

## Árbol Generador Mínimo

Eric Brandwein, Tomas Chimenti, Luci Ruz Veloso

DC - FCEN, UBA

16 de Octubre, 2024

UNIVERSIDAD  
DE BUENOS AIRES

## 1 Repaso AGM

## 2 Ejercicios





## Árbol Generador (AG)

Sea  $G$  un grafo. Decimos que  $T$  es un *árbol generador* de  $G$  si se cumplen estas condiciones:

- $T$  es subgrafo de  $G$
- $T$  es un árbol
- $T$  tiene todos los vértices de  $G$

## Árbol Generador (AG)

- $T$  es subgrafo de  $G$
- $T$  es un árbol
- $T$  tiene todos los vértices de  $G$

## Costo de un árbol generador

Sea  $G$  un grafo con pesos en sus aristas. Si  $T$  es un *árbol generador* de  $G$ , definimos el *costo* de  $T$  como la suma de los pesos de las aristas de  $T$ .

## 1 Repaso AGM

Árbol Generador

## Árbol Generador Mínimo

## Camino MaxiMin y MiniMax

## Vínculo entre MiniMax y AGM

## Aplicaciones

Kruskal

Prim

## 2 Ejercicios

## 7 / 74





## 1 Repaso AGM

Árbol Generador

## Árbol Generador Mínimo

# Camino MaxiMin y MiniMax

## Vínculo entre MiniMax y AGM

## Aplicaciones

## Kruskal

Prim

## 2 Ejercicios

## Definición (Camino MaxiMin)

Sea  $G$  un grafo ,  $c : E(G) \rightarrow \mathbb{R}$  su función de costos y  $v, w \in V(G)$ . Decimos que  $P$  es un *camino MaxiMin* de  $v$  a  $w$  si se cumple que  $P$  maximiza  $c_{min}$ :

$$c_{min}(P) = \min\{c(vw) \mid vw \in E(P)\}$$

## Definición (Camino MaxiMin)

Sea  $G$  un grafo,  $c : E(G) \rightarrow \mathbb{R}$  su función de costos y  $v, w \in V(G)$ . Decimos que  $P$  es un *camino MaxiMin* de  $v$  a  $w$  si se cumple que  $P$  maximiza  $c_{min}$ :

$$c_{min}(P) = \min\{c(vw) \mid vw \in E(P)\}$$

También se lo conoce cómo el problema del *camino más ancho*.<sup>a</sup>

<sup>a</sup>Si nos imaginamos los costos como capacidades, entonces queremos maximizar la capacidad que nuestro camino puede trasladar. Y la capacidad de un camino está dada por la capacidad de su arista más pequeña.

## Definición (Camino MiniMax)

Sea  $G$  un grafo ,  $c : E(G) \rightarrow \mathbb{R}$  su función de costos y  $v, w \in V(G)$ . Decimos que  $P$  es un *camino MiniMax* de  $v$  a  $w$  si se cumple que  $P$  minimiza  $c_{max}$ :

$$c_{max}(P) = \max\{c(vw) \mid vw \in E(P)\}$$

## 1 Repaso AGM

Árbol Generador

## Árbol Generador Mínimo

## Camino MaxiMin y MiniMax

## Vínculo entre MiniMax y AGM

## Aplicaciones

## Kruskal

Prim

## 2 Ejercicios

¿Que herramientas de las que vimos creen que podemos utilizar para mostrar que *camino AGM*  $\Rightarrow$  *camino Minimax*?

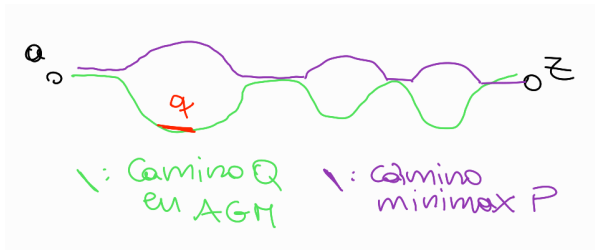
---

¿Que herramientas de las que vimos creen que podemos utilizar para mostrar que *camino AGM*  $\Rightarrow$  *camino Minimax*? ¡Exacto! Podemos usar cualquiera, pero en este caso vamos a utilizar el absurdo.



