Árbol Generador Mínimo

Fernando Nicolás Frassia Ferrari - Ezequiel Companeetz

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

1^{er} Cuatrimestre de 2024

Programa de hoy

1. Repaso de definiciones
2. Aplicaciones de AGM
3. Ejercicios introductorios
4. Tips de Modelado
5. Ejercicios más picantes
6. Implementación
7. Conclusiones

Repaso de definiciones

Definición (Árbol generador)

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

Repaso de definiciones

Definición (Árbol generador)

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

Definición (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.

¿Que es un AGM?

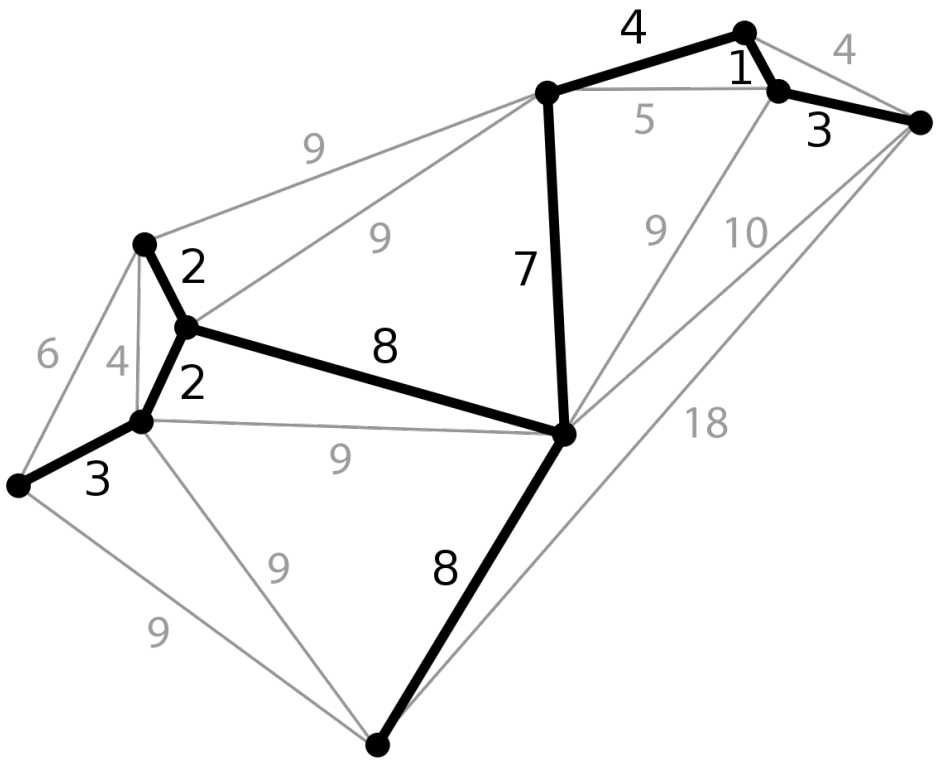
Definición (Árbol generador mínimo)

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

¿Que es un AGM?

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



Camino MaxiMin

Definición (Camino MaxiMin)

Sea G un grafo , $c : E(G) \to \mathbb{R}$ 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}:

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

MaxiMin = *MAXI*mizar la arista de *MINI*mo costo por la cuál pasas.

Camino MaxiMin

Definición (Camino MaxiMin)

Sea G un grafo , $c : E(G) \to \mathbb{R}$ 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}:

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

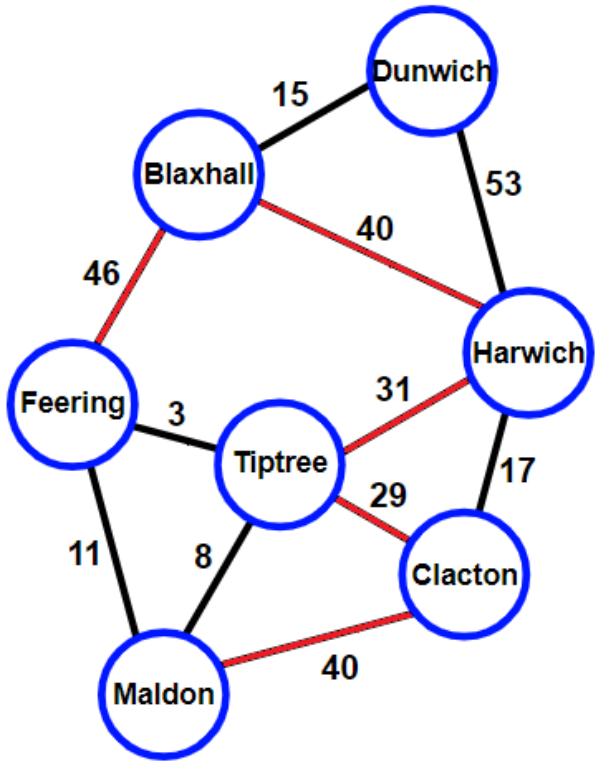
También se lo conoce cómo el problema del *camino más ancho*.[a]

