MST y algoritmo de Prim

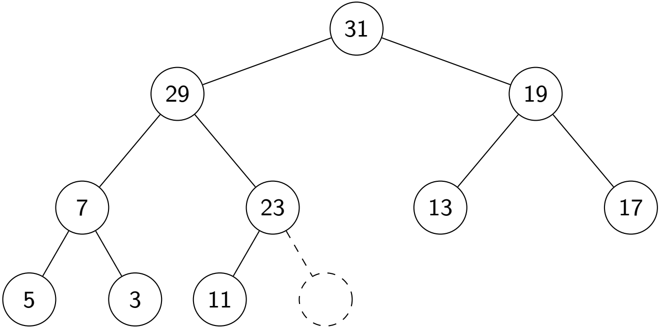
Clase 23

IIC 2133 - Sección 3

Prof. Eduardo Bustos

Sumario

Heaps binarios



Uso de la cola de prioridad

Uso de la cola de prioridad

Ya vimos que la cola de prioridad implementada como heap permite

Uso de la cola de prioridad

Ya vimos que la cola de prioridad implementada como heap permite

- Insertar valores con prioridad

Uso de la cola de prioridad

Ya vimos que la cola de prioridad implementada como heap permite

- Insertar valores con prioridad
- Extraer el más prioritario

Uso de la cola de prioridad

Ya vimos que la cola de prioridad implementada como heap permite

- Insertar valores con prioridad
- Extraer el más prioritario

En algunos contextos además necesitamos **actualizar** prioridad cuando cambie el costo óptimo

Uso de la cola de prioridad

Ya vimos que la cola de prioridad implementada como heap permite

- Insertar valores con prioridad
- Extraer el más prioritario

En algunos contextos además necesitamos **actualizar** prioridad cuando cambie el costo óptimo

1. Cambiamos prioridad del nodo del heap

Uso de la cola de prioridad

Ya vimos que la cola de prioridad implementada como heap permite

- Insertar valores con prioridad
- Extraer el más prioritario

En algunos contextos además necesitamos **actualizar** prioridad cuando cambie el costo óptimo

1. Cambiamos prioridad del nodo del heap
2. Hacemos intercambios hacia arriba si es necesario

Uso de la cola de prioridad

Ya vimos que la cola de prioridad implementada como heap permite

- Insertar valores con prioridad
- Extraer el más prioritario

En algunos contextos además necesitamos **actualizar** prioridad cuando cambie el costo óptimo

1. Cambiamos prioridad del nodo del heap
2. Hacemos intercambios hacia arriba si es necesario

La propiedad de heap permite que esta operación
solo afecte nodos en la ruta del nodo a la raíz

Uso de la cola de prioridad

input : heap representado como arreglo $H[0 \dots n-1]$,
índice i,
nueva prioridad $k > H[i]$

$\texttt{IncreaseKey}(H, i, k)$**:**

$H[i] \leftarrow k$

while $i > 0 \land H[\texttt{Parent}(i)] < H[i]$ **:**

$H[i] \leftrightharpoons H[\texttt{Parent}(i)]$

$i \leftarrow \texttt{Parent}(i)$

En tiempo $\mathcal{O}(\log(V))$ actualizamos la prioridad y mantenemos el heap...

Uso de la cola de prioridad

input : heap representado como arreglo $H[0 \ldots n-1]$,
índice i,
nueva prioridad $k > H[i]$

$\texttt{IncreaseKey}(H, i, k)$**:**

$H[i] \leftarrow k$

while $i > 0 \wedge H[\texttt{Parent}(i)] < H[i]$ **:**

$H[i] \leftrightharpoons H[\texttt{Parent}(i)]$

$i \leftarrow \texttt{Parent}(i)$

En tiempo $\mathcal{O}(\log(V))$ actualizamos la prioridad y mantenemos el heap...
¡Esta operación será necesaria en nuestro algoritmo de hoy!

Conectividad digital

Conectividad digital

Consideremos el problema de mejorar la conectividad digital de la Región del Maule

Conectividad digital

Consideremos el problema de mejorar la conectividad digital de la Región del Maule

- Objetivo: instalar fibra óptica subterránea entre pares de puntos relevantes

Conectividad digital

Consideremos el problema de mejorar la conectividad digital de la Región del Maule

- Objetivo: instalar fibra óptica subterránea entre pares de puntos relevantes
- Cada instalación de ese cableado tiene un costo

Conectividad digital

Consideremos el problema de mejorar la conectividad digital de la Región del Maule

- Objetivo: instalar fibra óptica subterránea entre pares de puntos relevantes
- Cada instalación de ese cableado tiene un costo
- Es prioritario conectar ciudades más pobladas

Conectividad digital

Consideremos el problema de mejorar la conectividad digital de la Región del Maule

- Objetivo: instalar fibra óptica subterránea entre pares de puntos relevantes
- Cada instalación de ese cableado tiene un costo
- Es prioritario conectar ciudades más pobladas

El desafío es **cubrir** con el menor costo

Conectividad digital

Consideremos el problema de mejorar la conectividad digital de la Región del Maule