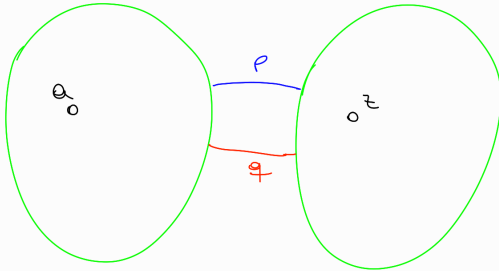
---

Los caminos  $P$  y  $Q$  pueden compartir algunas aristas, pero seguro que no comparten la máxima arista de  $Q$ , porque sino sus máximas aristas serían la misma. Llamemos  $q$  a la arista más grande de  $Q$ .



---

---



- 1  $T^*$  es generador, porque tiene los mismos vértices que  $T$ .
- 2  $T^*$  es conexo, porque  $p$  conecta las dos componentes conexas que resultan de eliminar  $q$  de  $T$ .
- 3  $T^*$  es un árbol, porque tiene la misma cantidad de aristas que  $T$ , y es conexo.

Esto significa que  $T^*$  es un AG. Como  $p$  es más chica que  $q$ , el peso de  $T^*$  es menor que el de  $T$ , y entonces encontramos un AG con peso menor que  $T$ . **ABSURDO**, que vino de suponer que  $T$  es un AGM con un camino que no es minimax.  $\square$

## 1 Repaso AGM

Árbol Generador

## Árbol Generador Mínimo

## Camino MaxiMin y MiniMax

## Vínculo entre MiniMax y AGM

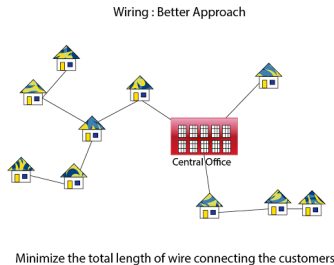
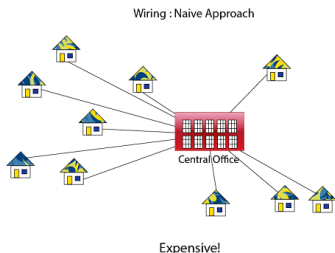
## Aplicaciones

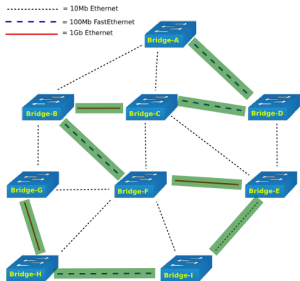
## Kruskal

Prim

## 2 Ejercicios

## Red de electricidad de una ciudad





### Figura 1: STP (Spanning Tree Protocol)



Figura 2: Aproximación para TSP (Traveling Salesman Problem)



## 1 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 2 Ejercicios



## 1 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 2 Ejercicios



## 1 Repaso AGM

## 2 Ejercicios

Viaje en peligro

Audífonos Defectuosos

Alimentando hormigas

Rutas y aeropuertos

## 1 Repaso AGM

## 2 Ejercicios

Viaje en peligro

Audífonos Defectuosos

Alimentando hormigas

Rutas y aeropuertos

# Ejercicio 1

## Viaje en peligro

Cifu vive en Kruskal, Rusia, y quiere visitar las ciudades locales, pero su localidad no tiene rutas que lo conecten con el resto. En total hay  $n$  ciudades, pero el presidente sólo le ofrece construir  $k \ll n$  rutas, poniéndole 2 condiciones:

- 1 Que queden conectadas  $k + 1$  localidades.
- 2 Que la red resultante sea una subred de la red que conecta a todas las localidades con costo mínimo.

Sabemos las ubicaciones de las localidades y que el costo de construir una ruta entre la localidad  $x$  y la  $y$  se calcula como  $r \cdot \text{distEnKm}(x, y) + c_{x,y}$ , donde  $c_{x,y}$  depende de las localidades. Además sabemos en qué localidad se encuentra Cifu. Queremos una red que cumpla lo pedido. ¿Es única?



