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, recorremos la lista de monedas y elegimos siempre la de mayor valor que nos acerca a nuestro objetivo!

Monedas

Posición → *Valor*

Posición		Valor
1	→	5
2	→	2
3	→	1
4	→	10
5	→	10
6	→	2

Monedas Ordenadas por Valor

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

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

Monedas

Posición — *Valor*

1 → 2
2 → 8
3 → 2
4 → 10
5 → 8
6 → 2

Monedas Ordenadas por Valor

Posición — *Valor*

4 ← 10
5 ← 8
2 ← 8
6 ← 2
1 ← 2
3 ← 2

Valor del cambio a devolver: 16

Solución óptima: 2 5

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

Posición $\left\{ \begin{array}{l} 1 \rightarrow 5 \\ 2 \rightarrow 7 \\ 3 \rightarrow 1 \\ 4 \rightarrow 10 \\ 5 \rightarrow 7 \end{array} \right\}$ *Valor*

Monedas Ordenadas por Valor

Posición $\left\{ \begin{array}{l} 4 \leftarrow 10 \\ 5 \leftarrow 7 \\ 2 \leftarrow 7 \\ 1 \leftarrow 5 \\ 3 \leftarrow 1 \end{array} \right\}$ *Valor*

Valor del cambio a devolver: 14

Solución óptima: 2 5

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.

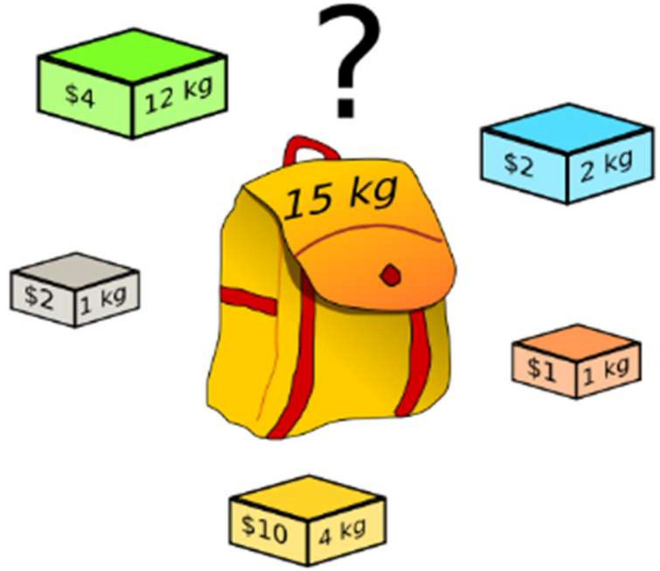
De esta forma, si en un paso no se eligiese la mejor opción habría que utilizar al menos dos monedas

Problema de la mochila 0/1

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

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

Elección binaria



Problema de la mochila

¿ 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

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

$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

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

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

Problema de la mochila

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

Evaluando las 4 alternativas básicas

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

¿ Hay alguna estrategia greedy mejor ?

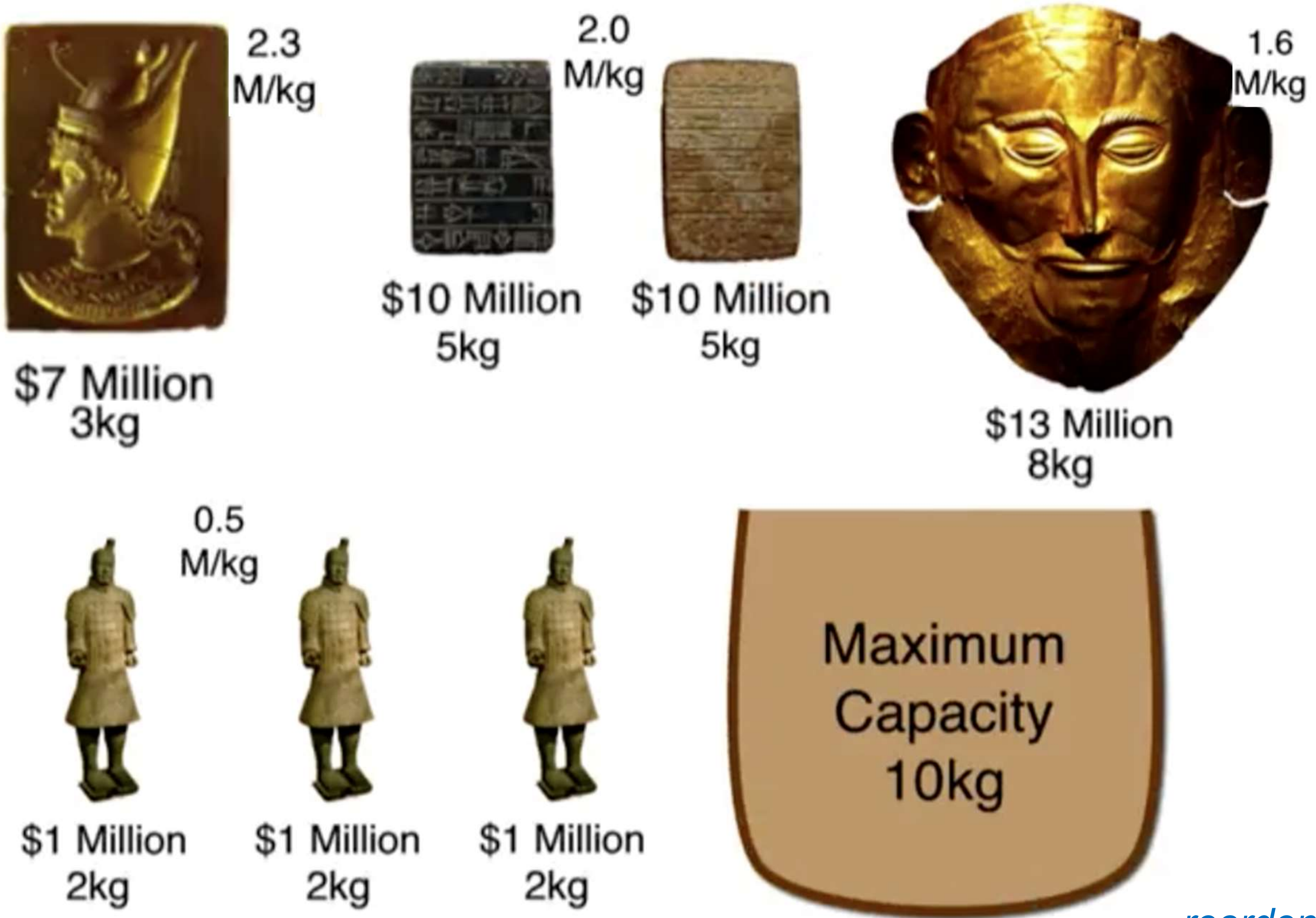
Problema de la mochila

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

Problema de la mochila

2.0
M/kg

$10 Million
5kg

1.6
M/kg

$13 Million
8kg

¿ Era la mejor elección ?

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

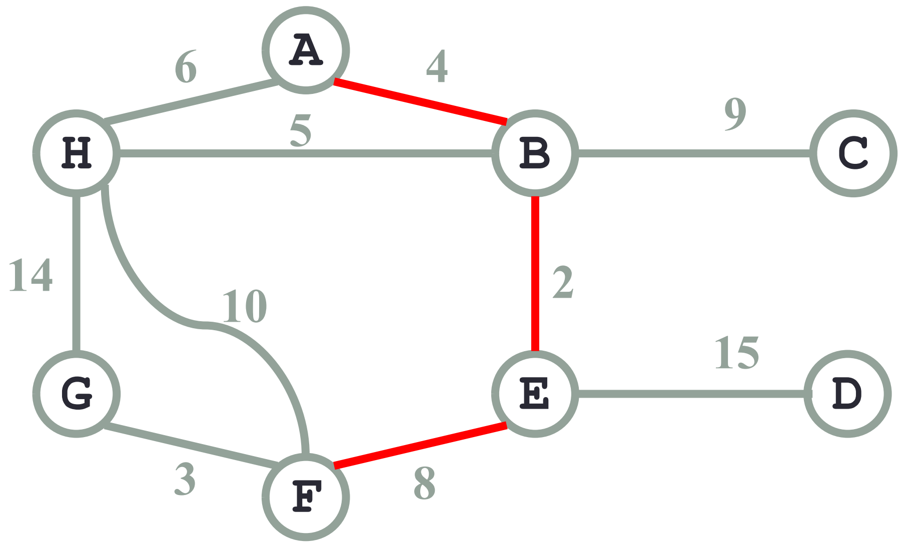
0.5
M/kg

$1 Million
2kg

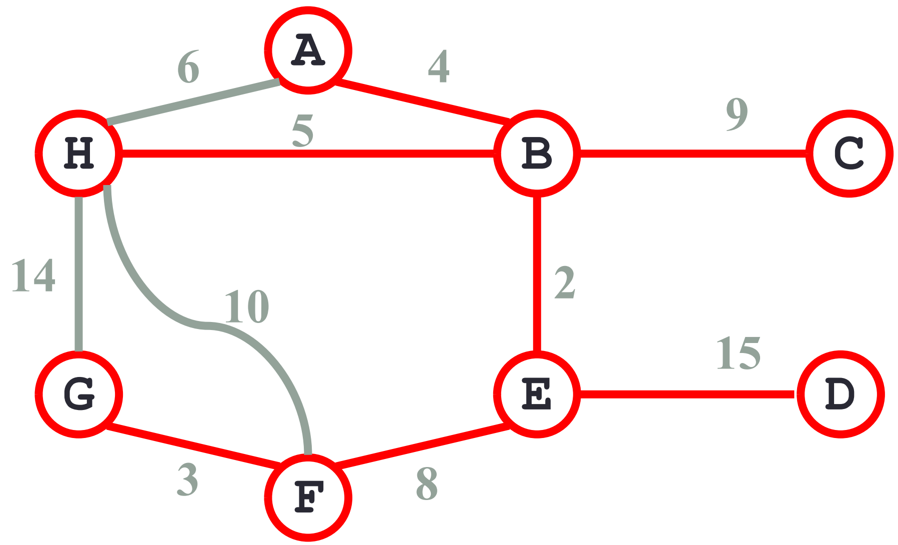
$1 Million
2kg

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 expansion 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 Dijskstra 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 vi. 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 $dt(vi) < Dvi \rightarrow Dvi = dt(vi)$

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.

Bucle Voraz

Una vez terminado al algoritmo, D estará completamente lleno.

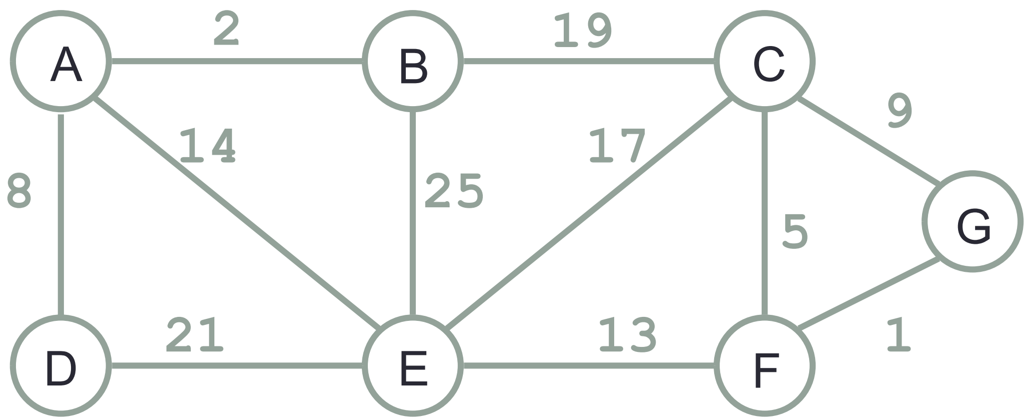
Algoritmo de Kruskal

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 <u>no se forme un ciclo</u>
4) Repetir hasta que el número de aristas sea n-1

