal. Además, quiere saber cuál es la mínima cantidad de portales que podría estar tomando el jugador que hiciese la máxima cantidad de puntos posible.

### 3.1. Resolución

Para empezar, tenemos que pensar cómo **modelar el problema utilizando grafos**. Una posibilidad sería que los mundos sean representados como nodos, y los portales de un mundo a otro como una arista dirigida de uno a otro. Es decir,  $V(G) = \{m_1, \dots, m_n\}$  y  $E(G) = \{(m_i, m_j) \mid \exists \text{ portal del mundo } m_i \text{ al } m_j\}$ . Además, a cada arista  $e_i$  le asignamos un peso  $p_i$  correspondiente a la cantidad de puntos que se obtiene en el juego al pasar por dicho portal.

El problema nos pide dos cosas: encontrar el puntaje máximo, y decidir cuál es la mínima cantidad de portales que podría haber tomado un jugador que obtuvo puntaje máximo. Concentrémonos en principio en el primero de estos problemas.

Pensemos un par de cosas antes de arrancar:

- ¿Qué sería una partida desde el punto de vista del digrafo? Sería un recorrido  $R$  que empieza desde el nodo  $m_1$  y termina en el  $m_n$ .
- ¿El puntaje máximo siempre va a ser un número finito? Si en algún punto hubiera un ciclo de peso positivo, un jugador podría tomar ese ciclo cuantas veces quiera y llegar a conseguir una puntuación arbitrariamente grande. En caso de que no haya ciclos con esta característica, sí podríamos concluir que la puntuación máxima va a ser un número finito.  
Es decir, la puntuación de las partidas no está acotada si y solamente si existe un ciclo  $C$  de longitud positiva alcanzable por  $m_1$  y que luego alcanza  $m_n$  (esto habría que demostrarlo).
- Si hubiera aristas que salen de  $m_n$ , estas nunca pertenecerían a ningún juego ya que al llegar a  $m_n$  el mismo termina. Por lo tanto podemos asumir sin pérdida de generalidad que no hay aristas de este estilo.

#### Demostración

La puntuación máxima de las partidas no está acotada  $\Leftrightarrow$  existe un ciclo  $C$  de longitud positiva alcanzable por  $m_1$  que luego alcanza  $m_n$ .

$\Rightarrow$ )

Supongamos que la puntuación máxima no está acotada, o sea, para todo  $s \in \mathbb{N}$  existe un recorrido entre  $m_1$  y  $m_n$  cuya puntuación es mayor a  $s$ . Tomemos un  $s$  gigante tal que no haya camino simple que pueda sumar eso. Por ejemplo, tomemos que  $s$  es la suma de todas las aristas positivas del digrafo. Como cada arista no puede aparecer más de una vez en un camino, es claro que ningún camino puede puntuar más que  $s$ . Por lo tanto, todo recorrido  $R$  que puntúe más que  $s$  repetirá alguna arista. Con esto es suficiente para decir que  $R$ , siendo un recorrido cualquiera que puntúe más que  $S$ , tiene un ciclo (¿Por qué?).

Supongamos que  $R$  no contiene ningún ciclo positivo. Es decir, todos los ciclos que contiene suman algo positivo o 0. Cada ciclo  $C$  adentro de  $R$  comenzará y terminará en el mismo vértice, digamos  $v$ . Podemos entonces tomar como  $R'$  el resultado de eliminar el ciclo  $C$  de  $R$ , y  $R'$  será



un recorrido válido. Como  $R$  debe tener una cantidad finita de ciclos, y eliminar un ciclo de un recorrido no genera nuevos ciclos, si realizamos esta operación hasta no poder más, el resultado será un camino. Además, como ningún ciclo es positivo, eliminar cada ciclo de  $R$  resulta en una puntuación mayor o igual a la anterior. Al final, entonces, tendremos un camino con puntuación mayor o igual a la de  $R$ . Pero  $R$  tenía una puntuación mayor a  $s$ , y por lo tanto el camino también la tendrá. **ABSRDO!**, ya que dijimos antes que ningún camino podría tener puntaje mayor a  $s$ . ¿Y dónde está lo que supusimos que nos llevó a la contradicción? Está en que dijimos que  $R$  no tenía ningún ciclo positivo. Por lo tanto,  $R$  sí tiene un ciclo positivo. Además, este ciclo positivo es alcanzable desde  $m_1$  y alcanza  $m_n$ , porque hay un recorrido  $R$  que va de  $m_1$  a  $m_n$  que lo alcanza. Finalmente, el ciclo pertenece al digrafo, y entonces llegamos al consecuente.

⇐)

Si existe un ciclo  $C$  de longitud positiva alcanzable por  $m_1$  que luego alcanza  $m_n$ , entonces un jugador podría partir de  $m_1$ , y tomar el ciclo tantas veces como quiera para luego ir a  $m_n$  y terminar el juego. Por lo tanto, podría llegar con un puntaje arbitrariamente grande, y la puntuación no estaría acotada.  $\square$

Entonces, traduciendo, vamos a querer encontrar el camino máximo entre el nodo  $m_1$  al  $m_n$  en un digrafo que podría tener ciclos positivos.

Pero, ¿cómo hacemos para encontrar el **camino máximo**? Nosotros conocemos algoritmos para encontrar caminos mínimos, no máximos. Lo que podemos intentar entonces es modificar el modelo para que estos algoritmos solucionen el problema. ¿Cómo? Multiplicamos todos los pesos por “-1”, y ahora el que antes fuera el peso máximo, pasará a ser el peso mínimo. Además, los ciclos de peso positivo ahora serán ciclos de peso negativo.