# Especificaciones

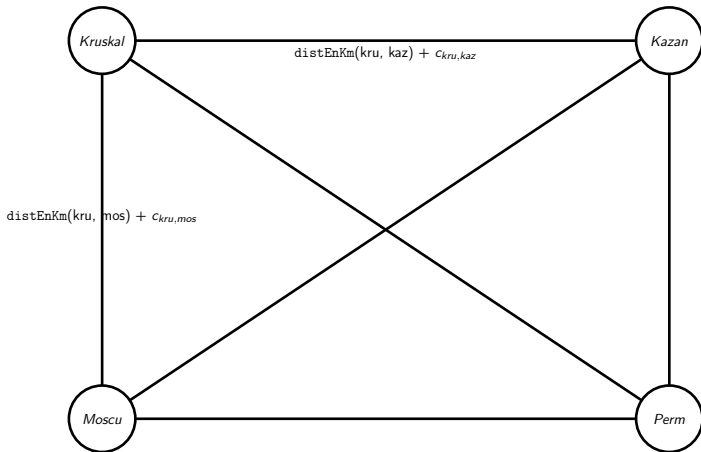
- El formato del input es una línea que contiene dos enteros  $n$  (cantidad de ciudades) y  $k$  (cantidad de rutas). Luego tenemos  $n$  líneas que tienen el formato  $x_i, y_i$ , las cuales representan la posición de la  $i$ -ésima ciudad.
- Tenemos que devolver la red de rutas que cumpla lo pedido.



# Analicemos el problema

- El problema nos pide que queden  $k + 1$  localidades conectadas a partir de  $k$  rutas, por lo que podemos inferir que nos está pidiendo generar un árbol.
- Luego sabemos que  $k \ll n$ , por lo que el árbol que vamos a generar va a ser un subgrafo del grafo completo.
- También nuestro algoritmo va a tener que comenzar desde la localidad de Cifu, puesto que queremos que la misma quede conectada.
- Por último como queremos que el presupuesto utilizado sea el menor posible vamos a modelar este problema utilizando *AGM*.

## Dibujemos un ejemplo



# Solución

- Creamos un grafo con un vértice para cada localidad.
- Calculamos las distancias en kilómetros entre las distintas localidades.
- A partir de las distancias y los costos  $c_{x,y}$  entre las localidades creamos las aristas con sus pesos asociados.  
$$\text{costo}(x, y) = \text{distEnKm}(x, y) + c_{x,y}$$
- Corremos *Prim* (¿por qué no Kruskal?) sobre nuestro grafo para obtener el *AGM* parcial de tamaño  $k$ . ¿Cuál es la complejidad?
- Construir el grafo nos va a costar  $\mathcal{O}(n^2)$  (pues es el grafo completo). Luego si usamos la implementación  $\mathcal{O}(n^2)$  de Prim la complejidad nos va a quedar  $\mathcal{O}(nk)$ . Como  $k \ll n$  la complejidad total es  $\mathcal{O}(n^2)$ . Con  $n =$  cantidad de ciudades.

# Aclaración

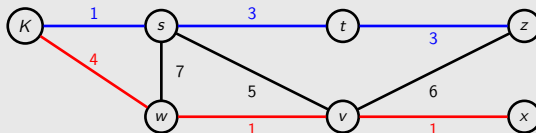
## Invariante de Prim

La consigna pide *que la red resultante sea una subred de la red que conecta a todas las localidades con costo mínimo*. ¿Por qué pide eso y no que sea directamente *la red que contenga a Kruskal que conecta a  $k$  localidades con costo mínimo*? Esto se debe a que el invariante de Prim cumple que, en su  $i$ -ésimo paso, va a tener un subgrafo de tamaño  $i$  del AGM, no el árbol de  $i$  aristas mínimo entre los que contengan a la raíz.

# Aclaración

## Invariante de Prim

En este grafo, por ejemplo, si arrancamos del nodo  $K$  el árbol de  $i$  aristas que nos va a generar Prim (el azul) no va a ser el mínimo (el rojo).



## 1 Repaso AGM

## 2 Ejercicios

Viaje en peligro

**Audífonos Defectuosos**

Alimentando hormigas

Rutas y aeropuertos

## Ejercicio 2

### Audífonos Defectuosos

Sasha vino de intercambio a Exactas y quiere ver cómo llegar desde su hogar hasta Ciudad Universitaria. Sus auriculares marca **AGM** (auriculares generalmente malos) están rotos, por lo que no tiene forma de cancelar el sonido. Conocemos todas las calles que conectan una esquina con otra en el mapa y tenemos una función  $d$  la cual nos dice el volumen de cada calle. Sasha sufre mucho los sonidos fuertes, por lo que quiere encontrar una ruta que minimice el ruido máximo.

Queremos determinar cuál es el máximo nivel de ruido que tiene que soportar para llegar a Ciudad Universitaria desde su casa.