[a] Si nos imaginamos los costos cómo capacidades, entonces queremos maximizar la capacidad que nuestro camino puede trasladar. Y la capacidad de un camino esta dada por la capacidad de su arista más pequeña.

MaxiMin = *MAXI*mizar la arista de *MINI*mo costo por la cuál pasas.

Camino MaxiMin

Figura: Ejemplo de camino *MaxiMin* entre Maldon y Feering



Camino MiniMax

Definición (Camino MiniMax)

Sea G un grafo , $c : E(G) \to \mathbb{R}$ 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}:

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

Vínculo entre MiniMax y AGM

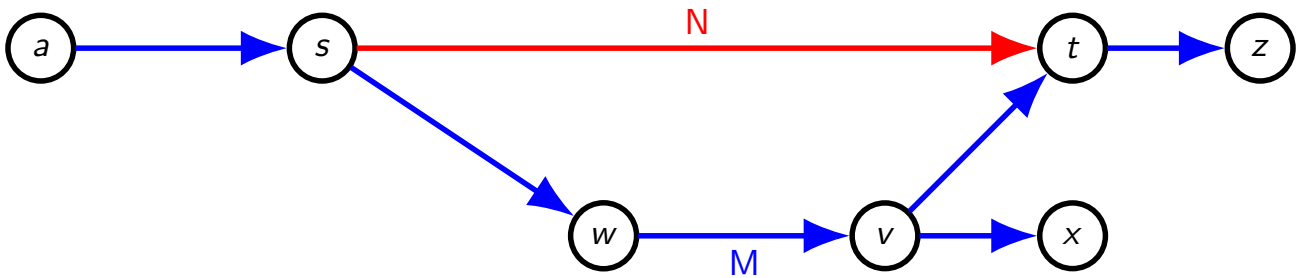
¿Que herrramientas de las que vimos creen que podemos utilizar para mostrar que *camino AGM* $\rightarrow$ *camino Minimax*?

Vínculo entre MiniMax y AGM

¿Que herrramientas 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 minimax P entre los vértices a y z que no está completamente en el AGM T, siendo las aristas de T las azules. Esto significa que hay una arista $N(nueva) = (s, t)$ en P que no está en T. Sea Q el camino en T desde s hasta t. Sea $M(mayor)$ la arista de mayor peso en Q.



Vínculo entre MiniMax y AGM

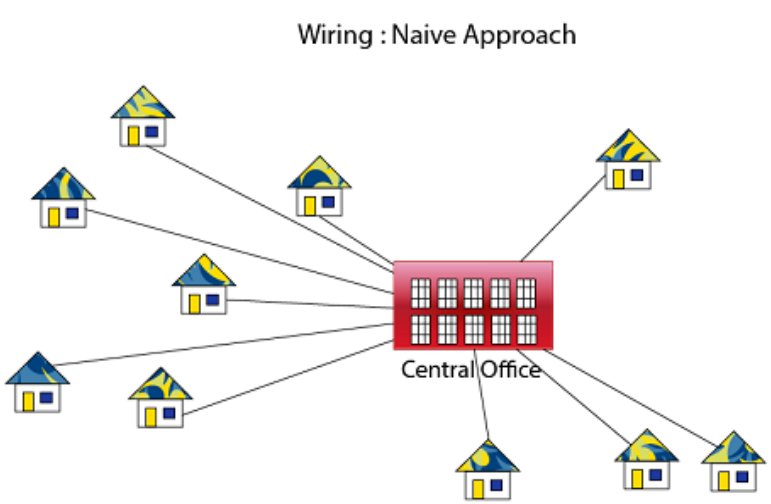
Ahora hay 2 condiciones a considerar:

1. $peso(N) < peso(M)$: En ese caso, T no es un AGM. Pues existe $T' = T - M + N$ tal que $peso(T') < peso(T)$. Absurdo!
2. $peso(M) \leq peso(N)$: En este caso podrías eliminar N de P y agregar las aristas de Q en su lugar, y seguiría siendo un camino minimax, ya que no incluimos una arista con un peso mayor que el que ya estaba en ese camino antes.