Bucle Voraz

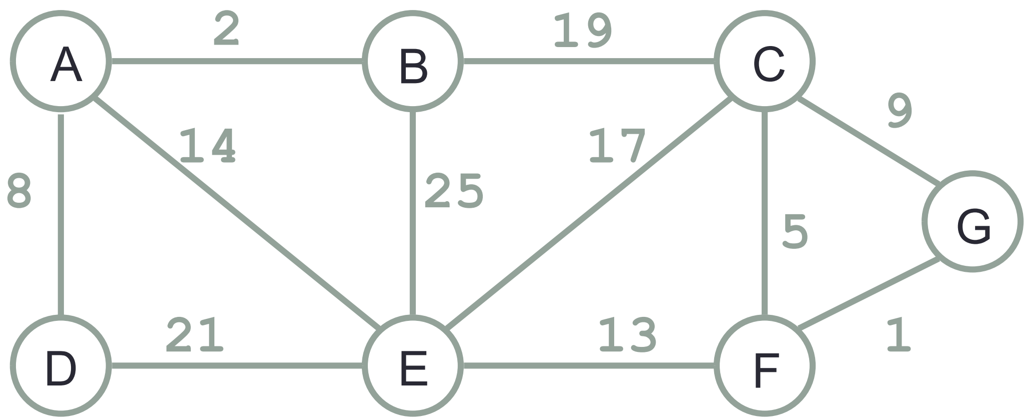
Algoritmo de Kruskal: Ejemplo



Aristas con su peso

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

Algoritmo de Kruskal: Ejemplo



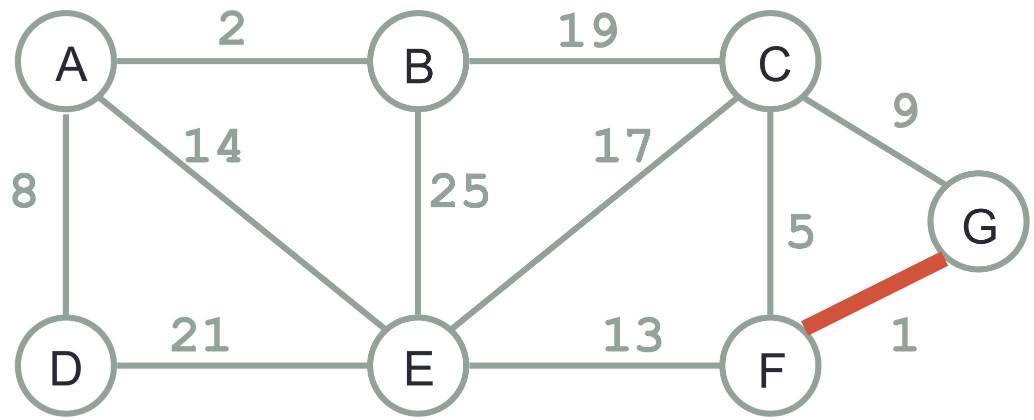
A ⇔ B ... 2
A ⇔ E ... 14
A ⇔ D ... 8
B ⇔ C ... 19
B ⇔ E ... 25
C ⇔ E ... 17
C ⇔ F ... 5
C ⇔ G ... 9
D ⇔ E ... 21
E ⇔ F ... 13
F ⇔ G ... 1

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

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

Algoritmo de Kruskal: Ejemplo

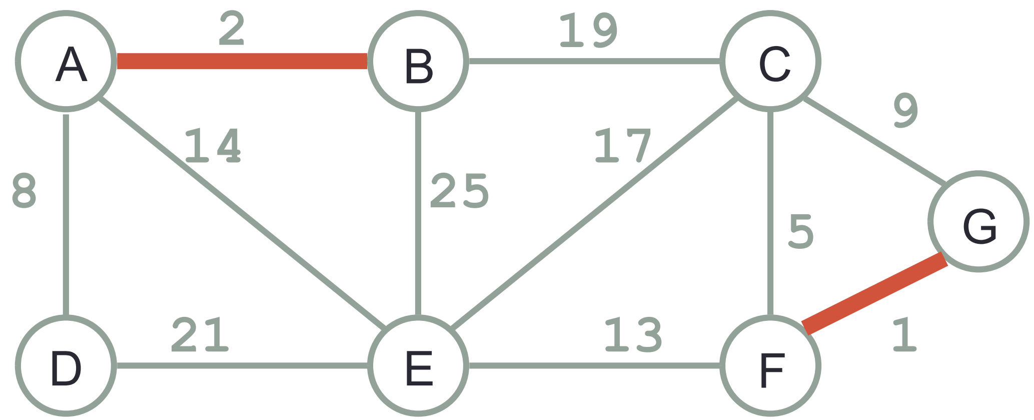


Aristas ordenadas

→ 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

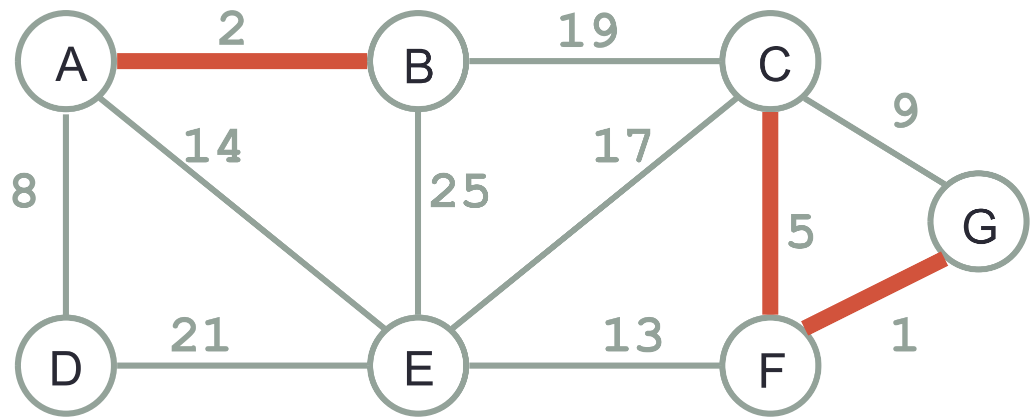
Algoritmo de Kruskal: Ejemplo



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

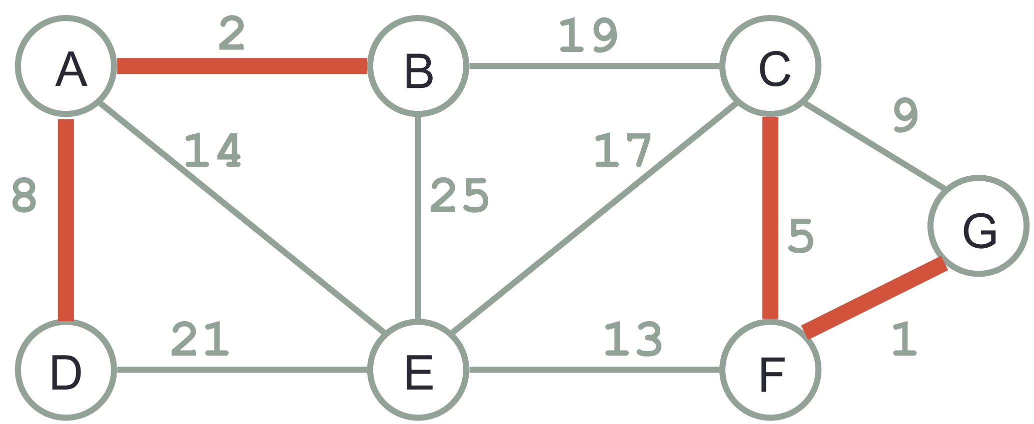
Algoritmo de Kruskal: Ejemplo



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

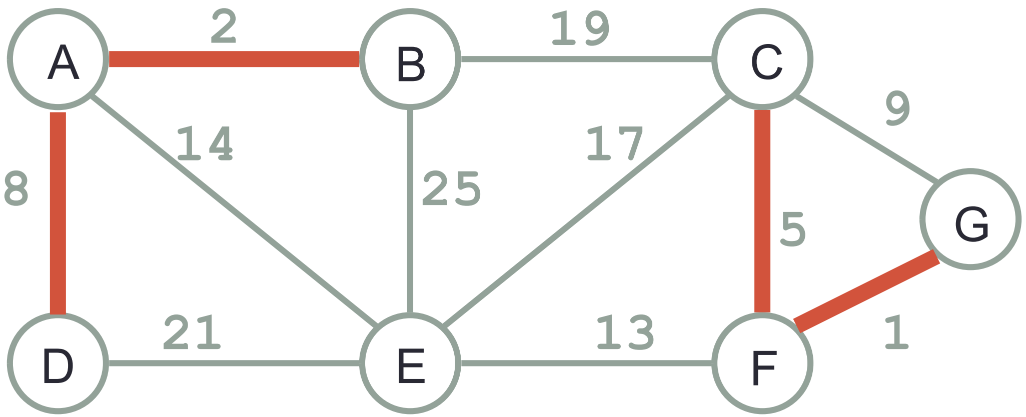
Algoritmo de Kruskal: Ejemplo



F ⇔ 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

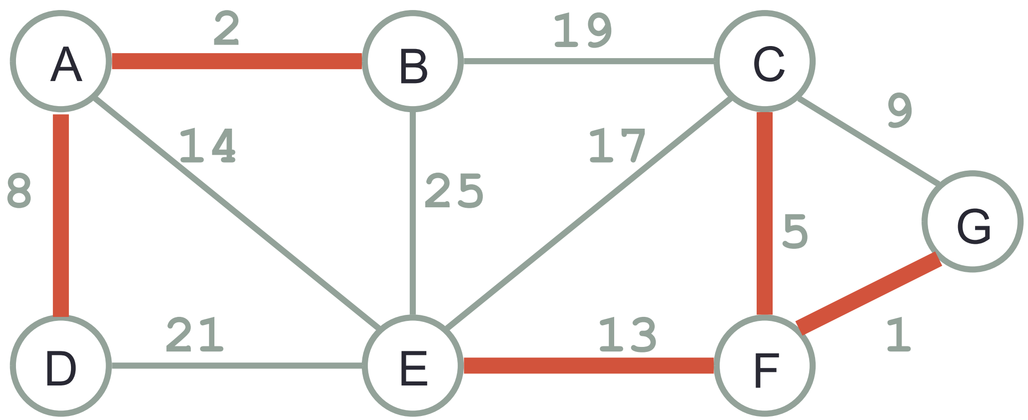
Algoritmo de Kruskal: Ejemplo



	F ⇔ G	... 1		
	A ⇔ B	... 2		
	C ⇔ F	... 5		
	A ⇔ D	... 8		
→	C ⇔ G	... 9	No	
	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

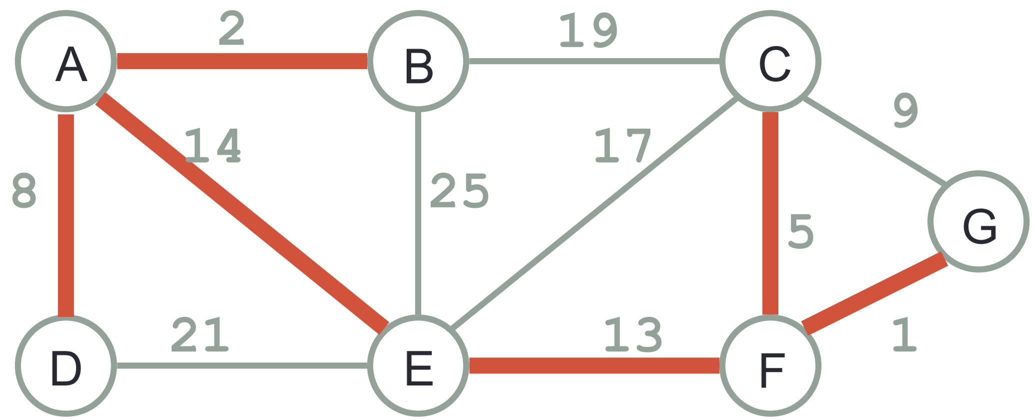
Algoritmo de Kruskal: Ejemplo



F ⇔ G ... 1
A ⇔ B ... 2
C ⇔ F ... 5
A ⇔ D ... 8
C ⇔ G ... 9 No
→ 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

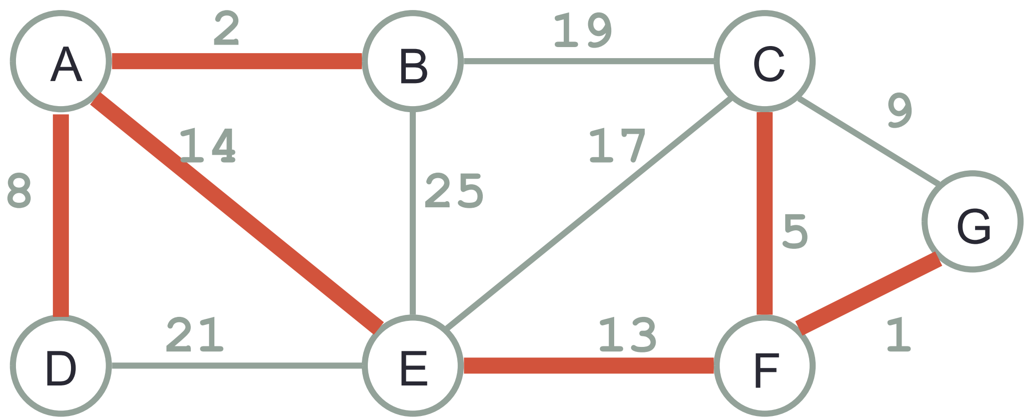
Algoritmo de Kruskal: Ejemplo



	F ⇔ G	... 1		
	A ⇔ B	... 2		
	C ⇔ F	... 5		
	A ⇔ D	... 8		
	C ⇔ G	... 9	No	
	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