# Especificaciones

- El formato del input es una línea que contiene dos enteros  $C$  (cantidad de calles) y  $E$  (cantidad de esquinas). Luego tenemos  $C$  líneas que tienen el formato  $e_i, e_j, d_{ij}$ , las cuales representan el nivel del ruido de la calle que conecta la esquina  $i$  con la  $j$ .
- Tenemos que devolver el umbral mínimo de tolerancia para Sasha.



## Propiedades a tener en cuenta

- ¿Que propiedad tiene que tener el camino que Sasha quiere encontrar?
- ¿Cómo se llama este tipo de camino?
- ¿Que herramienta tenemos para conocer este tipo de camino?

### Lema

Sea  $T$  un árbol generador de un grafo  $G$  y  $c : E(G) \rightarrow \mathbb{R}$  su función de costo. Entonces,  $T$  es un *AGM* de  $(G, c)$  si y sólo si todo camino de  $T$  es *MiniMax* de  $(G, c)$ .

# Solución

- Creamos un grafo con un vértice correspondiente a cada esquina.
- Agregamos una arista entre las esquinas que tengan una calle que las conecta y le colocamos un costo igual a su nivel de ruido.
- Buscamos el *AGM* del grafo con Prim o Kruskal. ¿Cuál es la complejidad?  $\mathcal{O}(m + n \log n)$ , con  $n = E$  y  $m = C$ .

# Solución

- Nos fijamos la arista de máximo costo del camino *MiniMax* entre la esquina de Sasha y Ciudad Universitaria. Esa es nuestra respuesta.



## Variaciones Ejercicio 2

### Posibles variaciones del ejercicio

- ¿Cómo cambiaría nuestra solución si ahora nos dicen que queremos encontrar los umbrales de tolerancia para todos los lugares (esquinas) a las que puede ir Sasha?

## Variaciones Ejercicio 2

### Posibles variaciones del ejercicio

- ¿Cómo cambiaría nuestra solución si ahora nos dicen que queremos encontrar los umbrales de tolerancia para todos los lugares (esquinas) a las que puede ir Sasha?
  - A partir del *AGM* generado encontramos el camino MiniMax entre todos los nodos. Pues todo camino del *AGM* es MiniMax.
  - Luego nos fijamos para cada nodo cuál es su umbral de tolerancia.

## Variaciones Ejercicio 2

### Posibles variaciones del ejercicio

- ¿Cómo cambiaría nuestra solución si ahora nos dicen que queremos encontrar los umbrales de tolerancia para todos los lugares (esquinas) a las que puede ir Sasha?
  - A partir del AGM generado encontramos el camino MiniMax entre todos los nodos. Pues todo camino del AGM es MiniMax.
  - Luego nos fijamos para cada nodo cuál es su umbral de tolerancia.
- ¿Qué ocurre si ahora Tasha (hermana de Sasha) se duerme si pasa por lugares con ruido menor a cierto umbral y el umbral es un dato? ¿Cómo conseguimos el camino?

# Variaciones Ejercicio 2

## Posibles variaciones del ejercicio

- ¿Cómo cambiaría nuestra solución si ahora nos dicen que queremos encontrar los umbrales de tolerancia para todos los lugares (esquinas) a las que puede ir Sasha?
  - A partir del AGM generado encontramos el camino MiniMax entre todos los nodos. Pues todo camino del AGM es MiniMax.
  - Luego nos fijamos para cada nodo cuál es su umbral de tolerancia.
- ¿Qué ocurre si ahora Tasha (hermana de Sasha) se duerme si pasa por lugares con ruido menor a cierto umbral y el umbral es un dato? ¿Cómo conseguimos el camino?
  - Hacemos lo mismo, solo que ahora conseguimos el Árbol Generador Máximo, es análogo.

# Variaciones Ejercicio 2

## Posibles variaciones del ejercicio

- ¿Cómo cambiaría nuestra solución si ahora nos dicen que queremos encontrar los umbrales de tolerancia para todos los lugares (esquinas) a las que puede ir Sasha?
  - A partir del AGM generado encontramos el camino MiniMax entre todos los nodos. Pues todo camino del AGM es MiniMax.
  - Luego nos fijamos para cada nodo cuál es su umbral de tolerancia.
- ¿Qué ocurre si ahora Tasha (hermana de Sasha) se duerme si pasa por lugares con ruido menor a cierto umbral y el umbral es un dato? ¿Cómo conseguimos el camino?
  - Hacemos lo mismo, solo que ahora conseguimos el Árbol Generador Máximo, es análogo.
- ¿Cómo cambiaría nuestra solución si el grafo resultante es ralo o denso?



