



ESTRATEGIAS DE PROGRAMACIÓN (CONTINUACIÓN-1)

Programación 3
Javier Miranda

Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

Estrategias básicas de programación

- Fuerza bruta
 - Vuelta atrás (*backtracking*)
 - Voráz (*greedy*)
-
- Divide y vencerás
 - reduce y vencerás
 - Programación Dinámica

Fuerza Bruta (*Brute Force*)

- Basada en la definición del problema
 - Evalúa *todas las posibles combinaciones* que resuelven el problema
- Fortalezas:
 - Amplia aplicabilidad
 - Simple
 - Genera soluciones razonables para algunos problemas
- *Debilidades:*
 - Algoritmos poco eficientes
 - En general, con poco esfuerzo podemos hacerlo mucho mejor !

https://en.wikipedia.org/wiki/Brute-force_search

Vuelta atrás (*Backtracking*)

- Estrategia de búsqueda que descarta (*poda del árbol*) las combinaciones que no llevan a la solución.

Algoritmo Genérico

1. Generamos una combinación componente a componente
2. Evaluamos si nos puede llevar hacia la solución
3. Si no satisface alguna restricción del problema descartamos esta solución (y todas las que dependan de ella)
Poda
4. Volvemos al paso 1

Backtracking es una estrategia de fuerza bruta con poda!

*Implementación
Iterativa*

Ejemplo: Sudoku con Backtracking

```
def solveSudoku(grid, i=0, j=0):  
    i,j = findNextCellToFill(grid, i, j)  
    if i == -1:  
        return True  
    for e in range(1,10):  
        if isValid(grid,i,j,e):  
            grid[i][j] = e  
            if solveSudoku(grid, i, j):  
                return True  
            # Undo the current cell for backtracking  
            grid[i][j] = 0  
    return False
```

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Estrategia Voraz (*Greedy*)

- También conocida como estrategia **ávida**, **miope**, o **avariciosa**

Algoritmo Genérico

- Se implementa mediante un bucle (*bucle voraz*)
- En cada paso:
 - Dispone de un conjunto de candidatos
 - Elige el “mejor”
 - La añade a la solución

Nunca deshace las decisiones tomadas

Devolución del cambio

- **Entrada:** Cantidad N a devolver
- **Salida:** Con las monedas disponibles, conjunto de monedas mínimo para devolver el cambio N



Devolución del cambio (algoritmo greedy)

- Reordenamos internamente las monedas de mayor a menor valor, recorreremos la lista de monedas y elegimos siempre la de mayor valor que nos acerca a nuestro objetivo!

<u>Monedas</u>		
<i>Posición</i>	1	→ 5
	2	→ 2
	3	→ 1
	4	→ 10
	5	→ 10
	6	→ 2
		<i>Valor</i>

<u>Monedas Ordenadas por Valor</u>		
<i>Posición</i>	4	← 10
	5	← 10
	1	← 5
	2	← 2
	6	← 2
	3	← 1
		<i>Valor</i>

Valor del cambio a devolver: 21

Solución óptima: 3 4 5

Elección: 3 4 5

¿ Funciona bien siempre ?



Sólo funciona bien en problemas donde la decisión del algoritmo voraz coincide con la decisión correcta para llegar a la solución óptima

Ejemplos donde no funciona bien (1/2)

- Primer ejemplo

<u>Monedas</u>	
Posición	Valor
1	→ 2
2	→ 8
3	→ 2
4	→ 10
5	→ 8
6	→ 2

Valor del cambio a devolver: 16

Solución óptima: 2 5

Monedas Ordenadas por Valor

Posición	Valor
4	← 10
5	← 8
2	← 8
6	← 2
1	← 2
3	← 2

Elección: 1 3 4 6

Con la estrategia voraz nos devuelve 4 monedas (en vez de 2)

Ejemplos donde no funciona bien (2/2)

- Segundo ejemplo

Monedas

<i>Posición</i>	1	→	5	<i>Valor</i>
	2	→	7	
	3	→	1	
	4	→	10	
	5	→	7	

Valor del cambio a devolver: 14

Solución óptima: 2 5

Monedas Ordenadas por Valor

<i>Posición</i>	4	←	10	<i>Valor</i>
	5	←	7	
	2	←	7	
	1	←	5	
	3	←	1	

Elección: No hay solución

Con la estrategia voraz no consigue encontrar ninguna solución

¿Cuando funciona bien Greedy en la devolución de cambio?

- Cuando se cumple la siguiente propiedad:

El valor de cada tipo de moneda es menor o igual que la mitad del valor que le precede.

De esta forma, si en un paso no se eligiese la mejor opción habría que utilizar al menos dos monedas

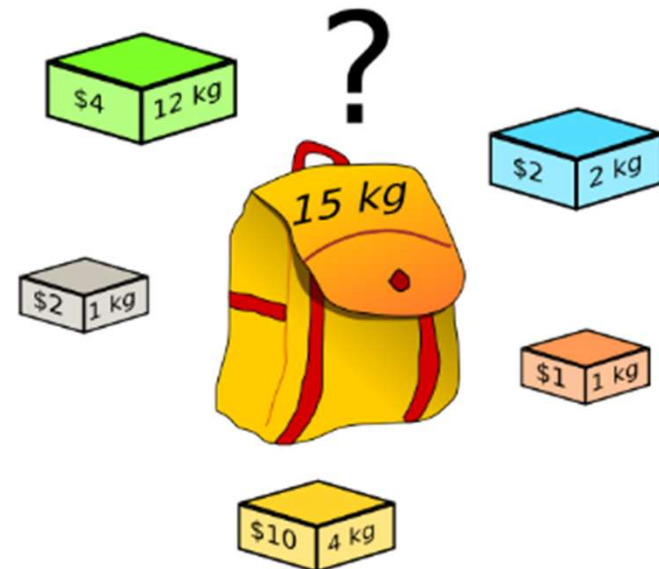


Problema de la mochila 0/1

Entrada: Peso de N items $\{w_1, w_2, \dots, w_n\}$
Coste de N items $\{c_1, c_2, \dots, c_n\}$
Mochila con un límite de peso S

Salida: Elección $\{x_1, x_2, \dots, x_n\}$
... donde $x_i \in \{0, 1\}$.

*Elección
binaria*



Problema de la mochila



The image illustrates the knapsack problem with various items and a knapsack. The items are arranged in two rows. The top row contains three identical Terracotta Warriors, each valued at \$1 Million and weighing 2kg, followed by two identical Egyptian scarabs, each valued at \$10 Million and weighing 5kg. The bottom row contains a golden mask valued at \$13 Million and weighing 8kg, and a golden coin valued at \$7 Million and weighing 3kg. To the right of these items is a large brown knapsack labeled 'Maximum Capacity 10kg'. Further to the right is a stick figure scratching its head with a question mark above it, indicating a decision problem.

Item	Value	Weight
Terracotta Warrior	\$1 Million	2kg
Terracotta Warrior	\$1 Million	2kg
Terracotta Warrior	\$1 Million	2kg
Egyptian Scarab	\$10 Million	5kg
Egyptian Scarab	\$10 Million	5kg
Golden Mask	\$13 Million	8kg
Golden Coin	\$7 Million	3kg
Knapsack Capacity	-	10kg

¿ Estrategia greedy ?

Problema de la mochila



Estrategia greedy #1: Primero la de más valor (reordenamos)

Problema de la mochila



Estrategia greedy #1: Primero la de más valor (reordenamos)

Problema de la mochila



\$13 Million
8kg



\$7 Million
3kg



\$10 Million
5kg



\$10 Million
5kg



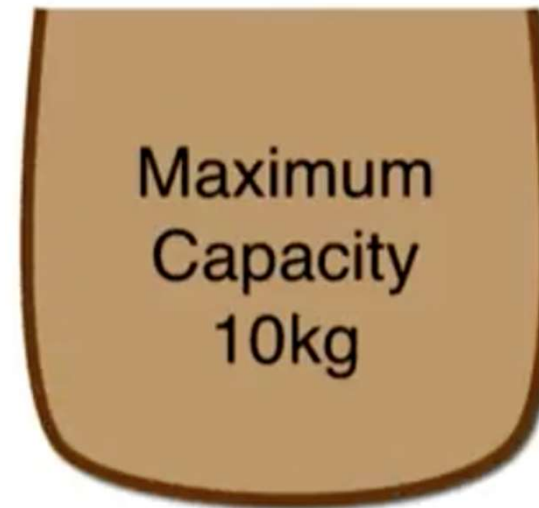
\$1 Million
2kg



\$1 Million
2kg



\$1 Million
2kg



Estrategia greedy #1: Primero la de más valor

Problema de la mochila



\$7 Million
3kg



\$10 Million
5kg



\$10 Million
5kg

\$13 Million
8kg

= 8 Kg



\$1 Million
2kg



\$1 Million
2kg



\$1 Million
2kg



Max = 10 Kg

Problema de la mochila



The image displays several ancient artifacts with their respective values and weights, arranged for a knapsack problem:

- Gold Coin:** \$7 Million, 3kg
- Papyrus-Bundle Tablet (left):** \$10 Million, 5kg
- Papyrus-Bundle Tablet (right):** \$10 Million, 5kg
- Combined Tablets:** \$13 Million, 8kg
- Terracotta Soldier (left):** \$1 Million, 2kg
- Terracotta Soldier (middle):** \$1 Million, 2kg
- Terracotta Soldier (right, highlighted with a dashed box):** \$1 Million, 2kg
- Golden Mask:** Max = 10 Kg

Additional annotations include:

- = 8 Kg:** A blue text annotation next to the combined tablets.
- Max = 10 Kg:** A blue text annotation next to the golden mask.

Problema de la mochila



\$7 Million
3kg



\$10 Million
5kg



\$10 Million
5kg

\$13 Million
8kg

\$1 Million
2kg

= 10 Kg



\$1 Million
2kg



\$1 Million
2kg



Max = 10 Kg

Problema de la mochila

14 millones de dolares
¿ Podemos hacerlo mejor ?



\$7 Million
3kg



\$10 Million
5kg



\$10 Million
5kg

\$13 Million
8kg

\$1 Million
2kg

= 10 Kg



\$1 Million
2kg



\$1 Million
2kg



Max = 10 Kg

Problema de la mochila



Estrategia greedy #2: Primero los menos pesados

Problema de la mochila



Item	Value	Weight
Terracotta Warrior	\$1 Million	2kg
Terracotta Warrior	\$1 Million	2kg
Terracotta Warrior	\$1 Million	2kg
Gold Coin	\$7 Million	3kg
Tablet with Hieroglyphs	\$10 Million	5kg
Stone Tablet	\$10 Million	5kg
Golden Mask	\$13 Million	8kg

Maximum Capacity 10kg

Estrategia greedy #2: Primero los menos pesados (reordenados)

Problema de la mochila



The image displays several ancient Egyptian artifacts, each with its value and weight listed below it:

- Three small statues (each): \$1 Million, 2kg
- Gold coin: \$7 Million, 3kg
- Papyrus tablet: \$10 Million, 5kg
- Wooden tablet: \$10 Million, 5kg
- Large golden mask: \$13 Million, 8kg

A brown bag icon represents the knapsack with the text: Maximum Capacity 10kg

Estrategia greedy #2: Primero los menos pesados

Problema de la mochila

Obtenemos un resultado peor:
10 millones de dolares



\$10 Million
5kg



\$10 Million
5kg



\$13 Million
8kg



\$1 Million \$1 Million \$1 Million \$7 Million
2kg 2kg 2kg 3kg

Evaluando las 4 alternativas básicas



\$1 Million
2kg



\$1 Million
2kg



\$1 Million
2kg



\$10 Million
5kg



\$10 Million
5kg



\$13 Million
8kg



\$7 Million
3kg

- 1) Primero la de más valor: \$14M
- 2) Primero la de menos valor: \$10M
- 3) Primero la de menos peso: \$10M
- 4) Primero la de más peso: \$14M

Problema de la mochila



The image illustrates the knapsack problem with various items and a knapsack. The items are arranged in two rows. The top row contains three Terracotta Warriors, two Egyptian scarabs, and a golden mask. The bottom row contains a golden coin and a large brown knapsack. To the right of the knapsack is a stick figure thinking.

Item	Value	Weight
Terracotta Warrior	\$1 Million	2kg
Terracotta Warrior	\$1 Million	2kg
Terracotta Warrior	\$1 Million	2kg
Egyptian Scarab	\$10 Million	5kg
Egyptian Scarab	\$10 Million	5kg
Golden Mask	\$13 Million	8kg
Golden Coin	\$7 Million	3kg
Knapsack Capacity	Maximum Capacity	10kg

¿ Hay alguna estrategia greedy mejor ?

Problema de la mochila



Three identical terracotta warrior figurines, each representing an item with a value of \$1 Million and a weight of 2kg.

\$1 Million
2kg

\$1 Million
2kg

\$1 Million
2kg



Two identical gold bars, each representing an item with a value of \$10 Million and a weight of 5kg.

\$10 Million
5kg

\$10 Million
5kg



A golden mask, representing an item with a value of \$13 Million and a weight of 8kg.

\$13 Million
8kg



A golden coin, representing an item with a value of \$7 Million and a weight of 3kg.

\$7 Million
3kg



A backpack representing the knapsack with a maximum capacity of 10kg.