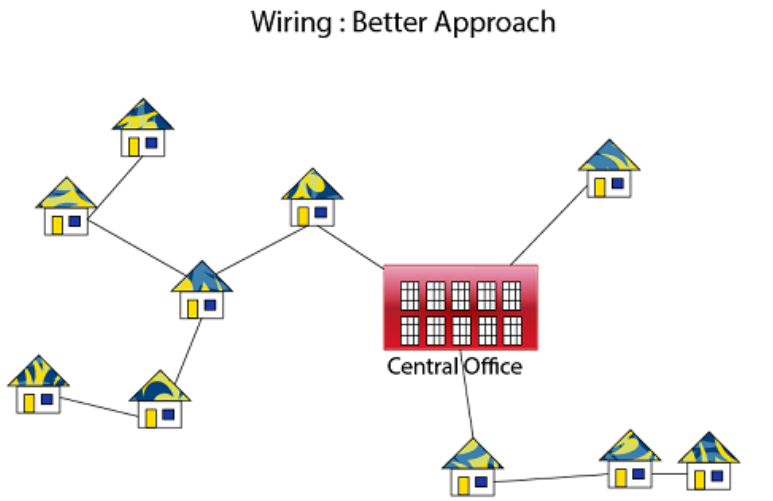
Por cada arista N en un camino minimax que no está en T, se puede realizar una sustitución de arista como se describe en el punto 2, creando así un camino minimax que estará completamente en T.

Aplicaciones del AGM

Red de electricidad de una ciudad



Expensive!



Minimize the total length of wire connecting the customers

Más aplicaciones del AGM!

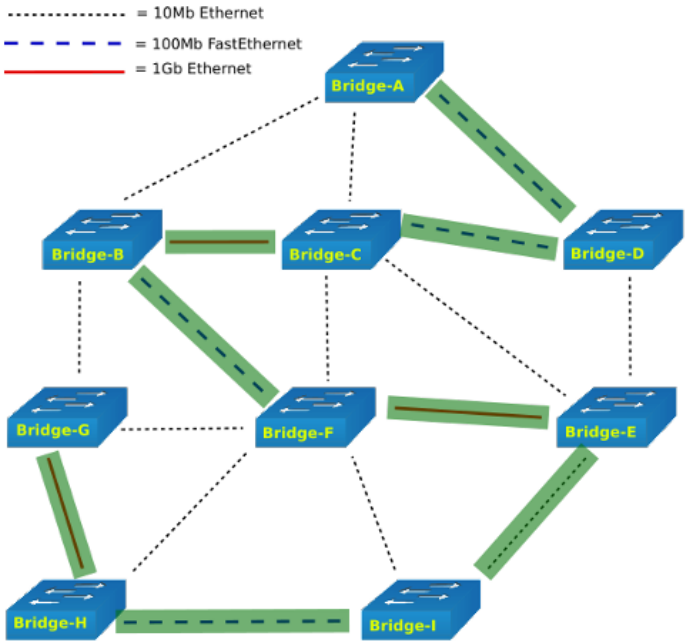


Figura: STP (Spanning Tree Protocol)

Figura: Aproximación para TSP (Traveling Salesman Problem)

Kahoot Time

Ahora vamos a repasar algunas propiedades útiles del *AGM* y de los algoritmos que utilizamos para obtenerlo.
Para eso van a tener que entrar https://kahoot.it/ e ingresar el código escrito en el pizarrón.

Tips de Modelado

Antes de arrancar con los ejercicios, una pregunta.

Ejercicio 1

Viaje en peligro

Sasha 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, poniendole 2 condiciones. Quiere que queden conectadas $k+1$ localidades y 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 a partir de *rublos* por *km* + un precio fijo $c_{x,y}$ que depende de las localidades. Además sabemos en qué localidad se encuentra Sasha. Queremos una red que cumpla lo pedido. ¿Es única?