## Variaciones Ejercicio 2

### Posibles variaciones del ejercicio

- ¿Cómo cambiaría nuestra solución si ahora nos dicen que queremos encontrar los umbrales de tolerancia para todos los lugares (esquinas) a las que puede ir Sasha?
  - A partir del AGM generado encontramos el camino MiniMax entre todos los nodos. Pues todo camino del AGM es MiniMax.
  - Luego nos fijamos para cada nodo cuál es su umbral de tolerancia.
- ¿Qué ocurre si ahora Tasha (hermana de Sasha) se duerme si pasa por lugares con ruido menor a cierto umbral y el umbral es un dato? ¿Cómo conseguimos el camino?
  - Hacemos lo mismo, solo que ahora conseguimos el Árbol Generador Máximo, es análogo.
- ¿Cómo cambiaría nuestra solución si el grafo resultante es ralo o denso?
  - Si es ralo, entonces nos conviene usar Kruskal; si es denso, nos conviene usar Prim.

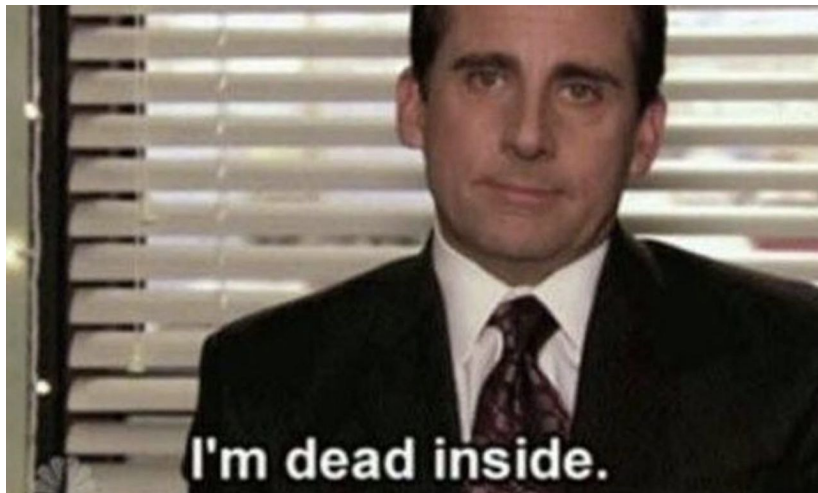
## Bonus track

Tarea: Ahora los auriculares de Sasha le bloquean el sonido pero sólo de 1 calle. ¿Cómo cambia el problema? <sup>1</sup>

---

<sup>1</sup>Spoiler: Se re pica.

# ¿Hacemos un break?



## 1 Repaso AGM

## 2 Ejercicios

Viaje en peligro

Audífonos Defectuosos

Alimentando hormigas

Rutas y aeropuertos

## Ejercicio 3

Martin fue a la NDJ (noche de juegos) y se cruzó con unos estudiantes de Biología. Ahí le contaron el problema que tenían con un hormiguero que estaban intentando mantener.

### Alimentando hormigas

- En el hormiguero hay  $n$  cuevas. La  $i$ -ésima cueva tiene coordenadas  $(x_i, y_i)$ , ninguna tiene comida.
- Hay dos formas de proveer comida a una cueva:
  - 1 Colocarle un tubo de comida.
  - 2 Construir un túnel entre una cueva  $i$  sin comida a otra cueva  $j$  que tenga comida. Para unir dos cuevas  $i, j$  necesitamos  $|x_i - x_j| + |y_i - y_j|$  centímetros de madera.

## Ejercicio 3

### Alimentando hormigas

- Sabemos que construir un tubo de comida cuesta  $T$  y un centímetro de madera cuesta  $M$ .
- ¿Cuál es la manera más barata de que todas las cuevas tengan acceso a la comida?

# Un modelo posible

- Crear un grafo con un vértice correspondiente a cada cueva.

# Un modelo posible

- Crear un grafo con un vértice correspondiente a cada cueva.
- Agregar aristas entre las cuevas con el costo de unir las cuevas con madera.



