Técnica de diseño #3: Algoritmos voraces

Análisis y Diseño de Algoritmos

Ing. Román Martínez M.

Técnicas de diseño de algoritmos

- Divide y vencerás
- Programación dinámica
- **Algoritmos voraces**
- Backtraking
- Branch and bound

Algoritmos voraces

- También conocidos como algoritmos ávidos, glotones o miopes…
- Su característica es que **toman decisiones basándose en la información que tienen en forma inmediata**, sin tener en cuenta los efectos que esto pueda tener en el futuro, es decir, nunca reconsidera su decisión, sea cual fuera la situación que ocurrirá más adelante…
- **Se confía que la decisión tomada sea la mejor para la solución general del problema**…

Algoritmos voraces

- Utilizados en aplicaciones de optimización...
- Son algoritmos fáciles de diseñar y de implementar…
- pero… **NO** siempre llevan a una solución correcta del problema…
- Cuando si obtienen la solución correcta, lo hacen de una manera eficiente…
- Sin embargo, es difícil demostrar formalmente cuando un algoritmo voraz es correcto o no...

Estructura general de un algoritmo voraz

- Se apoya en un conjunto original de datos (***C***), y el conjunto resultante que contendrá la solución (***S***)

S = ∅

Mientras C <> ∅ y no se haya encontrado la solución:

*x = __**selección**__ de mejor candidato de C*

C = C - {x}

*si es __**factible**__ S ∪ {x} entonces S = S ∪ {x}*

***__Si S tiene la solución__** devolver S, sino, no hay solución.*

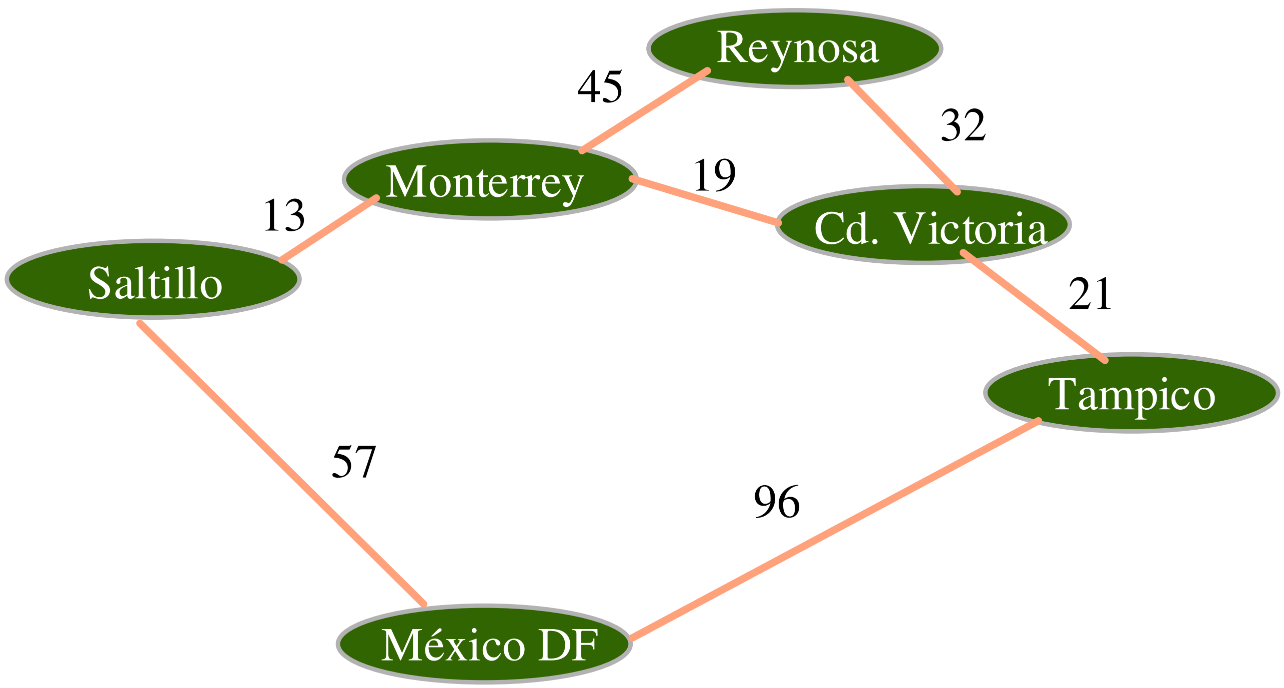
¿Se puede usar esta técnica para resolver el problema del viajero?

El problema del árbol de extensión mínima

- Dado un grafo no dirigido y ponderado...
- **¿Cuál es el costo MÍNIMO para tener a todos los vértices conectados?**
- La solución del problema, implica que se obtenga un subgrafo del grafo original, en el que NO se tengan ciclos...
- por lo tanto, la solución obtiene un ÁRBOL que es llamado de EXTENSIÓN MÍNIMA por la optimización que se realiza.

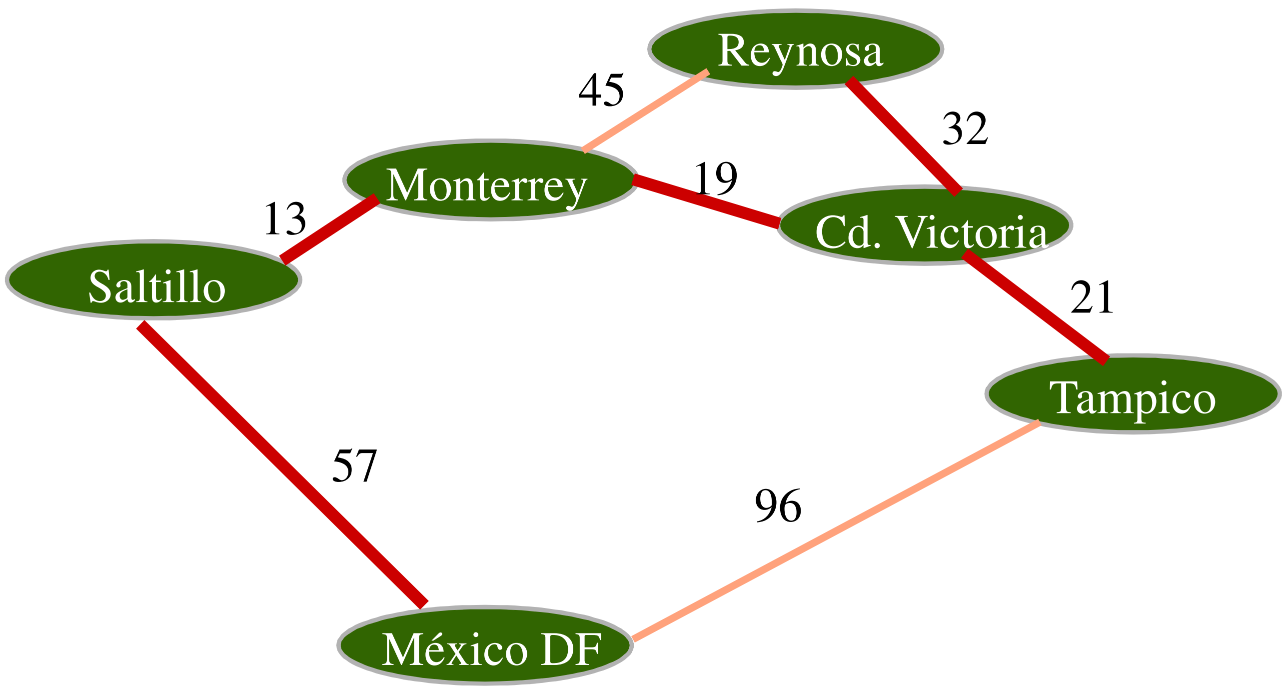
Ejemplo

- ¿Cómo puedo tener conectadas a las siguientes ciudades por un costo mínimo?