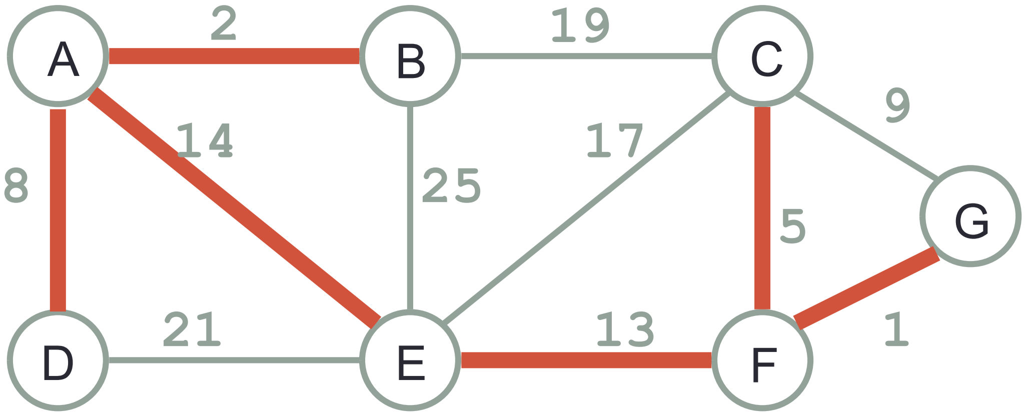
Algoritmo de Kruskal: Ejemplo



	F ⇔ G	... 1		
	A ⇔ B	... 2		
	C ⇔ F	... 5		
	A ⇔ D	... 8		
	C ⇔ G	... 9	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

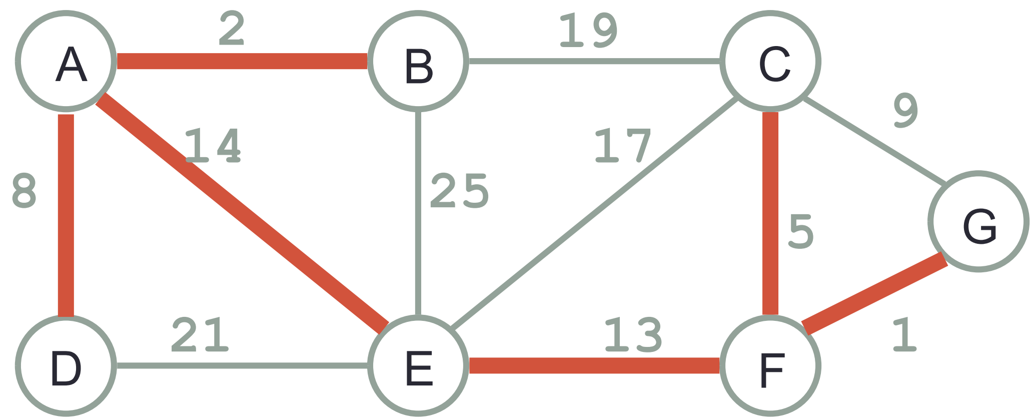
Algoritmo de Kruskal: Ejemplo



	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

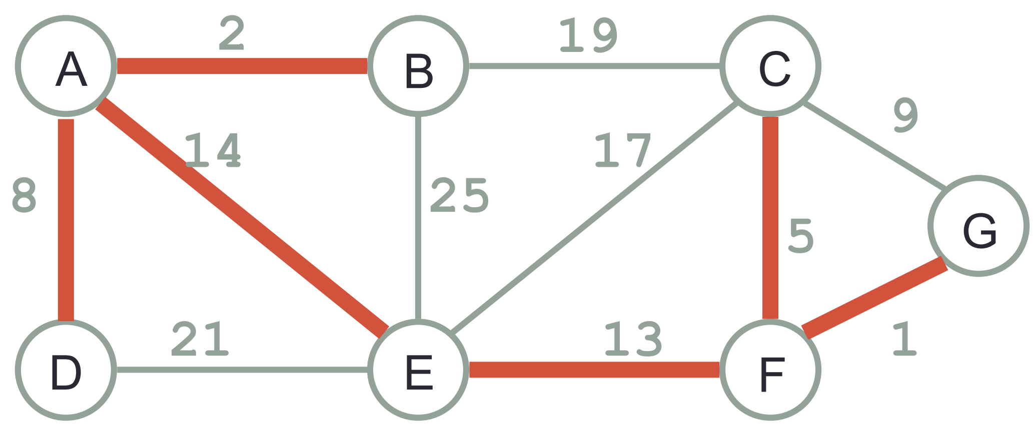
Algoritmo de Kruskal: Ejemplo



	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

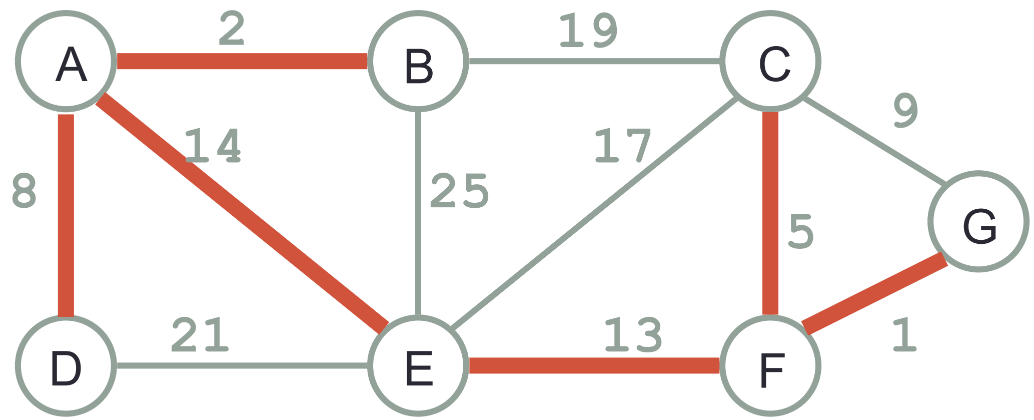
Algoritmo de Kruskal: Ejemplo



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

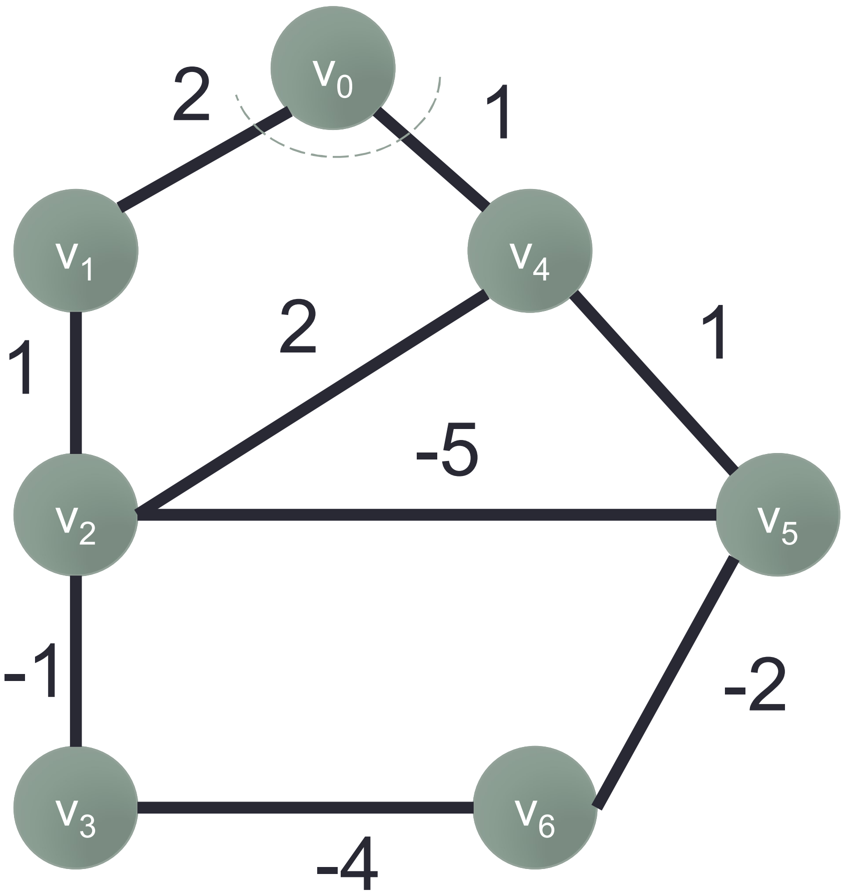
(1) u is in V_T

(2) v is in $V - V_T$

$V_T = V_T \cup \{v^*\}$
$E_T = E_T \cup \{e^*\}$
end loop

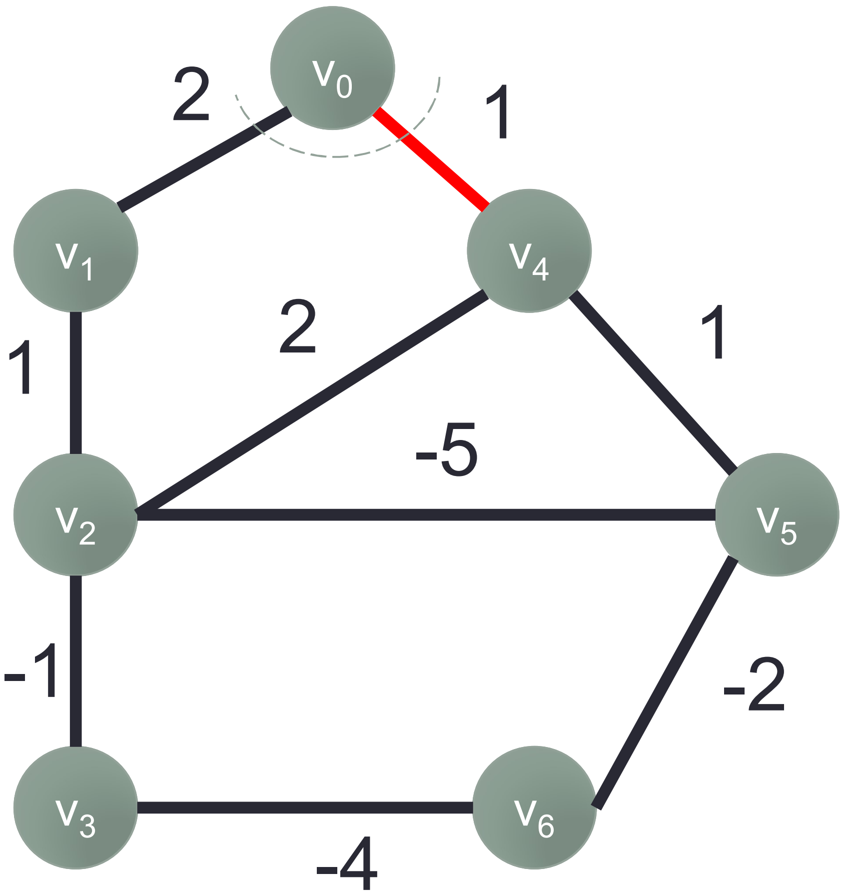
Son estrategias diferentes:
- *Kruskal recorre las aristas*
- *Prims recorre los vértices*