# Un modelo posible

- Crear un grafo con un vértice correspondiente a cada cueva.
- Agregar aristas entre las cuevas con el costo de unir las cuevas con madera.
- Un *AGM* en este grafo representa una solución al problema, puesto que es indistinto donde colocamos cualquiera de los tubos.

# Un modelo posible

- Crear un grafo con un vértice correspondiente a cada cueva.
- Agregar aristas entre las cuevas con el costo de unir las cuevas con madera.
- Un *AGM* en este grafo representa una solución al problema, puesto que es indistinto donde colocamos cualquiera de los tubos.
- Pero hay un problema...

# Un modelo posible

¡Cuidado! ¡Este modelo no funciona!

Siempre está bueno dibujar algunos ejemplos y pensar casos borde para nuestro algoritmo. En este caso si tenemos dos posibles componentes conexas que estén muy lejos, puede ser mejor colocar dos tubos de comida en vez de uno solo.

# Un modelo posible

¡Cuidado! ¡Este modelo no funciona!

Siempre está bueno dibujar algunos ejemplos y pensar casos borde para nuestro algoritmo. En este caso si tenemos dos posibles componentes conexas que estén muy lejos, puede ser mejor colocar dos tubos de comida en vez de uno solo.

- Entonces puede ser que nos esté faltando agregar información a nuestro grafo. ¿Cómo la podemos agregar?

# Una solución válida

- Crear un grafo con un vértice correspondiente a cada cueva y *crear un vértice adicional, llamémoslo **Tubo**.*

# Una solución válida

- Crear un grafo con un vértice correspondiente a cada cueva y *crear un vértice adicional, llamémoslo **Tubo***.
- Agregar aristas entre las cuevas con el costo de unir las cuevas con madera.

# Una solución válida

- Crear un grafo con un vértice correspondiente a cada cueva y *crear un vértice adicional, llamémoslo **Tubo**.*
- Agregar aristas entre las cuevas con el costo de unir las cuevas con madera.
- Agregar aristas entre cada cueva y el **Tubo** con costo equivalente a instalar un tubo de comida en la cueva.

# Una solución válida

- Crear un grafo con un vértice correspondiente a cada cueva y *crear un vértice adicional, llamémoslo **Tubo**.*
- Agregar aristas entre las cuevas con el costo de unir las cuevas con madera.
- Agregar aristas entre cada cueva y el **Tubo** con costo equivalente a instalar un tubo de comida en la cueva.
- Ahora sí: Un *AGM*  $T$  en este grafo sí que representa una solución al problema.



# Una solución válida

- A partir de las aristas de  $T$  podemos decirles a nuestros amigos de Biología qué hacer. Por cada arista  $(c_i, c_j)$  sabemos que tenemos que colocar un túnel entre la cueva  $i$  y la cueva  $j$ . Luego por cada arista  $(c_i, \mathbf{Tubo})$  sabemos que tenemos que colocar un tubo de comida en la cueva  $i$ .

# Una solución válida

- A partir de las aristas de  $T$  podemos decirles a nuestros amigos de Biología qué hacer. Por cada arista  $(c_i, c_j)$  sabemos que tenemos que colocar un túnel entre la cueva  $i$  y la cueva  $j$ . Luego por cada arista  $(c_i, \mathbf{Tubo})$  sabemos que tenemos que colocar un tubo de comida en la cueva  $i$ .
- Lo que hicimos fue: Problema  $\rightarrow$  Modelado  $\rightarrow$  Algoritmo sobre el grafo  $\rightarrow$  traducción de la salida del algoritmo a la solución del problema.

## 1 Repaso AGM

## 2 Ejercicios

Viaje en peligro

Audífonos Defectuosos

Alimentando hormigas

Rutas y aeropuertos

## Ejercicio 4

Cifu volvio de su estancia de investigacion en Emiratos Arabes. Como había salido tan bien su proyecto el presidente Mohammed bin *Prim* le pidio ayuda.

### Rutas y aeropuertos

Ahora Cifu tiene que conectar todas las ciudades de Emiratos Arabes. Puede poner la cantidad de rutas y aeropuertos que desee. Se puede volar desde una ciudad con aeropuerto a cualquier otra que tenga aeropuerto. Conocemos los costos de colocar una ruta entre la ciudad  $i$  y la  $j$ , que es  $c_{i,j}$ . También conocemos el costo de colocar un aeropuerto en la ciudad  $i$ ,  $a_i$ . Con toda esta información queremos lograr conectar todas las ciudades usando el menor presupuesto posible.

