

# BFS, Dijkstra, Greedy y AGM

Luciana Skakovsky, Luciano Ruz Veloso, Agustin Venegas,  
Ezequiel Companeetz

DC - FCEN, UBA

14 de Mayo, 2025



UNIVERSIDAD  
DE BUENOS AIRES

- ① BFS
- ② Dijkstra
- ③ Greedy
- ④ Repaso AGM
- ⑤ Ejercicios AGM

## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

## 5 Ejercicios AGM

## Enunciado del Ejercicio

Sea  $T$  un árbol generador de un grafo conexo  $G$  con raíz  $r$ . Sean  $P$  e  $I$  los vértices a distancia par e impar de  $r$ , respectivamente.

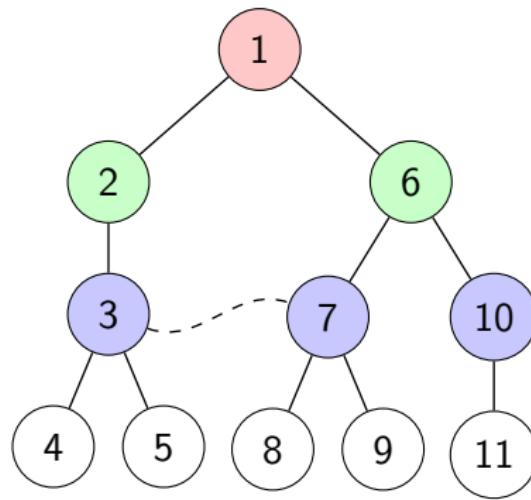
- ① Si existe  $vw \in E(G) \setminus E(T)$  tal que  $v, w \in P$  o  $v, w \in I$ , entonces  $T \cup \{vw\}$  contiene un ciclo impar.
  - ② Si todas las aristas en  $E(G) \setminus E(T)$  conectan un vértice de  $P$  con uno de  $I$ , entonces  $(P, I)$  es una bipartición de  $G$ .
  - ③ Diseñar un algoritmo lineal que determine si  $G$  es bipartito. Retornar una bipartición o un ciclo impar.
  - ④ Generalizar el algoritmo para grafos no conexos.

## Resolución del Ejercicio

Tomamos un ejemplo de grafo  $G$  y aplicamos BFS para construir su árbol generador  $T$ . Clasificamos vértices en  $P$  (distancia par) e  $I$  (impar).

- Luego analizamos qué ocurre al agregar una arista entre dos nodos del mismo conjunto (a).
  - Consideraremos el caso en el cuál todas las aristas adicionales unen nodos de distinto conjunto (b).

## Ejemplo visual



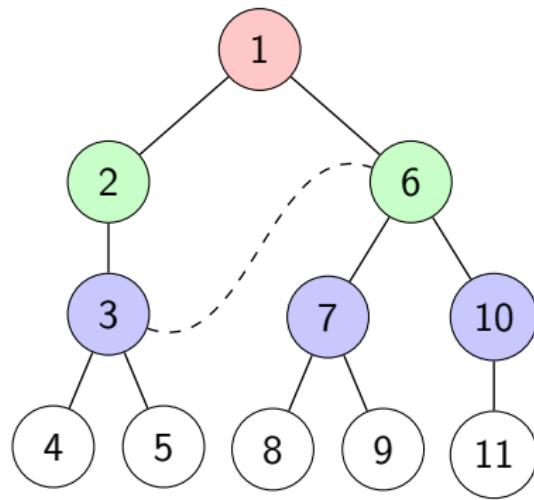
**Figura 1:** Árbol generador  $T$  obtenido mediante BFS. La arista punteada  $(3,7)$  pertenece al grafo  $G$  pero no al árbol  $T$ .

## Inciso a: Ciclos impares

Al agregar una arista  $uv$  con  $u, v$  en  $P$  o  $I$ , el único ciclo (llamémosle  $C$ ) generado tiene longitud impar.

- Porque  $d(u, r) = d(r, v) \Rightarrow \text{Longitud}(C) = 2 \cdot d(u, v) + 1$
- Como  $2 \cdot d(u, r)$  es par para cualquier  $u$ , al sumarle 1 siempre será impar.