Prim



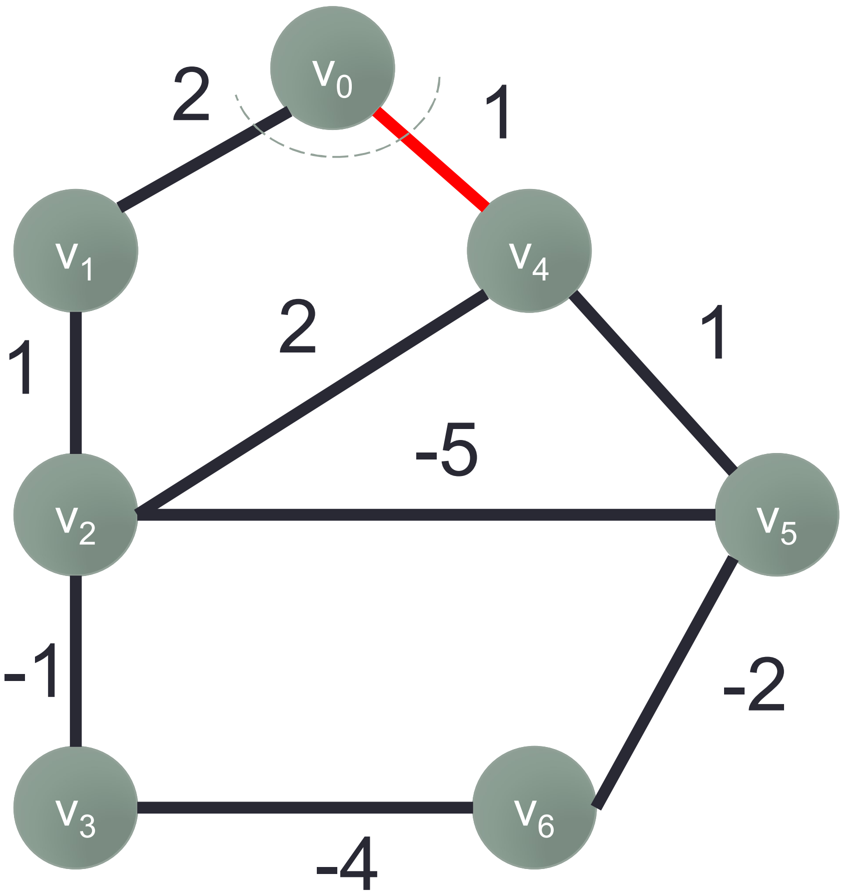
Árbol de Expansión Mínima

Prim

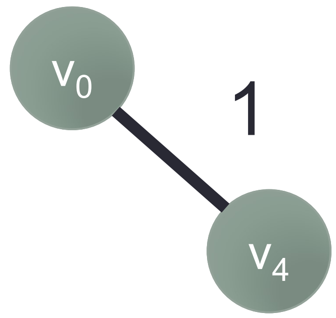


Árbol de Expansión Mínima

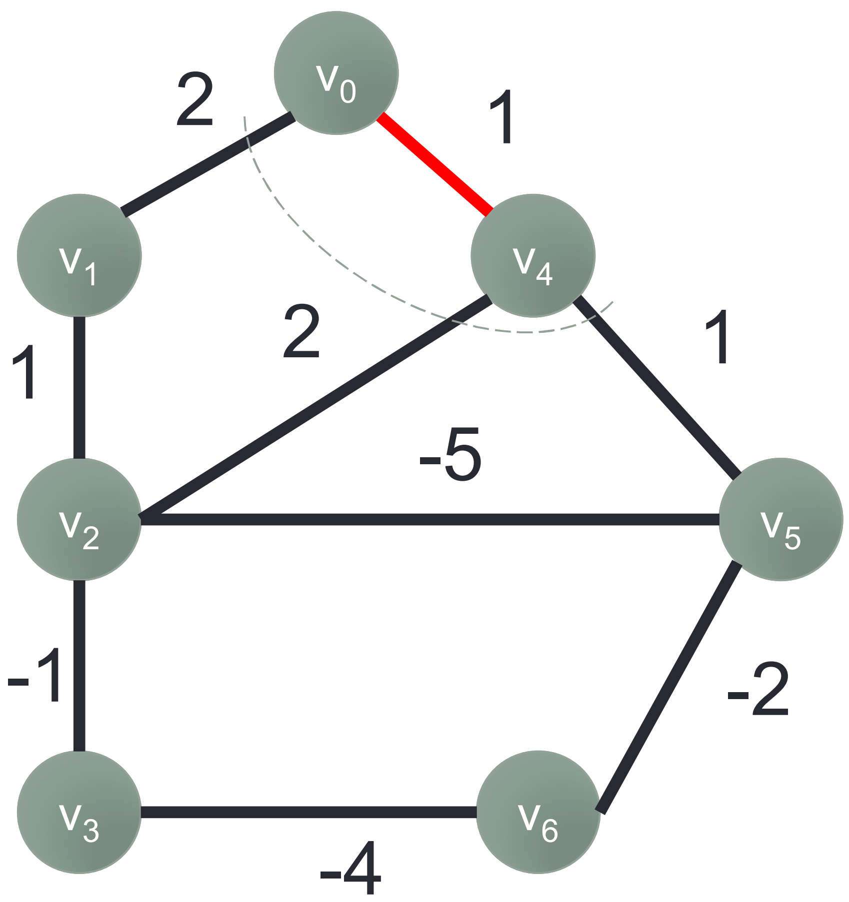
Prim



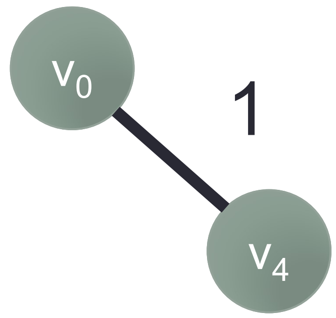
Árbol de Expansión Mínima



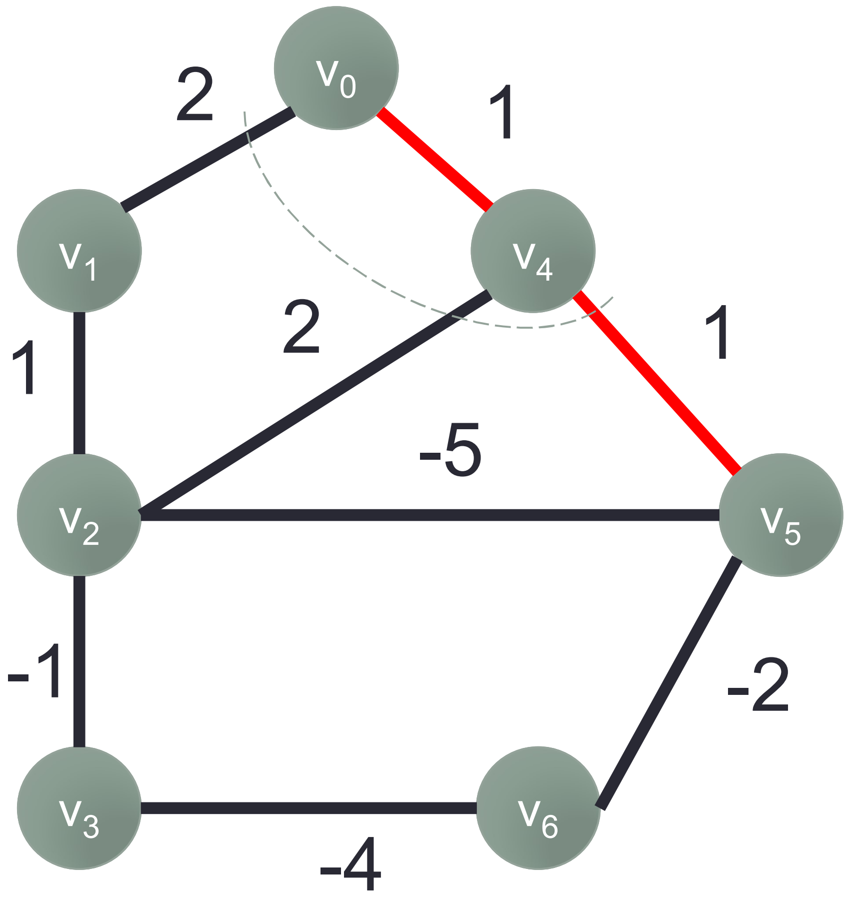
Prim



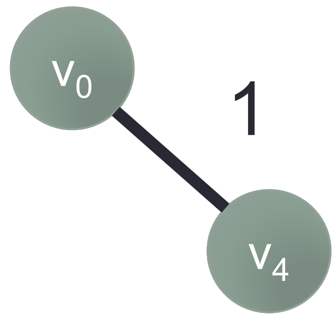
Árbol de Expansión Mínima