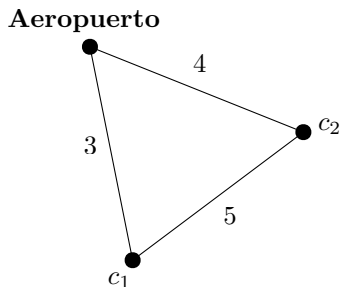
# Analicemos el problema

- Nuestro grafo  $G$  seguramente va a tener a las ciudades como vértices y van a estar conectadas con aristas de costo  $c_{i,j}$ .
- Ahora la pregunta que tenemos es: ¿Cómo modelamos los aeropuertos?

# Analicemos el problema

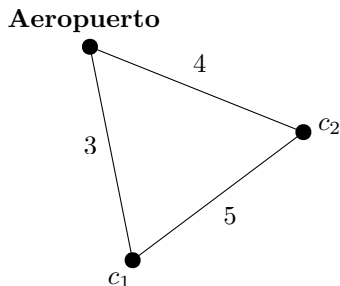
- Nuestro grafo  $G$  seguramente va a tener a las ciudades como vértices y van a estar conectadas con aristas de costo  $c_{i,j}$ .
- Ahora la pregunta que tenemos es: ¿Cómo modelamos los aeropuertos?
- Una opción posible es intentar hacer lo mismo que en el ejercicio anterior. Pero en este caso la relación es distinta, no nos sirve que haya sólo una ciudad con aeropuerto como podía suceder en el caso anterior.

# Analicemos el problema



¿Qué pasa si corremos AGM acá?

# Analicemos el problema



¿Qué pasa si corremos AGM acá? Nos va a decir que la respuesta óptima es agarrar las aristas que conectan con **Aeropuerto**, cuando en realidad la solución óptima del problema era agregar solamente la calle entre  $c_1$  y  $c_2$ .



# Solución

Separemos entonces las posibles soluciones en 2: las que usan algún aeropuerto y las que no. Vamos a encontrar la mejor que no use ningún aeropuerto, y la mejor que use al menos uno, y compararlas.

# Solución

Si una solución no usa ningún aeropuerto, necesariamente tiene que conectar todas las ciudades con rutas. Por lo tanto, debe ser un subgrafo del grafo que contiene solo las rutas como aristas. Como todos los pesos de las rutas son positivos, el AGM va a ser la solución óptima que **no** use aeropuertos.

# Solución

Usemos el modelo  $G'$  en el que agregamos un nodo que represente a todos los aeropuertos, llamémoslo **Aeropuerto**. ¿Qué pasa con el AGM de este grafo?

Supongamos que una solución  $S$  sí usa un aeropuerto. Por cada calle usada en  $S$ , seleccionamos la arista correspondiente en  $G'$ , y por cada aeropuerto agregado a una ciudad, seleccionamos la arista correspondiente que incida en **Aeropuerto**. Al grafo inducido por las aristas que seleccionamos lo llamamos  $H$ .

Sabemos que  $H$  es conexo, porque en  $S$  cada ciudad debe estar conectada a alguna ciudad que tenga un aeropuerto, y todas las ciudades que tienen un aeropuerto están conectadas a **Aeropuerto**. Además, si tenemos una solución  $S$  tal que  $H$  no es un árbol, podemos sacar alguna calle o aeropuerto de la solución y obtener una solución mejor. Por último, un árbol generador de  $G'$  es una solución válida.

Por lo tanto, la solución óptima que usa al menos un aeropuerto es el árbol generador mínimo de  $G'$ .

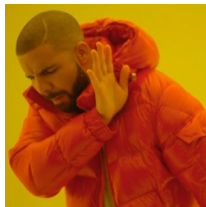
# Algoritmo

- 1 Crear un grafo  $G$  con un vértice correspondiente a cada ciudad.
- 2 Agregar aristas entre las ciudades con el costo de colocar una ruta entre ellas.
- 3 Correr Prim o Kruskal sobre  $G$  para obtener el AGM, y guardar el peso en `pesoSinAeropuertos`.
- 4 Agregar a  $G$  un nodo **Aeropuerto** y agregar aristas entre **Aeropuerto** y cada ciudad con el costo de colocar un aeropuerto en la ciudad.
- 5 Correr Prim o Kruskal sobre  $G'$  y guardar el peso en `pesoConAeropuertos`.
- 6 Devolver  $\min(\text{pesoSinAeropuertos}, \text{pesoConAeropuertos})$ .