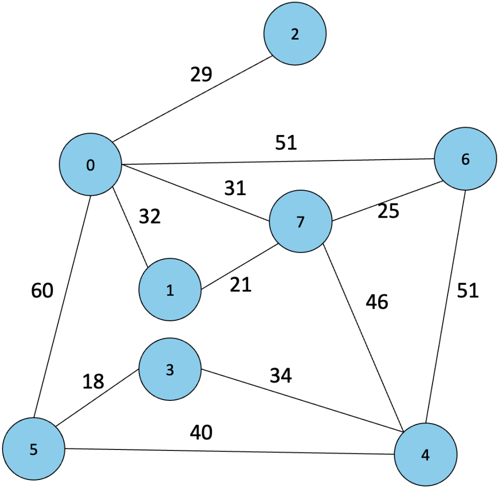
- Objetivo: instalar fibra óptica subterránea entre pares de puntos relevantes
- Cada instalación de ese cableado tiene un costo
- Es prioritario conectar ciudades más pobladas

El desafío es **cubrir** con el menor costo

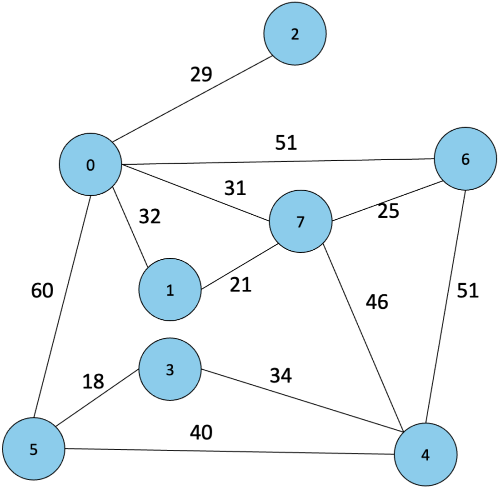
¿Ya tenemos una estrategia para atacar este problema?

Conectividad digital

Conectividad digital



Conectividad digital



Usamos un grafo **no dirigido con costos**

Conectividad digital

Usamos un grafo no dirigido con costos

Conectividad digital

Usamos un grafo no dirigido con costos

- Cada nodo es un punto a conectar (ciudad, pueblo, etc)

Conectividad digital

Usamos un grafo no dirigido con costos

- Cada nodo es un punto a conectar (ciudad, pueblo, etc)
- Cada arista tiene un costo de conexión

Conectividad digital

Usamos un grafo no dirigido con costos

- Cada nodo es un punto a conectar (ciudad, pueblo, etc)
- Cada arista tiene un costo de conexión
- Consideramos que cada nodo presente en el grafo, debe ser conectado

Conectividad digital

Usamos un grafo no dirigido con costos

- Cada nodo es un punto a conectar (ciudad, pueblo, etc)
- Cada arista tiene un costo de conexión
- Consideramos que cada nodo presente en el grafo, debe ser conectado

Para resolver el problema seleccionamos un subconjunto de aristas

Conectividad digital

Usamos un grafo no dirigido con costos

- Cada nodo es un punto a conectar (ciudad, pueblo, etc)
- Cada arista tiene un costo de conexión
- Consideramos que cada nodo presente en el grafo, debe ser conectado

Para resolver el problema seleccionamos un subconjunto de aristas

- La suma de los costos debe ser **mínima**

Conectividad digital

Usamos un grafo no dirigido con costos

- Cada nodo es un punto a conectar (ciudad, pueblo, etc)
- Cada arista tiene un costo de conexión
- Consideramos que cada nodo presente en el grafo, debe ser conectado

Para resolver el problema seleccionamos un subconjunto de aristas

- La suma de los costos debe ser **mínima**
- El **subgrafo** que solo considera esas aristas, debe ser **conexo**

Conectividad digital

Usamos un grafo no dirigido con costos

- Cada nodo es un punto a conectar (ciudad, pueblo, etc)
- Cada arista tiene un costo de conexión
- Consideramos que cada nodo presente en el grafo, debe ser conectado

Para resolver el problema seleccionamos un subconjunto de aristas

- La suma de los costos debe ser **mínima**
- El **subgrafo** que solo considera esas aristas, debe ser **conexo**

Buscamos un MST

Objetivos de la clase

Objetivos de la clase

- ☐ Modificar prioridad dentro de un heap

Objetivos de la clase

- ☐ Modificar prioridad dentro de un heap
- ☐ Comprender una segunda forma de resolver el problema de MST

Objetivos de la clase

- ☐ Modificar prioridad dentro de un heap
- ☐ Comprender una segunda forma de resolver el problema de MST
- ☐ Demostrar correctitud y complejidad del algoritmo de Prim

Sumario

Recordemos: cortes

Para atacar el problema algorítmicamente, dado G no dirigido

Recordemos: cortes

Para atacar el problema algorítmicamente, dado G no dirigido

- Llamamos **corte** a una **partición** (V_1, V_2) de $V(G)$

Recordemos: cortes

Para atacar el problema algorítmicamente, dado G no dirigido

- Llamamos **corte** a una **partición** (V_1, V_2) de $V(G)$

$$V_1, V_2 \neq \emptyset, \qquad V_1 \cup V_2 = V(G), \qquad V_1 \cap V_2 = \emptyset$$