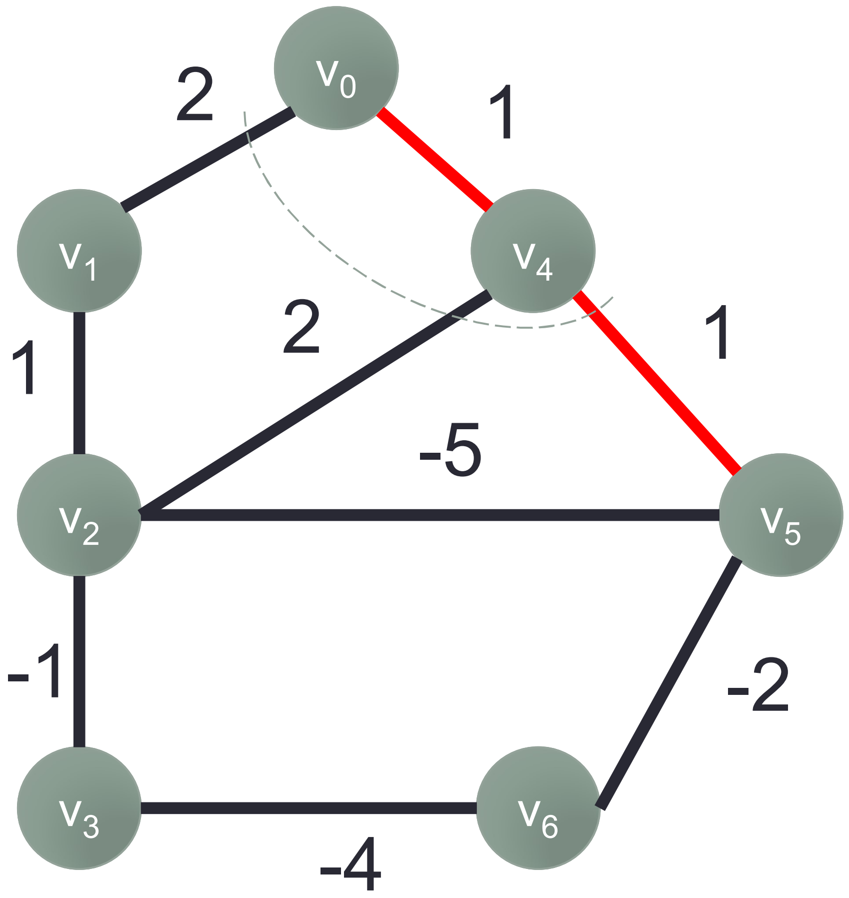
Prim



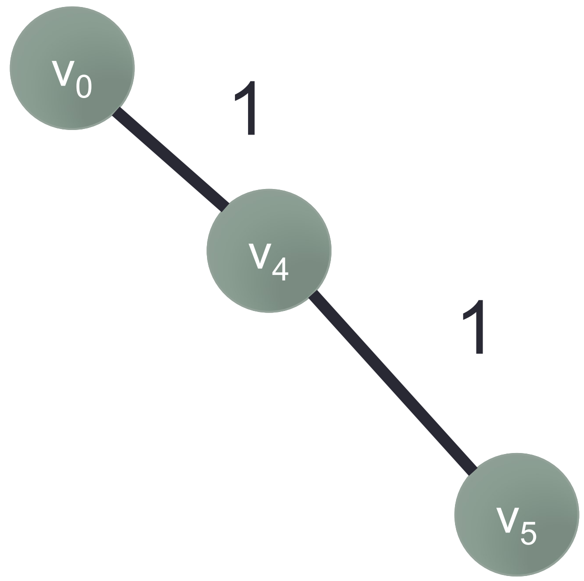
Árbol de Expansión Mínima



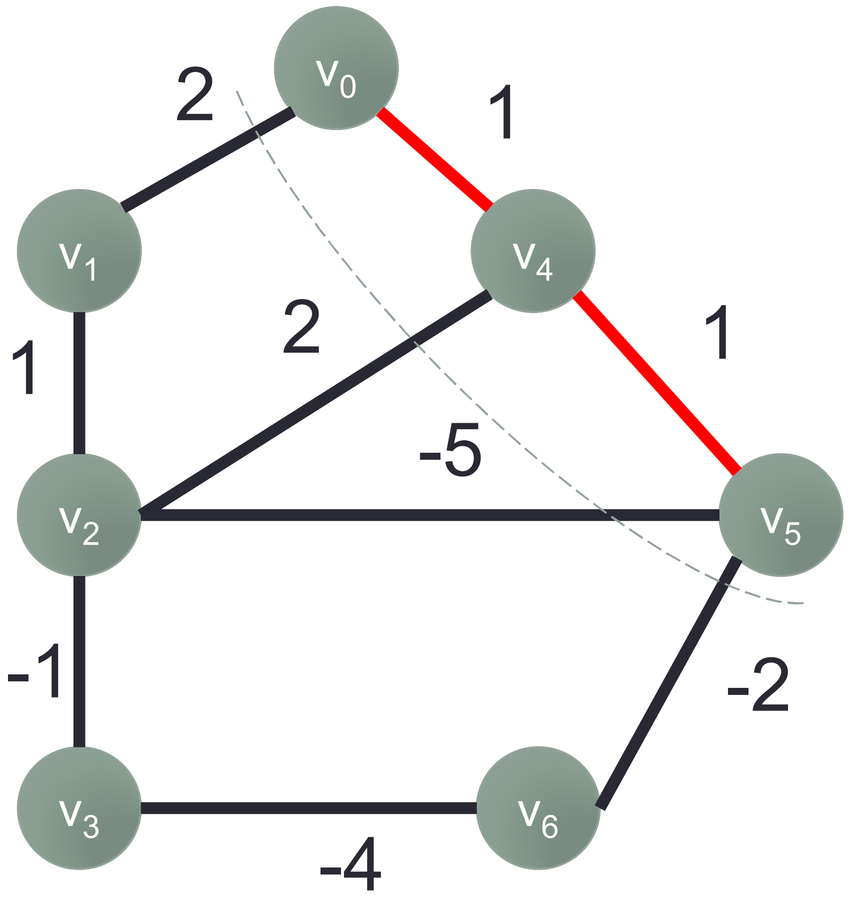
Prim



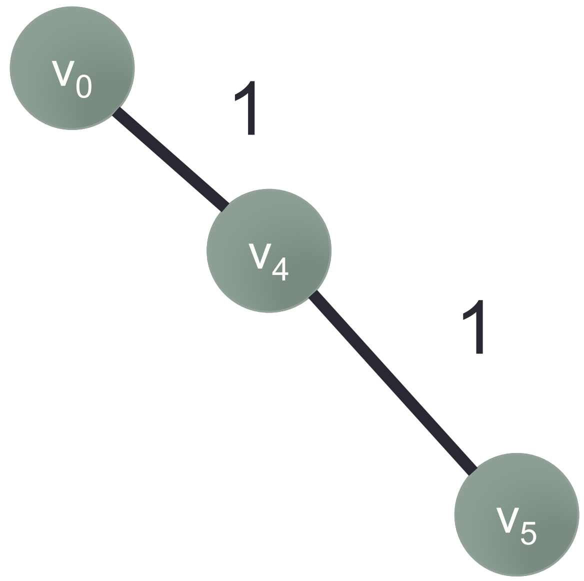
Árbol de Expansión Mínima



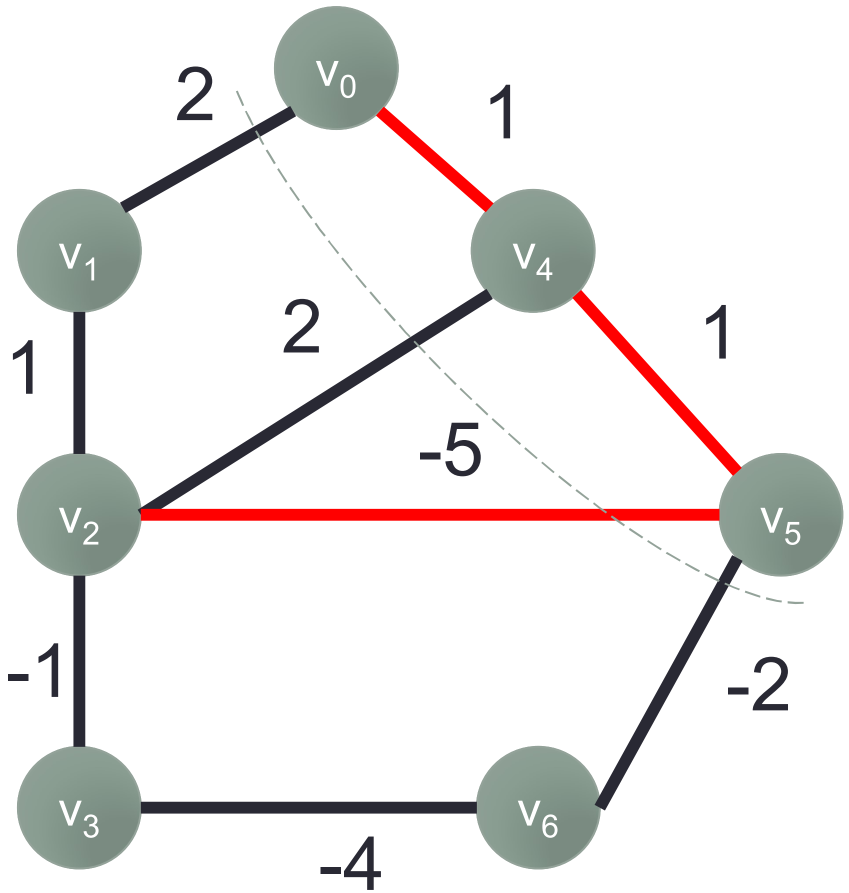
Prim



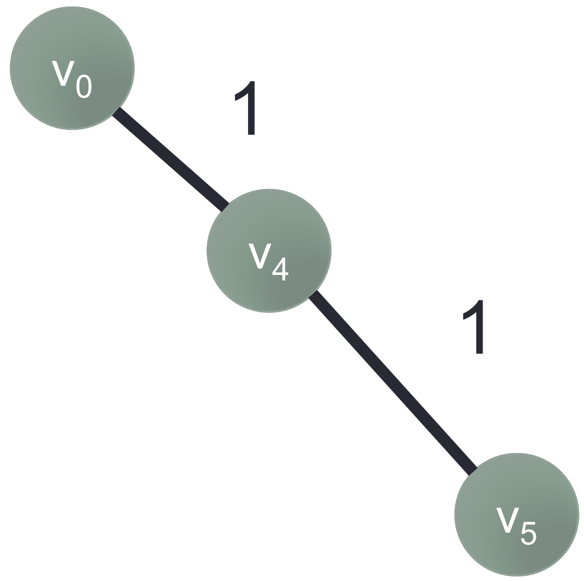
Árbol de Expansión Mínima



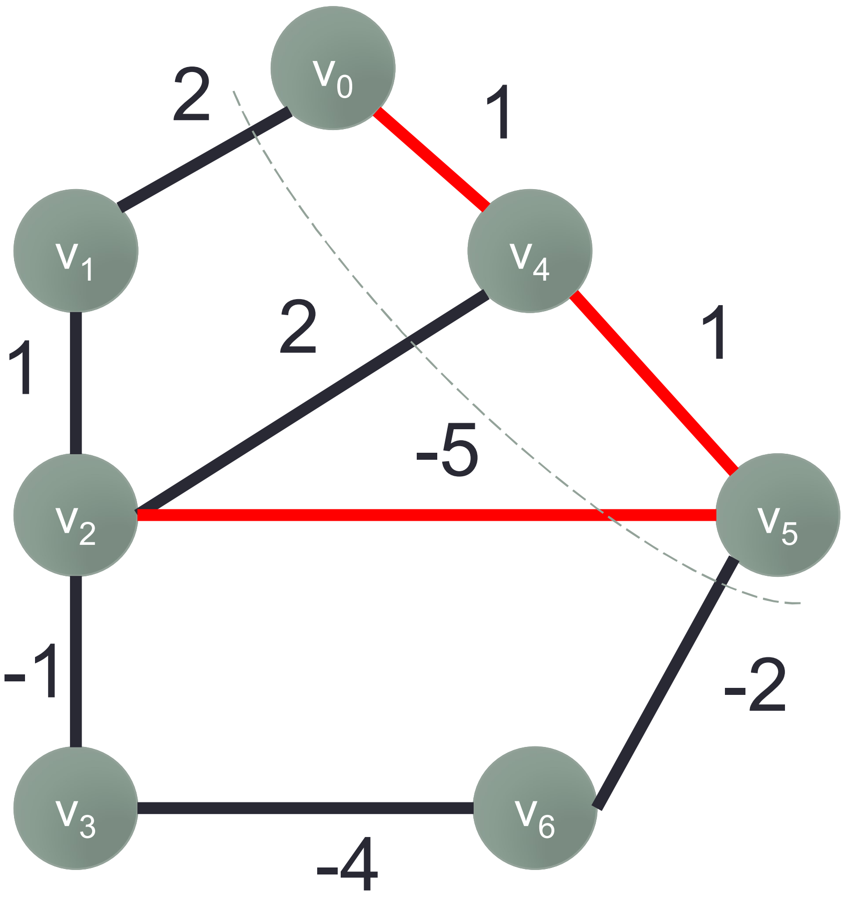
Prim