Ejemplo

- ¿Cómo puedo tener conectadas a las siguientes ciudades por un costo mínimo?



Algoritmo general para obtener el Árbol de extensión mínima

- Sea el grafo G = (V, A) en donde V es el conjunto de vértices y A el conjunto de arcos de la forma (v_i, v_j).

La forma en que se hace la selección determina el algoritmo específico

S = $\varnothing$

Mientras (no se haya resuelto el problema)

Seleccionar un arco de A de acuerdo a cierta política de optimización **--> SELECCION**

Si al agregar ese arco a S no genera un ciclo en el subgrafo, agregarlo a S **--> FACTIBILIDAD**

Si (V,S) es el árbol de extensión mínima, el problema se ha resuelto **--> VERIFICACIÓN DE LA SOLUCIÓN**

Algoritmo de Prim

$S = \varnothing$

$Y = \{v_1\}$

Mientras (no se haya resuelto el problema)

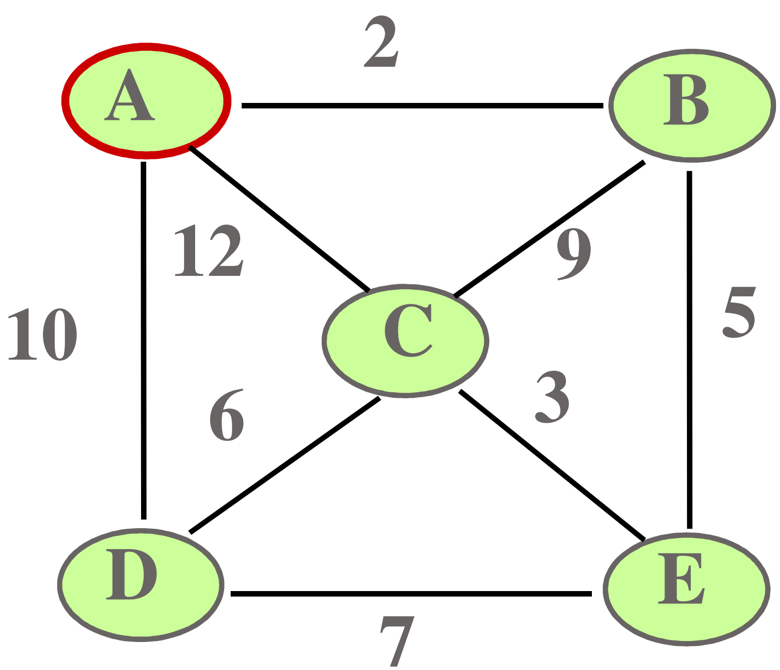
Seleccionar el vértice de V-Y que sea el más cercano (menor peso) a alguno de los vértices en Y.

Agregar el vértice a Y.

Agregar el arco correspondiente a S.