Maximum
Capacity
10kg

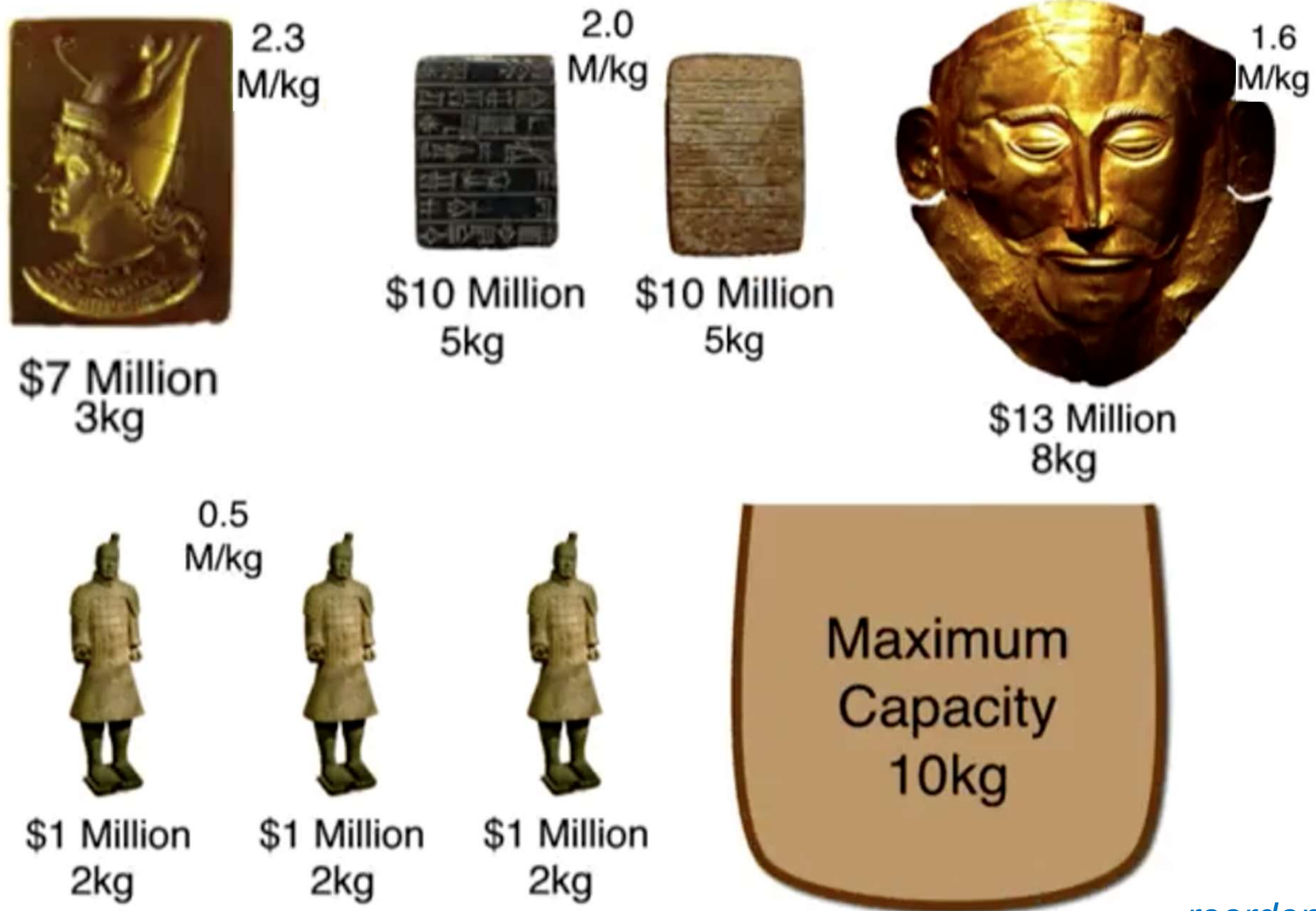
Estrategia greedy #3: Calculamos dolar por Kg

Problema de la mochila



Estrategia greedy #3: Calculamos dolar por Kg

Problema de la mochila



... reordenados

Problema de la mochila



Problema de la mochila



Problema de la mochila

Con esta estrategia conseguimos elegir 18 millones de dolares

2.0
M/kg



\$10 Million
5kg

1.6
M/kg



\$13 Million
8kg

0.5
M/kg



\$1 Million
2kg



\$1 Million
2kg



\$7 Million
3kg

\$10 Million
5kg

\$1 Million
2kg

Problema de la mochila

¿ Era la mejor elección ?

No. Eligiendo las dos tabletas habríamos tenido justo los 10 Kg y \$20M

2.0
M/kg



\$10 Million
5kg

1.6
M/kg



\$13 Million
8kg

0.5
M/kg



\$1 Million
2kg



\$1 Million
2kg



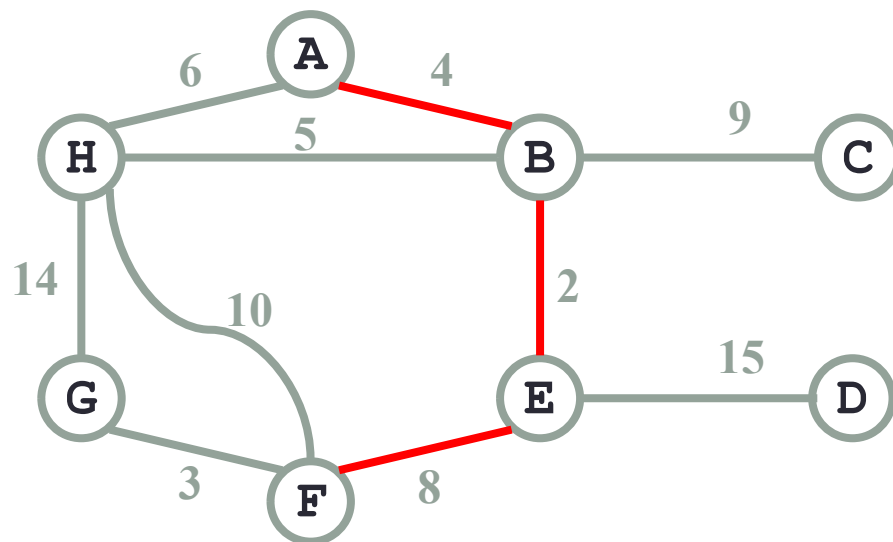
\$7 Million
3kg

\$10 Million
5kg

\$1 Million
2kg

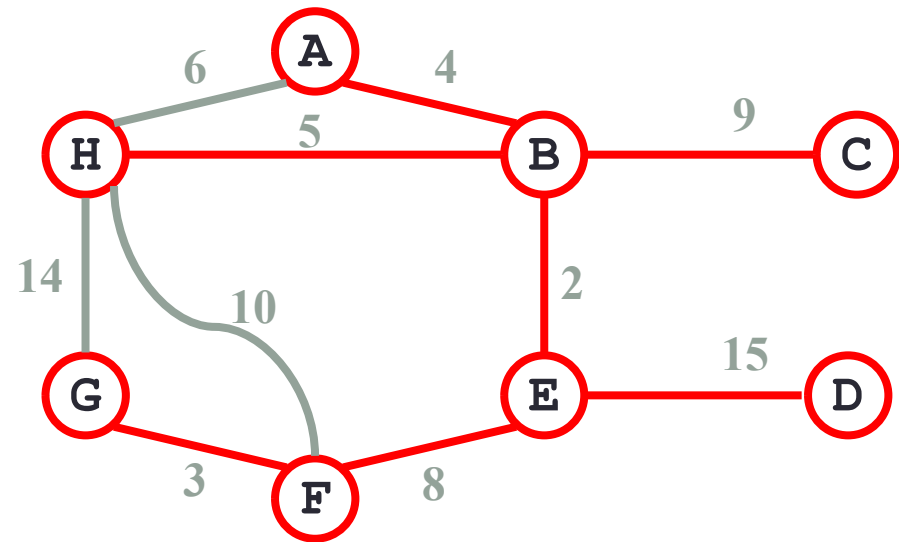
Algoritmos Greedy que ya conocemos

- En el curso anterior (en la asignatura *Matemática Discreta*) hemos visto:
 - Algoritmo de *Dijkstra* (ruta más corta)
 - Algoritmo de *Kruskal* (árbol de expansión mínimo)



Ruta más corta entre A y F

https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra



Árbol de expansión mínimo

https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal

Algoritmo Dijkstra ruta más corta

Identificando Greedy

Teniendo un grafo dirigido ponderado de N nodos no aislados, sea x el nodo inicial. Un vector D de tamaño N guardará al final del algoritmo las distancias desde x hasta el resto de los nodos.

1) Inicializar todas las distancias en D con un valor infinito relativo, ya que son desconocidas al principio, exceptuando la de x , que se debe colocar en 0, debido a que la distancia de x a x sería 0.

2) Sea $a = x$ (Se toma a como nodo actual.)

3) Se recorren todos los nodos adyacentes de a , excepto los nodos marcados. Se les llamará nodos no marcados V_i .

4) Para el nodo actual, se calcula la distancia tentativa desde dicho nodo hasta sus vecinos con la siguiente fórmula: $dt(V_i) = D_a + d(a, V_i)$. Es decir, la distancia tentativa del nodo ' V_i ' es la distancia que actualmente tiene el nodo en el vector D más la distancia desde dicho nodo ' a ' (el actual) hasta el nodo V_i . Si la distancia tentativa es menor que la distancia almacenada en el vector, entonces se actualiza el vector con esta distancia tentativa. Es decir, si $dt(V_i) < D_{V_i} \rightarrow D_{V_i} = dt(V_i)$

5) Se marca como completo el nodo a .

Elección Greedy

6) Se toma como próximo nodo actual el de menor valor en D (puede hacerse almacenando los valores en una cola de prioridad) y se regresa al paso 3, mientras existan nodos no marcados.

Una vez terminado el algoritmo, D estará completamente lleno.

Bucle Voraz

Algoritmo de Kruskal

Identificando Greedy

1) Ordenar la lista de aristas según su peso en orden creciente

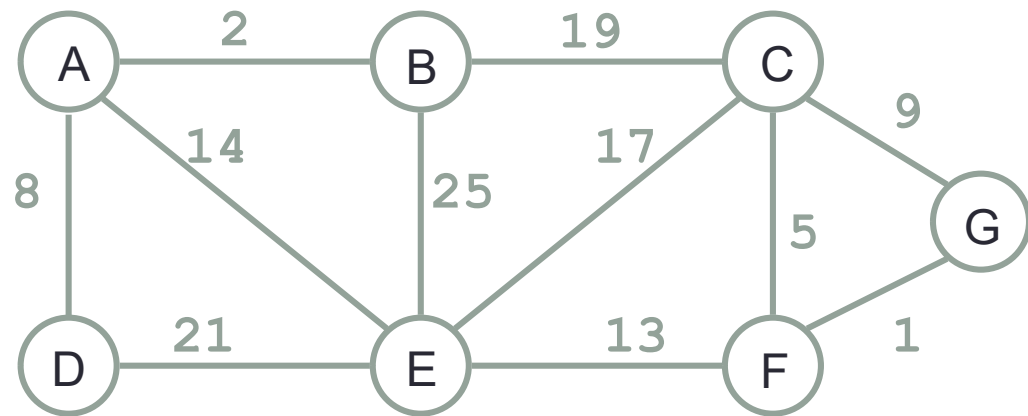
2) [Añadir la primera arista al subgrafo H] *Elección Greedy*

3) Seguir añadiendo aristas a H mientras no se forme un ciclo

4) Repetir hasta que el número de aristas sea $n-1$

Bucle Voraz

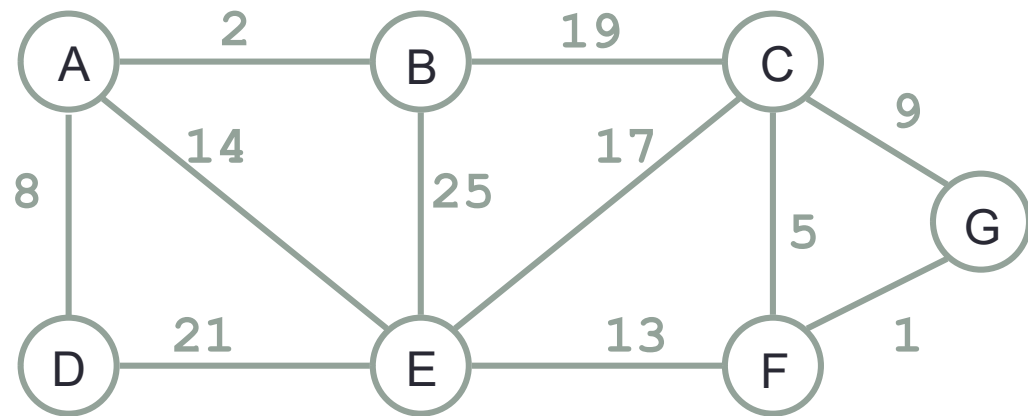
Algoritmo de Kruskal: Ejemplo



Aristas con su peso

A \leftrightarrow B ... 2
A \leftrightarrow E ... 14
A \leftrightarrow D ... 8
B \leftrightarrow C ... 19
B \leftrightarrow E ... 25
C \leftrightarrow E ... 17
C \leftrightarrow F ... 5
C \leftrightarrow G ... 9
D \leftrightarrow E ... 21
E \leftrightarrow F ... 13
F \leftrightarrow G ... 1

Algoritmo de Kruskal: Ejemplo

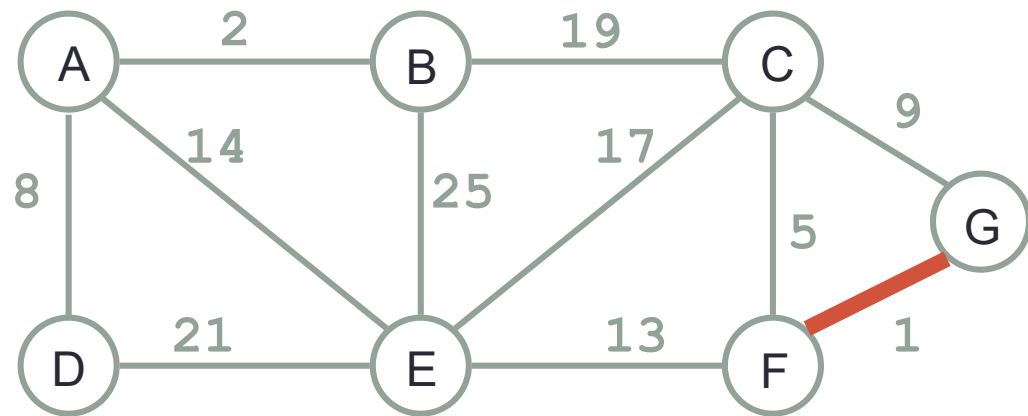


Aristas ordenadas

A ⇔ B ... 2	F ⇔ G ... 1
A ⇔ E ... 14	A ⇔ B ... 2
A ⇔ D ... 8	C ⇔ F ... 5
B ⇔ C ... 19	A ⇔ D ... 8
B ⇔ E ... 25	C ⇔ G ... 9
C ⇔ E ... 17	E ⇔ F ... 13
C ⇔ F ... 5	A ⇔ E ... 14
C ⇔ G ... 9	C ⇔ E ... 17
D ⇔ E ... 21	B ⇔ C ... 19
E ⇔ F ... 13	D ⇔ E ... 21
F ⇔ G ... 1	B ⇔ E ... 25

Paso 1: Ordenamos las aristas por su peso en orden ascendente

Algoritmo de Kruskal: Ejemplo

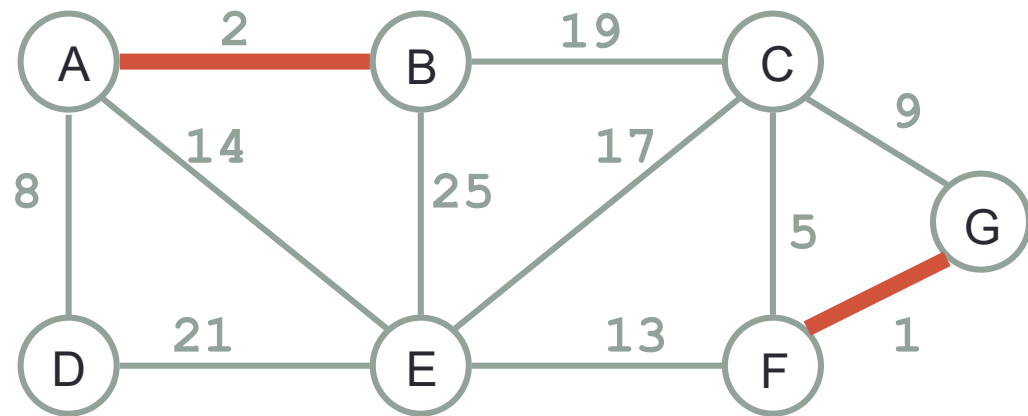


Aristas ordenadas

→ F ↔ G ... 1
A ↔ B ... 2
C ↔ F ... 5
A ↔ D ... 8
C ↔ G ... 9
E ↔ F ... 13
A ↔ E ... 14
C ↔ E ... 17
B ↔ C ... 19
D ↔ E ... 21
B ↔ E ... 25