## Inciso b: Bipartición - Ejemplo visual



**Figura 2:** Árbol generador  $T$  obtenido mediante BFS. La arista punteada  $(3,6)$  pertenece al grafo  $G$  pero no al árbol  $T$ .

## Inciso b: Bipartición

Si toda arista que no está en  $T$  une un nodo de  $P$  con uno de  $I$ , entonces:

- Si usamos BFS, las aristas solo pueden conectar nodos de niveles consecutivos.
- No existen aristas entre nodos del mismo conjunto.
- $(P, I)$  es una bipartición válida de  $G$ .
- Por lo tanto,  $G$  es bipartito.

## Inciso c: Algoritmo Lineal

## Pasos del algoritmo:

- ① Obtener  $T$  usando BFS (lineal)
  - ② Clasificar nodos en  $P$  e  $I$
  - ③ Recorrer aristas que no están en  $T$  y verificar si conectan nodos del mismo conjunto
  - ④ Si encontramos tal arista, devolver el ciclo impar. Si no, devolver bipartición.

# Complejidad del Algoritmo

- BFS:  $O(n + m)$
- Clasificación por niveles:  $O(n)$
- Revisión de aristas:  $O(m)$

**Total:**  $O(n + m)$  (lineal)

## Inciso d: Generalización

- Un grafo es bipartito ssi todas sus componentes conexas lo son.
- Aplicamos el algoritmo anterior sobre cada componente conexa.
- Si alguna contiene ciclo impar,  $G$  no es bipartito.

## Para pensar

- ¿Cómo resolverían este problema utilizando DFS en vez de BFS?

## Para pensar

- ¿Cómo resolverían este problema utilizando DFS en vez de BFS?

1 BFS

2 Dijkstra

3 Greedy

4 Repaso AGM

5 Ejercicios AGM

# Enunciado del Ejercicio

## Camino Mínimo

Tuki plantó una planta de papas en su jardín, y recientemente se percató de que las hormigas se la están comiendo. Ya descubrió todos los surcos que armaron las hormigas, y pudo medir la longitud de cada uno de ellos. Llamamos  $E$  a este conjunto de surcos, donde cada uno de ellos une un par de intersecciones. Al conjunto de intersecciones lo llamamos  $V$ , y la planta de papas se encuentra en la intersección  $p \in V$ .

En una inspección rápida observó que en algunos surcos hay hormigas cargando con pedazos de hoja de la planta. Llamamos  $E^*$  a este conjunto, el cual cumple que  $E^* \subseteq E$  (no necesariamente son estos todos los surcos usados por hormigas, puede ser que también usen algunos de los surcos en  $E \setminus E^*$ ). Gracias a sus conocimientos de Biología, Tuki sabe que las hormigas siempre siguen un camino mínimo para moverse entre su hormiguero y la fuente de comida. Teniendo en cuenta este comportamiento, les gustaría saber en cuáles intersecciones es posible que se encuentre el hormiguero.

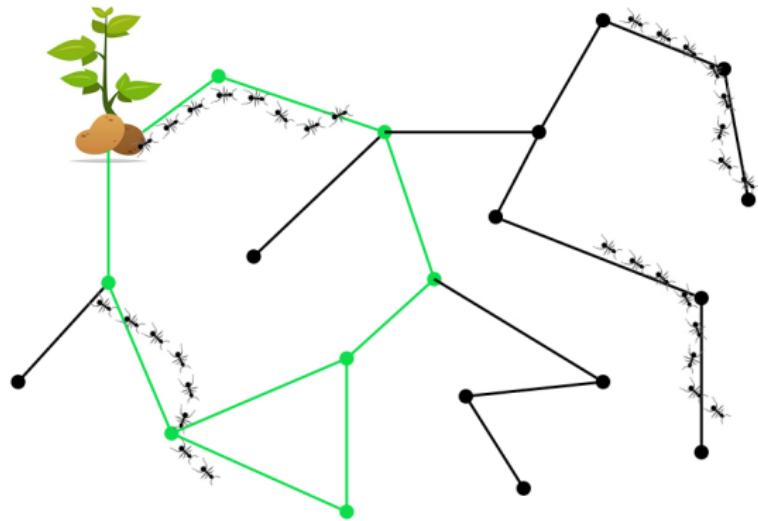
Más formalmente, se quiere decidir para cuáles intersecciones  $h \in V$  vale que todos los surcos en  $E^*$  pertenecen a un camino mínimo entre los que van de  $p$  a  $h$ .