Si $Y = V$ *el problema se ha resuelto.*

Ejemplo



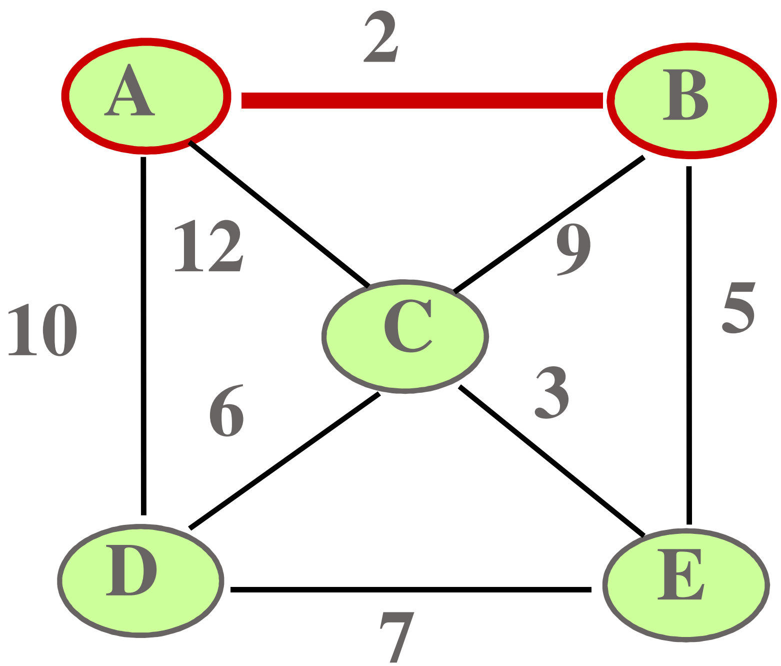
$S = \varnothing$

$Y = \{v_A\}$

$V\text{-}Y = \{v_B, v_C, v_D, v_E\}$

Seleccionar el arco de menor costo de Y a V-Y

Ejemplo



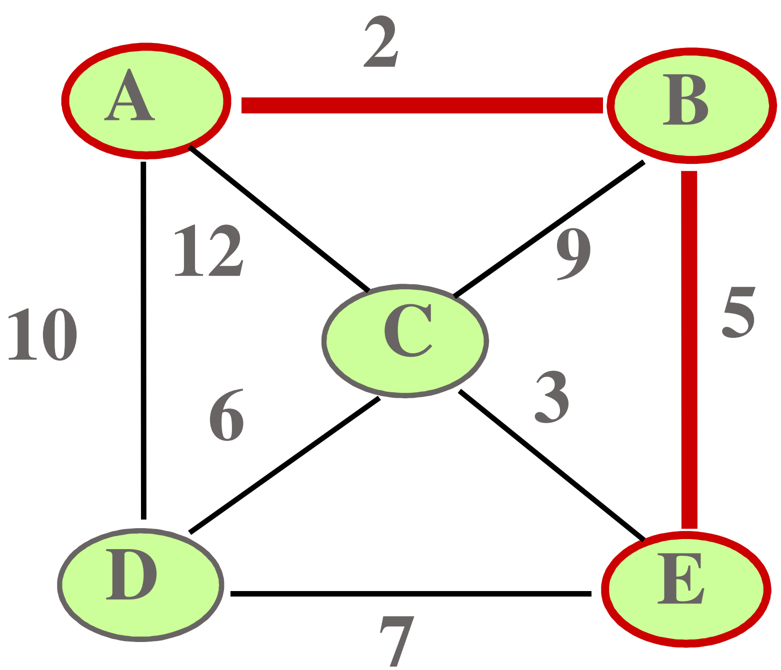
$S = \{(v_A, v_B)\}$

$Y = \{v_{A,}, v_B\}$

$V\text{-}Y = \{v_C, v_D, v_E\}$

Seleccionar el arco de menor costo de Y a V-Y

Ejemplo



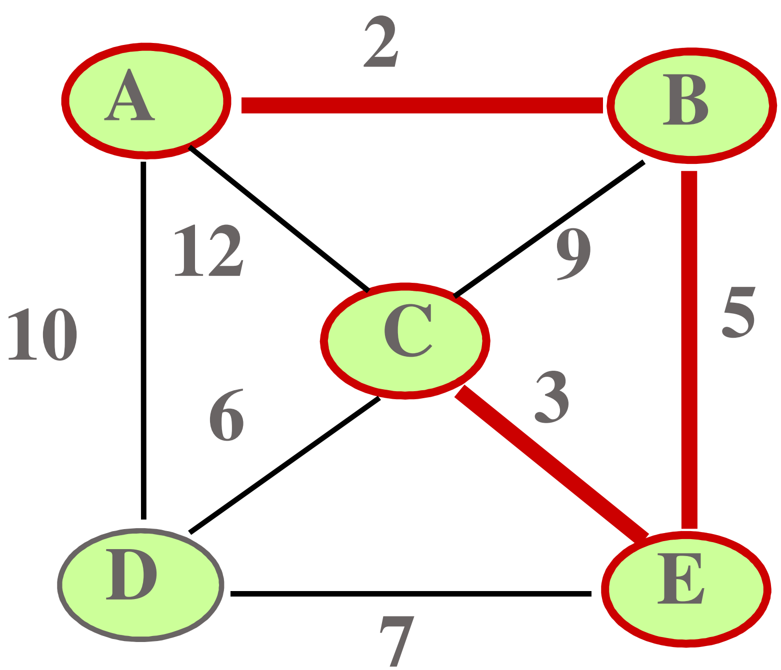
$S = \{(v_A, v_B),(v_B, v_E)\}$

$Y = \{v_A, v_B, v_E\}$

$V\text{-}Y = \{v_C, v_D\}$

Seleccionar el arco de menor costo de Y a V-Y

Ejemplo



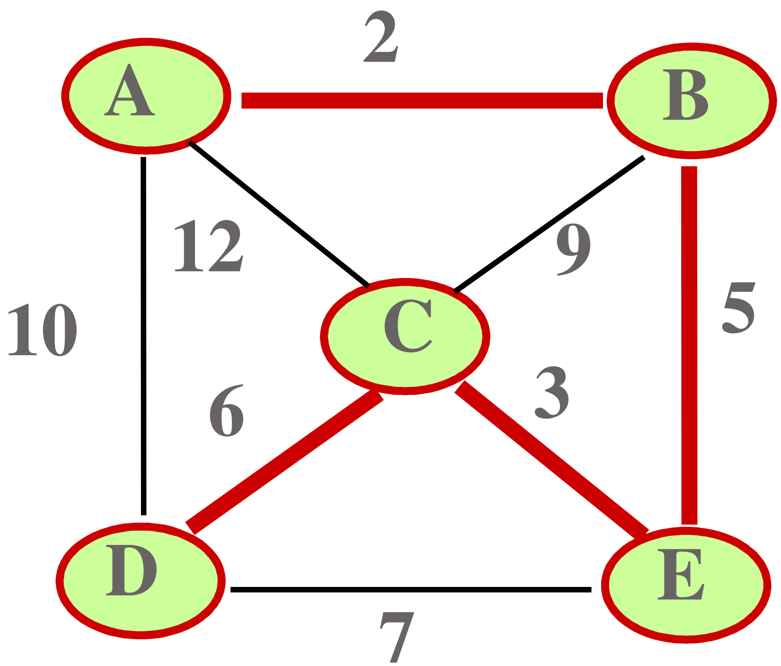
$S = \{(v_A, v_B),(v_B, v_E), (v_E, v_C)\}$

$Y = \{v_A, v_B, v_E, v_C\}$

$V\text{-}Y = \{v_D\}$

Seleccionar el arco de menor costo de Y a V-Y

Ejemplo



$S = \{(v_A, v_B),(v_B, v_E),$
$(v_E, v_C), (v_C, v_D) \}$

$Y = \{v_A, v_B, v_C, v_D, v_E\}$

$V\text{-}Y = \varnothing$

Puesto que Y es igual a V, se ha encontrado la solución

Algoritmo de Kruskal

S = ∅

Y = subconjuntos disjuntos de V, cada uno con cada uno de los vértices de V

Ordenar los arcos en A en forma ascendente de acuerdo a su peso.

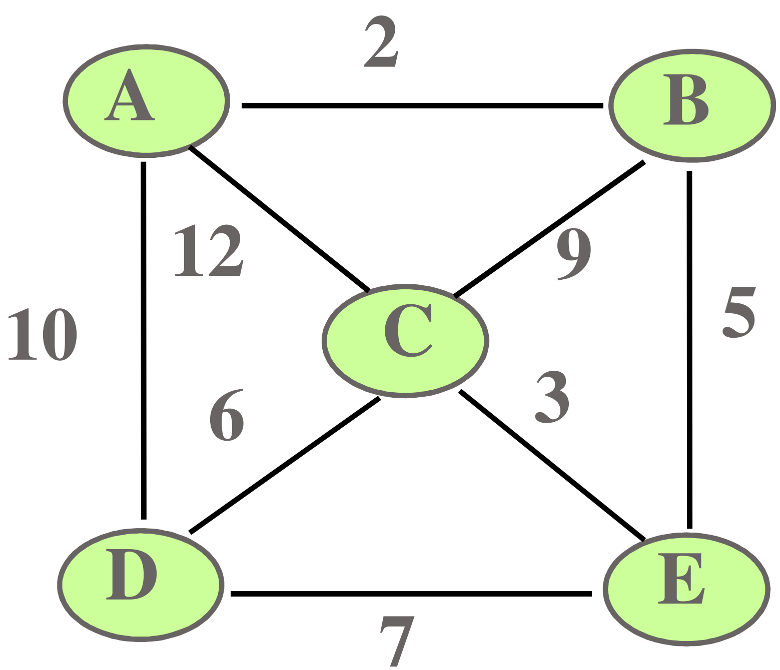
Mientras (no se haya resuelto el problema)

Seleccionar el siguiente arco de A.

Si el arco conecta 2 vértices de Y, unir los subconjuntos y añadir el arco a S.

Si todos los subconjuntos se han unido, el problema se ha resuelto.

Ejemplo



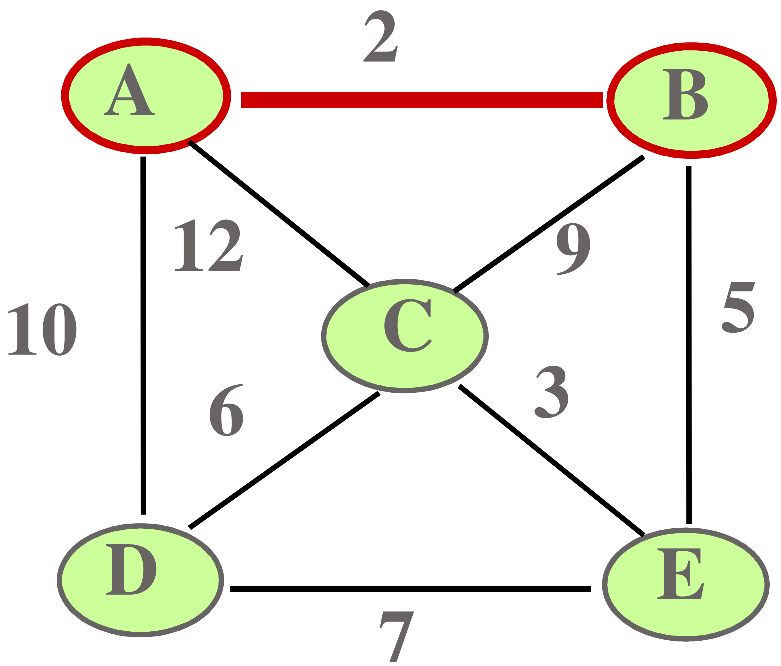
$S = \varnothing$

$Y = \{\{v_A\}\ \{v_B\}\ \{v_C\}\ \{v_D\}\ \{v_E\}\}$

$A = \{(v_A, v_B), (v_C, v_E), (v_B, v_E), (v_C, v_D), (v_D, v_E), (v_B, v_C), (v_A, v_D), (v_A, v_C)\}$