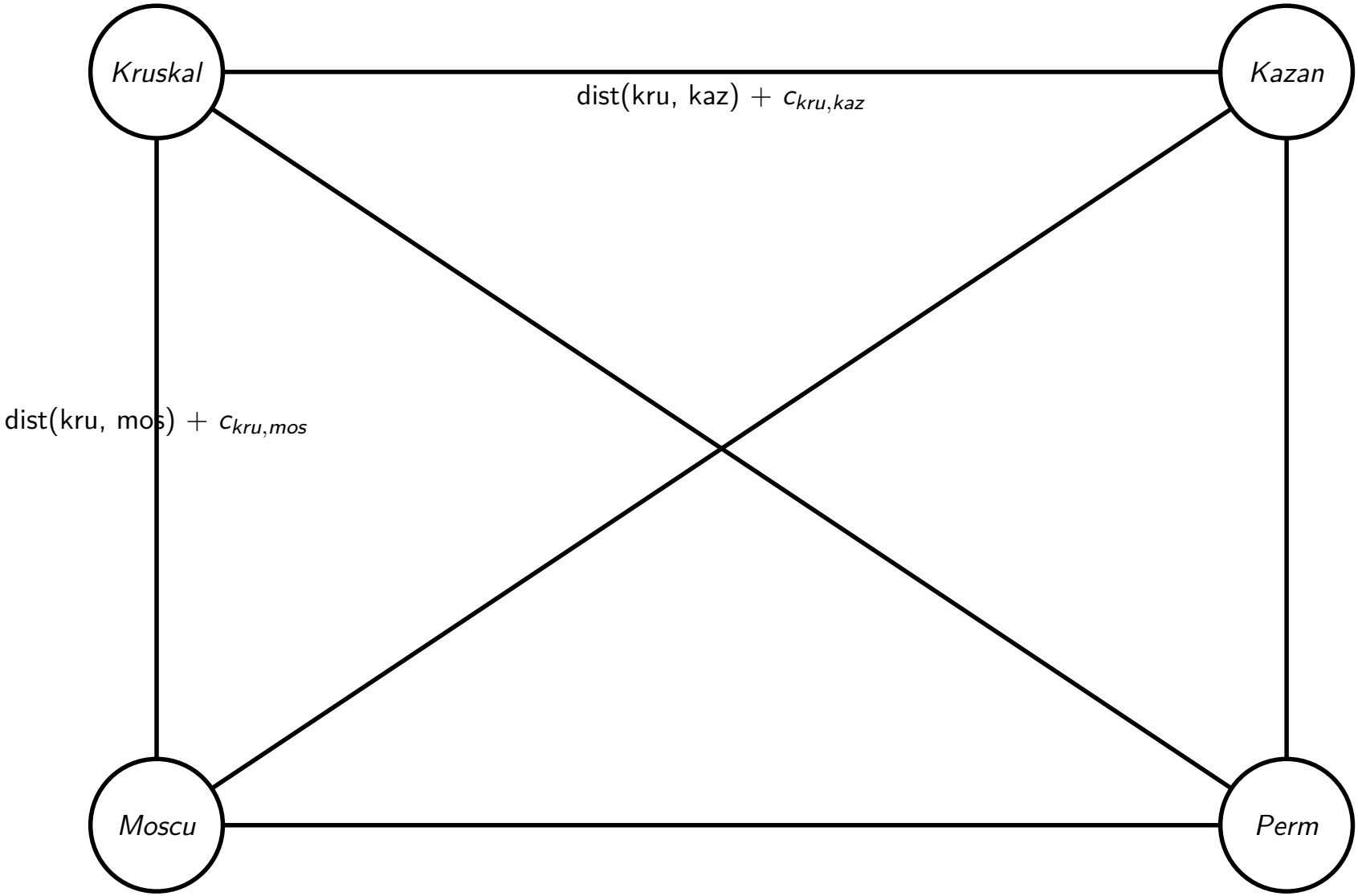
Especificaciones

- El formato del input es una línea que contiene dos enteros n (cantidad de ciudades) y k (cantidad de rutas). Luego tenemos n lineas que tienen el formato x_i, y_i, las cuáles 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 Sasha, puesto que queremos que la misma quede conectada.
- Por último cómo 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 *km* 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.
  $costo((x, y)) = dist(x, y) + c_{x,y}$
- 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 vas a quedar $\mathcal{O}(nk)$. Cómo $k << n$ la complejidad total es $\mathcal{O}(n^2)$. Con n =cantidad de ciudades.