Recordemos: cortes

Para atacar el problema algorítmicamente, dado G no dirigido

- Llamamos **corte** a una **partición** (V_1, V_2) de $V(G)$

$$V_1, V_2 \neq \varnothing, \qquad V_1 \cup V_2 = V(G), \qquad V_1 \cap V_2 = \varnothing$$

- Diremos que una arista **cruza el corte** si uno de sus extremos está en V_1 y el otro en V_2

Recordemos: cortes

Para atacar el problema algorítmicamente, dado G no dirigido

- Llamamos **corte** a una **partición** (V_1, V_2) de $V(G)$

$$V_1, V_2 \neq \varnothing, \qquad V_1 \cup V_2 = V(G), \qquad V_1 \cap V_2 = \varnothing$$

- Diremos que una arista **cruza el corte** si uno de sus extremos está en V_1 y el otro en V_2

¿Qué podemos afirmar respecto a los MST y
las aristas que cruzan un corte dado?

Recordemos: cortes

Si T es un MST de G, y $P = (V_1, V_2)$ es un corte de G

Recordemos: cortes

Si T es un MST de G, y $P = (V_1, V_2)$ es un corte de G

- Sabemos que al menos una arista que cruza P está incluida en T

Recordemos: cortes

Si T es un MST de G, y $P = (V_1, V_2)$ es un corte de G

- Sabemos que al menos una arista que cruza P está incluida en T
- De lo contrario, no habría camino entre nodos de V_1 y V_2

Recordemos: cortes

Si T es un MST de G, y $P = (V_1, V_2)$ es un corte de G

- Sabemos que al menos una arista que cruza P está incluida en T
- De lo contrario, no habría camino entre nodos de V_1 y V_2
- Por lo tanto: T no sería de cobertura

Recordemos: cortes

Si T es un MST de G, y $P = (V_1, V_2)$ es un corte de G

- Sabemos que al menos una arista que cruza P está incluida en T
- De lo contrario, no habría camino entre nodos de V_1 y V_2
- Por lo tanto: T no sería de cobertura

Si $P = (V_1, V_2)$ es un corte de G

Recordemos: cortes

Si T es un MST de G, y $P = (V_1, V_2)$ es un corte de G

- Sabemos que al menos una arista que cruza P está incluida en T
- De lo contrario, no habría camino entre nodos de V_1 y V_2
- Por lo tanto: T no sería de cobertura

Si $P = (V_1, V_2)$ es un corte de G

- Sabemos que cada arista de corte tiene un costo asociado

Recordemos: cortes

Si T es un MST de G, y $P = (V_1, V_2)$ es un corte de G

- Sabemos que al menos una arista que cruza P está incluida en T
- De lo contrario, no habría camino entre nodos de V_1 y V_2
- Por lo tanto: T no sería de cobertura

Si $P = (V_1, V_2)$ es un corte de G

- Sabemos que cada arista de corte tiene un costo asociado
- La arista más barata **siempre** se incluye en **algún** MST

Recordemos: cortes

Si T es un MST de G, y $P = (V_1, V_2)$ es un corte de G

- Sabemos que al menos una arista que cruza P está incluida en T
- De lo contrario, no habría camino entre nodos de V_1 y V_2
- Por lo tanto: T no sería de cobertura

Si $P = (V_1, V_2)$ es un corte de G

- Sabemos que cada arista de corte tiene un costo asociado
- La arista más barata **siempre** se incluye en **algún** MST

¿Cómo podemos usar estos hechos para construir un MST desde cero?

Algoritmo de Prim

La idea detrás del **algoritmo de Prim** es utilizar las aristas que cruzan cortes para guiar la construcción

Algoritmo de Prim

La idea detrás del **algoritmo de Prim** es utilizar las aristas que cruzan cortes para guiar la construcción

Para un grafo $G = (V, E)$ y un nodo inicial v

Algoritmo de Prim

La idea detrás del **algoritmo de Prim** es utilizar las aristas que cruzan cortes para guiar la construcción

Para un grafo $G = (V, E)$ y un nodo inicial v

1. Sean $R = \{v\}$ y $\bar{R} = V - R$

Algoritmo de Prim

La idea detrás del **algoritmo de Prim** es utilizar las aristas que cruzan cortes para guiar la construcción

Para un grafo $G = (V, E)$ y un nodo inicial v

1. Sean $R = \{v\}$ y $\bar{R} = V - R$
2. Sea e la arista de menor costo que cruza de R a $\bar{R}$

Algoritmo de Prim

La idea detrás del **algoritmo de Prim** es utilizar las aristas que cruzan cortes para guiar la construcción

Para un grafo $G = (V, E)$ y un nodo inicial v

1. Sean $R = \{v\}$ y $\bar{R} = V - R$
2. Sea e la arista de menor costo que cruza de R a $\bar{R}$
3. Sea u el nodo de e que pertenece a $\bar{R}$