Seleccionar el arco de menor costo de A y si une a dos subconjuntos de Y, unirlos y agregar el arco a S

Ejemplo



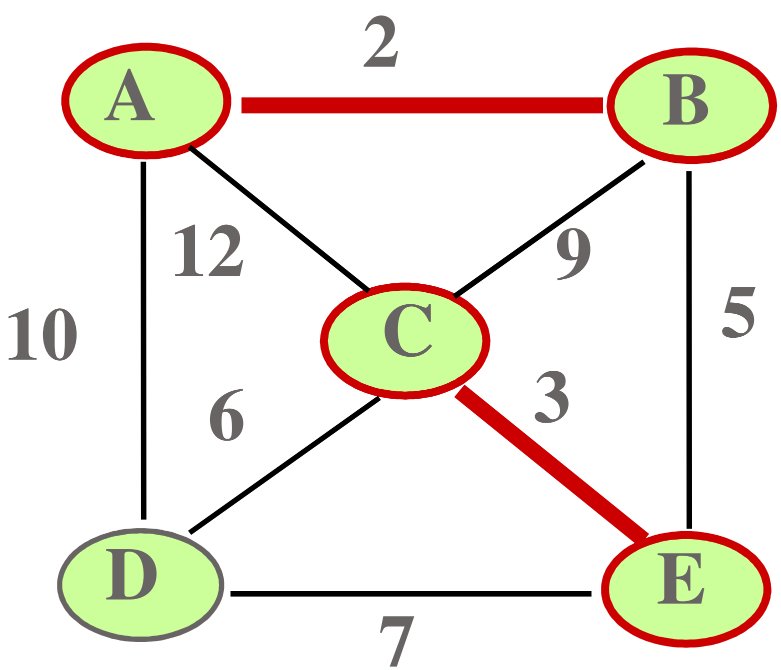
$S = \{(v_A, v_B)\}$

$Y = \{\{v_A, v_B\}\ \{v_C\}\ \{v_D\}\ \{v_E\}\}$

$A = \{(v_C, v_E), (v_B, v_E), (v_C, v_D), (v_D, v_E), (v_B, v_C), (v_A, v_D), (v_A, v_C)\}$

Seleccionar el arco de menor costo de A y si une a dos subconjuntos de Y, unirlos y agregar el arco a S

Ejemplo



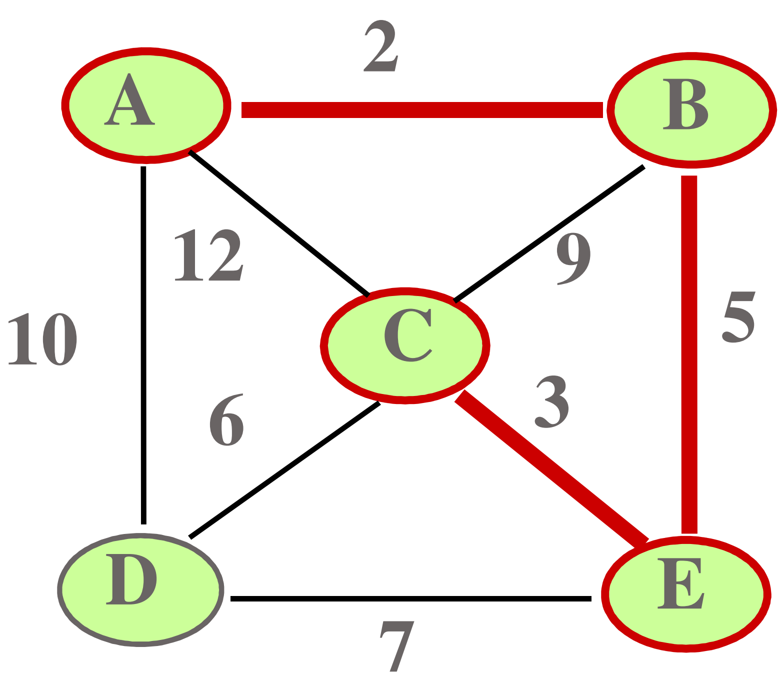
$S = \{(v_A, v_B), (v_C, v_E)\}$

$Y = \{\{v_A, v_B\}, \{v_C, v_E\} \{v_D\}\}$

$A = \{(v_B, v_E), (v_C, v_D), (v_D, v_E), (v_B, v_C), (v_A, v_D), (v_A, v_C)\}$

Seleccionar el arco de menor costo de A y si une a dos subconjuntos de Y, unirlos y agregar el arco a S

Ejemplo



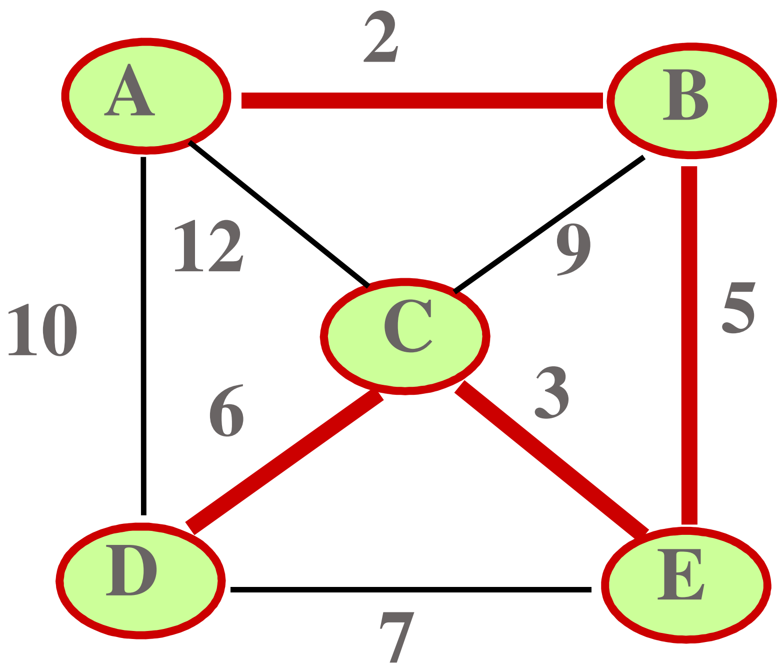
$S = \{(v_A, v_B), (v_C, v_E), (v_B, v_E)\}$

$Y = \{\{v_A, v_B, v_C, v_E\} \{v_D\}\}$

$A = \{(v_C, v_D), (v_D, v_E), (v_B, v_C), (v_A, v_D), (v_A, v_C)\}$

Seleccionar el arco de menor costo de A y si une a dos subconjuntos de Y, unirlos y agregar el arco a S

Ejemplo



$S = \{(v_A, v_B), (v_C, v_E), (v_B, v_E), (v_C, v_D)\}$

$Y = \{\{v_A, v_B, v_C, v_D, v_E\}\}$

$A = \{(v_D, v_E), (v_B, v_C), (v_A, v_D), (v_A, v_C)\}$

Puesto que Y sólo contiene un subconjunto con todos los vértices, S contiene la solución.