Aclaración

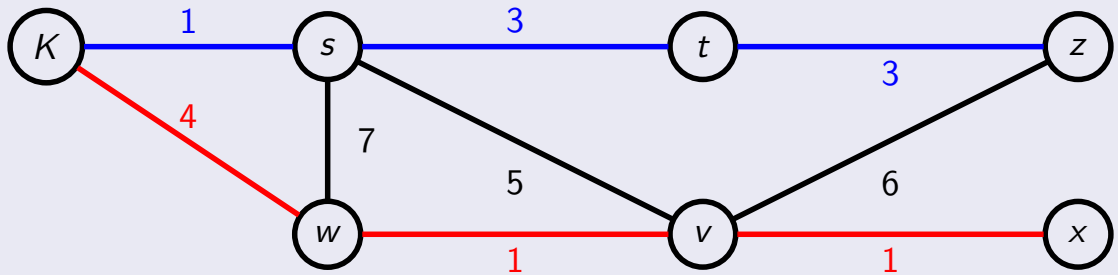
Invariante de Prim

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 conecta a todas las localidades con costo mínimo*? Esto se debe a que el invariante de Prim cumple que, en su i-ésimo paso, vas a tener un subgrafo de tamaño i del AGM, no el árbol de i aristas mínimo.

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).



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. Pero Sasha sufre mucho los sonidos fuertes, por lo que quiere encontrar una ruta con la menor cantidad de ruido. Conocemos todas las calles que conectan una esquina con otra en el mapa. Y tenemos una función d la cuál nos dice el volumen de cada calle.
Queremos determinar cuál es el mínimo 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 lineas que tienen el formato e_i, e_j, d_{ij}, las cuáles 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.

Ejemplo

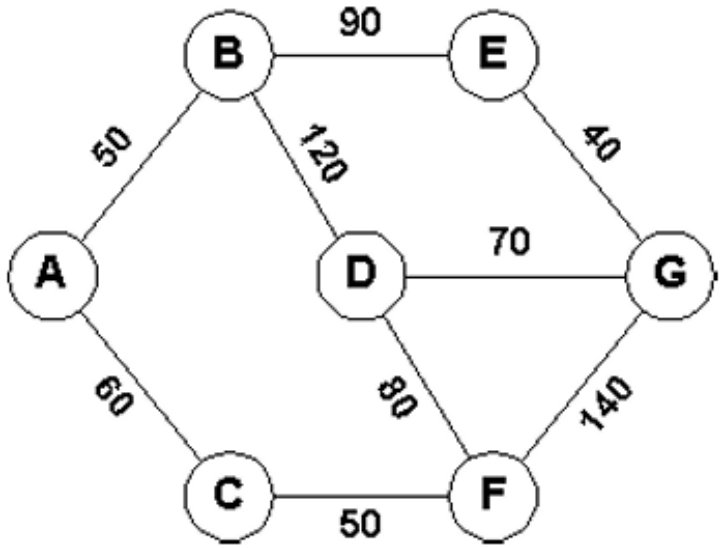


Figura: Para ir de A a G el mejor camino es A-C-F-D-G, y la tolerancia que uno necesita es de 80.

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 $c : E(G) \to \mathbb{R}$ 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? $\mathcal{O}(min(n^2, m \log n))$, 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?
  - 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.
- ¿Que 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 sólo 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. (Al final de la clase hay una pequeña aclaración sobre esto)

Bonus track

Tarea: Ahora los auriculares de Sasha le bloquean el sonido pero sólo de 1 calle. ¿Cómo cambia el problema? [1]

[1] Spoiler: Se re pica.

¿Sale un break?

Figura: A descansar las neuronas

Modelado

Tips para modelar

- Primero conviene analizar cuáles pueden ser nuestros posibles vértices, aristas y costos asociados del problema, es clave tener un *modelo* que funcione.
- Luego de analizar el enunciado hay que ver si hay alguna *propiedad* o *problema* que ya vimos, o si tiene una pequeña variación.
- Por último a partir de eso podemos pensar en los *algoritmos* que conocemos y comenzar a ver cómo resolver el problema.
- *Dibujen* el grafo para poder visualizarlo mejor.

Ejercicio 3

Sasha 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 que 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 $(c_i, tubo)$ 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 que 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 $(c_i, tubo)$ 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.