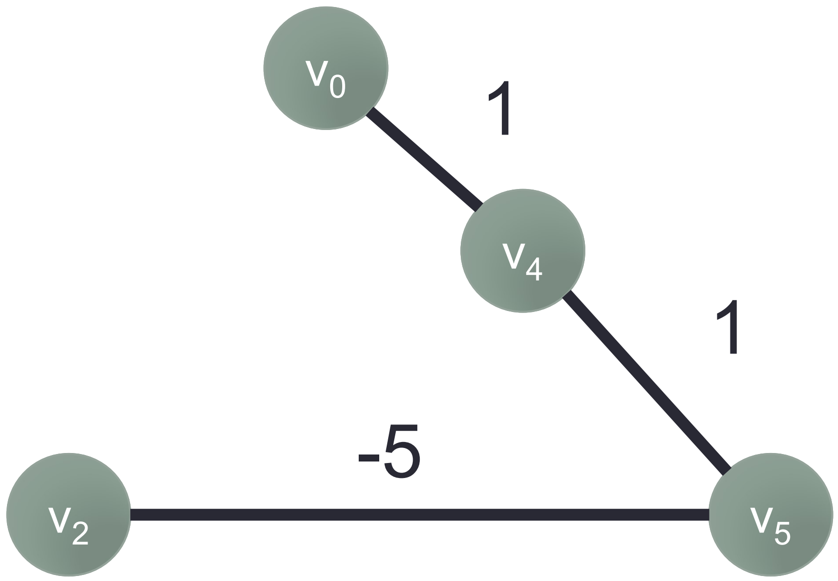
Árbol de Expansión Mínima



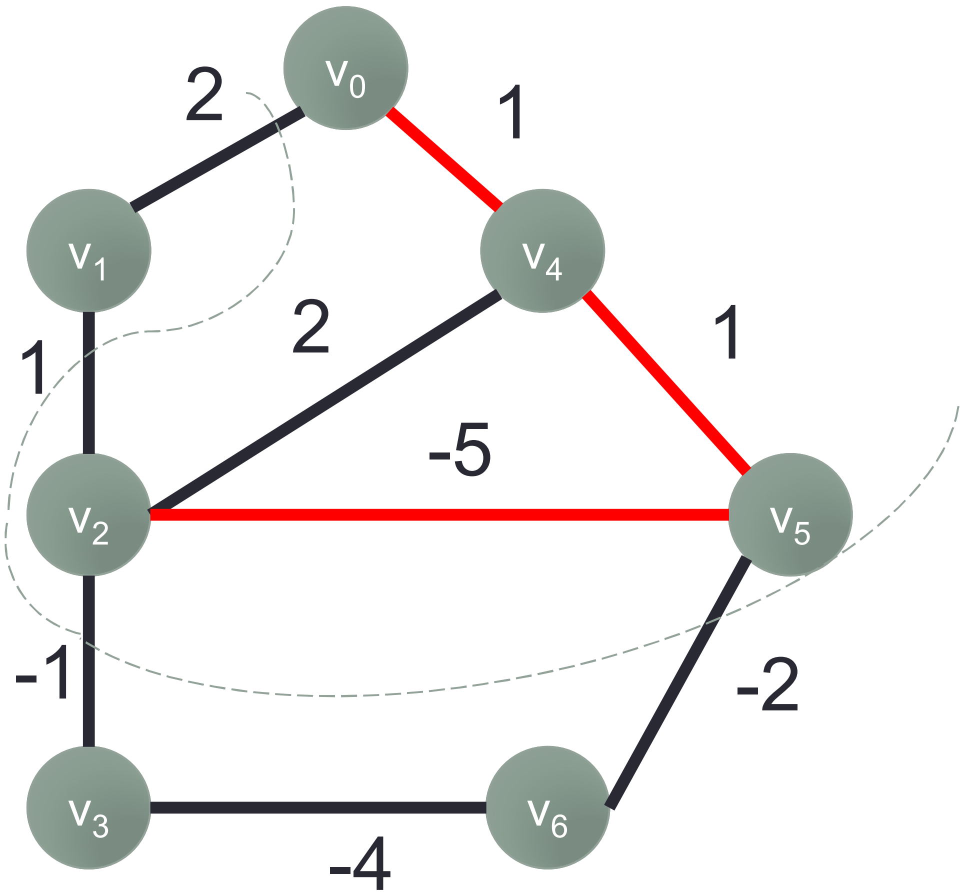
Prim



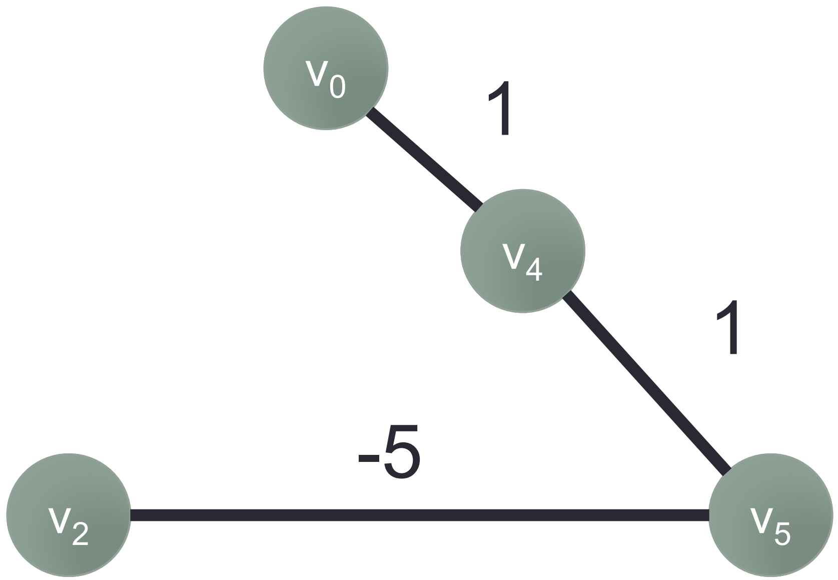
Árbol de Expansión Mínima



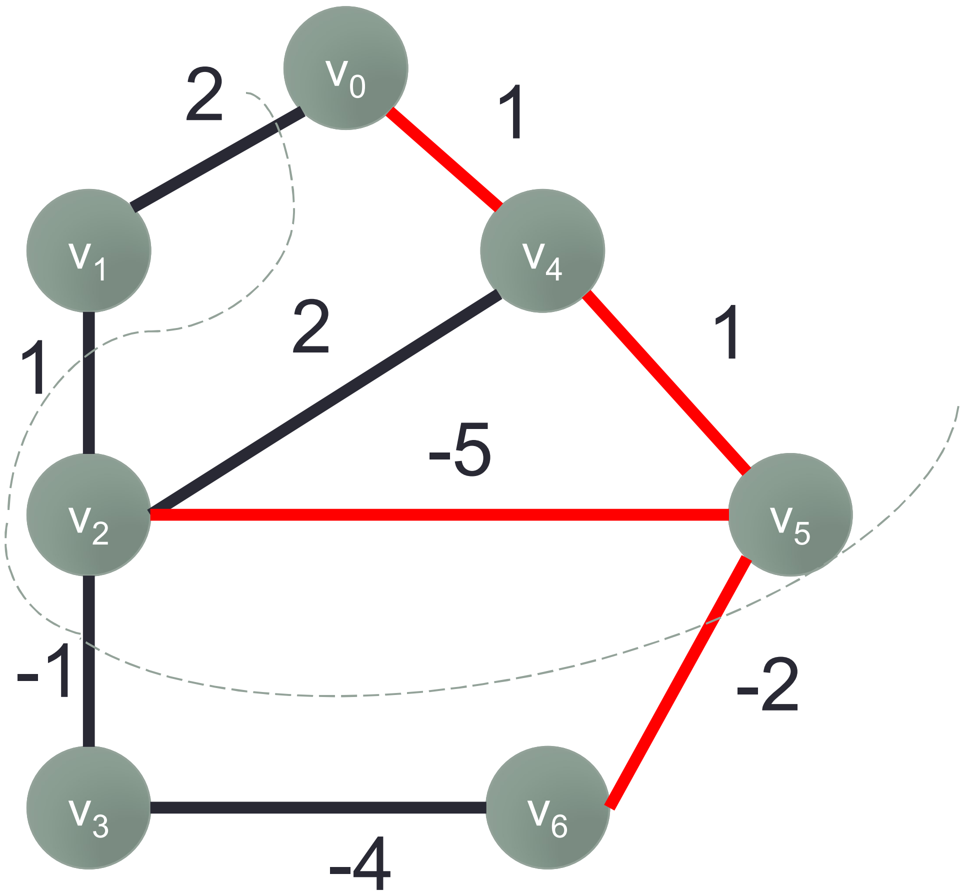
Prim



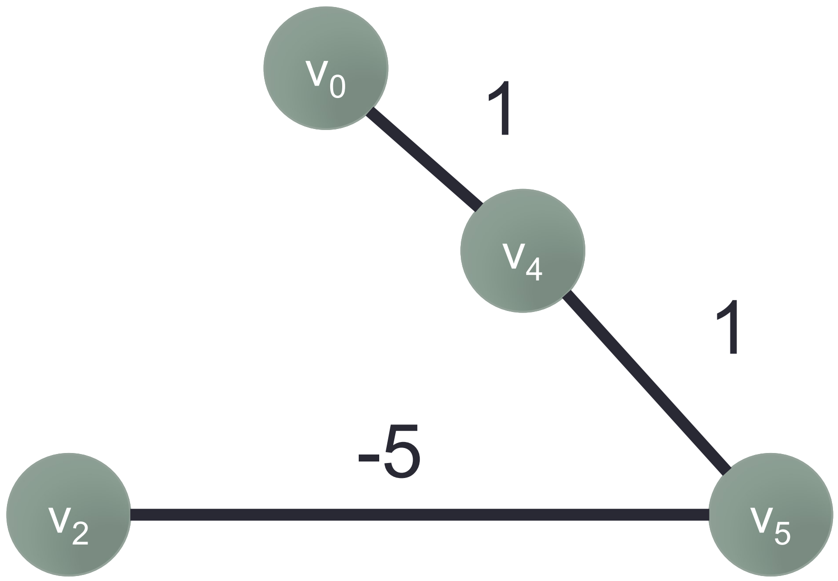
Árbol de Expansión Mínima



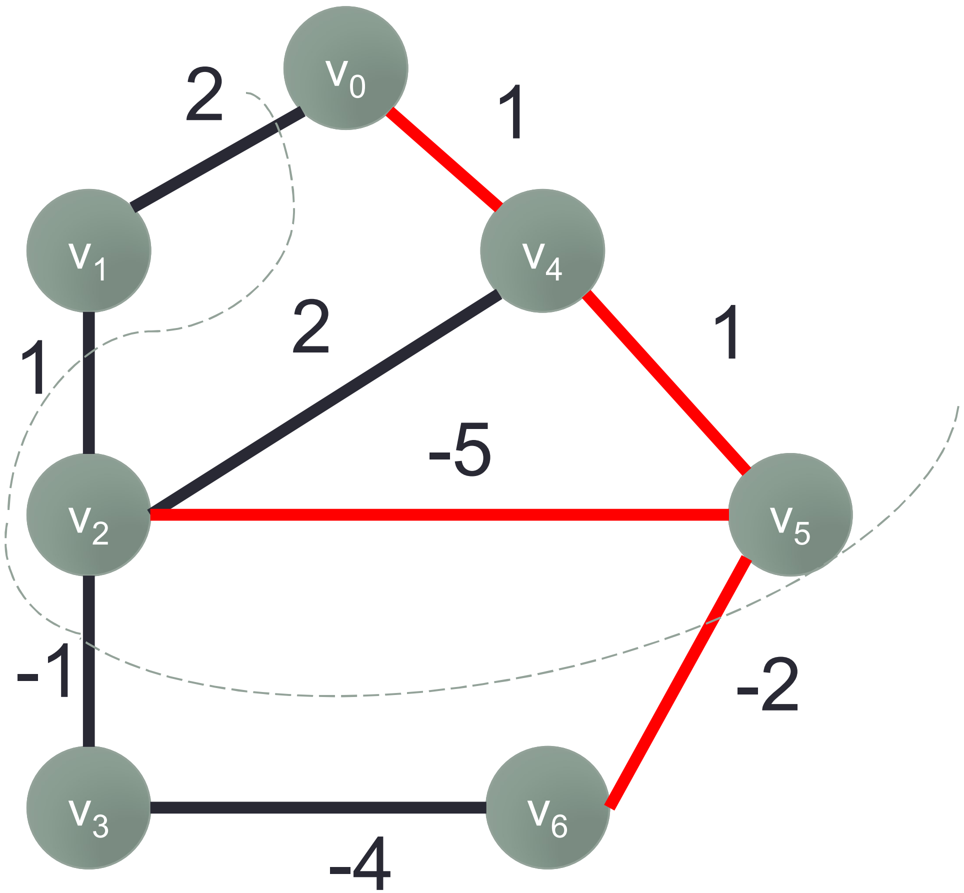
Prim



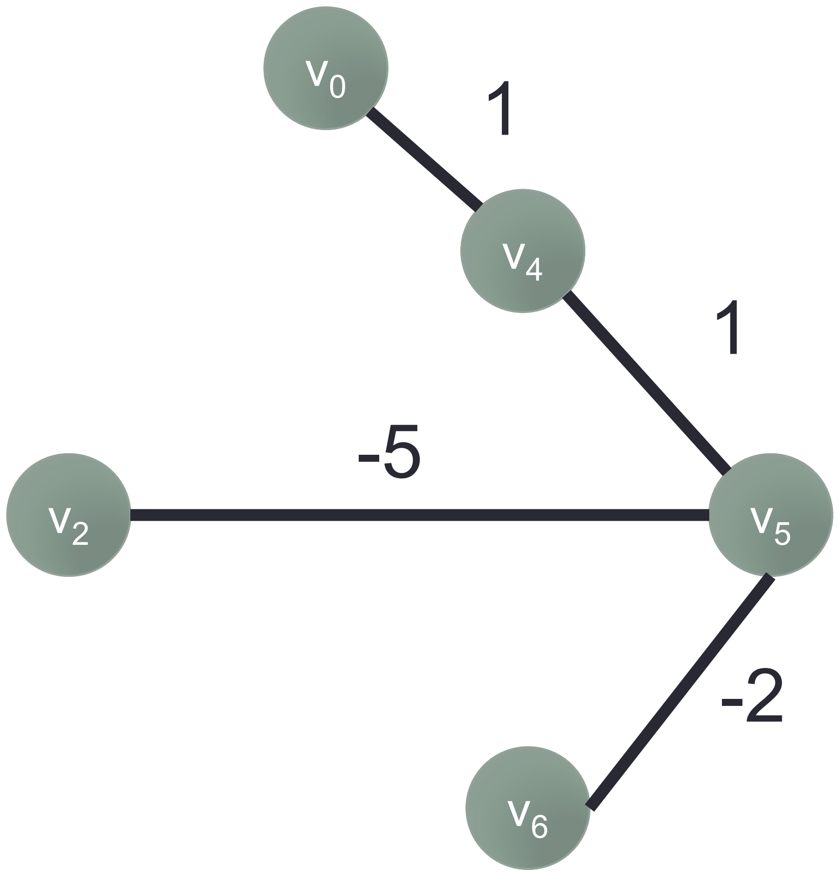
Árbol de Expansión Mínima