- Diseñar un algoritmo que lo resuelva cuya complejidad no sea superior a  $O(|V|^3)$
- Asumir que todos los surcos tienen la misma longitud. Dar un algoritmo para resolver el mismo problema con complejidad no superior a  $O(|V||E|)$ .

## ¿Cómo modelamos?

- Armamos un grafo cuyos nodos son la planta  $p$  y las intersecciones de los surcos.
- Cada surco es una arista. Asumimos que son bidireccionales  $\Rightarrow d(v, w) = d(w, v)$  para todo par de nodos  $v, w$ .
- El peso de cada arista es la longitud de cada surco.
- De todas las aristas  $E$ , nos reservamos alguna estructura para saber cuáles son las  $E^*$ .

# Ejemplo visual



## ¿Qué necesitamos?

- Queremos saber cuáles de todas las intersecciones son candidatas para ser el hormiguero ( $h$ ).
- Para ello, queremos encontrar todos los nodos  $h \in V$  tales que todos los surcos en  $E^*$  pertenecen a algún camino mínimo de  $h$  a  $p$ . Es decir, un nodo  $h$  será candidato si para toda arista  $vw \in E^*$  esta pertenece a algún camino  $ph$ -eficiente (concepto que sale en la guía 5).
- Queremos que  $\forall vw \in E^*$ ,  
$$d(p, h) = d(p, v) + c(v, w) + d(w, h)$$

## Idea del algoritmo

- Queremos obtener  $\forall h \in V$  el valor de  $d(p, h)$ . Para ello hacemos Dijkstra desde cada  $h$ . Esto tiene complejidad  $O(|V| \cdot |V|^2) = O(|V|^3)$ .
- Ahora, por cada nodo  $h$  tenemos que ver si cada arista  $vw \in E^*$  es  $ph$ -eficiente. Esto tiene complejidad  $O(|E| \cdot |V|)$ .
- Aquellos  $h$  que cumplan con lo anterior serán candidatos a ser el hormiguero.
- ¿Complejidad total?  $O(|V|^3 + |V| \cdot |E|) = O(|V|^3)$

# LAS HORMIGAS



LAS HORMIGAS VIENDO QUE TUKI  
ENCONTRÓ EL HORMIGUERO

## Item B

Ahora que los surcos tienen la misma longitud podemos modelar el mismo grafo pero sin peso en sus aristas. Ahora la función distancia  $d(v, w)$  dará como resultado la cantidad mínima de surcos que separan a  $v$  de  $w$ .

¿Qué algoritmo conocemos que nos permite calcular distancia entre nodos en un grafo sin pesos? Exacto: BFS!

Entonces podemos hacer lo mismo que en el punto anterior solo que esta vez para hacer el todos a todos usamos BFS en vez de Dijkstra.

## Idea del algoritmo

- Hacemos BFS desde cada nodo  $h$  para obtener  $d(p, h)$ . Esto tiene complejidad  $O(|V| \cdot (|V| + |E|)) = O(|V|^2 + |V| \cdot |E|) = O(|V| \cdot |E|)$ .
- Repetimos procedimiento del punto (a), cuya complejidad es  $O(|V| \cdot |E|)$ .
- ¿Complejidad final?  $O(|V| \cdot |E|)$ .