Algoritmo de Prim

La idea detrás del **algoritmo de Prim** es utilizar las aristas que cruzan cortes para guiar la construcción

Para un grafo $G = (V, E)$ y un nodo inicial v

1. Sean $R = \{v\}$ y $\bar{R} = V - R$
2. Sea e la arista de menor costo que cruza de R a $\bar{R}$
3. Sea u el nodo de e que pertenece a $\bar{R}$
4. Agregar e al MST. Eliminar u de $\bar{R}$ y agregarlo a R

Algoritmo de Prim

La idea detrás del **algoritmo de Prim** es utilizar las aristas que cruzan cortes para guiar la construcción

Para un grafo $G = (V, E)$ y un nodo inicial v

1. Sean $R = \{v\}$ y $\bar{R} = V - R$
2. Sea e la arista de menor costo que cruza de R a $\bar{R}$
3. Sea u el nodo de e que pertenece a $\bar{R}$
4. Agregar e al MST. Eliminar u de $\bar{R}$ y agregarlo a R
5. Si quedan elementos en $\bar{R}$, volver al paso 2.

Algoritmo de Prim

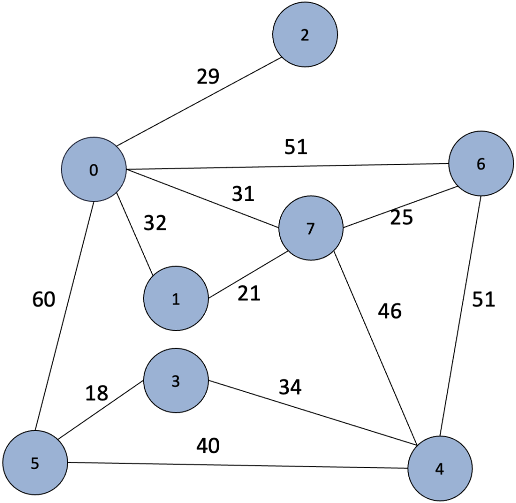
La idea detrás del **algoritmo de Prim** es utilizar las aristas que cruzan cortes para guiar la construcción

Para un grafo $G = (V, E)$ y un nodo inicial v

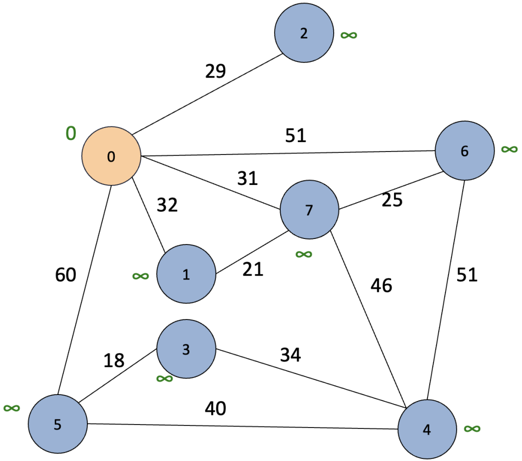
1. Sean $R = \{v\}$ y $\bar{R} = V - R$
2. Sea e la arista de menor costo que cruza de R a $\bar{R}$
3. Sea u el nodo de e que pertenece a $\bar{R}$
4. Agregar e al MST. Eliminar u de $\bar{R}$ y agregarlo a R
5. Si quedan elementos en $\bar{R}$, volver al paso 2.

¿Cómo hacer eficiente el paso 2?