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

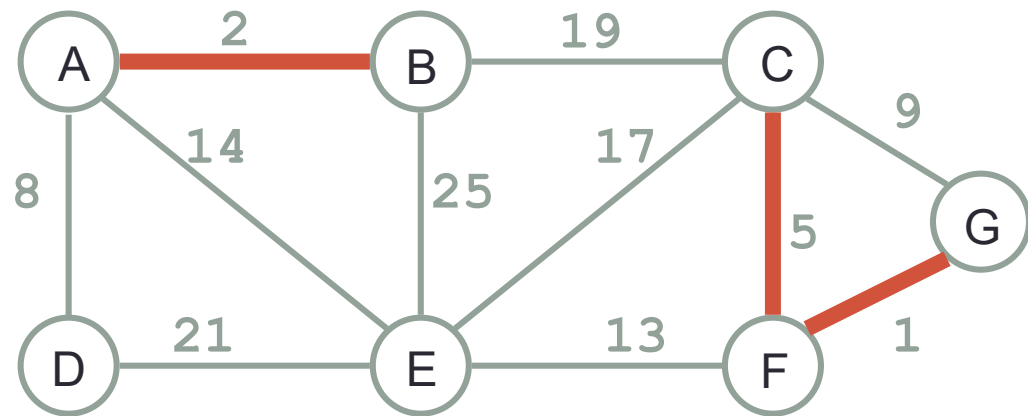
Algoritmo de Kruskal: Ejemplo



→ F ⇔ G ... 1
A ⇔ B ... 2
C ⇔ F ... 5
A ⇔ D ... 8
C ⇔ G ... 9
E ⇔ F ... 13
A ⇔ E ... 14
C ⇔ E ... 17
B ⇔ C ... 19
D ⇔ E ... 21
B ⇔ E ... 25

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

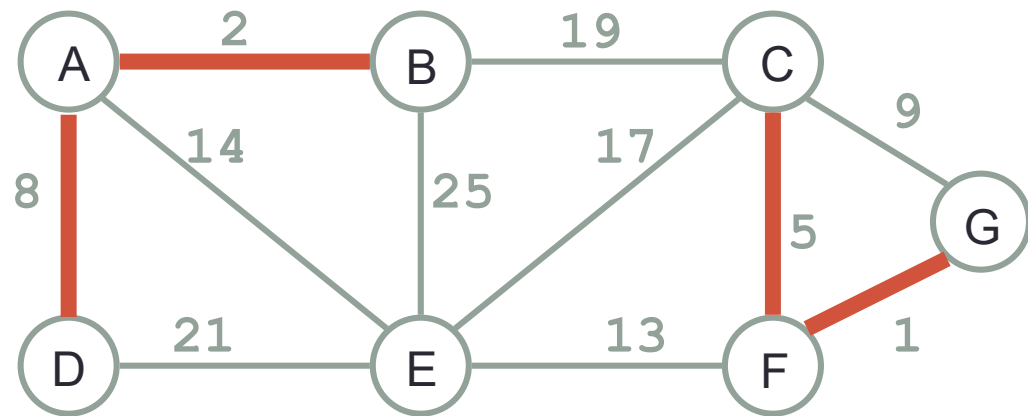
Algoritmo de Kruskal: Ejemplo



F ⇔ G ... 1
A ⇔ B ... 2
→ C ⇔ F ... 5
A ⇔ D ... 8
C ⇔ G ... 9
E ⇔ F ... 13
A ⇔ E ... 14
C ⇔ E ... 17
B ⇔ C ... 19
D ⇔ E ... 21
B ⇔ E ... 25

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

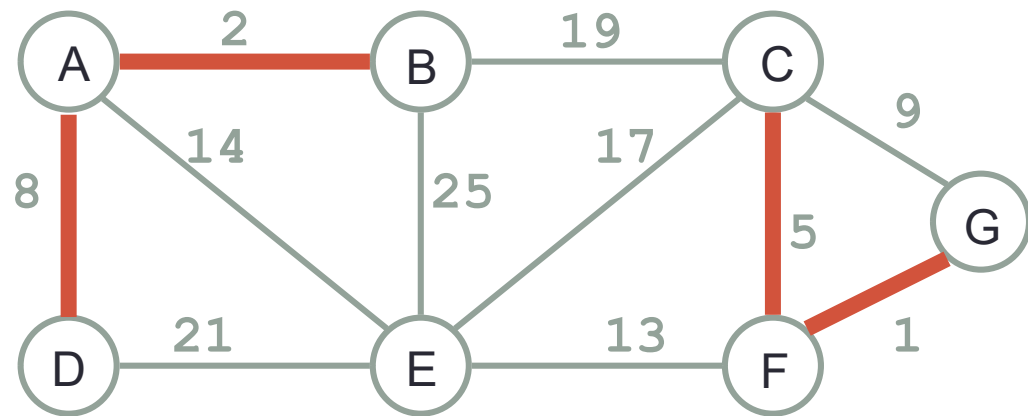
Algoritmo de Kruskal: Ejemplo



F \leftrightarrow G ... 1
A \leftrightarrow B ... 2
C \leftrightarrow F ... 5
→ A \leftrightarrow D ... 8
C \leftrightarrow G ... 9
E \leftrightarrow F ... 13
A \leftrightarrow E ... 14
C \leftrightarrow E ... 17
B \leftrightarrow C ... 19
D \leftrightarrow E ... 21
B \leftrightarrow E ... 25

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

Algoritmo de Kruskal: Ejemplo

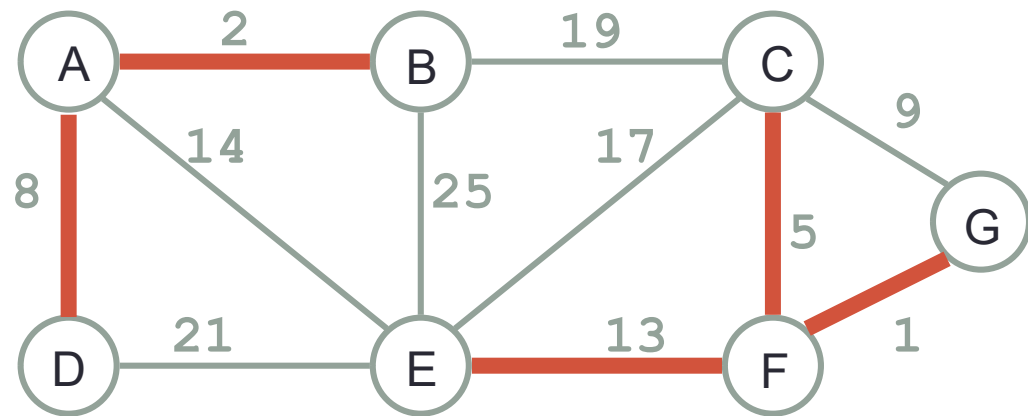


F \leftrightarrow G ... 1
A \leftrightarrow B ... 2
C \leftrightarrow F ... 5
A \leftrightarrow D ... 8
→ C \leftrightarrow G ... 9
E \leftrightarrow F ... 13
A \leftrightarrow E ... 14
C \leftrightarrow E ... 17
B \leftrightarrow C ... 19
D \leftrightarrow E ... 21
B \leftrightarrow E ... 25

No

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

Algoritmo de Kruskal: Ejemplo

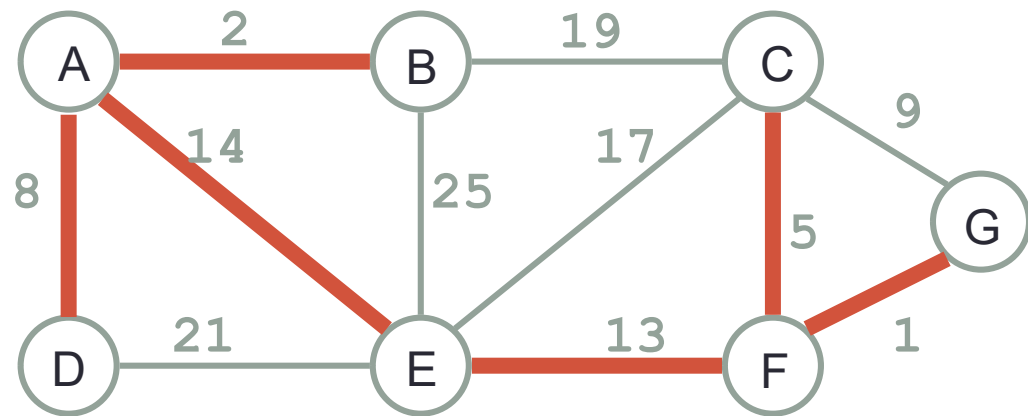


F ↔ G ... 1
A ↔ B ... 2
C ↔ F ... 5
A ↔ D ... 8
C ↔ G ... 9
→ E ↔ F ... 13
A ↔ E ... 14
C ↔ E ... 17
B ↔ C ... 19
D ↔ E ... 21
B ↔ E ... 25

No

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

Algoritmo de Kruskal: Ejemplo

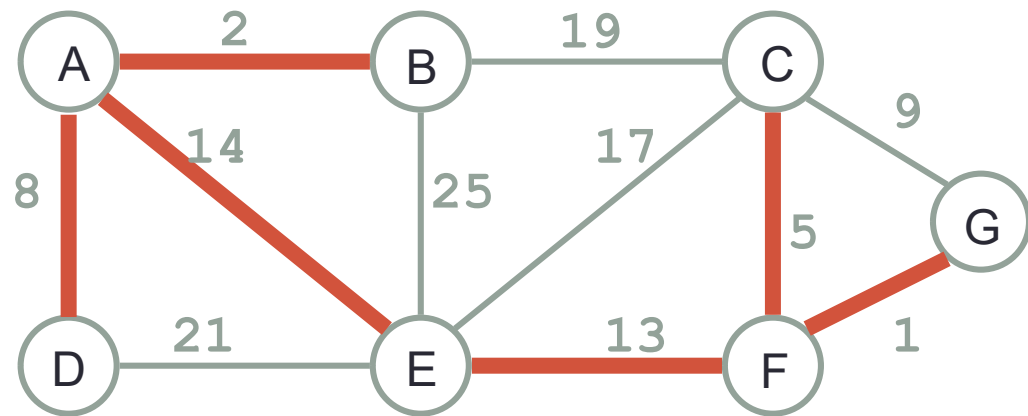


F \leftrightarrow G ... 1
A \leftrightarrow B ... 2
C \leftrightarrow F ... 5
A \leftrightarrow D ... 8
C \leftrightarrow G ... 9
E \leftrightarrow F ... 13
→ A \leftrightarrow E ... 14
C \leftrightarrow E ... 17
B \leftrightarrow C ... 19
D \leftrightarrow E ... 21
B \leftrightarrow E ... 25

No

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

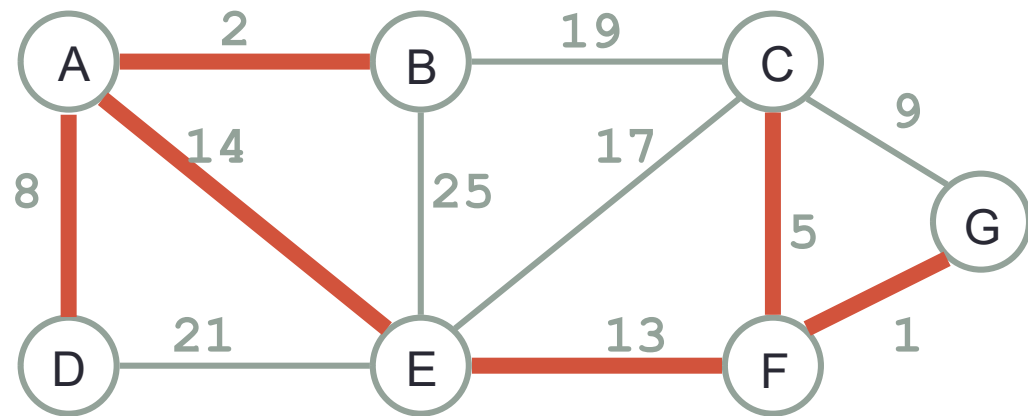
Algoritmo de Kruskal: Ejemplo



F ↔ G	... 1	
A ↔ B	... 2	
C ↔ F	... 5	
A ↔ D	... 8	
C ↔ G	... 9	No
E ↔ F	... 13	
A ↔ E	... 14	
→ C ↔ E	... 17	No
B ↔ C	... 19	
D ↔ E	... 21	
B ↔ E	... 25	

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

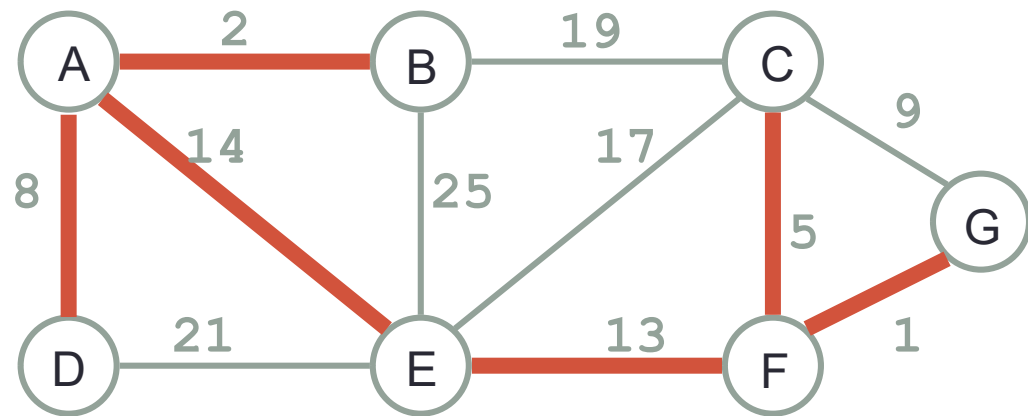
Algoritmo de Kruskal: Ejemplo



F ↔ G	... 1	
A ↔ B	... 2	
C ↔ F	... 5	
A ↔ D	... 8	
C ↔ G	... 9	No
E ↔ F	... 13	
A ↔ E	... 14	
C ↔ E	... 17	No
→ B ↔ C	... 19	No
D ↔ E	... 21	
B ↔ E	... 25	

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

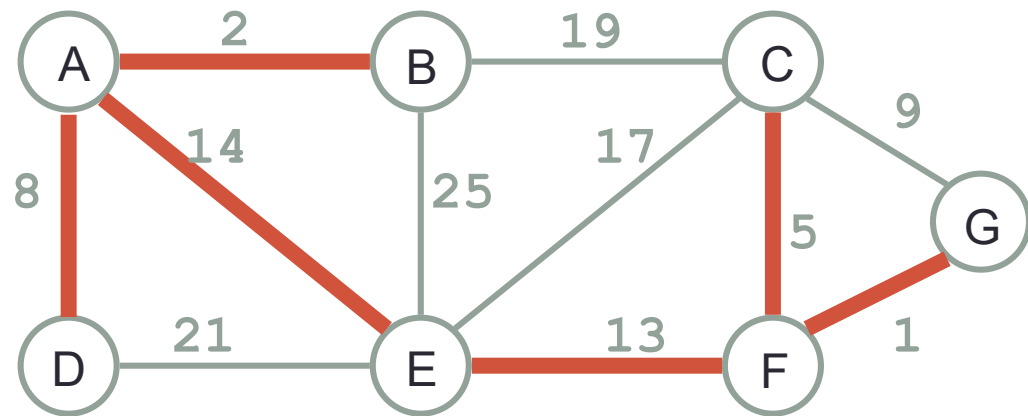
Algoritmo de Kruskal: Ejemplo



F ↔ G	... 1	
A ↔ B	... 2	
C ↔ F	... 5	
A ↔ D	... 8	
C ↔ G	... 9	No
E ↔ F	... 13	
A ↔ E	... 14	
C ↔ E	... 17	No
B ↔ C	... 19	No
→ D ↔ E	... 21	No
B ↔ E	... 25	