Ejercicio 4

Sasha aprobó Algo3 y volvió de intercambio. Cómo había salido tan bien el proyecto anterior el presidente Juan Jose *Prim* le volvió a pedir ayuda.

Rutas y aeropuertos

Ahora Sasha tiene que conectar todas las ciudades de Rusia. 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 cómo podía suceder en el caso anterior.

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 cómo podía suceder en el caso anterior.
  - Otra opción entonces es pensar qué algoritmos conocemos para resolver este problema y si alguno de ellos está vinculado.

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 cómo podía suceder en el caso anterior.
  - Otra opción entonces es pensar qué algoritmos conocemos para resolver este problema y si alguno de ellos está vinculado.
  - Este problema particular se puede ver cómo un problema de bosque generador mínimo.

Solución

- Creamos un grafo G con las ciudades cómo vértices y las rutas con costo $c_{i,j}$ cómo sus aristas.

Solución

- Creamos un grafo G con las ciudades cómo vértices y las rutas con costo $c_{i,j}$ cómo sus aristas.
- Corremos una versión modificada del algoritmo de Kruskal para obtener los bosques generadores mínimos de este grafo y donde colocaríamos los aeropuertos.

Solución

- Creamos un grafo G con las ciudades cómo vértices y las rutas con costo $c_{i,j}$ cómo sus aristas.
- Corremos una versión modificada del algoritmo de Kruskal para obtener los bosques generadores mínimos de este grafo y donde colocaríamos los aeropuertos.
- Utilizamos Kruskal para construir el AGM, pero vamos guardando:

Solución

- Creamos un grafo G con las ciudades cómo vértices y las rutas con costo $c_{i,j}$ cómo sus aristas.
- Corremos una versión modificada del algoritmo de Kruskal para obtener los bosques generadores mínimos de este grafo y donde colocaríamos los aeropuertos.
- Utilizamos Kruskal para construir el AGM, pero vamos guardando:
  - Cuáles son nuestras componentes conexas
  - A que componente conexa pertenece cada vértice.
  - Cuál es el aeropuerto de menor costo de la componente conexa, o si es que tiene un aeropuerto.

Solución

- Para unir dos ciudades nos fijamos la opción más barata entre:
  - Unir las dos ciudades con una ruta
  - Colocar un aeropuerto en ambas (o una sola, si una ya tiene) de las componentes conexas.
- Al agregar una ciudad a una componente conexa con aeropuerto tenemos que ver si es más barato construir un aeropuerto en ella, antes que donde ya estaba construido.

Kruskal - Invariante

- Invariante:

Kruskal - Invariante

- Invariante: Tenemos un bosque generador de i aristas

Kruskal - Invariante

- Invariante: Tenemos un bosque generador de i aristas que es subgrafo de algún AGM.

Kruskal - Invariante

- Invariante: Tenemos un bosque generador de i aristas que es subgrafo de algún AGM.
- Invariante alternativo:

Kruskal - Invariante

- Invariante: Tenemos un bosque generador de i aristas que es subgrafo de algún AGM.
- Invariante alternativo: Tenemos un bosque generador de i aristas que es mínimo entre los bosques de i aristas.

Kruskal - Invariante

- Invariante: Tenemos un bosque generador de i aristas que es subgrafo de algún AGM.
- Invariante alternativo: Tenemos un bosque generador de i aristas que es mínimo entre los bosques de i aristas.
- Observar que el primer invariante no implica el invariante alternativo, y que Prim no cumple el invariante alternativo. Pensar ejemplos en casa.

Kruskal - Idea

Idea:

1. Inicialización: Comenzamos con todos los vértices y ninguna arista.

Kruskal - Idea

Idea:

1. Inicialización: Comenzamos con todos los vértices y ninguna arista.
2. Iteración: De las aristas que podemos agregar y seguir teniendo un bosque (osea que no genera ciclos), agregamos la más barata.

Kruskal - Idea

- Podemos empezar ordenando las aristas de manera creciente por costo.

Kruskal - Idea

- Podemos empezar ordenando las aristas de manera creciente por costo.
- Para cada arista nos fijamos si al agregarla al bosque no genera un ciclo.

Kruskal - Idea

- Podemos empezar ordenando las aristas de manera creciente por costo.
- Para cada arista nos fijamos si al agregarla al bosque no genera un ciclo.
- Esto sucede sii sus extremos están en diferentes componentes conexas.