---



# Kruskal - Idea



con  
BFS o DFS?



con  
Disjoint-Set

## Disjoint Set

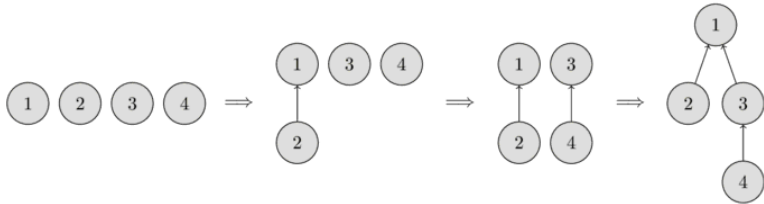
Es una estructura de datos que nos provee las siguientes operaciones eficientemente:

- $find-set(u)$ : Dado un vértice nos dice a qué componente conexa pertenece.
- $union(u,v)$ : Une las componentes conexas a las que pertenecen  $u$  y  $v$ .



## Disjoint Set

Vamos a representar a las componentes conexas en forma de árboles: Cada árbol va a corresponder a una componente conexa. La raíz del árbol representará al representante de la componente conexa. Al principio empezamos con  $n$  nodos donde cada uno es su propia componente conexa, donde el representante es uno mismo. Luego, cuando hacemos la unión entre dos nodos, ambos nodos van a tener el mismo representante. La complejidad del `find()` es  $O(n)$  en peor caso, por lo tanto la complejidad total sería  $O(m * n)$ .



## Disjoint Set

Se suelen implementar dos optimizaciones a esta estructura de datos:

- *Union by rank*
- *Path Compression*


## Union by rank

A cada nodo se le asigna un *rank*, que indica el tamaño de su subarbol mas grande.

- Al principio, cada nodo tiene *rank* 0.
- Cuando se hace una union, si el *rank* de ambos nodos es igual, se incrementa el rank del representante en 1.
- En caso contrario, el nodo con *rank* menor apunta hacia el de *rank* mayor, sin alterar el valor del *rank* del representante.


\_\_\_\_\_

**rank = 1**

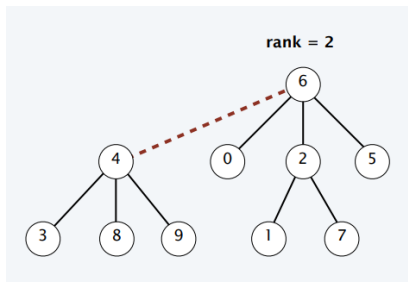


```
graph TD; 4((4)) --- 3((3)); 4 --- 8((8)); 4 --- 9((9));
```

**rank = 2**



```
graph TD; 6((6)) --- 0((0)); 6 --- 2((2)); 6 --- 5((5)); 2 --- 1((1)); 2 --- 7((7));
```



## Union by rank

La complejidad en el peor caso para:

- Find queda  $O(\log n)$
- union queda  $O(1)$ , ya que solamente hacemos apuntar un representante a otro.

Veamos por qué para Find es esa complejidad. La estrategia será ver en que el árbol resultante que nos queda de unir dos componentes conexas, es un árbol que tiene una altura acotada.

9

---

---

## Union by rank

## Lema

*El rank mas alto de un nodo es menor o igual a  $\log_2(n)$*

## Demostración.

Por la proposición anterior, un nodo de *rank*  $x$  tiene a lo sumo  $2^x$ . Sea  $n$  la cantidad de nodos totales dentro del árbol, entonces se cumple que  $2^x \leq n$  que es lo mismo que  $x \leq \log_2 n$ .



## Union by rank

Esto nos dice que el *rank* máximo está acotado por el logaritmo de la cantidad de nodos. Con lo cual, la altura máxima de nuestro árbol es a lo sumo logarítmica. Entonces si queremos buscar el representante de un elemento, basta que recorra todo el árbol hasta la raíz. Por eso nos queda en complejidad  $O(\log n)$ .

# Path Compression

## Path Compression

Empezando desde un nodo  $x$ , ir hasta la raíz para encontrar el representante. En el camino, cambiar los punteros de todos los nodos dentro del camino para apuntar directamente al representante.

A set of small navigation icons typically found in Beamer presentations, including symbols for back, forward, search, and other slide controls.

---