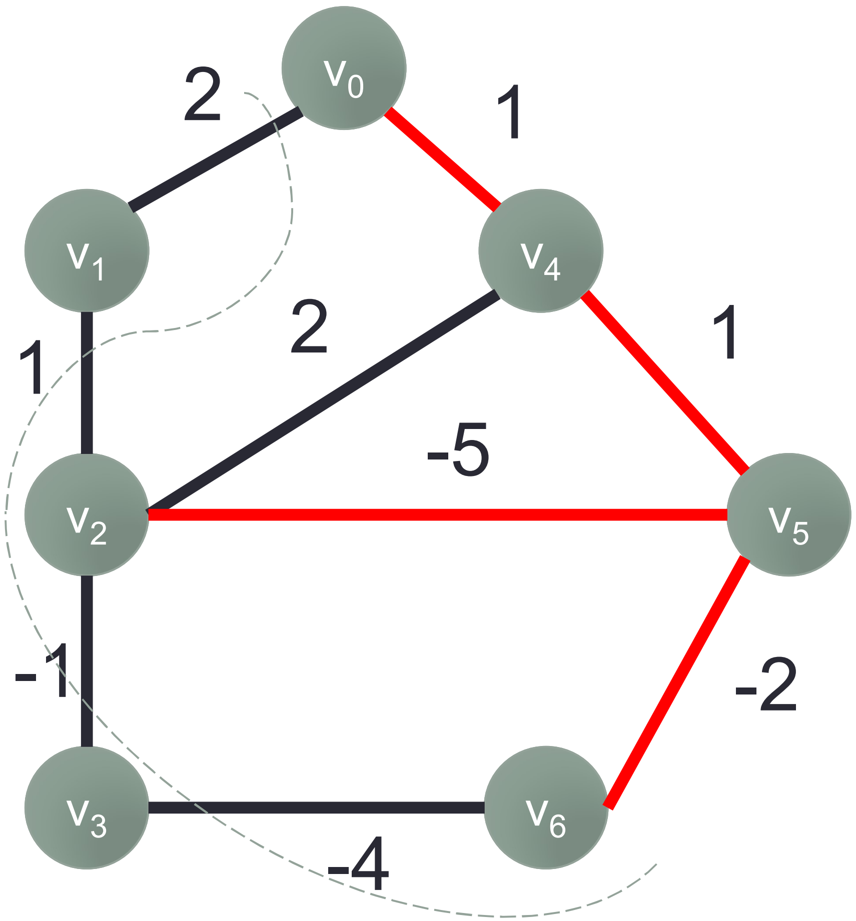
Prim



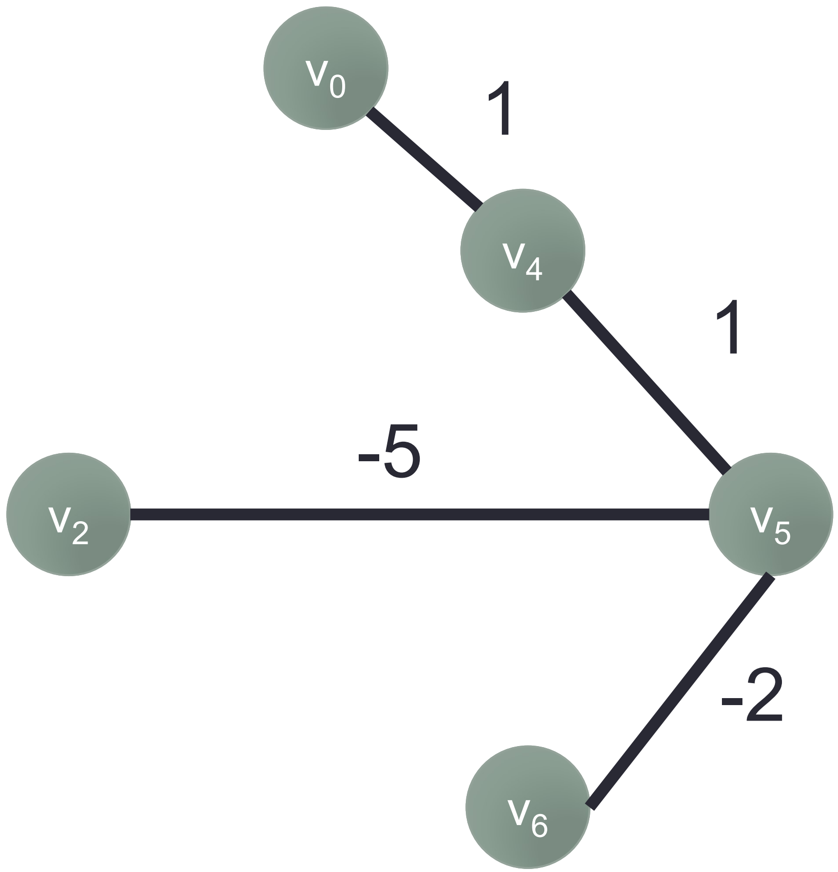
Árbol de Expansión Mínima



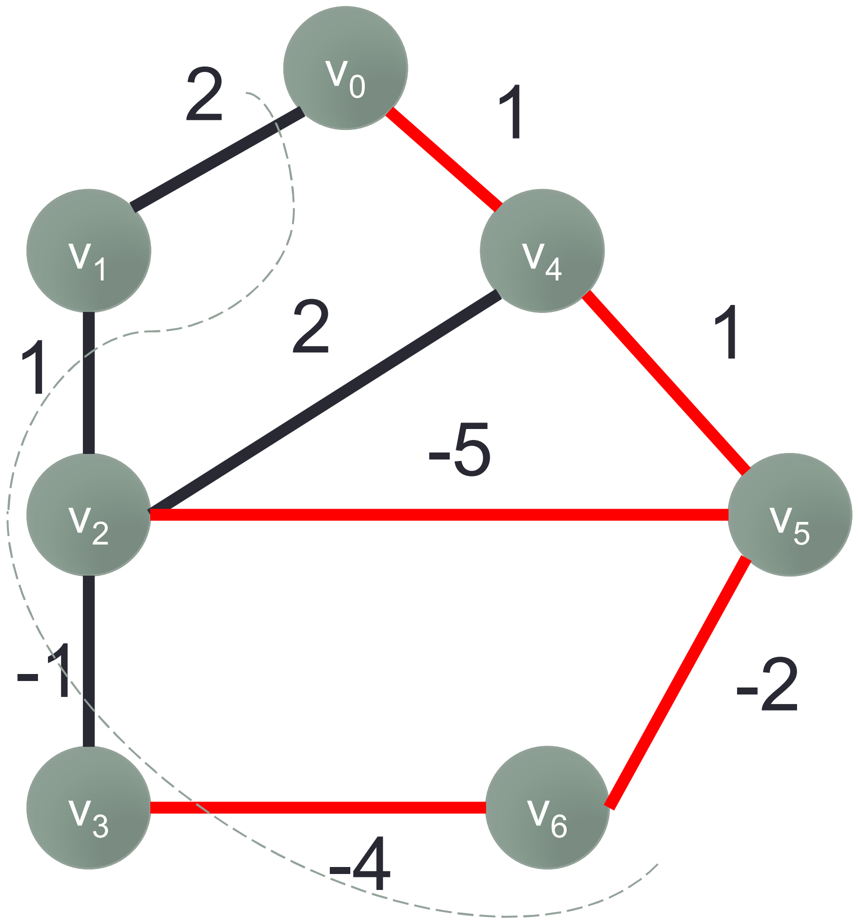
Prim



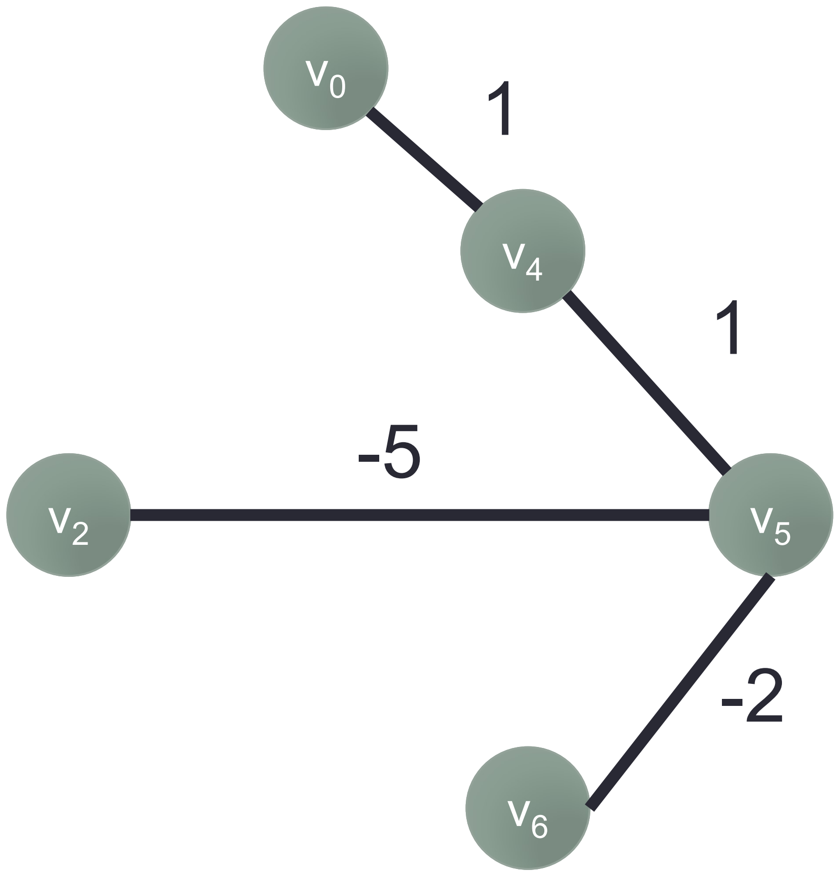
Árbol de Expansión Mínima



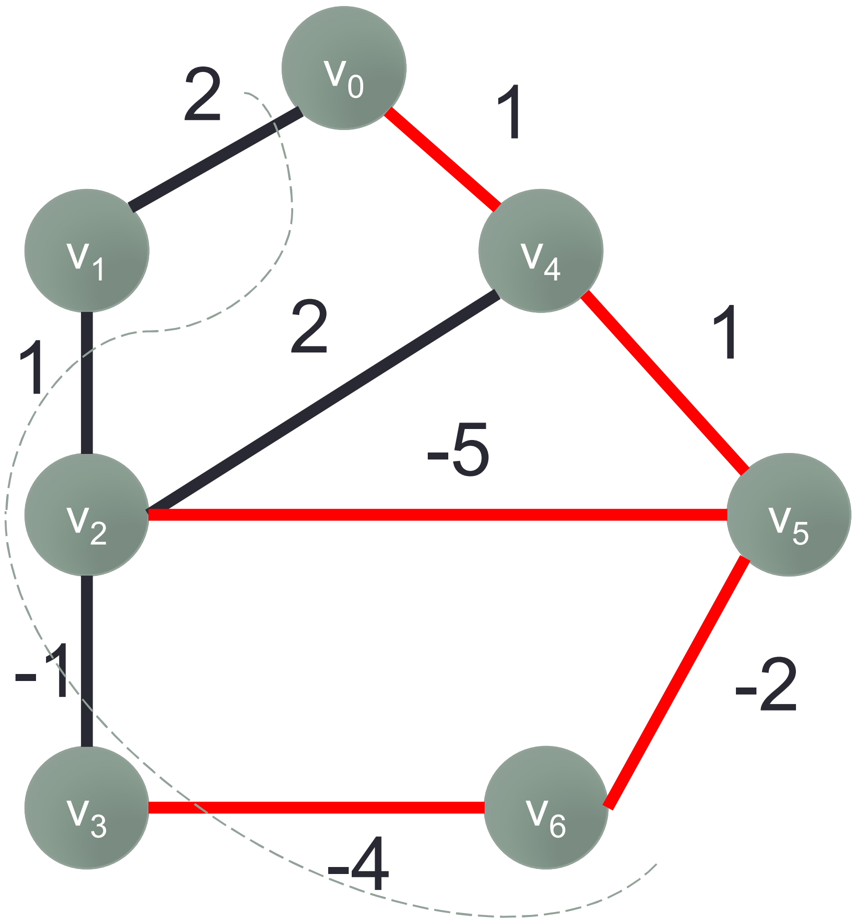
Prim



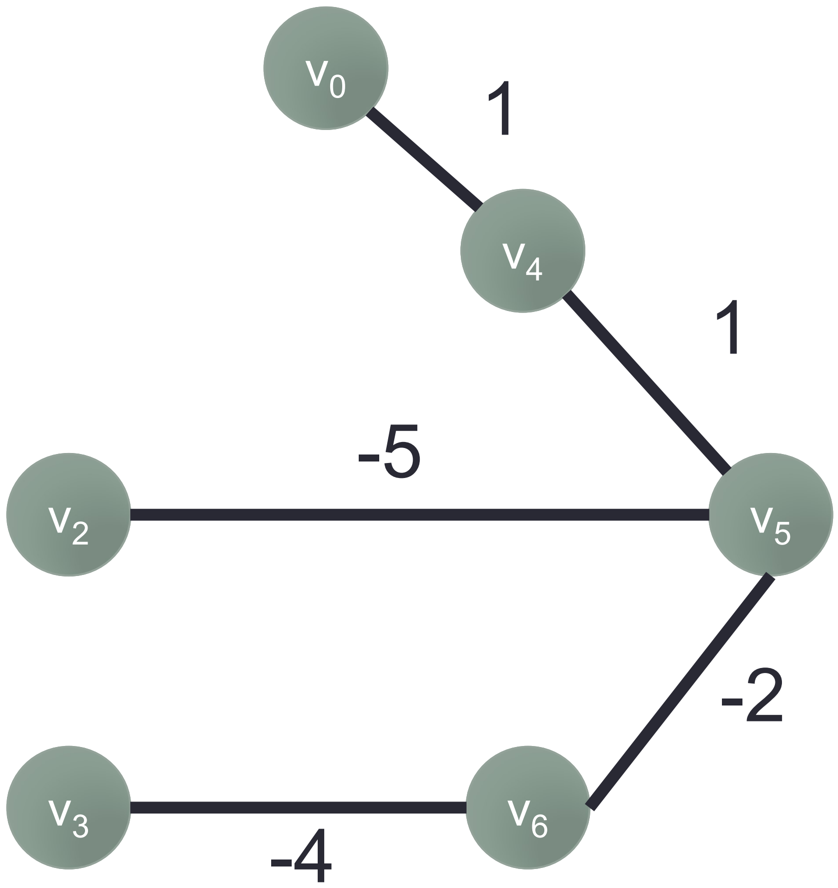
Árbol de Expansión Mínima



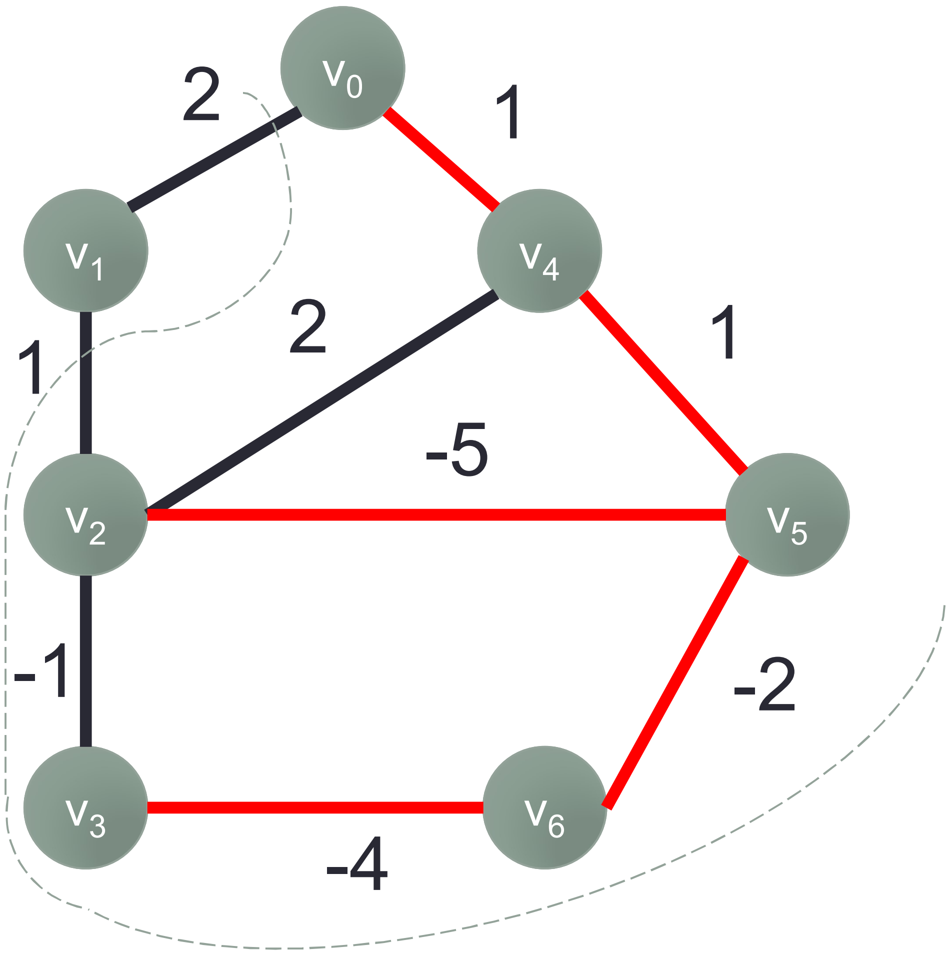