1 BFS

2 Dijkstra

3 Greedy

4 Repaso AGM

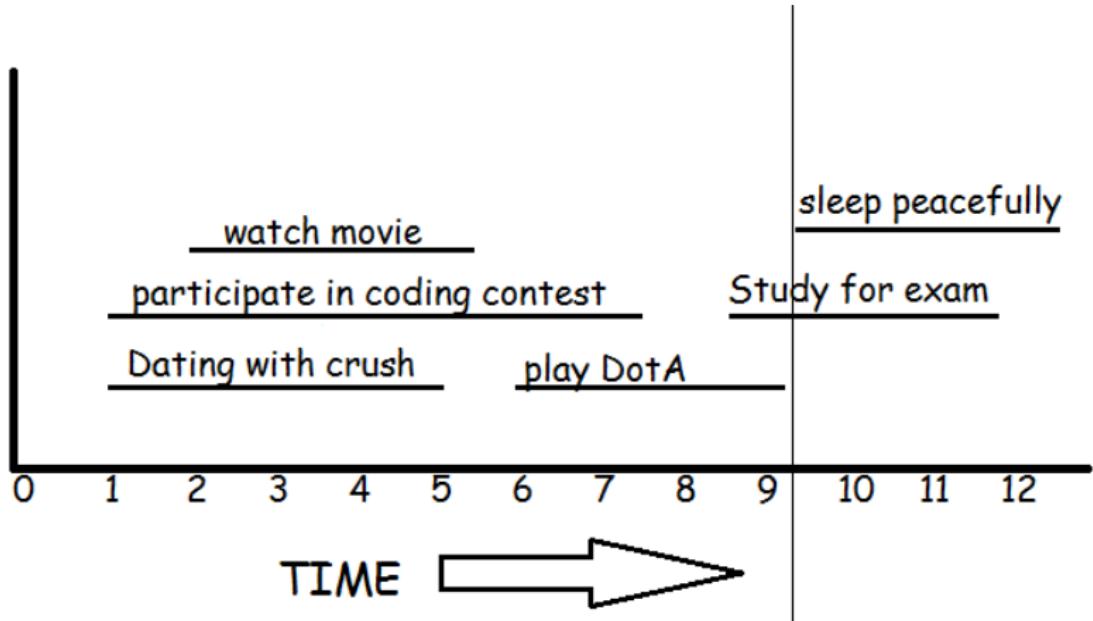
5 Ejercicios AGM

# I'm very busy - Busyman

## Algoritmos Golosos

Chimi es una persona muy ocupada, pero como es muy ordenado tiene un cronograma con todas las actividades que podría realizar. Sin embargo, muchas de estas actividades se superponen entre sí, por lo que debe decidir cuidadosamente cuáles hacer y cuáles no. Su objetivo es realizar la mayor cantidad posible de actividades, siempre que ninguna se solape con otra.

Dado un conjunto de actividades, cada una con un horario de inicio y fin, ¿cuál es el máximo número de actividades que Chimi puede llevar a cabo sin que se superpongan?



## Relación con grafos

Este problema se puede pensar como un problema de **grafo de intervalos**, donde:

- A cada vértice se le asocia un intervalo  $[t_i^j, t_f^j]$  para cada actividad  $j$ .
- Existe una arista entre dos vértices si sus intervalos se superponen.

**Ejemplo:** Si el intervalo de  $u$  es  $[1, 4]$  y el de  $v$  es  $[3, 5]$ , entonces existe una arista que conecta  $u$  y  $v$ .

Entonces, encontrar la solución que quiere Chimi es equivalente al **Problema de conjunto independiente máximo**.

## Problema de conjunto independiente

Dado un grafo  $G = (V, E)$ , un **conjunto independiente** es un conjunto de vértices  $I \subseteq V$  tal que:

- Ningún par de vértices en el conjunto está conectado por una arista.

El **problema del conjunto independiente máximo** consiste en encontrar el mayor conjunto posible con esta propiedad.

En general, encontrar un conjunto independiente máximo en un grafo es **NP-hard** (no ahondaremos en esto, alentamos a que lo investiguen por su cuenta), pero ya que nuestro grafo es de intervalos se puede encontrar una solución polinomial.

## Idea de la solución Greedy

- ① Ordenar las actividades por su tiempo de finalización en orden creciente.
- ② Inicializar un conjunto vacío  $B$  para las actividades seleccionadas.
- ③ Inicializar una variable  $t \leftarrow -\infty$  que representa el final de la última actividad seleccionada.
- ④ Para cada actividad  $(t_i, t_f)$  en el orden dado:
  - Si  $t_i > t$ , entonces:
    - Agregar  $(t_i, t_f)$  al conjunto  $B$ .
    - Actualizar  $t \leftarrow t_f$ .
- ⑤ Devolver el conjunto  $B$ .