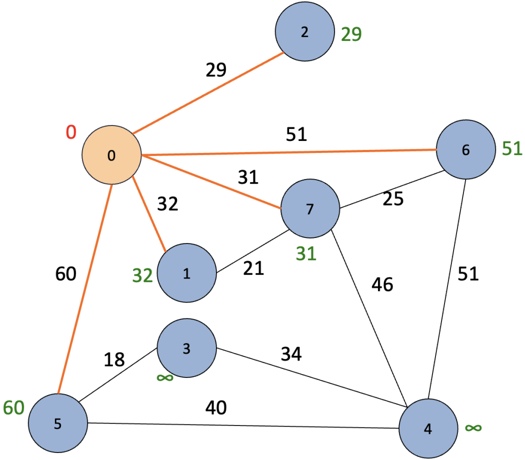
Árboles de cobertura mínimos



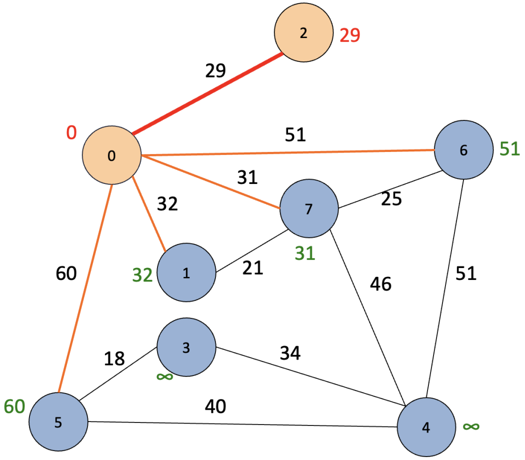
Árboles de cobertura mínimos



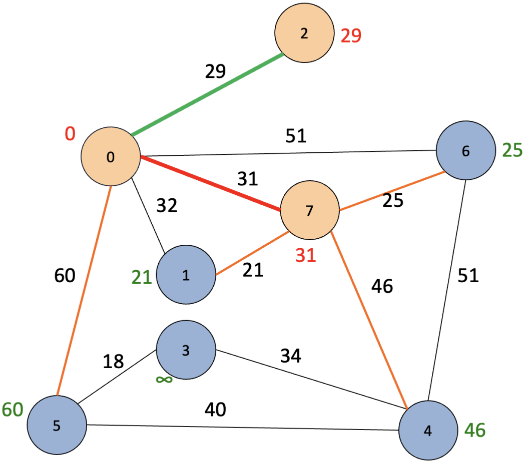
Árboles de cobertura mínimos



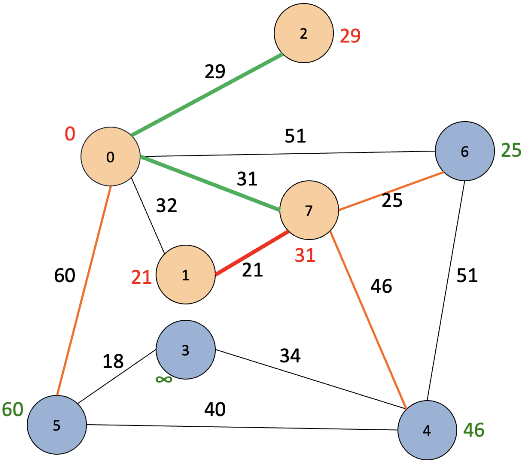
Árboles de cobertura mínimos



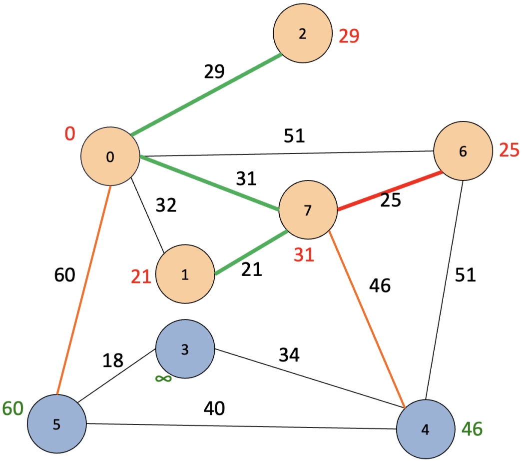
Árboles de cobertura mínimos



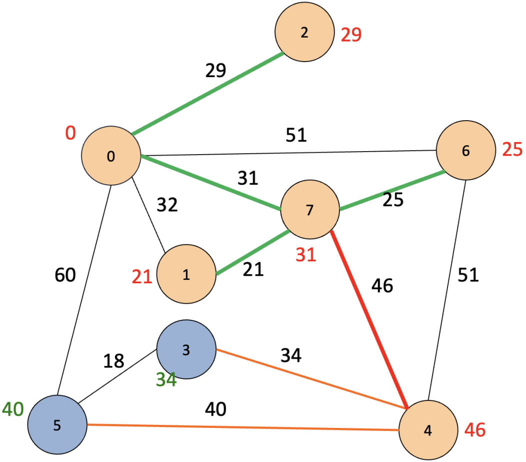
Árboles de cobertura mínimos



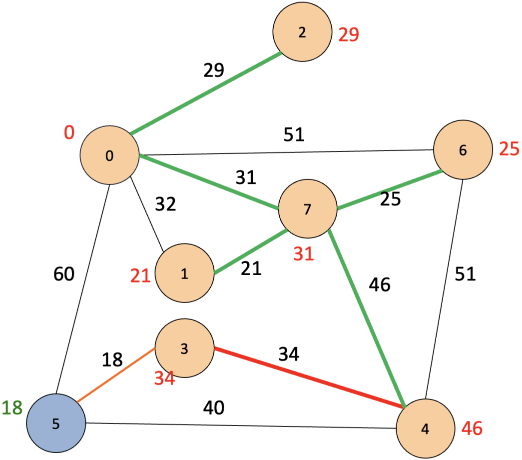
Árboles de cobertura mínimos



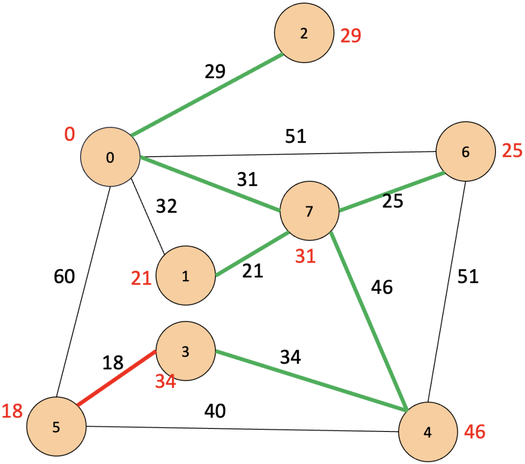
Árboles de cobertura mínimos



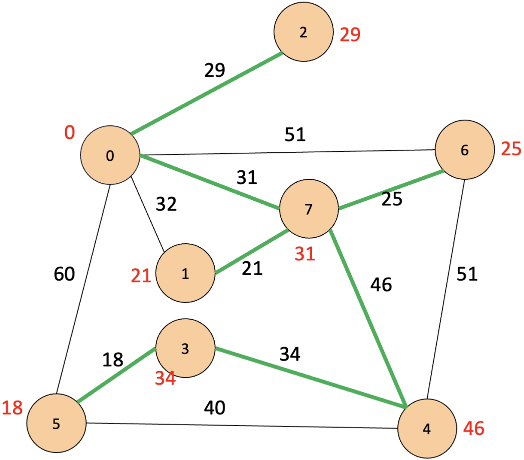
Árboles de cobertura mínimos



Árboles de cobertura mínimos



Árboles de cobertura mínimos



Algoritmo de Prim

```
Prim(s):
    Q ← cola de prioridades vacía (Min Heap)
    T ← lista vacía
    for u ∈ V − {s} :
        d[u] ← ∞;  π[u] ← ∅;  Insert(Q, u)
    d[s] ← 0;  π[s] ← ∅;  Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        T ← T ∪ {(π[u], u)}
        for v ∈ α[u] :
            if v ∈ Q :
                if d[v] > cost(u, v) :
                    d[v] ← cost(u, v);  π[v] ← u
                    DecreaseKey(Q, v, d[v])
    return T
```

Q usa como prioridad el valor $d[v]$.
DecreaseKey(Q, v, k) cambia la prioridad del elemento v

Sumario

Correctitud

Demostración

Correctitud.

Correctitud

Demostración

Correctitud. Fijaremos nuestra atención en la línea 8 del algoritmo, justo luego de agregar a T una nueva arista. Denotaremos por G_n al subgrafo de G tal que considera solo los primeros n nodos extraídos de Q con todas sus aristas de G.

Correctitud

Demostración

Correctitud. Fijaremos nuestra atención en la línea 8 del algoritmo, justo luego de agregar a T una nueva arista. Denotaremos por G_n al subgrafo de G tal que considera solo los primeros n nodos extraídos de Q con todas sus aristas de G.

Probaremos por inducción sobre el número de iteraciones la propiedad

$$P(n) := \quad \text{En la iteración } n\text{-ésima, la línea 8 guarda en } T \text{ un MST para el subgrafo } G_n$$

Correctitud

Demostración

C.B. Para $n = 1$,

Correctitud

Demostración