Implementación de los algoritmos

- Requieren de un tipo de dato conjunto, que permita trabajar con las operaciones de conjuntos (unión, diferencia, añadir).
- **Prim** se apoya en la matriz de transiciones, y en arreglos auxiliares *(ver detalle en libro)*.
- **Kruskal** requiere de la implementación del tipo de dato conjunto disjunto *(ver detalle en libro)*.
- Los **HEAPS** pueden ayudar a obtener otras versiones eficientes de implementación.

¿Cómo se comprueba que los algoritmos son correctos?

- En el caso de la **programación dinámica**, basta comprobar que se cumple el principio de optimalidad para saber que se tiene un algoritmo válido…
- En el caso de **divide y vencerás**, la recursividad está fundamentada, y comprueba la validez del algoritmo…
- En el caso de **algoritmos voraces**, se requiere una demostración matemática específica dependiendo del problema, y normalmente es más compleja *(ver demostración de Prim y Kruskal en libro).*

¿Cómo es el comportamiento de los algoritmos?

- **Prim** es un algoritmo con un comportamiento igual para todos los casos, que tiene una complejidad de tiempo de orden $\mathbf{O(n^2)}$, siendo ***n*** la cantidad de vértices en el grafo *(ver detalle en el libro)*.
- **Kruskal** es un algoritmo en el que influye la cantidad de arcos en su comportamiento. Su peor caso, tiene una complejidad de tiempo de orden $\mathbf{O(n^2 \log_2 n)}$ o $\mathbf{O(m \log_2 m)}$, siendo ***n*** la cantidad de vértices y ***m*** la cantidad de arcos *(ver detalle en el libro)*.

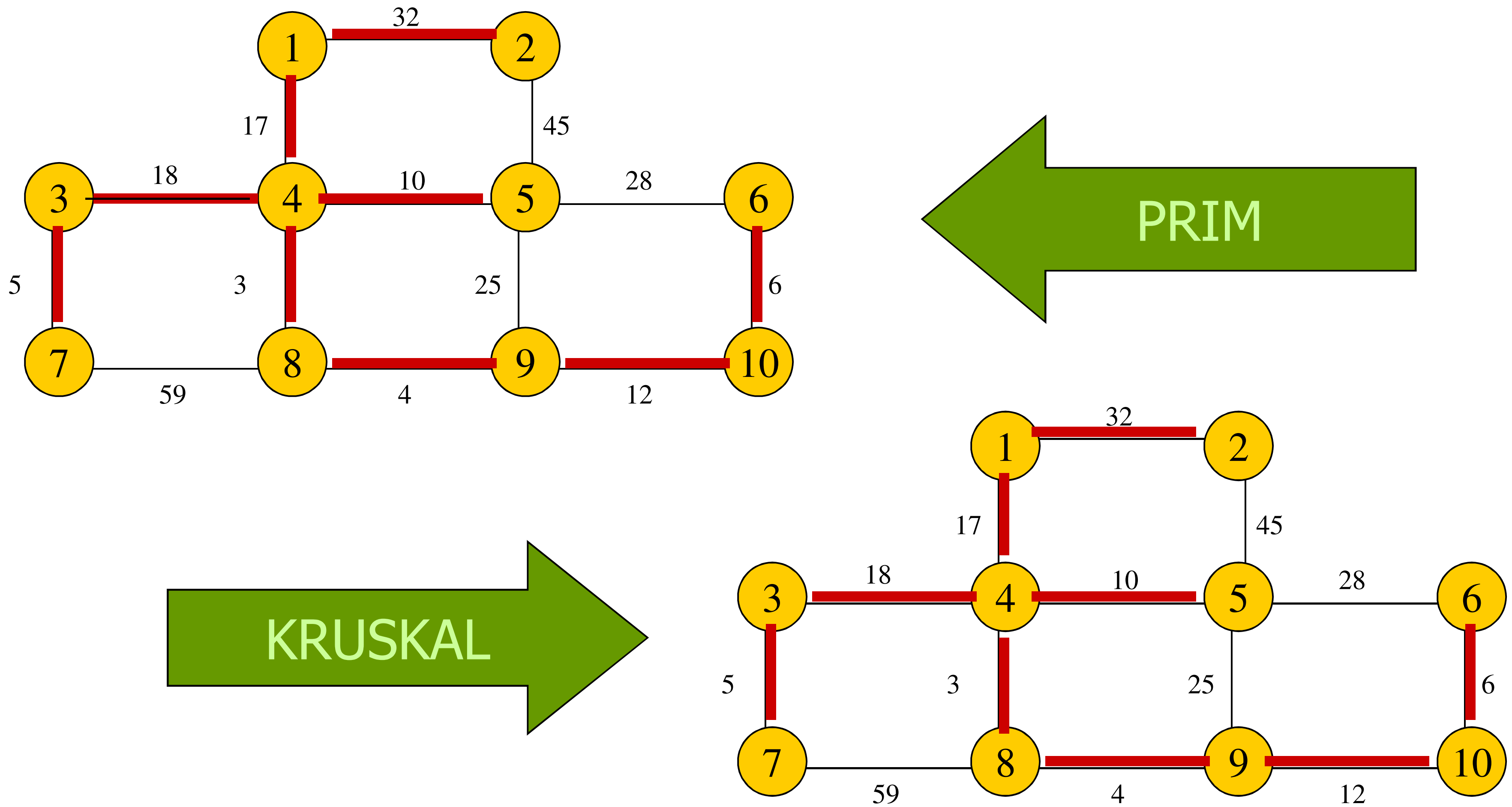
¿Cuál de los dos algoritmos es mejor utilizar?

- ¿Cuántos arcos puede tener un grafo no dirigido conectado de ***n*** vértices?
  - MINIMO: ***n - 1*** arcos
  - MAXIMO: ***n (n - 1) / 2*** arcos

  Prim $O(n^2)$ ***vs. Kruskal*** $O(m \log_2 m)$

- Por lo tanto, para un grafo con pocos arcos, **Kruskal** resultará más eficiente, y
- para un grafo muy denso (altamente conectado), **Prim** funcionará mejor...

EJEMPLO



El problema del camino más corto

- ¿Qué pasaría si en determinada aplicación se requiere conocer el camino más corto de un vértice a otro?
- El ***Algoritmo de Floyd*** obtiene el camino más corto de TODOS los vertices hacia TODOS los vértices y tiene un comportamiento de **O(n^3)**…
- Si sólo se requiere el análisis para un sólo camino, ¿podría hacerse de una manera más eficiente?…
- **SI**… la propuesta del ***Algoritmo de Dijkstra*** resuelve el problema en **O(n^2)**.