## Demostración

Vamos a hacerla por inducción. Usaremos la siguiente notación:

- Sea  $B = (B_1, B_2, \dots, B_k)$  la solución construida por el algoritmo goloso, donde  $B_i$  es la i-ésima actividad seleccionada.
- Sea  $O = (O_1, O_2, \dots, O_k)$  una solución óptima (pueden existir varias).

## Demostración

### Caso base ( $i = 1$ ):

Nuestro algoritmo elige como primera actividad  $B_1$ , la que termina antes que todas las otras.

- Sabemos que  $\text{fin}(B_1) \leq \text{fin}(O_1)$ , entonces si definimos  $O' = (B_1, O_2, \dots, O_k)$  tenemos una nueva solución óptima, ya que si  $O_1$  no se solapaba con ninguna otra, entonces  $B_1$  tampoco.
- Por lo tanto, el algoritmo comienza con una elección que puede ser parte de una solución óptima

### Hipótesis inductiva:

Suponemos que luego de  $i$  pasos, el algoritmo ha seleccionado una secuencia  $B_1, \dots, B_i$ , que puede extenderse a una solución óptima  $O = (B_1, \dots, B_i, O_{i+1}, \dots, O_k)$

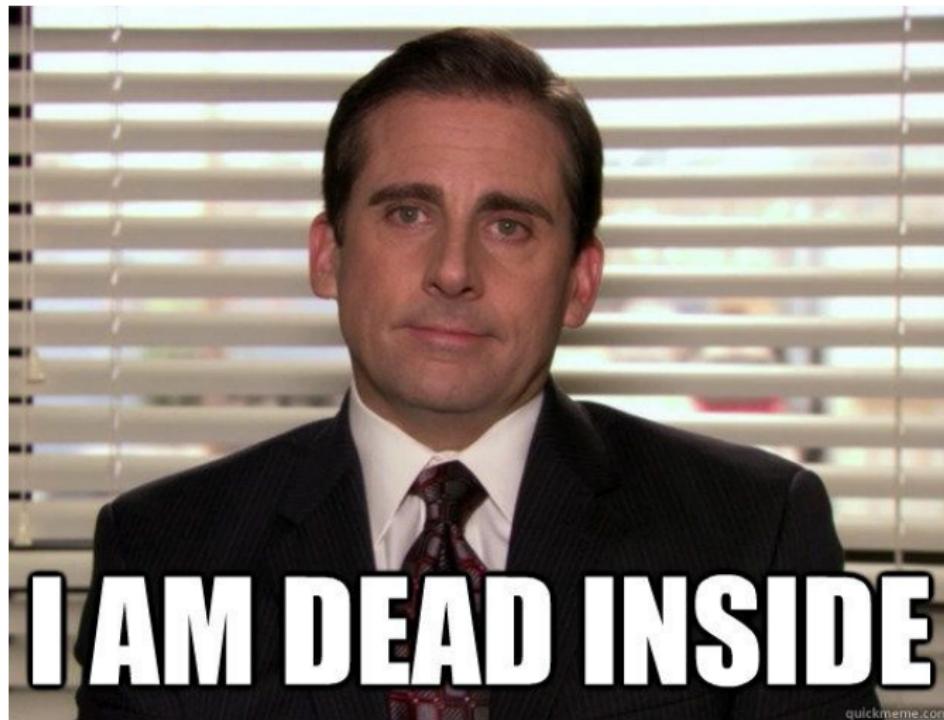
## Demostración

### Paso inductivo:

Queremos probar que podemos extender la secuencia  $B_1, \dots, B_i$  agregando  $B_{i+1}$  y que a partir de esta se puede obtener una solución óptima.