C.B. Para $n = 1$, extraemos de Q el nodo inicial y agregamos una arista *dummy*. Como el grafo que debemos cubrir es $G_1 = (\{s\}, \emptyset)$ y T no cubre nada más que el nodo inicial, es efectivamente un MST para G_1.

Correctitud

Demostración

C.B. Para $n = 1$, extraemos de Q el nodo inicial y agregamos una arista *dummy*. Como el grafo que debemos cubrir es $G_1 = (\{s\}, \emptyset)$ y T no cubre nada más que el nodo inicial, es efectivamente un MST para G_1.

H.I. Suponemos que en la iteración n, T tiene guardado un MST para cubrir el subgrafo G_n con costo mínimo.

Correctitud

Demostración

C.B. Para $n = 1$, extraemos de Q el nodo inicial y agregamos una arista *dummy*. Como el grafo que debemos cubrir es $G_1 = (\{s\}, \varnothing)$ y T no cubre nada más que el nodo inicial, es efectivamente un MST para G_1.

H.I. Suponemos que en la iteración n, T tiene guardado un MST para cubrir el subgrafo G_n con costo mínimo.

T.I. Sea u el $(n+1)$-ésimo nodo extraído de la cola Q. Sea además T la variable guardado en la iteración anterior.

Correctitud

Demostración

C.B. Para $n = 1$, extraemos de Q el nodo inicial y agregamos una arista *dummy*. Como el grafo que debemos cubrir es $G_1 = (\{s\}, \varnothing)$ y T no cubre nada más que el nodo inicial, es efectivamente un MST para G_1.

H.I. Suponemos que en la iteración n, T tiene guardado un MST para cubrir el subgrafo G_n con costo mínimo.

T.I. Sea u el $(n+1)$-ésimo nodo extraído de la cola Q. Sea además T la variable guardado en la iteración anterior.

- Como u fue extraído en la iteración $n+1$, significa que es el nodo con la arista más barata para conectar directamente con **algún nodo** de G_n.

Correctitud

Demostración

C.B. Para $n = 1$, extraemos de Q el nodo inicial y agregamos una arista *dummy*. Como el grafo que debemos cubrir es $G_1 = (\{s\}, \varnothing)$ y T no cubre nada más que el nodo inicial, es efectivamente un MST para G_1.

H.I. Suponemos que en la iteración n, T tiene guardado un MST para cubrir el subgrafo G_n con costo mínimo.

T.I. Sea u el $(n+1)$-ésimo nodo extraído de la cola Q. Sea además T la variable guardado en la iteración anterior.