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

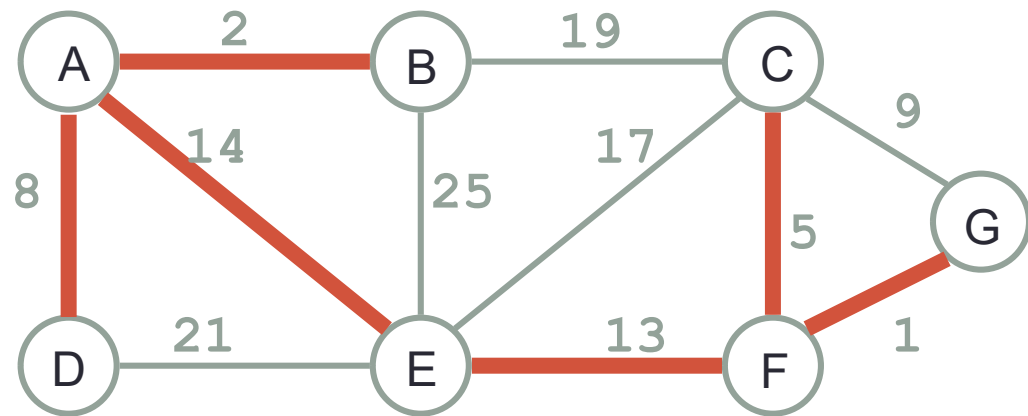
Algoritmo de Kruskal: Ejemplo



F ↔ G	... 1	
A ↔ B	... 2	
C ↔ F	... 5	
A ↔ D	... 8	
C ↔ G	... 9	No
E ↔ F	... 13	
A ↔ E	... 14	
C ↔ E	... 17	No
B ↔ C	... 19	No
D ↔ E	... 21	No
→ B ↔ E	... 25	No

Bucle voraz: Recorremos la lista ordenada de aristas añadiendo las que no forman ciclos

Algoritmo de Kruskal: Ejemplo



Descripción, ejemplo, e implementación en C++, Java y Python

<https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/>

Árbol de Expansión Mínima: Algoritmo de Prim

Given $G = (V, E)$

$T = (V_T, \{\}) = (\{v_0\}, \{\})$

while V_T is not equal to V

Select $e^* = (u^*, v^*) = \min \text{Weight}(u, v)$ where

(1) u is in V_T

(2) v is in $V - V_T$

$V_T = V_T \cup \{v^*\}$

$E_T = E_T \cup \{e^*\}$

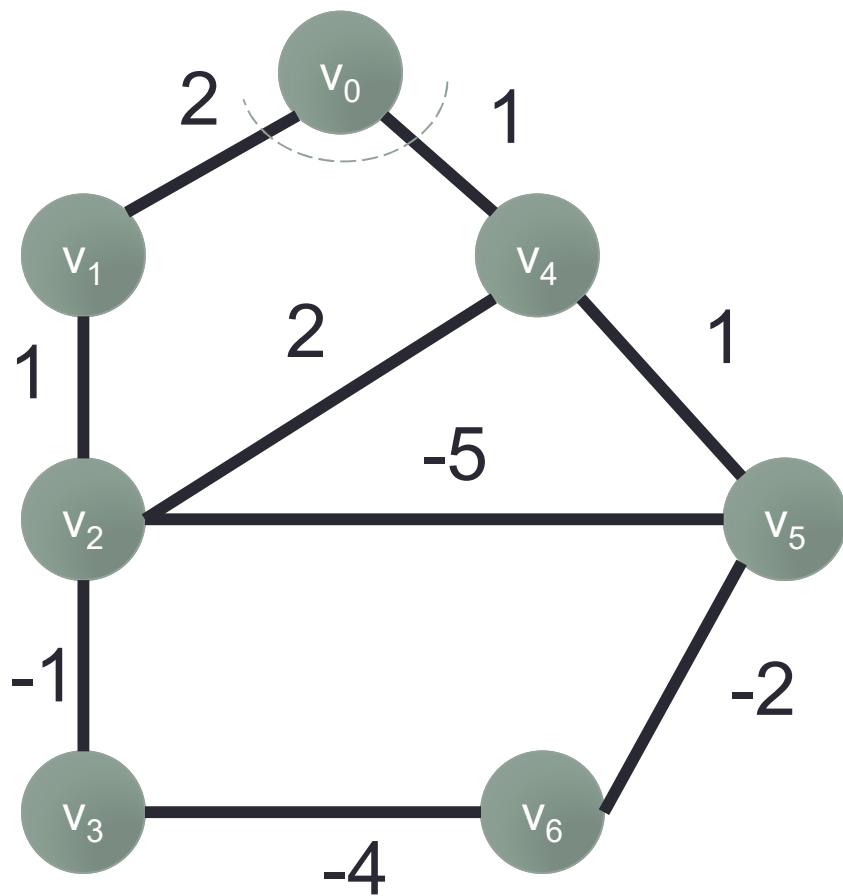
end loop

Son estrategias diferentes:

- *Kruskal recorre las aristas*
- *Prims recorre los vértices*

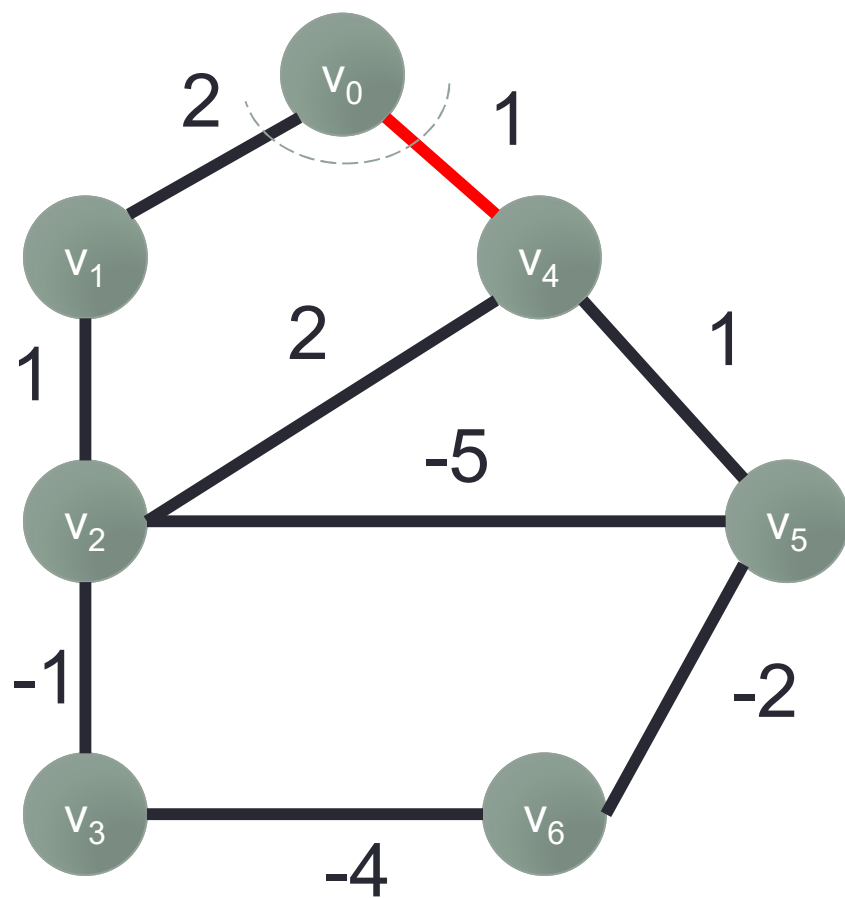
Prim

Árbol de Expansión Mínima

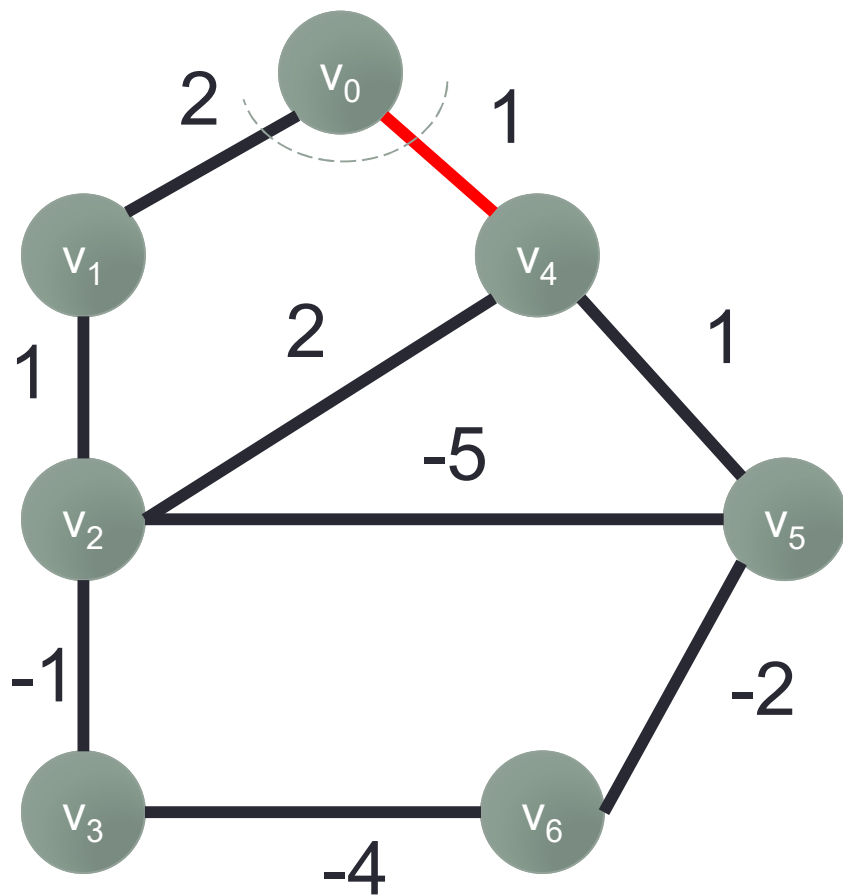


Prim

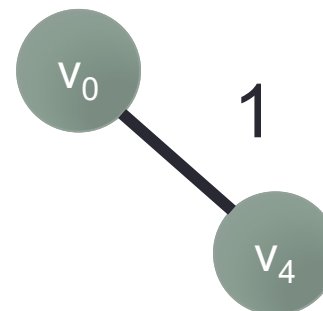
Árbol de Expansión Mínima



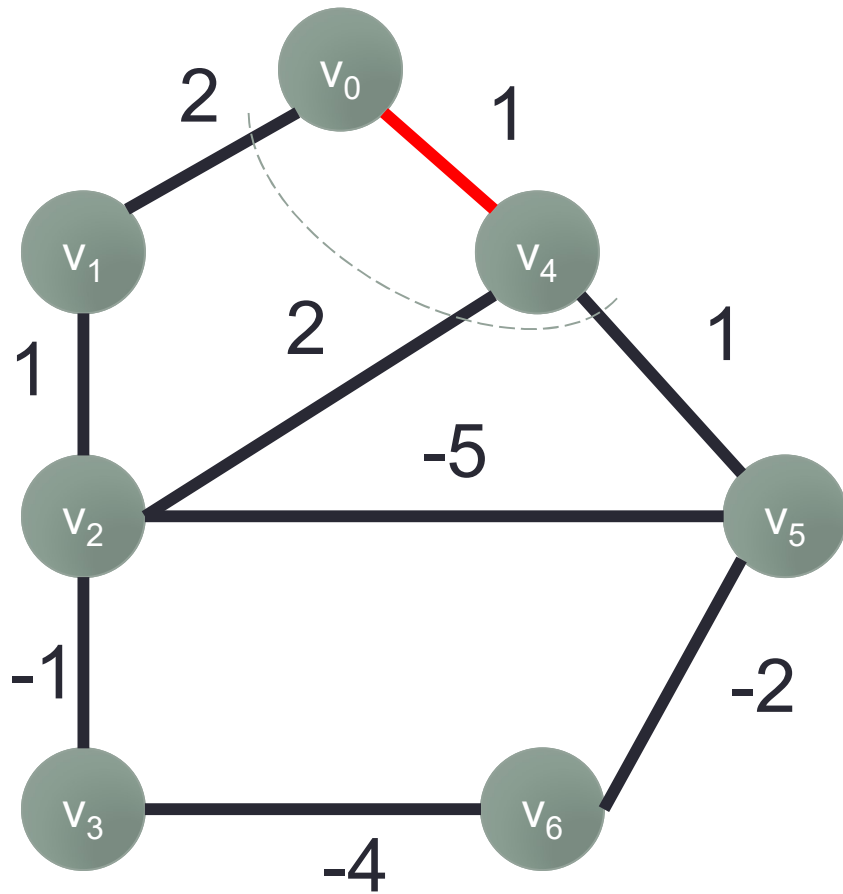
Prim



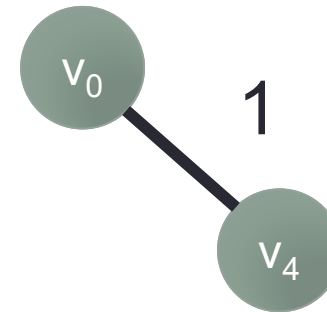
Árbol de Expansión Mínima



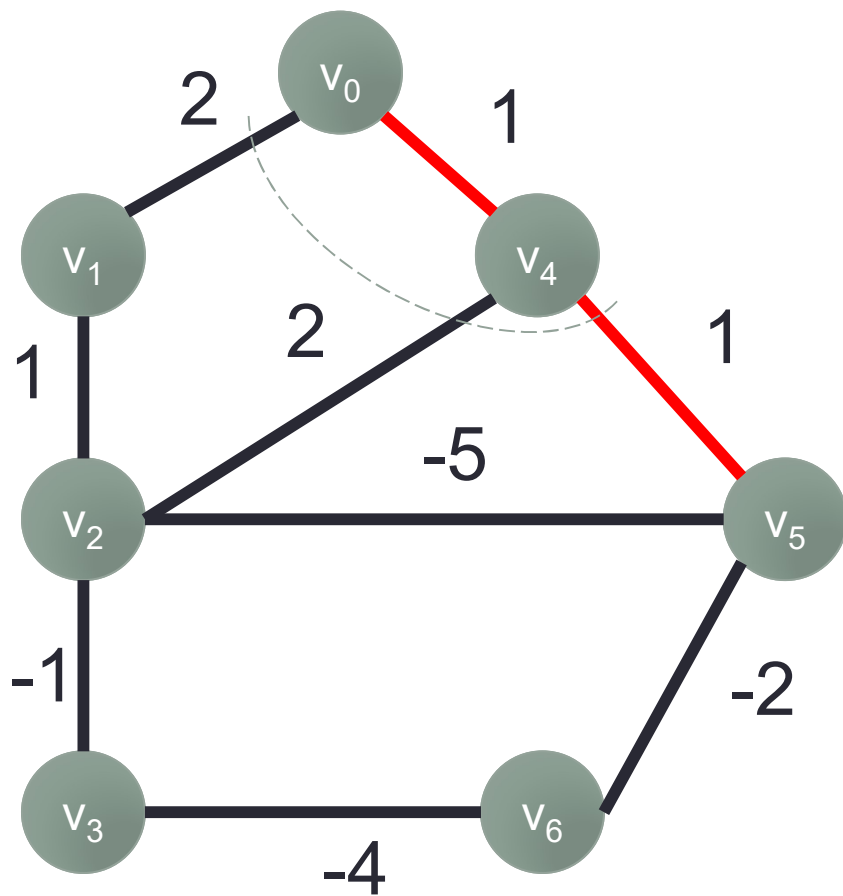
Prim



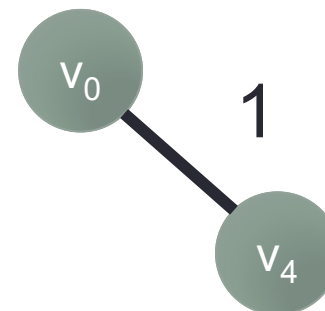
Árbol de Expansión Mínima



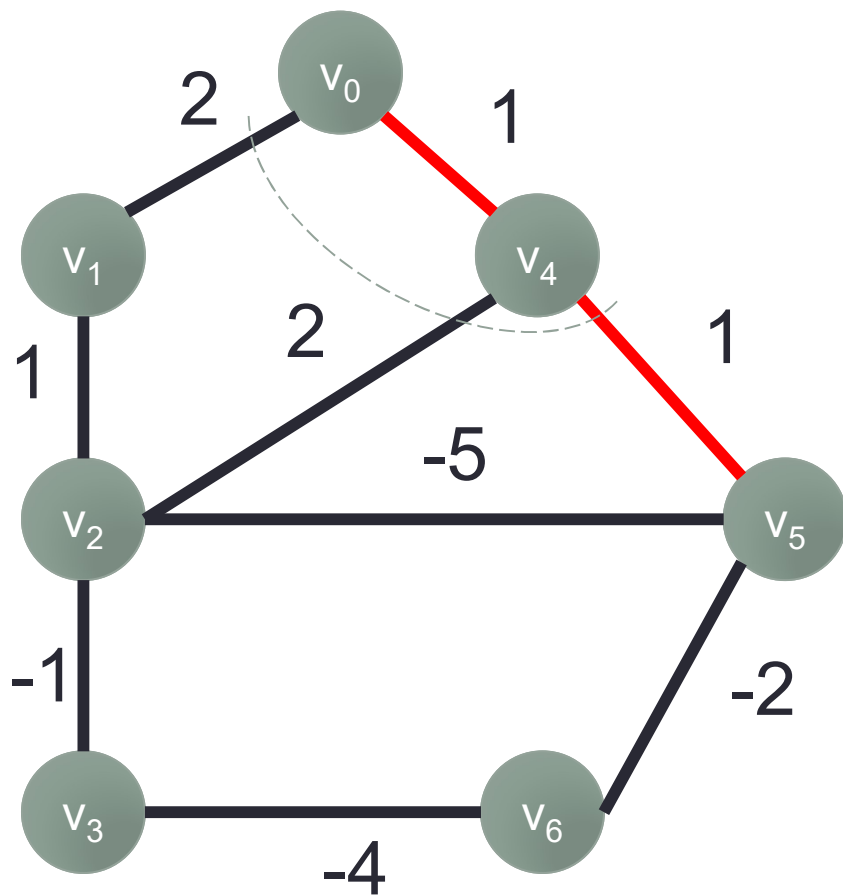
Prim



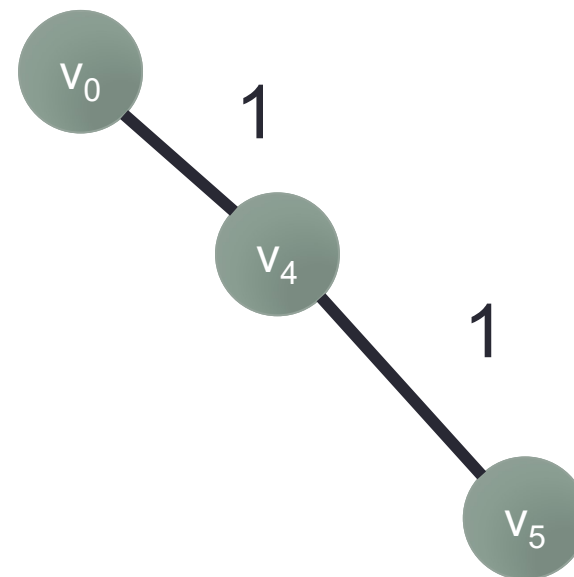
Árbol de Expansión Mínima



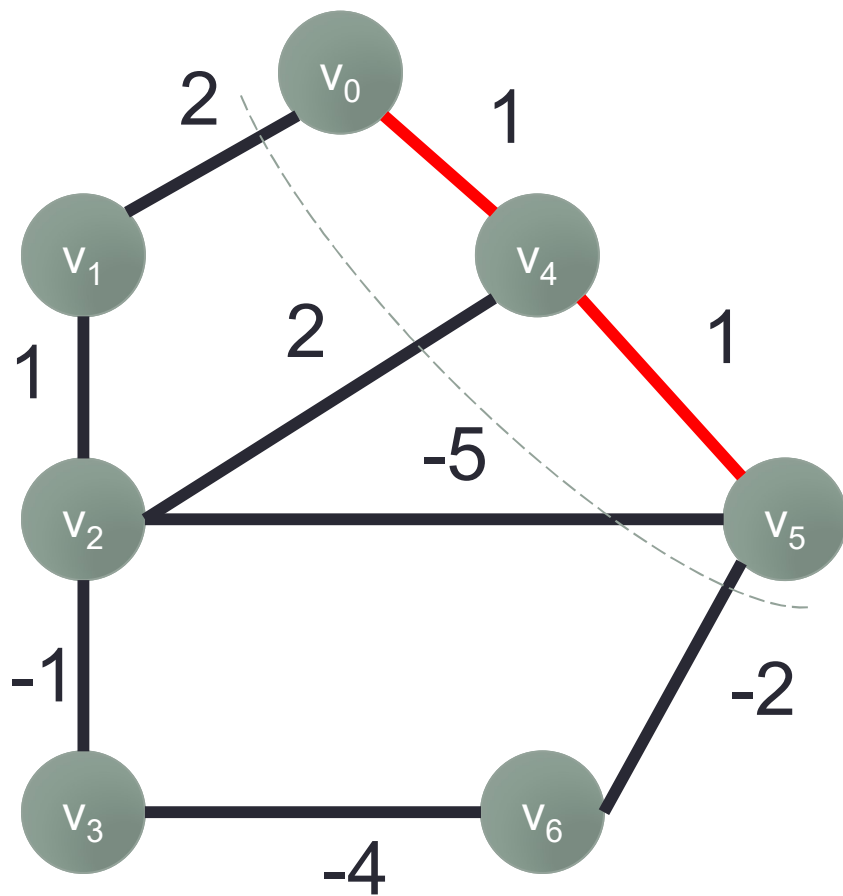
Prim



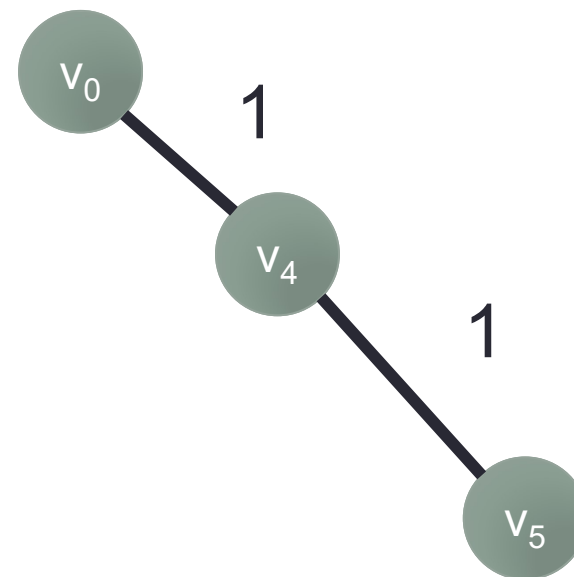