Algoritmo de Dijkstra

$S = \varnothing$

$Y = \{v_1\}$

Mientras (no se haya resuelto el problema)

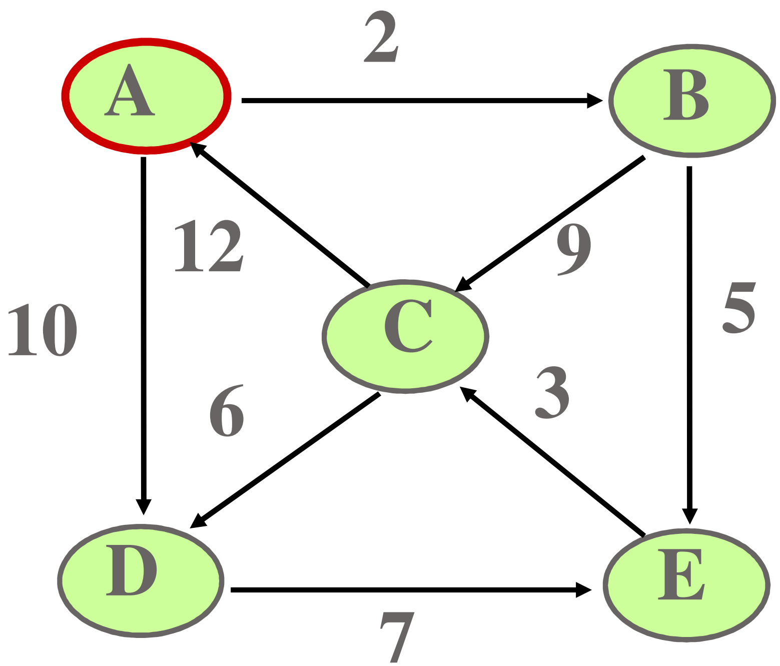
Seleccionar el vértice de V-Y que tenga el camino más corto desde v_1 usando sólo a los vértices en Y como intermediarios.

Agregar el vértice a Y.

Agregar el arco que llega a al vértice seleccionado a S.

Si $Y = V$ el problema se ha resuelto.

Ejemplo



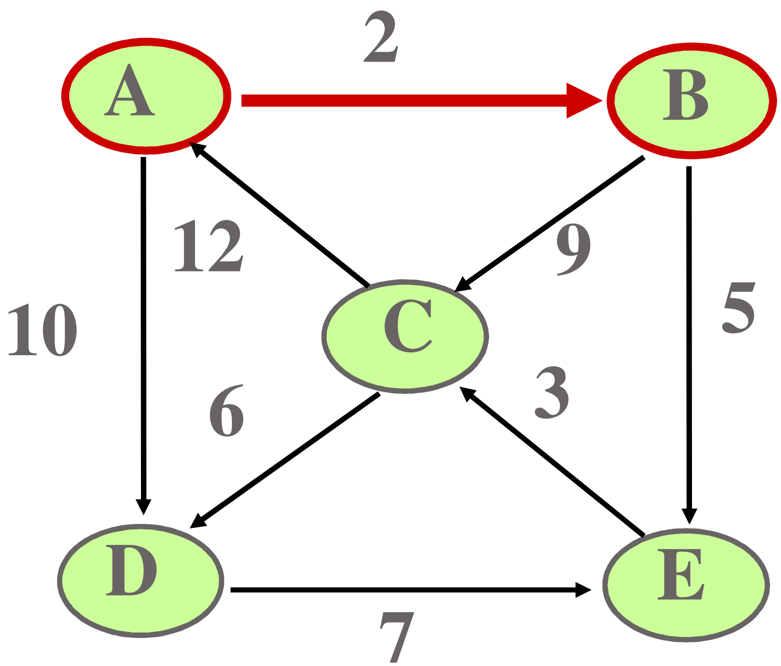
$S = \varnothing$

$Y = \{v_A\}$

$V\text{-}Y = \{v_B, v_C, v_D, v_E\}$

Seleccionar el vértice de V-Y que tenga el camino más corto desde v_A*, pasando por los vértices de Y*

Ejemplo



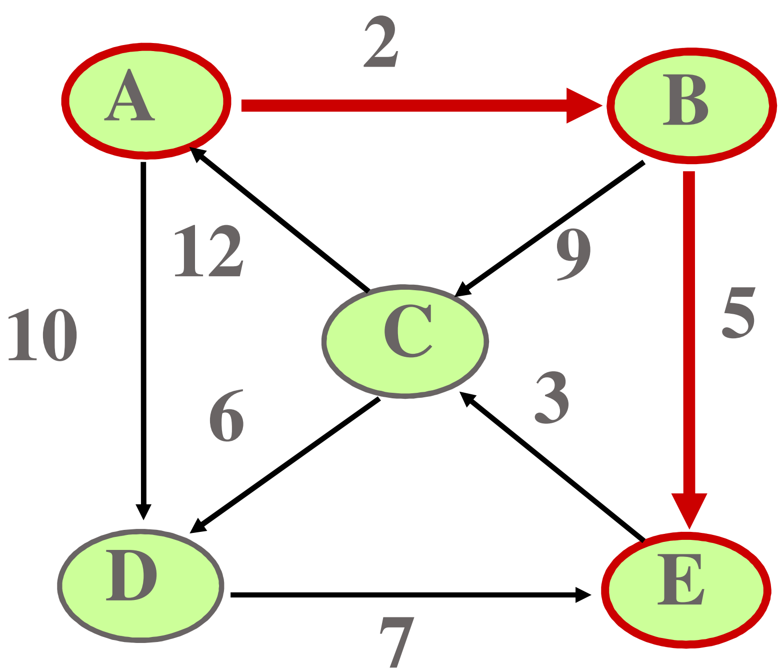
$S = \{(v_A, v_B)\}$

$Y = \{v_A, v_B\}$

$V\text{-}Y = \{v_C, v_D, v_E\}$

Seleccionar el vértice de V-Y que tenga el camino más corto desde v_A, pasando por los vértices de Y

Ejemplo



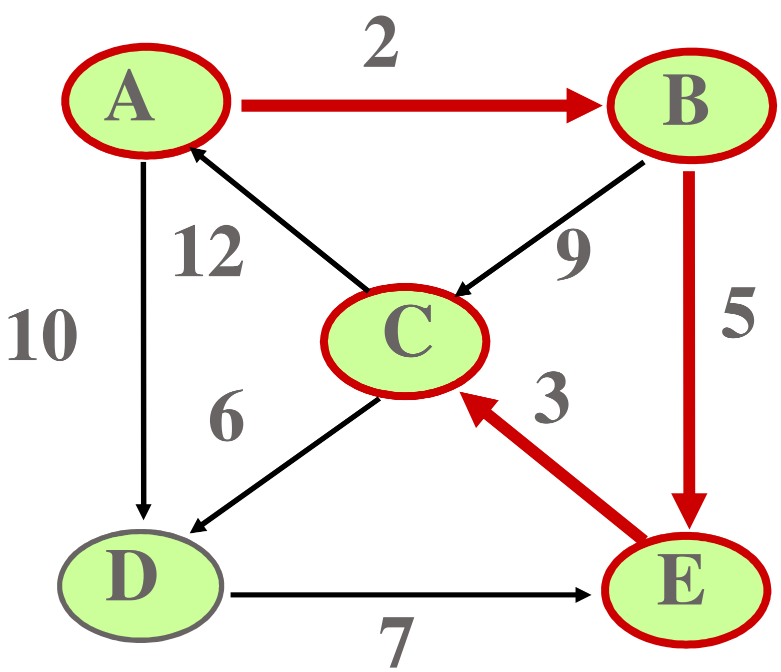
$S = \{(v_A, v_B),(v_B, v_E)\}$

$Y = \{v_A, v_B, v_E\}$

$V\text{-}Y = \{v_C, v_D\}$

Seleccionar el vértice de V-Y que tenga el camino más corto desde v_A, pasando por los vértices de Y