- Como u fue extraído en la iteración $n+1$, significa que es el nodo con la arista más barata para conectar directamente con **algún nodo** de G_n.
- A saber, dicho nodo es $\pi[u]$.

Correctitud

Demostración

C.B. Para $n = 1$, extraemos de Q el nodo inicial y agregamos una arista *dummy*. Como el grafo que debemos cubrir es $G_1 = (\{s\}, \emptyset)$ y T no cubre nada más que el nodo inicial, es efectivamente un MST para G_1.

H.I. Suponemos que en la iteración n, T tiene guardado un MST para cubrir el subgrafo G_n con costo mínimo.

T.I. Sea u el $(n+1)$-ésimo nodo extraído de la cola Q. Sea además T la variable guardado en la iteración anterior.

- Como u fue extraído en la iteración $n+1$, significa que es el nodo con la arista más barata para conectar directamente con **algún nodo** de G_n.
- A saber, dicho nodo es $\pi[u]$.
- Al agregar la arista $(\pi[u], u)$ a T, obtenemos un nuevo conjunto T' de G_{n+1}. Basta argumentar sus propiedades.

Correctitud

Demostración

T' es **cubrimiento**

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).
- Al agregar $(\pi[u], u)$, se conecta también a u. Por lo tanto, T' cubre todo G_{n+1}

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).
- Al agregar $(\pi[u], u)$, se conecta también a u. Por lo tanto, T' cubre todo G_{n+1}

T' es **árbol**

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).
- Al agregar $(\pi[u], u)$, se conecta también a u. Por lo tanto, T' cubre todo G_{n+1}

T' es **árbol**

- Por **H.I.**, T es árbol.

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).
- Al agregar $(\pi[u], u)$, se conecta también a u. Por lo tanto, T' cubre todo G_{n+1}

T' es **árbol**

- Por **H.I.**, T es árbol.
- Al agregar $(\pi[u], u)$, se conecta un nodo ya incluído en G_n con uno **no incluído**, de forma que no se producen ciclos y T' no es árbol

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).
- Al agregar $(\pi[u], u)$, se conecta también a u. Por lo tanto, T' cubre todo G_{n+1}

T' es **árbol**

- Por **H.I.**, T es árbol.
- Al agregar $(\pi[u], u)$, se conecta un nodo ya incluído en G_n con uno **no incluído**, de forma que no se producen ciclos y T' no es árbol

T' es de **costo mínimo**

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).
- Al agregar $(\pi[u], u)$, se conecta también a u. Por lo tanto, T' cubre todo G_{n+1}

T' es **árbol**

- Por **H.I.**, T es árbol.
- Al agregar $(\pi[u], u)$, se conecta un nodo ya incluído en G_n con uno **no incluído**, de forma que no se producen ciclos y T' no es árbol

T' es de **costo mínimo**

- Por **H.I.**, T es de costo mínimo para G_n.

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).
- Al agregar $(\pi[u], u)$, se conecta también a u. Por lo tanto, T' cubre todo G_{n+1}

T' es **árbol**

- Por **H.I.**, T es árbol.
- Al agregar $(\pi[u], u)$, se conecta un nodo ya incluído en G_n con uno **no incluído**, de forma que no se producen ciclos y T' no es árbol

T' es de **costo mínimo**

- Por **H.I.**, T es de costo mínimo para G_n.
- La elección de u asegura que se agrega la arista más barata para conectar u a G_n. Se concluye que T' es de costo mínimo.

Correctitud

Demostración

T' es **cubrimiento**

- Como T es MST para G_n por **H.I.**, entonces conecta todos los nodos de G_n (incluyendo a $\pi[u]$).
- Al agregar $(\pi[u], u)$, se conecta también a u. Por lo tanto, T' cubre todo G_{n+1}

T' es **árbol**

- Por **H.I.**, T es árbol.
- Al agregar $(\pi[u], u)$, se conecta un nodo ya incluído en G_n con uno **no incluído**, de forma que no se producen ciclos y T' no es árbol

T' es de **costo mínimo**

- Por **H.I.**, T es de costo mínimo para G_n.
- La elección de u asegura que se agrega la arista más barata para conectar u a G_n. Se concluye que T' es de costo mínimo.

Correctitud

Demostración

Finitud.

Correctitud

Demostración

Finitud. Es claro que el algoritmo termina, pues no visita nodos ya descubiertos y cada arista es revisada a lo más una vez. Como el grafo es finito, el algoritmo es finito y termina cuando $n = |V|$.

Correctitud

Demostración

Finitud. Es claro que el algoritmo termina, pues no visita nodos ya descubiertos y cada arista es revisada a lo más una vez. Como el grafo es finito, el algoritmo es finito y termina cuando $n = |V|$.

De lo anterior se concluye que $P(n)$ es cierta. En particular, como $G_{|V|} = G$

$$P(|V|) \text{ verdadera} \quad \Leftrightarrow \quad \texttt{Prim} \text{ entrega MST para } G$$

$\square$

Correctitud

Demostración

Finitud. Es claro que el algoritmo termina, pues no visita nodos ya descubiertos y cada arista es revisada a lo más una vez. Como el grafo es finito, el algoritmo es finito y termina cuando $n = |V|$.