Prim



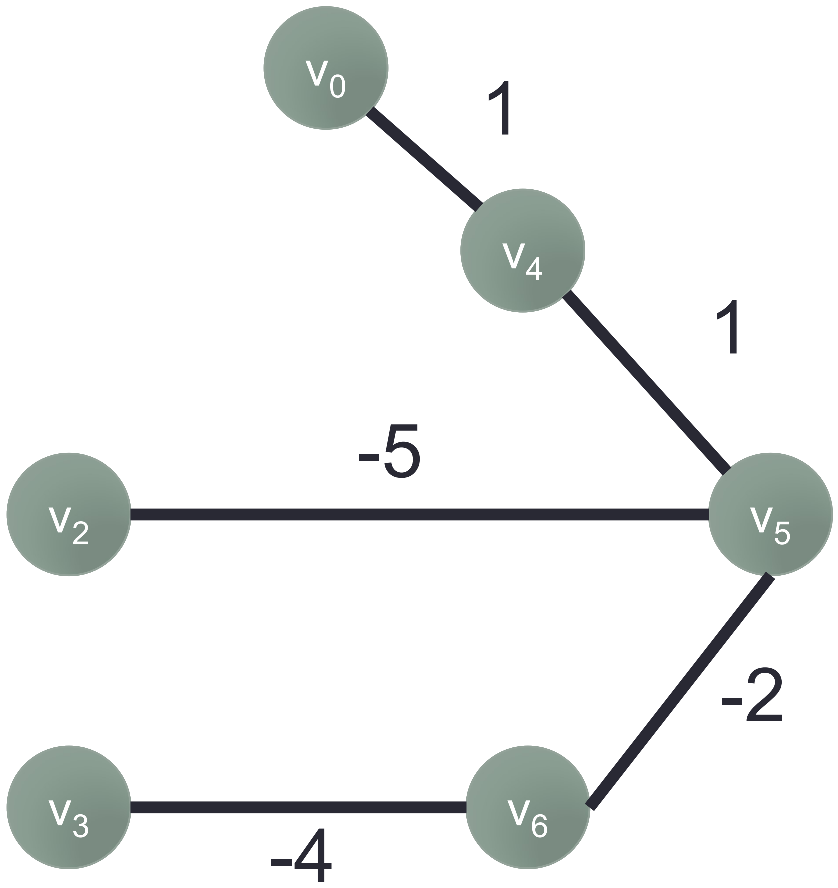
Árbol de Expansión Mínima



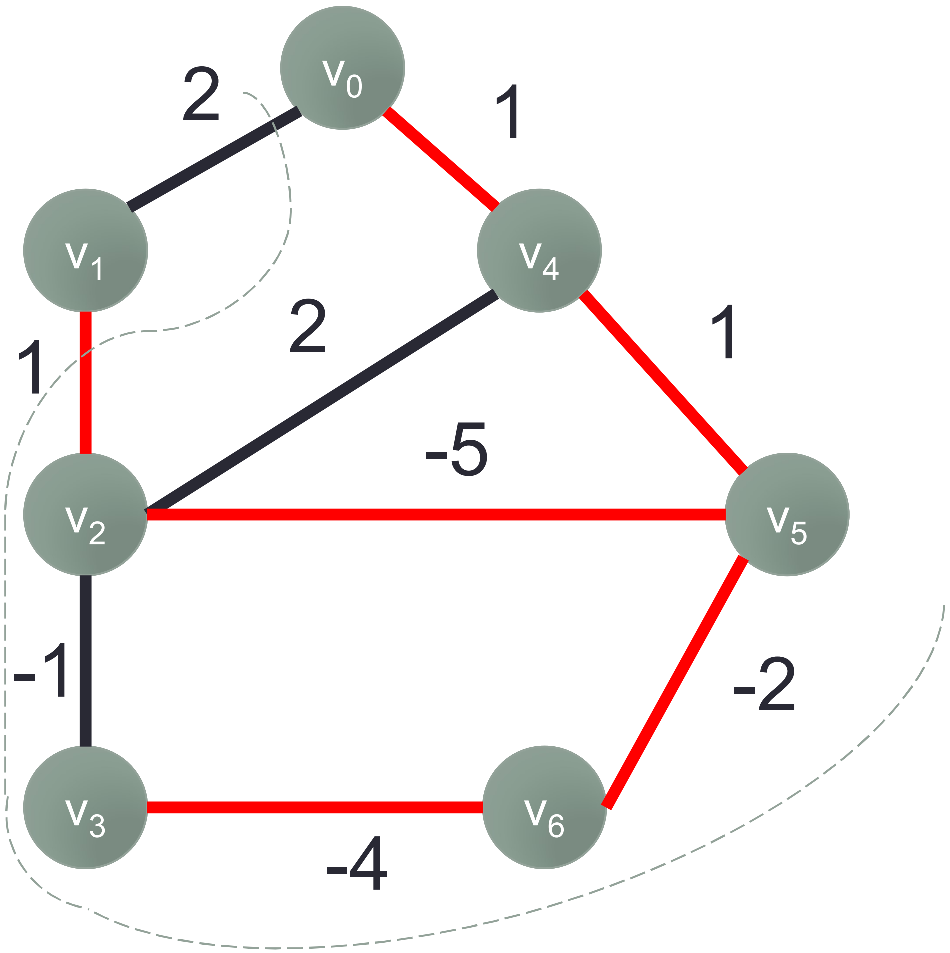
Prim



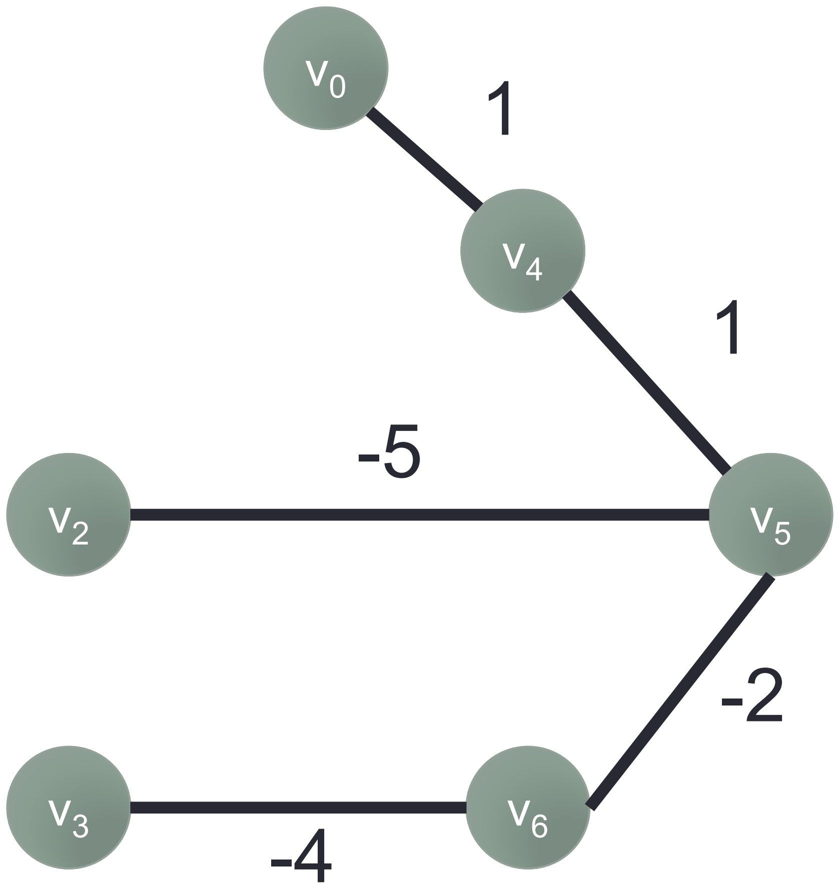
Árbol de Expansión Mínima



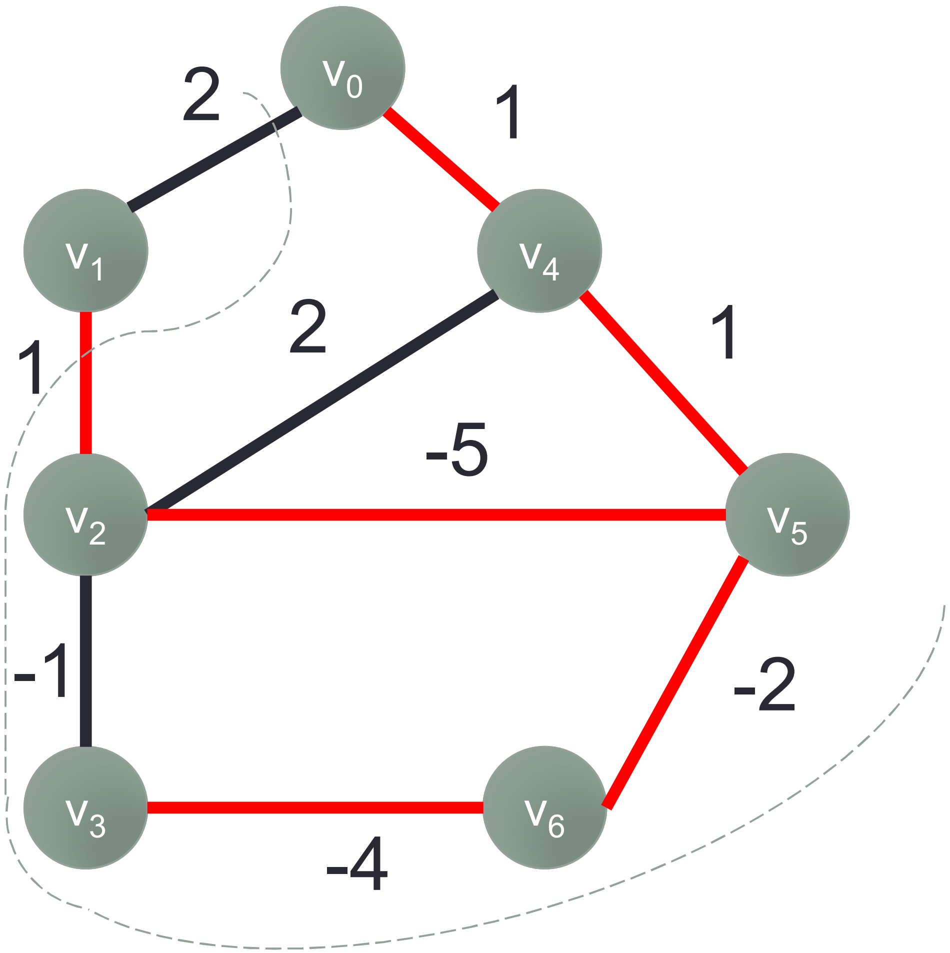
Prim

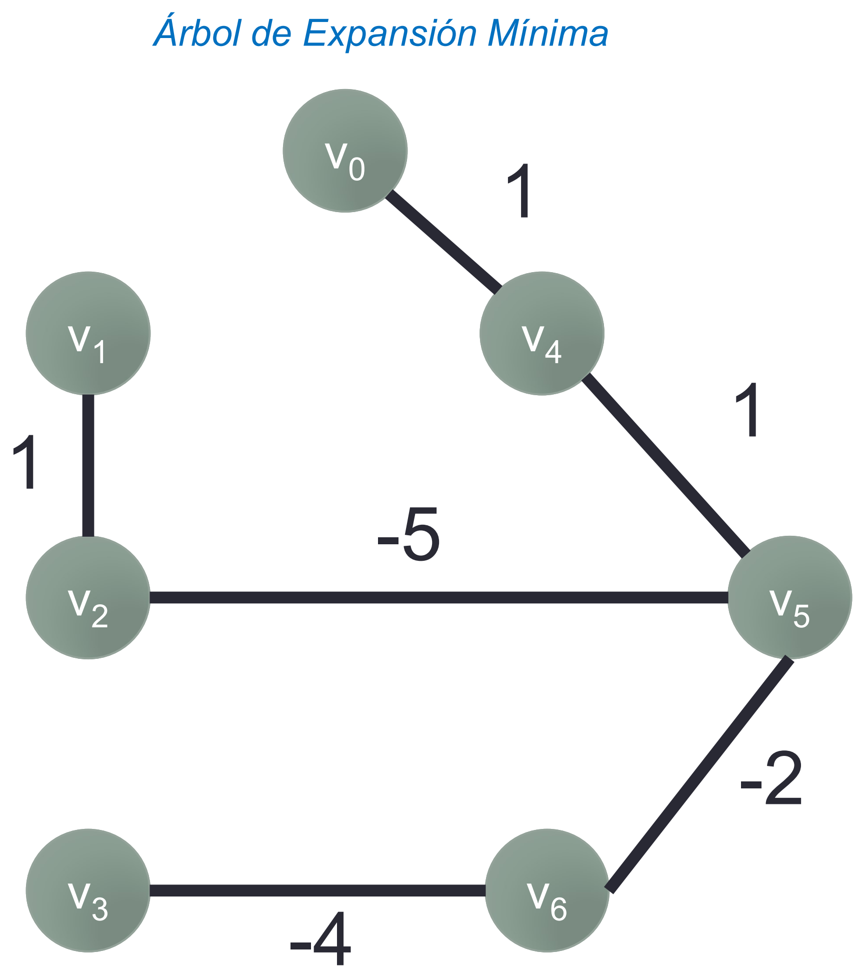


Árbol de Expansión Mínima



Prim





Árbol de Expansión Mínima

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!