Ejemplo



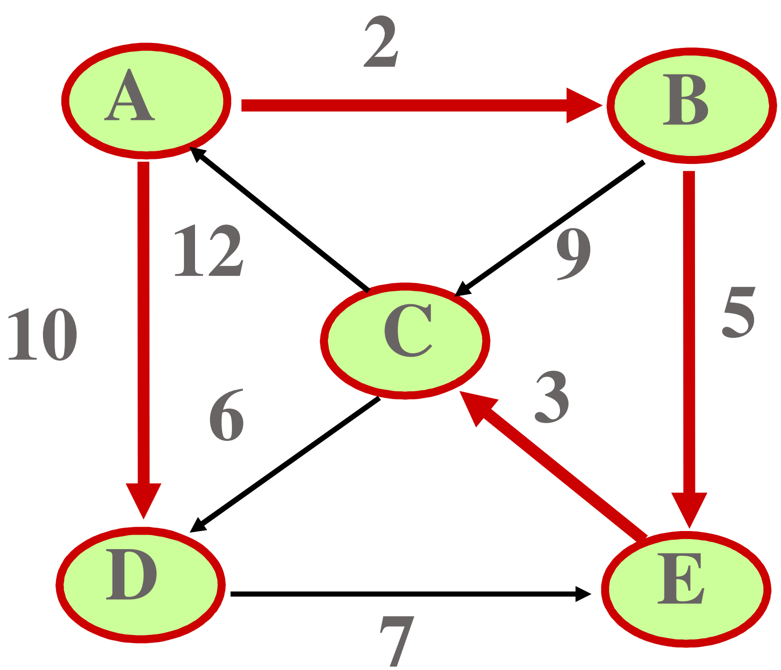
$S = \{(v_A, v_B), (v_B, v_E), (v_E, v_C)\}$

$Y = \{v_A, v_B, v_C, v_E\}$

$V\text{-}Y = \{v_D\}$

Seleccionar el vértice de V-Y que tenga el camino más corto desde v_A, pasando por los vértices de Y

Ejemplo



$S = \{(v_A, v_B), (v_B, v_E), (v_E, v_C), (v_A, v_D)\}$

$Y = \{v_A, v_B, v_C, v_D, v_E\}$

$V\text{-}Y = \varnothing$

Puesto que Y es igual a V, se ha encontrado la solución

Implementación del Algoritmo de Dijkstra

- Utiliza a la matriz de adyacencias del grafo (W).
- Se auxilia de un arreglo **L**, indexado de 2 a *n*, en donde guardará la longitud de los caminos más cortos del vértice v_1 al vértice v_i, usando sólamente a los vértices del conjunto Y como intermediarios.
- Se auxilia de un arreglo **T**, indexado de 2 a *n*, en donde guardará el índice del vértice v, cuyo arco (v,v_i) es el último arco en el camino más corto de v_1 a v_i usando solamente a los vértices del conjunto Y como intermediarios.

Algoritmo de Dijkstra

$S = \varnothing;$

for (i = 2; i <=n; i++)
{ L[i] = W[1][i];
T[i] = 1; }

Inicializa los arreglos auxiliares:
L con los caminos directos a partir de v_1
T con v_1 pues no se ha pasado por otro vértice

Repetir n-1 veces: //para incluir los vértices en Y

{ min = ∞;
for (i = 2; i<=n; i++)
if (0 <= L[i] <= min)
{ min = L[i]; vmin = i; }

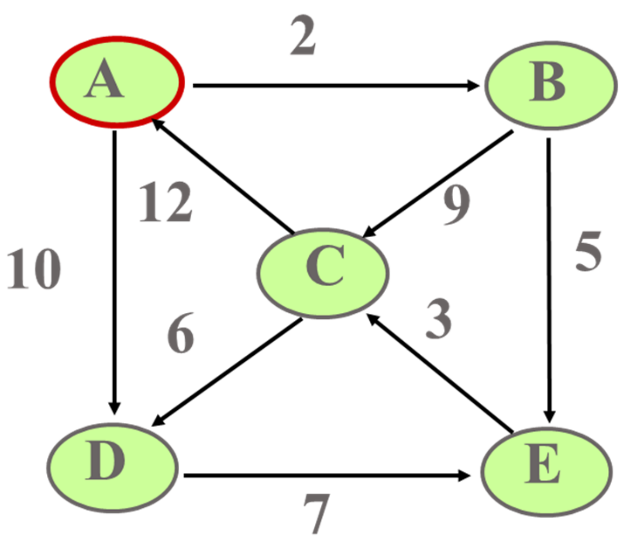
Busca el menor de los caminos más cortos a partir de v_1, descartando a los ya alcanzados

continua...

Algoritmo de Dijkstra

```
e = arco formado por T[vmin] y vmin;
Añadir e a S;
for (i=2; i<=n; i++)
  if (L[vmin]+W[vmin][i] < L[i])
 {    L[i] = L[vmin]+W[vmin][i];
      T[i] = vmin; }
L[vmin] = -1; //control para que ya no se considere en la búsqueda del menor
}
```

Ejemplo



$S = \varnothing;$

$for\ (i = 2;\ i <= n;\ i++)$

$\{\ L[i] = W[1][i];$

$T[i] = 1;\ \}$

	1 (A)	2 (B)	3 (C)	4 (D)	5 (E)
1 (A)	0	2	∝	10	∝
2 (B)	∝	0	9	∝	5
3 (C)	12	∝	0	6	∝
4 (D)	∝	∝	∝	0	7
5 (E)	∝	∝	3	∝	0

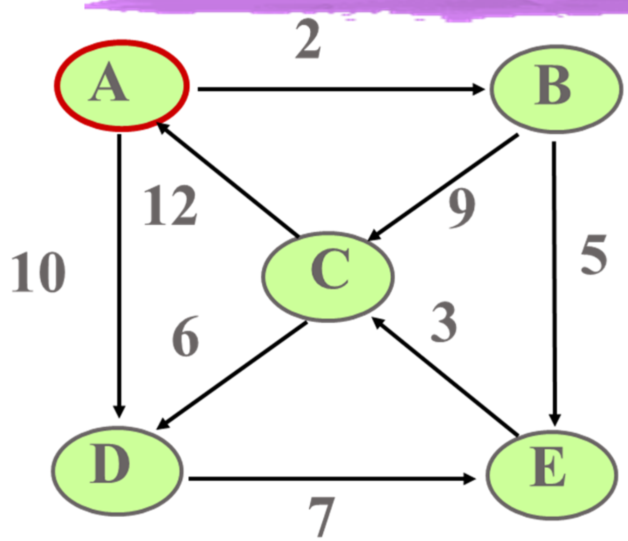
S = { }

Los arreglos L y T se muestran a partir de la posición 2:

L = [2, ∝, 10, ∝]

T = [1, 1, 1, 1]