- Por **H.I.** sabemos que existe la solución óptima  $O = (B_1, \dots, B_i, O_{i+1}, \dots, O_k)$
- Por cómo elige el algoritmo sabemos que  $B_{i+1}$  es la actividad compatible con  $B_i$  que **termina más temprano**.
- Luego,  $O_{i+1}$  también es compatible con  $B_i$  pero por cómo elige nuestro algoritmo sabemos que  $\text{fin}(B_{i+1}) \leq \text{fin}(O_{i+1})$ .
- Entonces podemos construir una nueva solución óptima  $O' = (B_1, \dots, B_i, B_{i+1}, O_{i+2}, \dots, O_k)$ .
- Por inducción, podemos extender paso a paso la solución parcial obtenida por el algoritmo goloso hacia una solución completa que es óptima. Esto demuestra que el algoritmo selecciona el **máximo de actividades sin que estas su superpongan**.

¿Hacemos un break?



## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 5 Ejercicios AGM

## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 5 Ejercicios AGM

## Definiciones

### 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  $G$

## Definiciones

### 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  $G$

### 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 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 5 Ejercicios AGM

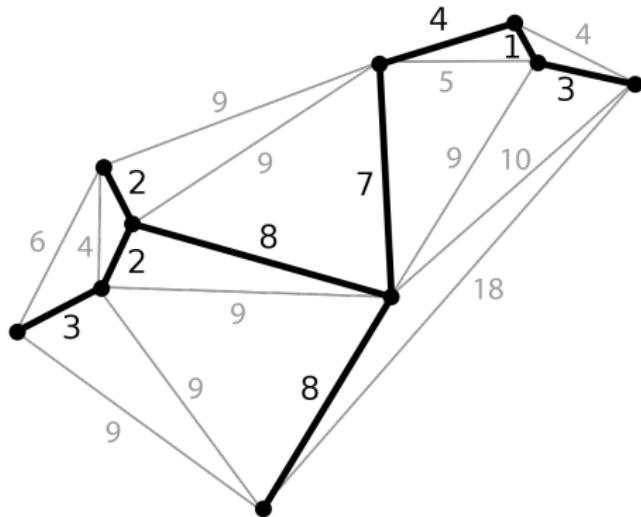
## Definiciones

### Árbol generador mínimo

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

- $T$  es árbol generador de  $G$
- El costo de  $T$  es mínimo con respecto a todos los árboles generadores de  $G$

Acá vemos un grafo  $G$  (en gris) y un subgrafo  $T$  (en negro) que es AGM de  $G$ .



## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 5 Ejercicios AGM

# Camino MaxiMin

## 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  $c_{min}$ :

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

## Camino MaxiMin

### 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  $c_{min}$ :

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

También se lo conoce como 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.

# Camino MiniMax

## 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  $c_{max}$ :

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

## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 5 Ejercicios AGM

## Vínculo entre MiniMax y AGM

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

## Vínculo entre MiniMax y AGM

¿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.

## Vínculo entre MiniMax y AGM

Supongamos que existe un camino  $Q$  en un AGM  $T$  entre dos vértices  $a$  y  $z$  que no es minimax. Existe entonces otro camino  $P$  entre  $a$  y  $z$  que es minimax, cuya máxima arista será más chica que la máxima arista de  $Q$ .



## Vínculo entre MiniMax y AGM

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  $Q$ .

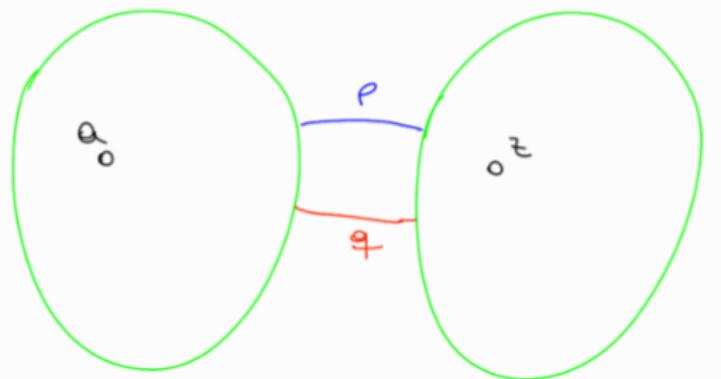


## Vínculo entre MiniMax y AGM

Vamos a querer reemplazar  $q$  por alguna arista de  $P$  para conseguir otro AG. Como la máxima arista de  $P$  es más chica que  $q$ , esto nos resultaría en un AG de menor peso que  $T$ , implicando que  $T$  no era un AGM, un absurdo. Veamos si podemos encontrar una arista de  $P$  que nos sirva.