¡Genial! Ahora buscamos ciclos negativos y si no los hay devolvemos el camino mínimo y ya, ¿no?...

Más o menos, porque si corriéramos los algoritmos que conocemos, cualquier ciclo negativo que hubiera nos frenaría la búsqueda del camino máximo. Sin embargo a nosotros **solo nos van a interesar algunos ciclos negativos**: los que se alcancen desde  $m_1$ , y luego alcancen a  $m_n$ .

Luego nos queda resolver el problema de cómo hacer para considerar solamente estos ciclos. Lo que podemos hacer es eliminar todos los nodos que nunca puedan alcanzar a  $m_n$  (es decir, no tienen un camino a  $m_n$ ). ¿Cómo lo hacemos? Damos vuelta todas las aristas y corremos DFS desde  $m_n$ . Todos los nodos que no logremos alcanzar los borramos.

Ahora sí, finalmente, la **solución** para nuestro problema va a ser correr un algoritmo de camino mínimo sobre el grafo con los pesos modificados como dijimos. Ahora bien ¿Qué algoritmo podemos usar? BFS no va a poder ser porque hay pesos distintos. Dijkstra tampoco, porque los pesos son todos negativos. No queda opción más que usar Bellman-Ford, y de paso podemos avisarle a Manuel cuando su juego está permitiendo que el puntaje máximo no tenga cota.

Resumiendo, la solución es:

1. Armar el grafo.
2. Multiplicar los pesos por -1.
3. Eliminar todos los nodos que no llegan a  $m_n$ .
4. Ejecutar Bellman-Ford desde  $m_1$  y devolver infinito si encuentra un ciclo, y en caso contrario el valor opuesto a lo que devuelva Bellman-Ford.



Pasemos ahora a la segunda parte del problema. Queremos saber cuál es la mínima cantidad de portales que podría estar tomando el jugador que hiciese la máxima cantidad de puntos posible. Esta pregunta solo tiene sentido en el caso en el que no hubiera ciclos de puntaje positivo en algún camino entre  $m_1$  y  $m_n$ , ya que sino no existiría tal puntaje máximo.

Para ello nos sería muy útil tener una noción de cuáles son todos los caminos posibles de puntaje máximo, y luego quedarnos con el más corto de todos ellos. ¿Cómo podemos conseguir algo así?

Lo que podemos hacer es obtener el “DAG de caminos mínimos” del grafo que representa nuestro problema. Esto es, un digrafo acíclico que contenga solamente las aristas que pertenezcan a algún camino mínimo desde un nodo  $s$  a un nodo  $t$ . En este caso nos interesaría el DAG de camino mínimo de  $m_1$  a  $m_n$ .

¿Cómo lo conseguimos? Lo que podemos hacer es lo siguiente:

1. Corremos Bellman-Ford (\*) desde  $m_1$  y obtenemos la distancia mínima de  $m_1$  a todos los nodos, y nos la guardamos en un vector “DistanciaAm $_1$ ”.
2. Corremos Bellman-Ford (\*) desde  $m_n$  en  $G^t$  (el grafo traspuesto) y obtenemos la distancia mínima de todos los nodos a  $m_n$ , y nos la guardamos en un vector “DistanciaAm $_n$ ”.
3. Armamos un DAG con los mismos nodos de  $G$  pero (todavía) sin aristas.
4. Luego, para cada arista  $(u, v) \in E$ 
  - Si  $\text{DistanciaAm}_1[u] + \text{peso}(u, v) + \text{DistanciaAm}_n[v] = \text{DistanciaAm}_1[m_n]$ , quiere decir que  $(u, v)$  pertenece a un camino mínimo de  $m_1$  a  $m_n$  y por ende la agrego al DAG.

(\*) **Observación:** en este caso, como estamos asumiendo que no hay ciclos negativos, en lugar de Bellman-Ford también podríamos usar un algoritmo basado en DAGs (Correr topological sort, y relajar las aristas en ese orden).

¿Cómo hacemos para saber ahora el largo mínimo de un camino máximo? Como ya lo veníamos haciendo: simplemente corremos BFS desde  $m_1$  en el DAG, y nos fijamos la altura a la que quedó  $m_n$ .

#### Demostración

Una arista está en el DAG que construimos  $\Leftrightarrow$  pertenecía a un camino mínimo de  $G$ .

$\Rightarrow$ )

Si una arista  $(u, v)$  está en el DAG es porque  $\text{DistanciaAm}_1[u] + \text{peso}(u, v) + \text{DistanciaAm}_n[v] = \text{DistanciaAm}_1[m_n]$ . Con lo cual existe un camino  $R$  de la forma  $m_1 \rightarrow \dots \rightarrow u \rightarrow v \rightarrow \dots \rightarrow m_n$  cuyo peso es  $\text{DistanciaAm}_1[m_n]$ , es decir que  $(u, v)$  pertenece a un camino mínimo, siendo  $R$  ese camino.

$\Leftarrow$ )

Si  $(u, v)$  pertenece a un camino mínimo, entonces vale  $\text{DistanciaAm}_1[u] + \text{peso}(u, v) + \text{DistanciaAm}_n[v] = \text{DistanciaAm}_1[m_n]$ , y por lo tanto el algoritmo la incluyó en el DAG.

Luego, el DAG efectivamente incluyó todas y solo las aristas que pertenecen a un camino mínimo de  $m_1$  a  $m_n$ . Y al hacer BFS en el DAG desde  $m_1$ , por BFS,  $m_n$  va a tener la mínima distancia posible en el DAG a  $m_1$ , y por ende esa es la mínima distancia posible con un camino



mínimo.