De lo anterior se concluye que $P(n)$ es cierta. En particular, como $G_{|V|} = G$

$$P(|V|) \text{ verdadera} \quad \Leftrightarrow \quad \texttt{Prim} \text{ entrega MST para } G$$

$\square$

No olvidar: no necesariamente hay un único MST para G

Complejidad

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo $\mathcal{O}(\log(V))$

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo $\mathcal{O}(\log(V))$
- Cada arista se revisa exactamente una vez

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo $\mathcal{O}(\log(V))$
- Cada arista se revisa exactamente una vez $\mathcal{O}(E)$

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo $\mathcal{O}(\log(V))$
- Cada arista se revisa exactamente una vez $\mathcal{O}(E)$
- Por cada revisión se hace una actualización de costo

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo $\mathcal{O}(\log(V))$
- Cada arista se revisa exactamente una vez $\mathcal{O}(E)$
- Por cada revisión se hace una actualización de costo $\mathcal{O}(\log(V))$

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo $\mathcal{O}(\log(V))$
- Cada arista se revisa exactamente una vez $\mathcal{O}(E)$
- Por cada revisión se hace una actualización de costo $\mathcal{O}(\log(V))$
- Total

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo $\mathcal{O}(\log(V))$
- Cada arista se revisa exactamente una vez $\mathcal{O}(E)$
- Por cada revisión se hace una actualización de costo $\mathcal{O}(\log(V))$
- Total $\mathcal{O}((V+E)\log(V))$

Complejidad

Considerando que usamos una cola de prioridad implementada como heap

- Extracción del más prioritario $\mathcal{O}(\log(V))$
- Actualización de prioridad cuando cambia el costo $\mathcal{O}(\log(V))$

La complejidad en el peor caso viene dada por las iteraciones del **while**

- El loop ocurre $|V|$ veces $\mathcal{O}(V)$
- En cada loop se extrae un nodo $\mathcal{O}(\log(V))$
- Cada arista se revisa exactamente una vez $\mathcal{O}(E)$
- Por cada revisión se hace una actualización de costo $\mathcal{O}(\log(V))$
- Total $\mathcal{O}((V + E)\log(V))$

Podemos simplificar este último resultado

Complejidad

Recordemos que para grafos conexos existe una relación entre $|V|$ y $|E|$

Complejidad

Recordemos que para grafos conexos existe una relación entre $|V|$ y $|E|$

- Si G es un árbol, $E = V - 1$

Complejidad

Recordemos que para grafos conexos existe una relación entre $|V|$ y $|E|$

- Si G es un árbol, $E = V - 1$
- Si G es conexo cualquiera, $E \geq V - 1$

Complejidad

Recordemos que para grafos conexos existe una relación entre $|V|$ y $|E|$

- Si G es un árbol, $E = V - 1$
- Si G es conexo cualquiera, $E \geq V - 1$

Concluímos que $V \in \mathcal{O}(E)$ para grafos conexos

Complejidad

Recordemos que para grafos conexos existe una relación entre $|V|$ y $|E|$

- Si G es un árbol, $E = V - 1$
- Si G es conexo cualquiera, $E \geq V - 1$

Concluímos que $V \in \mathcal{O}(E)$ para grafos conexos

- Luego, $(V + E) \in \mathcal{O}(E)$

Complejidad

Recordemos que para grafos conexos existe una relación entre $|V|$ y $|E|$

- Si G es un árbol, $E = V - 1$
- Si G es conexo cualquiera, $E \geq V - 1$

Concluímos que $V \in \mathcal{O}(E)$ para grafos conexos

- Luego, $(V + E) \in \mathcal{O}(E)$

El algoritmo de Prim toma tiempo $\mathcal{O}(E \log(V))$

Algoritmo de Prim: una versión más concreta

```
Prim(s):
    Q ← cola de prioridades con s (Min Heap)
    T ← lista vacía
    s.key ← 0;  s.parent ← ∅
    while Q no está vacía :
        u ← Extract(Q);  u.color ← negro
        if u.parent ≠ ∅ :
            T ← T ∪ {(u.parent, u)}
        for v ∈ α[u] ∧ v.color ≠ negro :
            if v ∉ Q :
                Insert(Q, v)
            if v.key > cost(u, v) :
                v.key ← cost(u, v);  v.parent ← u
                DecreaseKey(Q, v, v.key)
    return T
```

Suponemos que inicialmente $v.key \leftarrow \infty$ para todo v

Sumario

Objetivos de la clase

Objetivos de la clase

- ☐ Modificar prioridad dentro de un heap

Objetivos de la clase

- ☐ Modificar prioridad dentro de un heap
- ☐ Comprender una segunda forma de resolver el problema de MST

Objetivos de la clase

- ☐ Modificar prioridad dentro de un heap
- ☐ Comprender una segunda forma de resolver el problema de MST
- ☐ Demostrar correctitud y complejidad del algoritmo de Prim