## Vínculo entre MiniMax y AGM

Si sacamos la arista  $q$  del AGM  $T$  para conseguir un  $T'$ , nos van a quedar dos componentes conexas en  $T'$ : una que contenga  $a$ , y otra que contenga  $z$ . Necesariamente hay una arista  $p$  en  $P$  que conecta estas dos componentes conexas, porque sino  $P$  no conectaría a  $a$  y  $z$ . Además,  $p$  no está contenida en el AGM  $T$ , porque sino también estaría en  $T'$ , o sea que  $T'$  no tendría dos componentes conexas.



Nos generamos entonces el grafo  $T^*$  que se obtiene de reemplazar la arista  $q$  por la arista  $p$  en el AGM  $T$ . Vemos lo siguiente:

- 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. □

## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

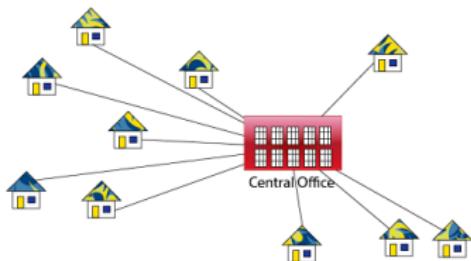
Prim

## 5 Ejercicios AGM

# Aplicaciones del AGM

## Red de electricidad de una ciudad

Wiring : Naive Approach



Expensive!

Wiring : Better Approach



Minimize the total length of wire connecting the customers

# Más aplicaciones del AGM!

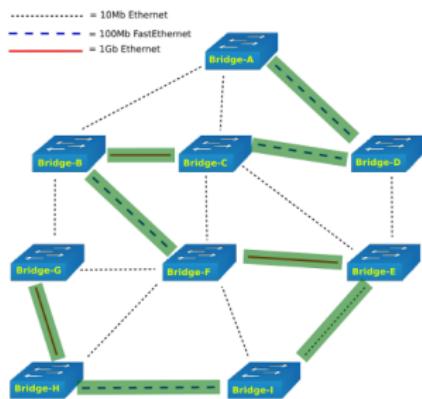


Figura 3: STP (Spanning Tree Protocol)

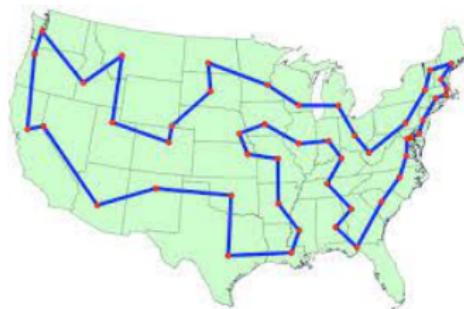


Figura 4: Aproximación para TSP (Traveling Salesman Problem)

## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 5 Ejercicios AGM

# Algoritmo de Kruskal

**MST-KRUSKAL( $G, w$ )**

- 1  $A = \emptyset$
- 2 **for** each vertex  $v \in G.V$   
3     MAKE-SET( $v$ )
- 4 create a single list of the edges in  $G.E$
- 5 sort the list of edges into monotonically increasing order by weight  $w$
- 6 **for** each edge  $(u, v)$  taken from the sorted list in order  
7     **if** FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )  
8          $A = A \cup \{(u, v)\}$   
9         UNION( $u, v$ )
- 10 **return**  $A$

## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

Árbol Generador

Árbol Generador Mínimo

Caminos MaxiMin y MiniMax

Vínculo entre MiniMax y AGM

Aplicaciones

Kruskal

Prim

## 5 Ejercicios AGM

# Algoritmo de Prim

MST-PRIM( $G, w, r$ )

```

1  for each vertex  $u \in G.V$ 
2     $u.key = \infty$ 
3     $u.\pi = \text{NIL}$ 
4     $r.key = 0$ 
5     $Q = \emptyset$ 
6  for each vertex  $u \in G.V$ 
7    INSERT( $Q, u$ )
8  while  $Q \neq \emptyset$ 
9     $u = \text{EXTRACT-MIN}(Q)$       // add  $u$  to the tree
10   for each vertex  $v$  in  $G.Adj[u]$  // update keys of  $u$ 's non-tree neighbors
11     if  $v \in Q$  and  $w(u, v) < v.key$ 
12        $v.\pi = u$ 
13        $v.key = w(u, v)$ 
14       DECREASE-KEY( $Q, v, w(u, v)$ )