Árbol de Expansión Mínima



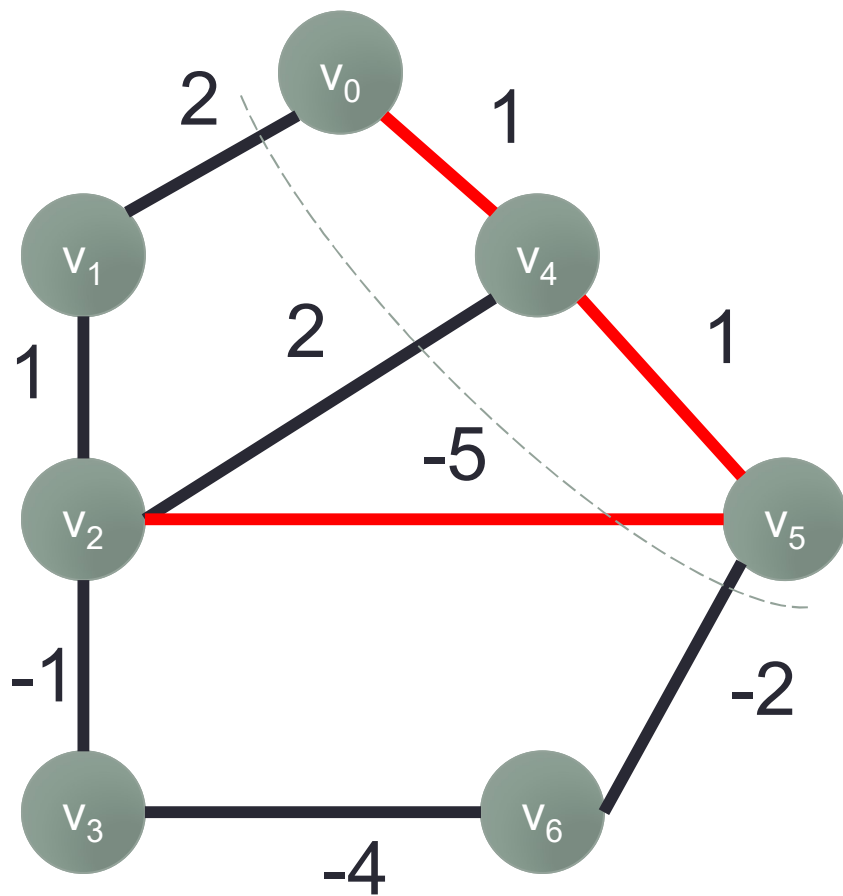
Prim



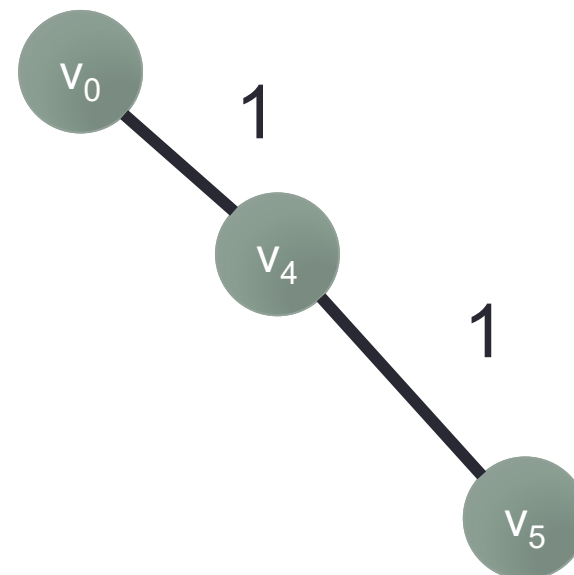
Árbol de Expansión Mínima



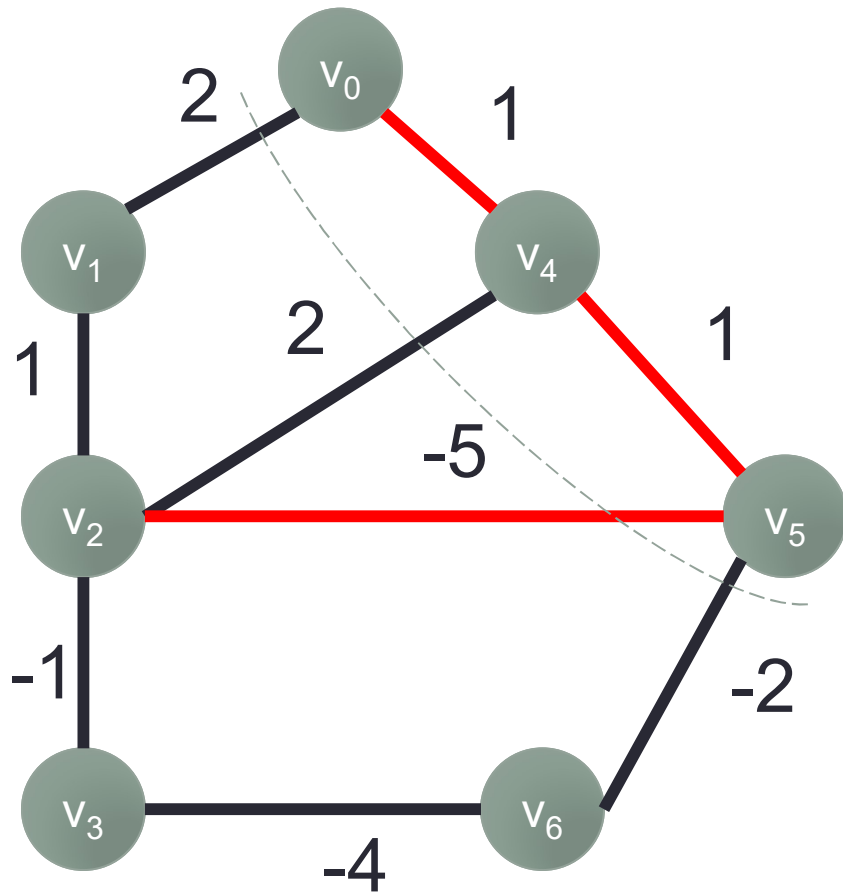
Prim



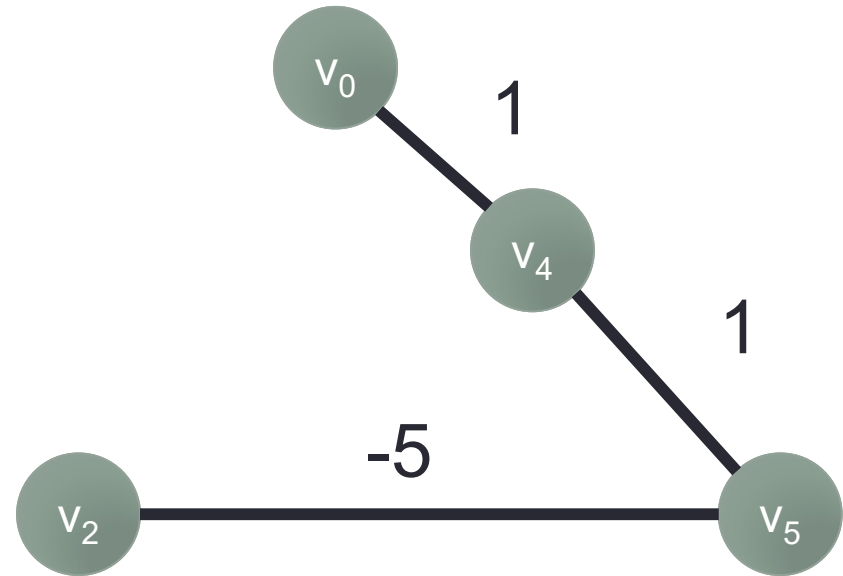
Árbol de Expansión Mínima



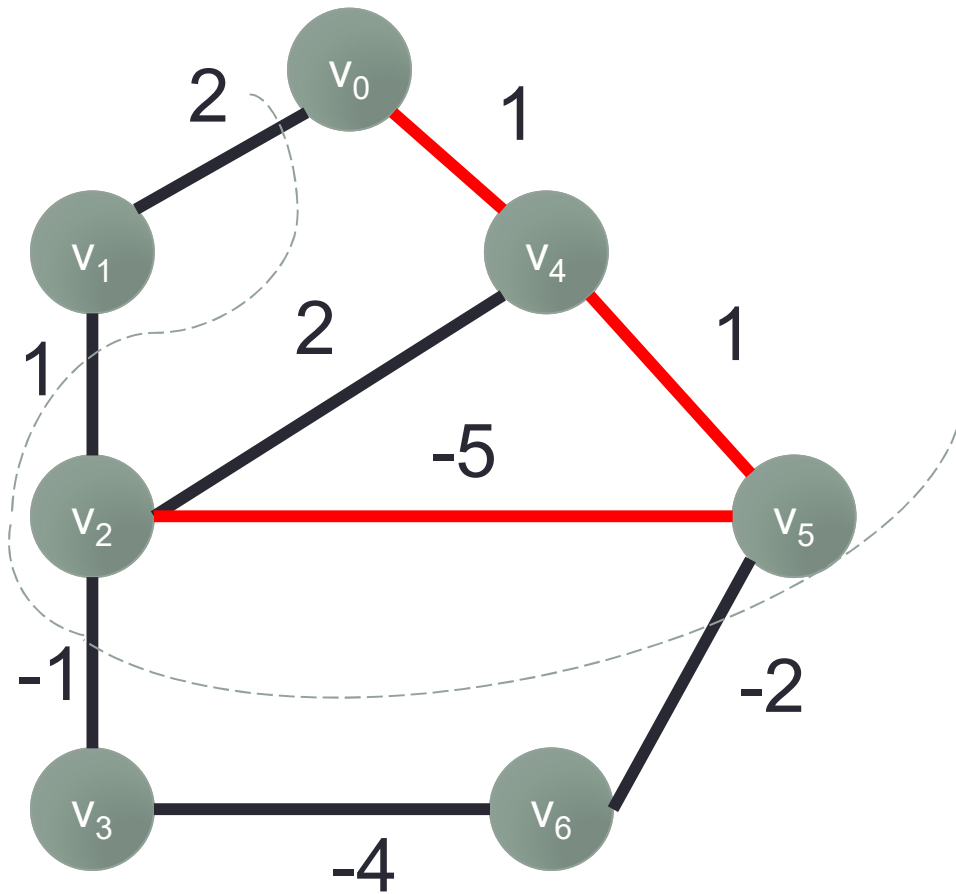
Prim



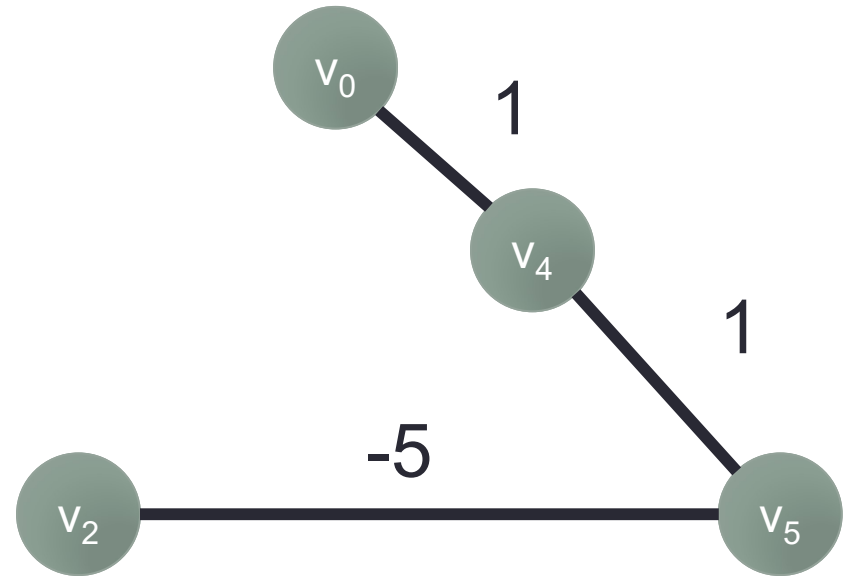
Árbol de Expansión Mínima



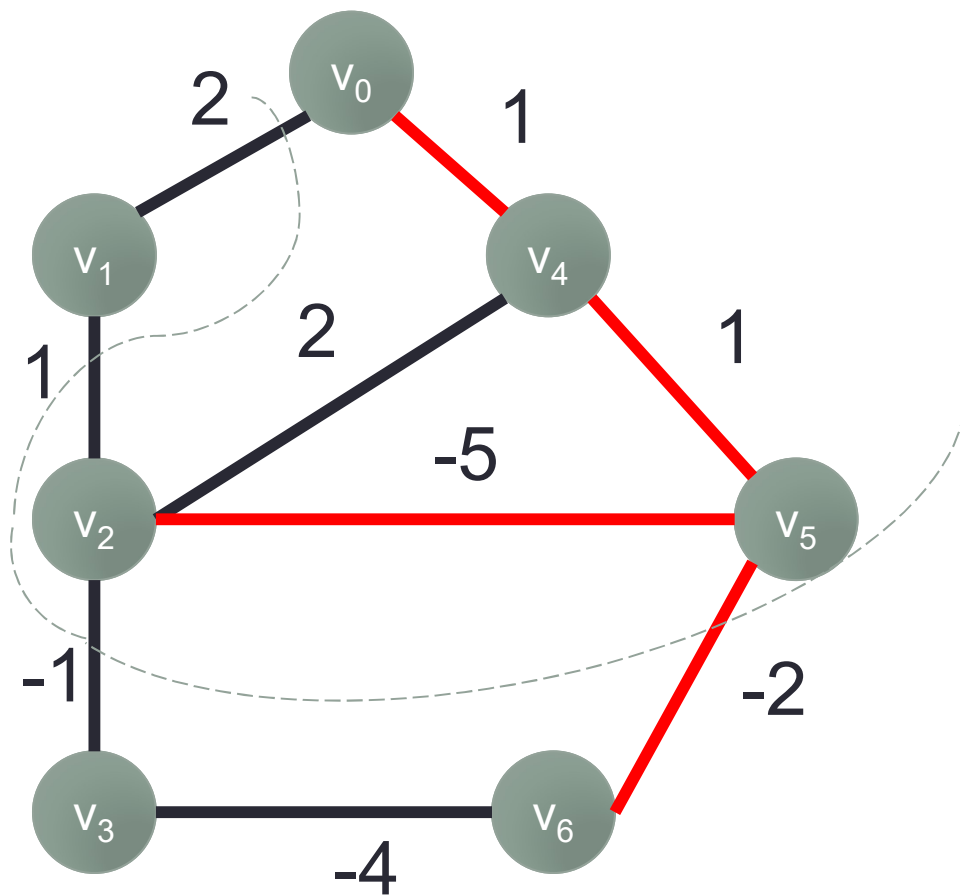
Prim



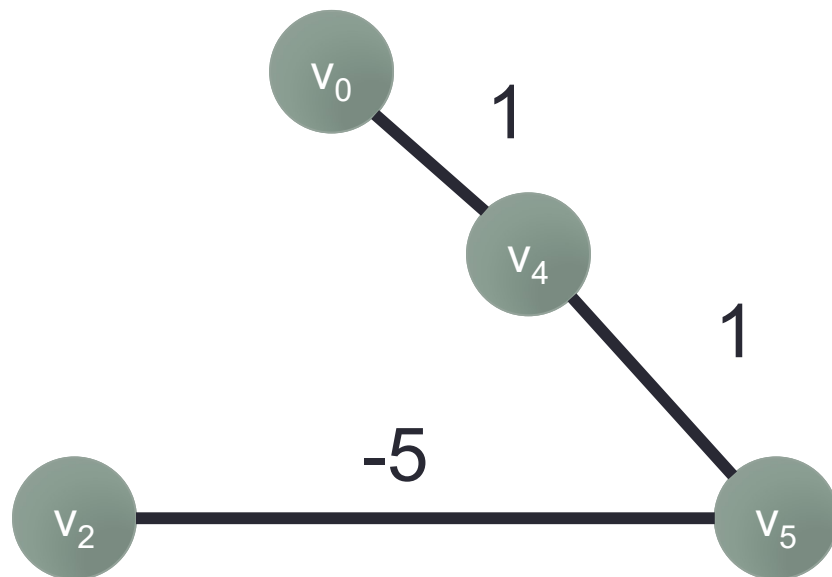
Árbol de Expansión Mínima



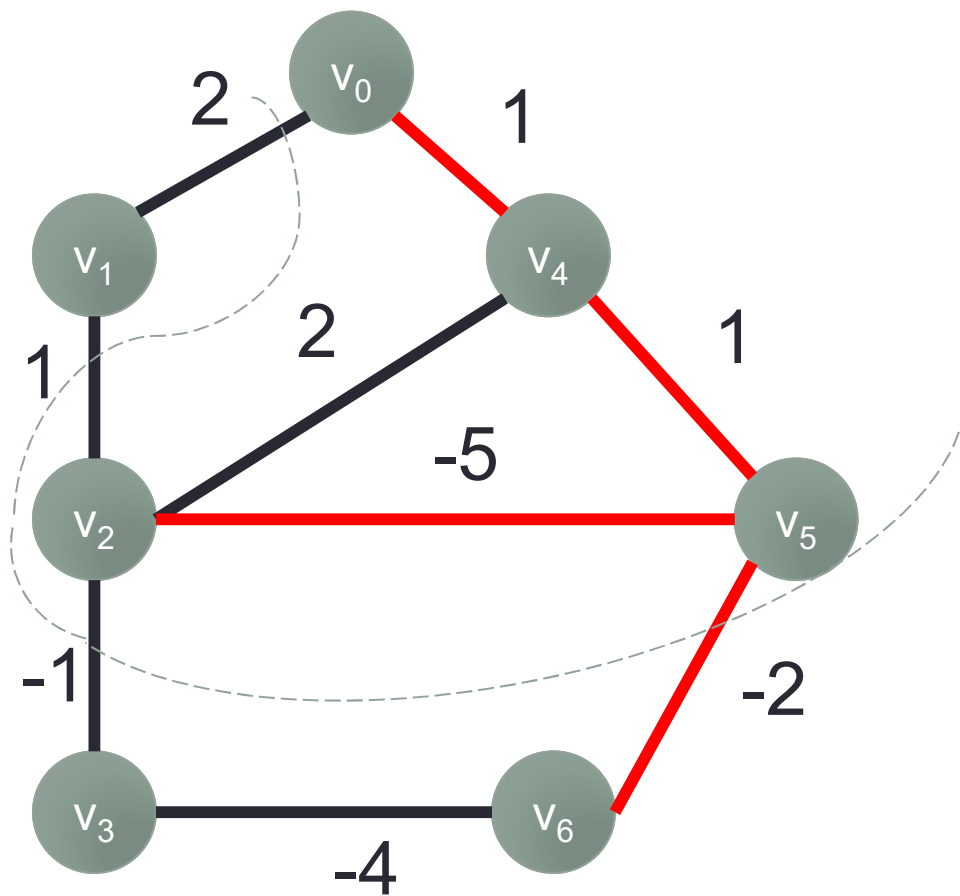
Prim



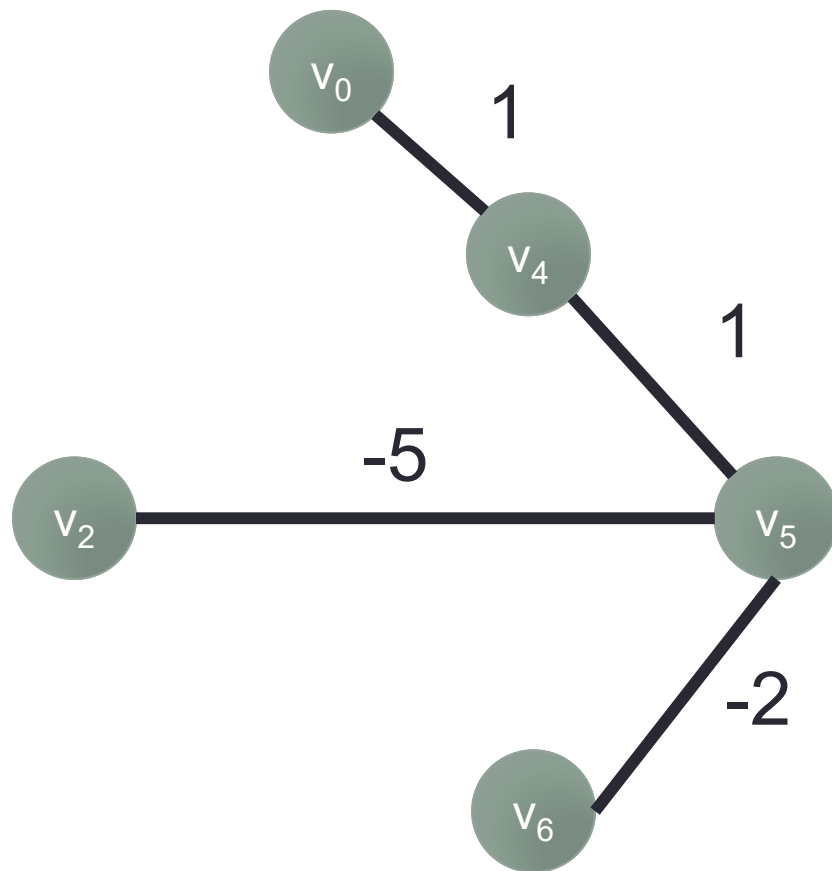
Árbol de Expansión Mínima



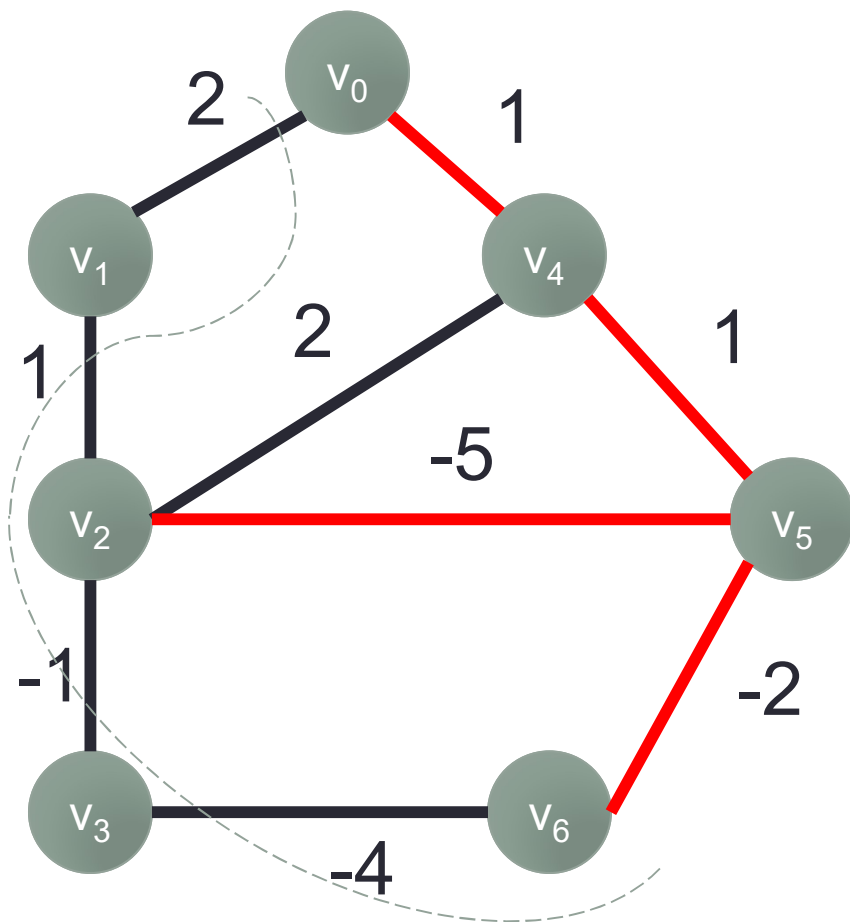
Prim