Ejemplo



	1 (A)	2 (B)	3 (C)	4 (D)	5 (E)
1 (A)	0	2	∝	10	∝
2 (B)	∝	0	9	∝	5
3 (C)	12	∝	0	6	∝
4 (D)	∝	∝	∝	0	7
5 (E)	∝	∝	3	∝	0

Repetir n-1 veces: *//para incluir los vértices en Y*

```
{   min = ∞;
    for (i = 2; i<=n; i++)
      if ( 0 <= L[i] <= min)
      {  min = L[i];  vmin = i; }
    e = arco formado por T[vmin] y vmin;
    Añadir e a S;
    for (i=2; i<=n; i++)
      if (L[vmin]+W[vmin][i] < L[i])
     {      L[i] = L[vmin]+W[vmin][i];
            T[i] = vmin; }
    L[vmin] = -1; //control para que ya no se considere en la
    búsqueda del menor
}
```

S = { (A, B) }

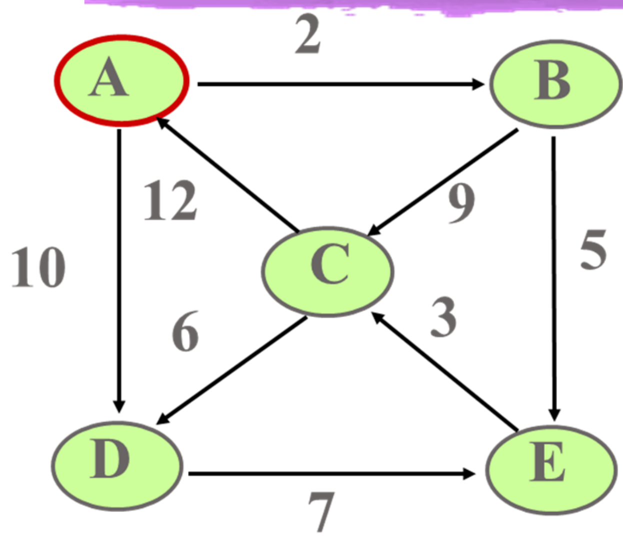
L = [2, ∝, 10, ∝]

L = [-1, 11, 10, 7]

T = [1, 1, 1, 1]

T = [1, 2, 1, 2]

Ejemplo



	1 (A)	2 (B)	3 (C)	4 (D)	5 (E)
1 (A)	0	2	∝	10	∝
2 (B)	∝	0	9	∝	5
3 (C)	12	∝	0	6	∝
4 (D)	∝	∝	∝	0	7
5 (E)	∝	∝	3	∝	0

Repetir n-1 veces: *//para incluir los vértices en Y*

{ min = ∞;

for (i = 2; i<=n; i++)

if (0 <= L[i] <= min)

{ min = L[i]; vmin = i; }

e = arco formado por T[vmin] y vmin;

Añadir e a S;

for (i=2; i<=n; i++)

if (L[vmin]+W[vmin][i] < L[i])

{ L[i] = L[vmin]+W[vmin][i];

T[i] = vmin; }

L[vmin] = -1; *//control para que ya no se considere en la búsqueda del menor*

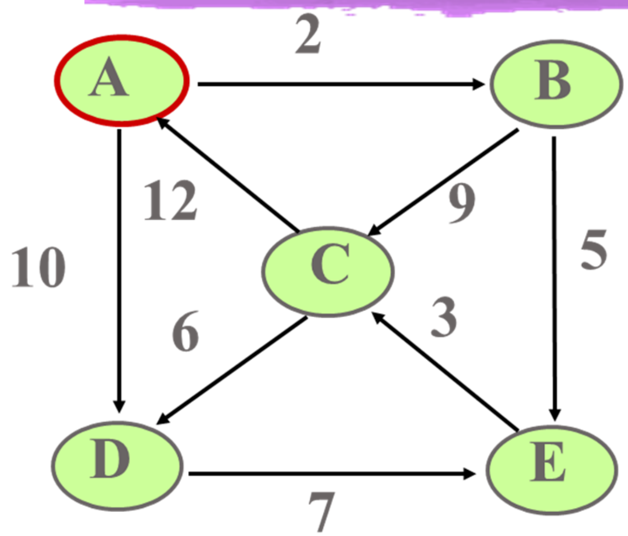
}

S = { (A,B),(B,E) }

L = [-1, 11, 10, 7] **T = [1, 2, 1, 2]**

L = [-1, 10, 10, -1] **T = [1, 5, 1, 2]**

Ejemplo



	1 (A)	2 (B)	3 (C)	4 (D)	5 (E)
1 (A)	0	2	∝	10	∝
2 (B)	∝	0	9	∝	5
3 (C)	12	∝	0	6	∝
4 (D)	∝	∝	∝	0	7
5 (E)	∝	∝	3	∝	0

Repetir n-1 veces: *//para incluir los vértices en Y*

{ min = ∞;

for (i = 2; i<=n; i++)

if (0 <= L[i] <= min)

{ min = L[i]; vmin = i; }

e = arco formado por T[vmin] y vmin;

Añadir e a S;

for (i=2; i<=n; i++)

if (L[vmin]+W[vmin][i] < L[i])

{ L[i] = L[vmin]+W[vmin][i];

T[i] = vmin; }

L[vmin] = -1; *//control para que ya no se considere en la búsqueda del menor*

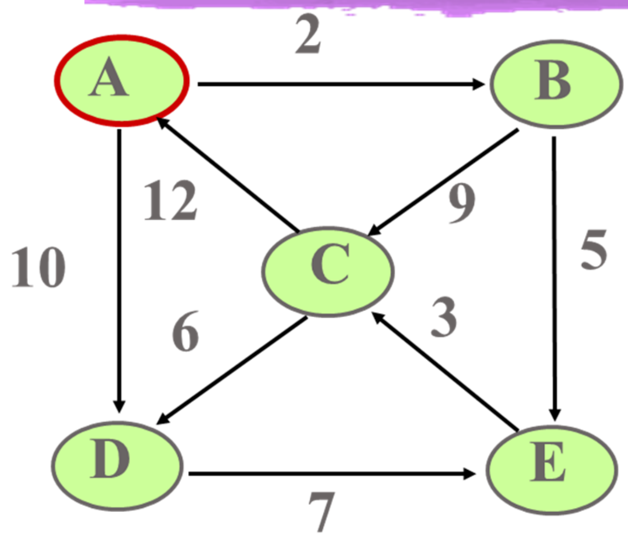
}

S = { (A,B),(B,E),(E,C) }

L = [-1, 10, 10, -1] **T = [1, 5, 1, 2]**

L = [-1, -1, 10, -1] **T = [1, 5, 1, 2]**

Ejemplo



	1 (A)	2 (B)	3 (C)	4 (D)	5 (E)
1 (A)	0	2	∝	10	∝
2 (B)	∝	0	9	∝	5
3 (C)	12	∝	0	6	∝
4 (D)	∝	∝	∝	0	7
5 (E)	∝	∝	3	∝	0

```
Repetir n-1 veces:   //para incluir los vértices en Y
{    min = ∞;
     for (i = 2; i<=n; i++)
       if ( 0 <= L[i] <= min)
       {  min = L[i];  vmin = i; }
     e = arco formado por T[vmin] y vmin;
     Añadir e a S;
     for (i=2; i<=n; i++)
       if (L[vmin]+W[vmin][i] < L[i])
      {       L[i] = L[vmin]+W[vmin][i];
              T[i] = vmin; }
     L[vmin] = -1; //control para que ya no se considere en la
     búsqueda del menor
}
```

S = { (A,B),(B,E),(E,C),(A,D) }

L = [-1, -1, 10, -1]   **T = [1, 5, 1, 2]**

L = [-1, -1, -1, -1]   **T = [1, 5, 1, 2]**