Kruskal - Idea

Pregunta: Dada una arista $e = uv$, cómo sabemos si u y v están en componentes conexas diferentes?

Kruskal - Idea

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.

Kruskal: Pseudocódigo (según el Cormen)

```
MST-KRUSKAL(G, w)
 1 A = Ø
 2 for each vertex v ∈ 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) ≠ FIND-SET(v)
 8         A = A ∪ {(u, v)}
 9         UNION(u, v)
10 return A
```

Kruskal: Complejidad

- Con BFS o DFS:

Kruskal: Complejidad

- Con BFS o DFS: $O(n^2)$
- Con Disjoint-Set:

Kruskal: Complejidad

- Con BFS o DFS: O(n^2)
- Con Disjoint-Set: O(m log n)

Kruskal: Justificando la complejidad con Disjoint-Set

- Sin optimizaciones: la complejidad del find() es O(n) en peor caso, por lo tanto la complejidad total sería O(m * n).
- Con *union by size/by rank* la complejidad de cada find es O(log n) en peor caso.
- Si además se usa *Path Compression* la complejidad amortizada de cada find() es O(α(n)), donde α representa a la función inversa de Ackermann. (Igual, si se ordenan las aristas al principio en O(m log n), ese paso domina la complejidad).

Prim - Invariante

- Invariante: Tenemos un árbol de i aristas

Prim - Invariante

- Invariante: Tenemos un árbol de i aristas que es subgrafo de algún AGM.

Prim - Idea

1. Inicialización: Empezamos con un sólo vértice v arbitrario.

Prim - Idea

1. Inicialización: Empezamos con un sólo vértice v arbitrario.
2. Iteración: De las aristas que podemos agregar y seguir teniendo un árbol, agregamos la más barata.

Prim: Pseudocódigo (según el Cormen)

```
MST-PRIM(G, w, r)
 1 for each vertex u ∈ G.V
 2     u.key = ∞
 3     u.π = NIL
 4 r.key = 0
 5 Q = Ø
 6 for each vertex u ∈ G.V
 7     INSERT(Q, u)
 8 while Q ≠ Ø
 9     u = 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 ∈ Q and w(u, v) < v.key
12             v.π = u
13             v.key = w(u, v)
14             DECREASE-KEY(Q, v, w(u, v))
```

Prim - Complejidad

- Con BFS o DFS:

Prim - Complejidad

- Con BFS o DFS: $O(n^2)$
- Con Cola de Prioridad sobre Min-Heap:

Prim - Complejidad

- Con BFS o DFS: O(n^2)
- Con Cola de Prioridad sobre Min-Heap: O(m log n)

Prim - Complejidad

- Con BFS o DFS: O(n^2)
- Con Cola de Prioridad sobre Min-Heap: O(m log n)
- Tip: Si esa cola de prioridad se implementa sobre Fibonacci Heap la complejidad baja a O(m + n log n)

Tips para el TP de AGM

Conclusiones

- Leer bien el enunciado y anotar todos los datos que se nos presentan para poder modelar el problema
- Según si el grafo es denso o ralo conviene usar Prim o Kruskal. [2]

[2] Aún así para los parciales pueden asumir que tienen un algoritmo *mágico* que resuelve AGM en $O(min(\ m \log n,\ n^2)\ )$. También existen versiones de Prim y Kruskal para grafos ralos/densos.

Conclusiones

- No siempre el modelo es lineal, tal que cada entidad tiene un vértice y su relación es una arista. Estén atentos a crear nuevos vértices o aristas con la información dada.
- Recuerden escribir todos los pasos: problema $\rightarrow$ modelo $\rightarrow$ algoritmo $\rightarrow$ solución al problema.
- En general es preferible modificar el modelo que los algoritmos. Si aún así tienen que modificarlos justifiquen bien que no rompen el invariante y que mantienen su complejidad.
- Hay propiedades interesantes de AGM que no vimos, hagan las guías para conocerlas!

Referencias

- Ejercicio 1 : https://onlinejudge.org/index.php?option=onlinejudge&Itemid=8&page=show_problem&problem=989
- Ejercicio 3 : https://codeforces.com/problemset/problem/1245/D
- Ejercicio de yapa : https://vjudge.net/problem/UVA-1265
- Ejercicio bonus: Ejercicio mega picante

Gracias! 🎄