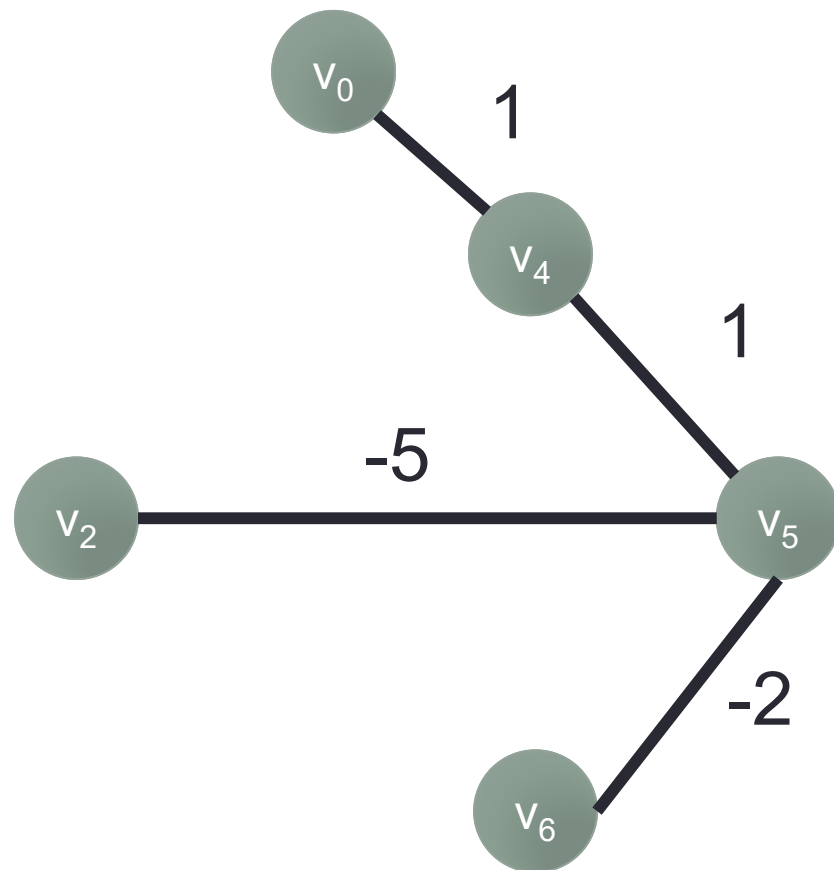
Árbol de Expansión Mínima



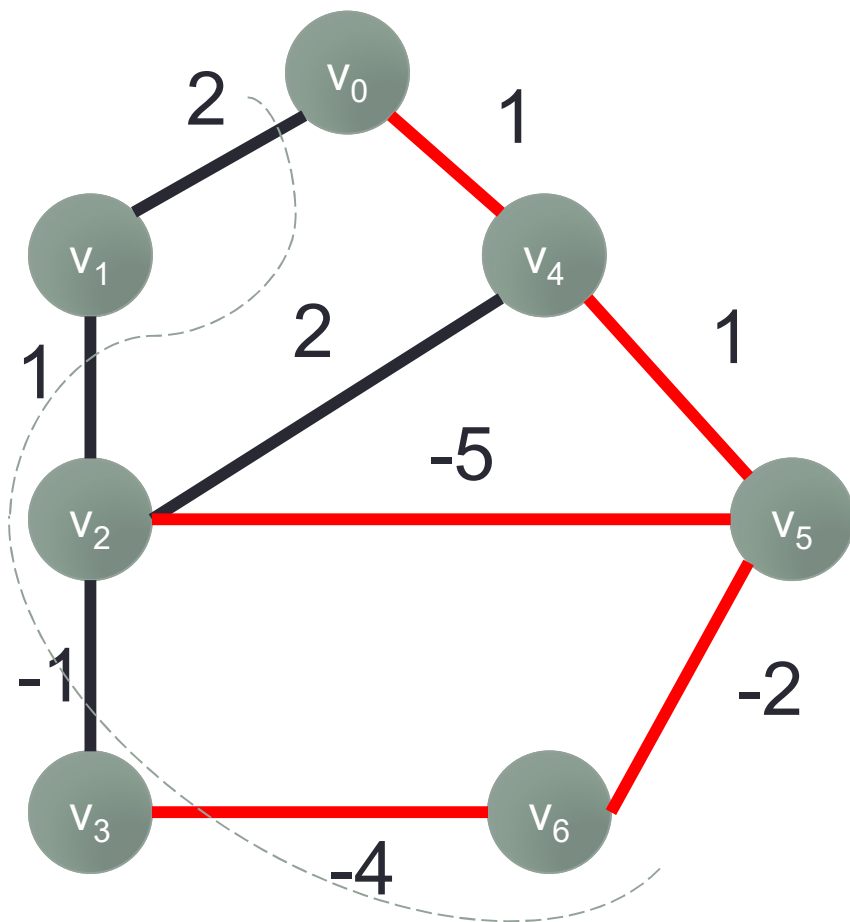
Prim



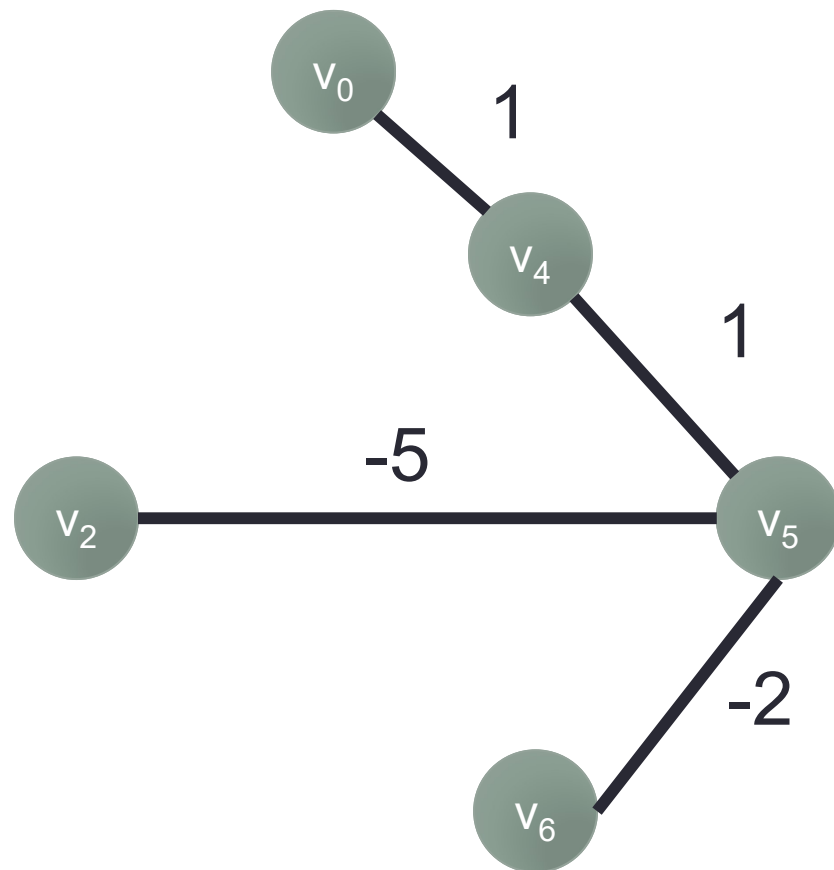
Árbol de Expansión Mínima



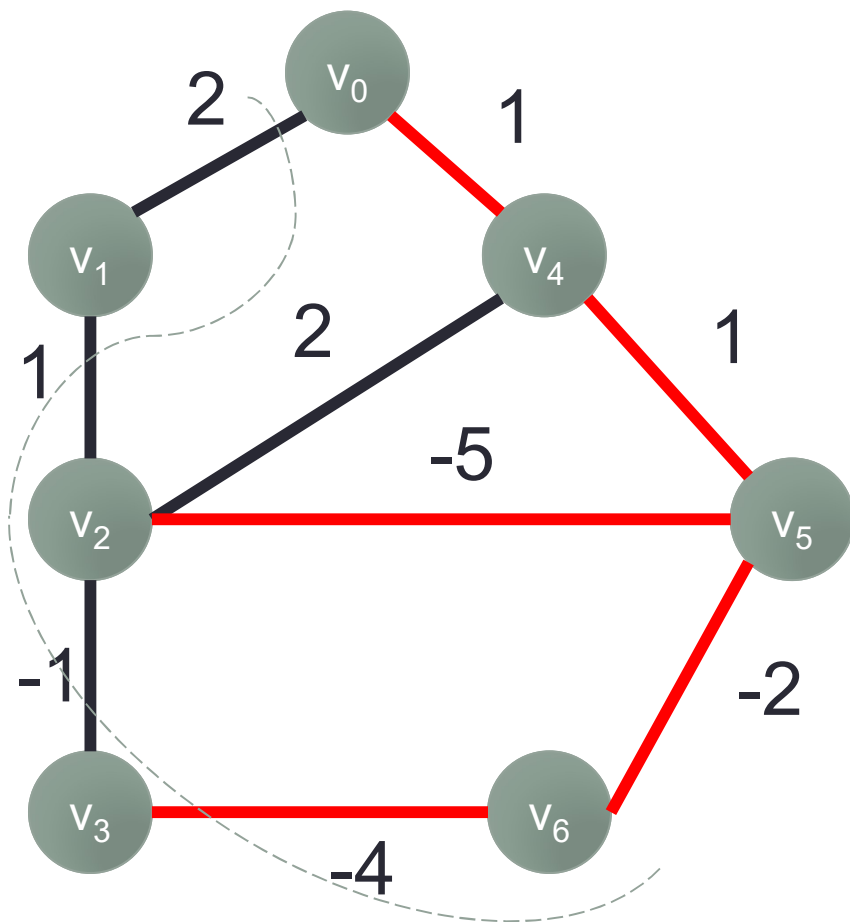
Prim



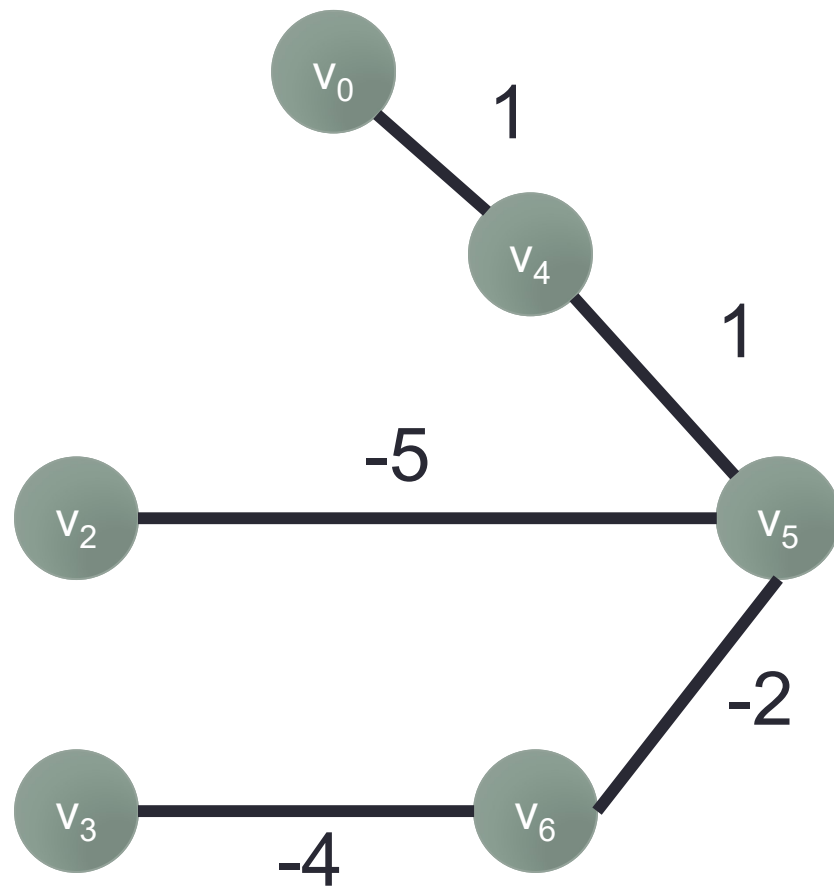
Árbol de Expansión Mínima



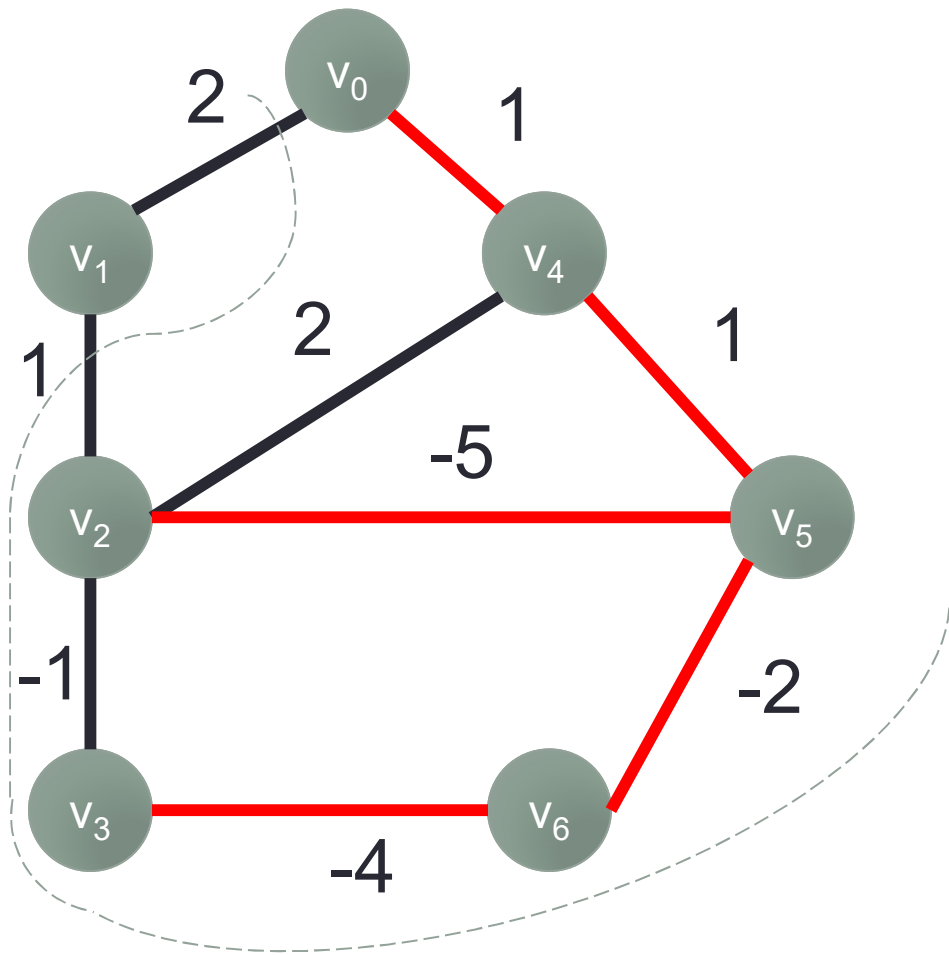
Prim



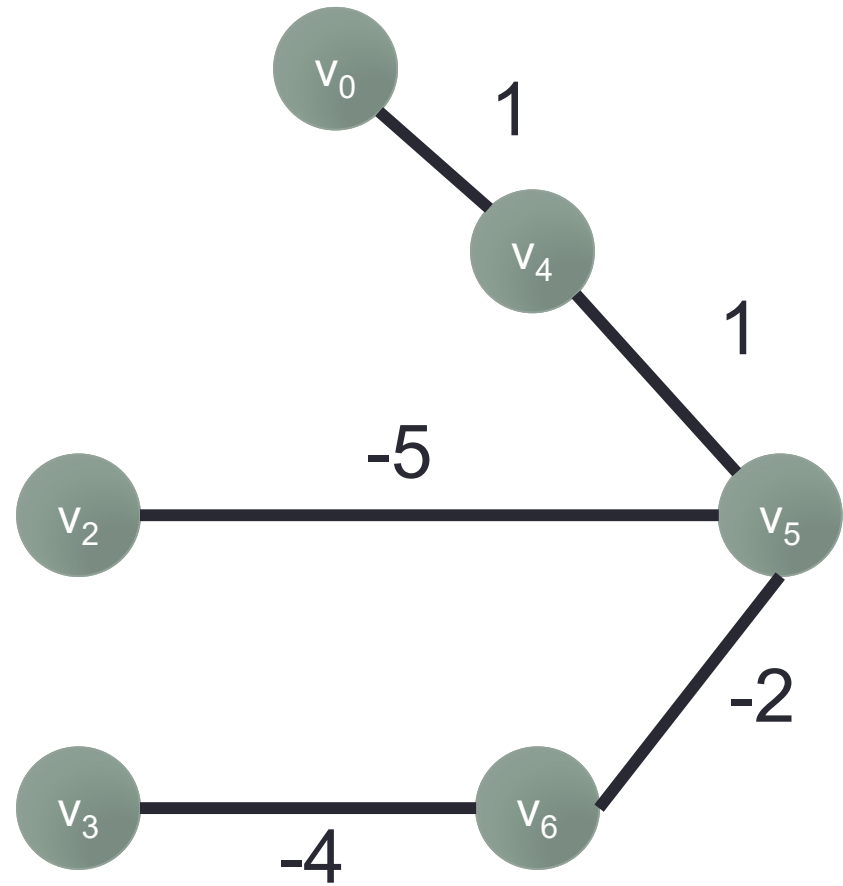
Árbol de Expansión Mínima



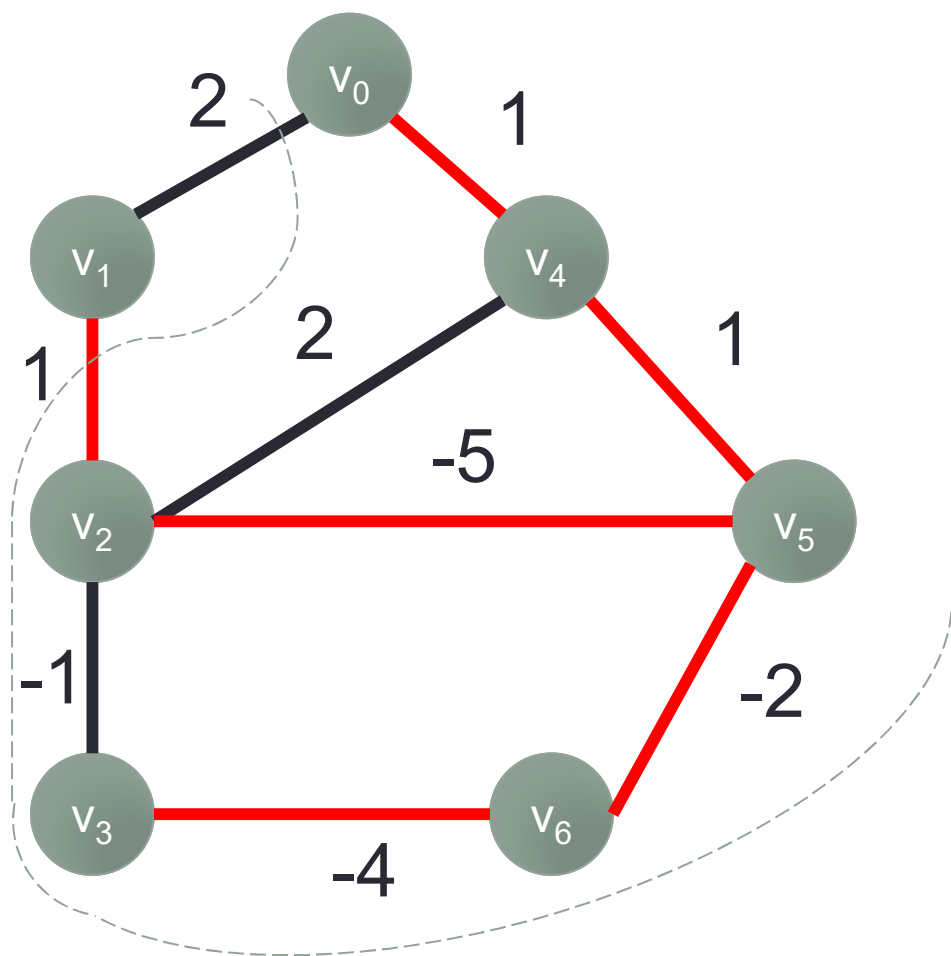
Prim



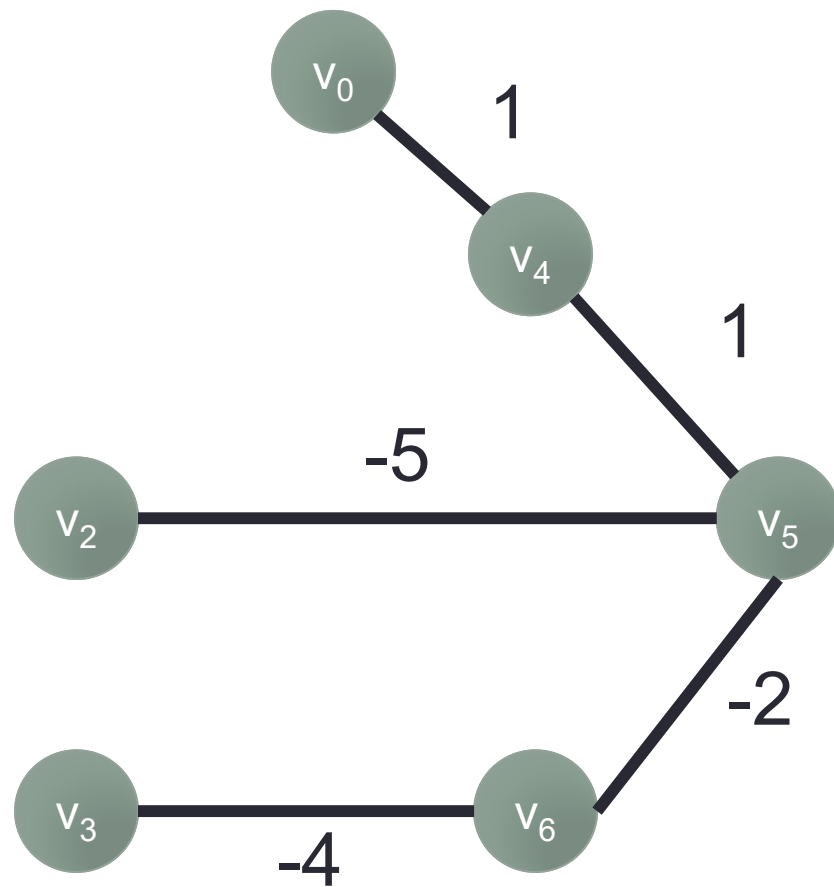
Árbol de Expansión Mínima



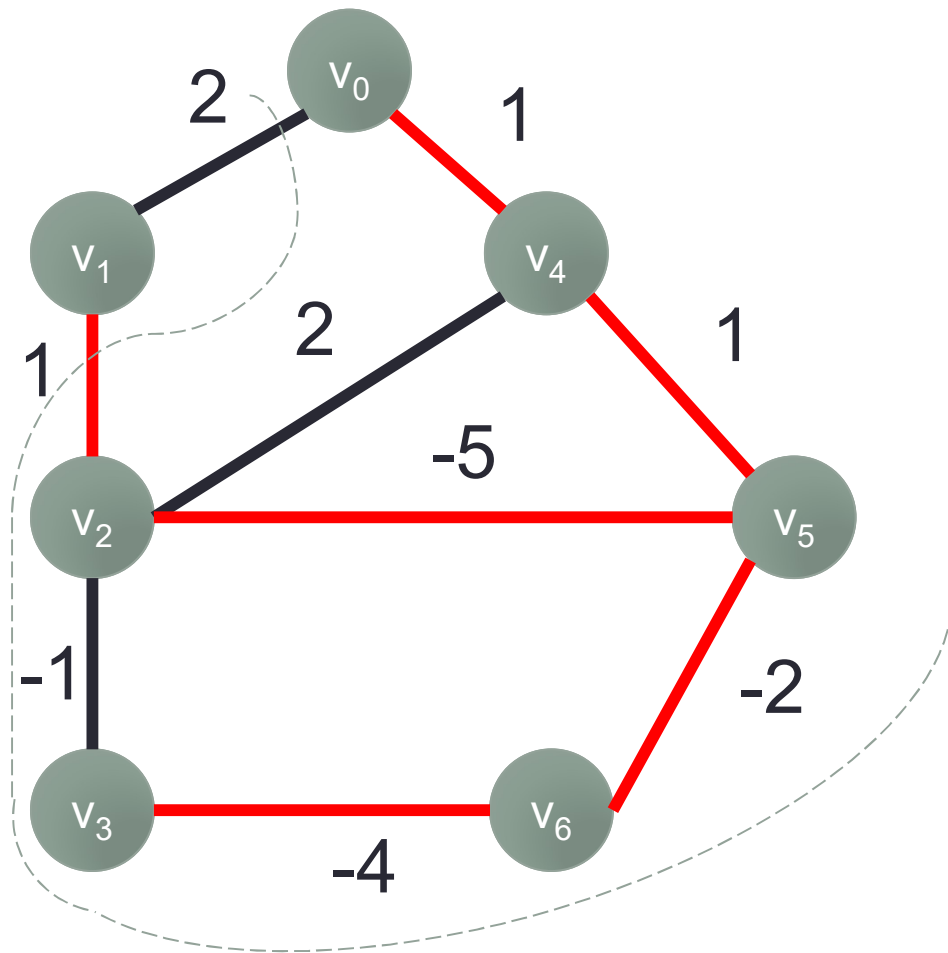
Prim



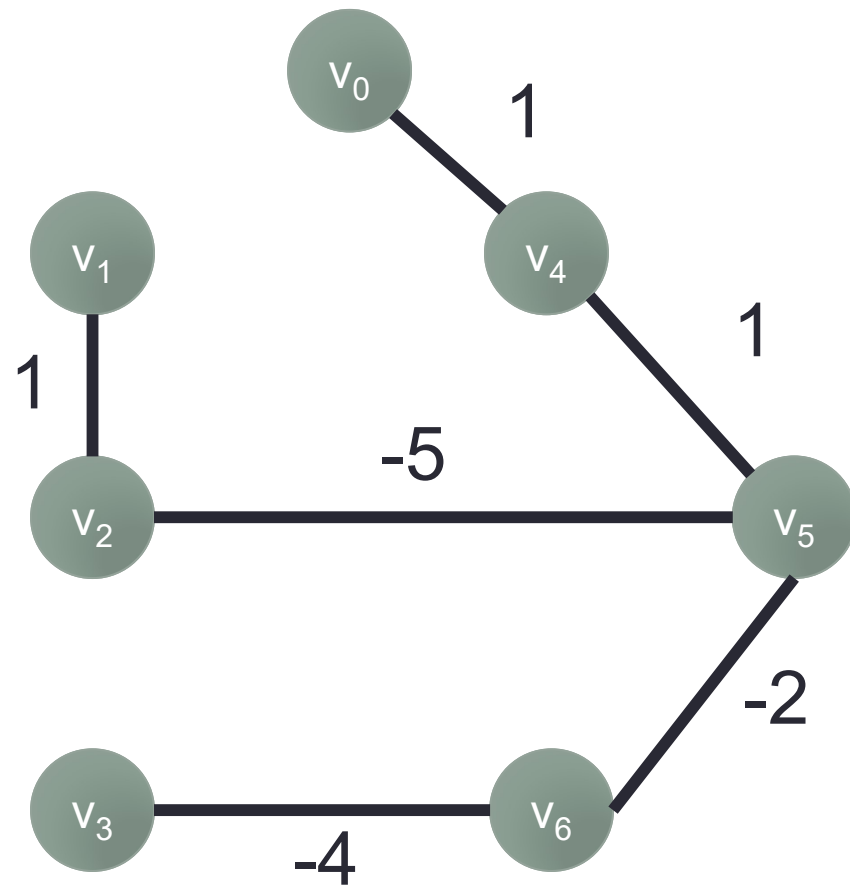
Árbol de Expansión Mínima



Prim



Árbol de Expansión Mínima



Árbol de Expansión Mínima

Estrategia Voraz: Fortalezas

- Simple y fácil de implementar
- Cuando funciona es realmente muy eficiente

Estrategia Voraz: Debilidades

- No siempre funciona (lo vimos con el problema de las monedas)
- En algunos casos tomar decisiones a corto plazo puede llevarnos a:
 - soluciones muy costosas a largo plazo
 - no encontrar solución
- Se basa en la experiencia (*heurística*) y no es fácil demostrar matemáticamente que calcula la solución óptima

Uso de Greedy

- En problemas complejos la estrategia Greedy se utiliza para encontrar rápidamente una posible solución
 - ... que utilizan otros algoritmos para mejorar la solución



... como veremos más adelante en esta asignatura!