```

## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

## 5 Ejercicios AGM

Viaje en peligro

Audífonos Defectuosos

## 1 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

## 5 Ejercicios AGM

Viaje en peligro

Audífonos Defectuosos

## 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

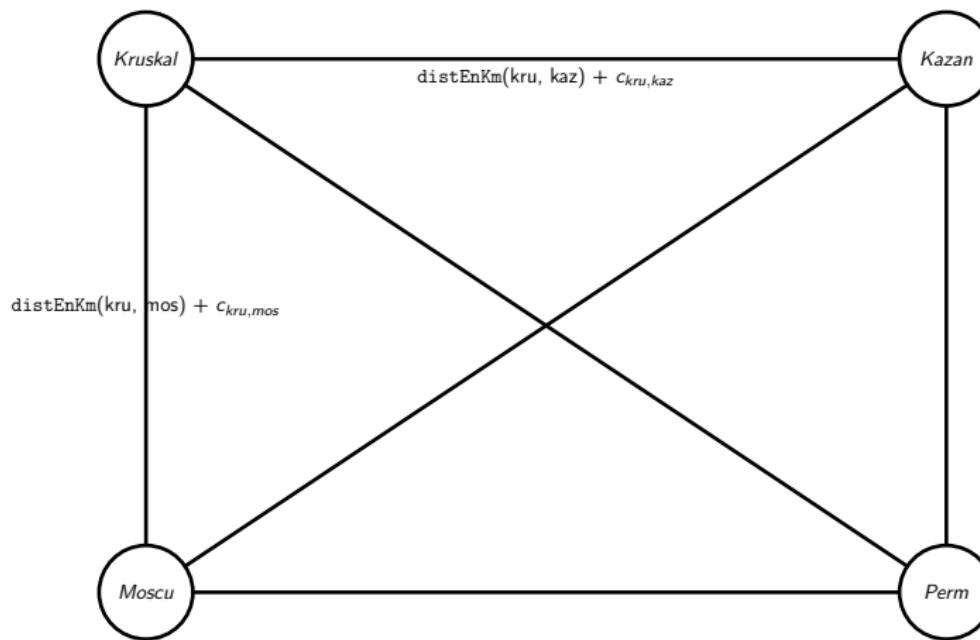
- El problema nos pide que queden  $k + 1$  localidades conectadas a partir de  $k$  rutas, por lo que podemos inferir que nos está pidiendo generar un árbol.
- Luego sabemos que  $k \ll n$ , por lo que el árbol que vamos a generar va a ser un subgrafo del grafo completo.
- También nuestro algoritmo 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.

# Modelado



Figura 5: Ayudante de 2da explicando modelado en TDA

## 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.

## Solución final



**Figura 6:** No se olviden de explicar cómo su modelo se traduce a la solución buscada

## Aclaración

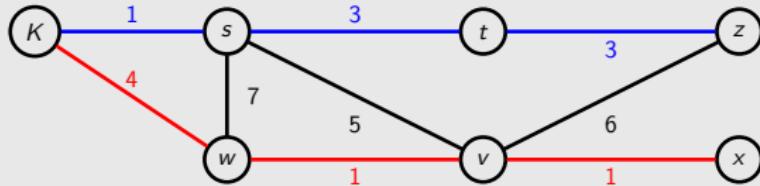
### Invariante de Prim

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 BFS

## 2 Dijkstra

## 3 Greedy

## 4 Repaso AGM

## 5 Ejercicios AGM

Viaje en peligro

Audífonos Defectuosos

## 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

### Possibles 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

### Possibles 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

### Possibles 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

### Possibles 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

### Possibles 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.

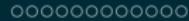
## Bonus track

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

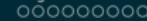
---

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

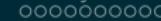
BFS



Dijkstra



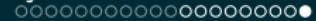
Greedy



Repaso AGM



Ejercicios AGM



¿Fin?

