

Semana 8

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

09-2021

Travelling Salesman Problem (TSP)

Usando Ramificación y Poda

Minimum Spanning Tree

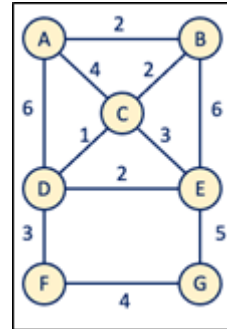
Algoritmo de Prim

Algoritmo de Kruskal

Travelling Salesman Problem (TSP)

El problema del agente viajero es que, dado un grafo ponderado y no dirigido, se deberá encontrar el ciclo Hamiltoniano con el menor costo, es decir, salir de un vértice, visitar todos los vértices restantes una sola vez y regresar al mismo punto de partida, con el menor costo.

- ▶ A-B-C-E-G-F-D-A con un costo de 25.
- ▶ A-B-E-G-F-D-C-A con un costo de 25.
- ▶ A-C-B-E-G-F-D-A con un costo de 30.
- ▶ A-C-D-F-G-E-B-A con un costo de 25.
- ▶ A-D-F-E-G-B-C-A con un costo de 30.
- ▶ A-D-F-E-G-C-B-A con un costo de 25.



Usando Ramificación y Poda

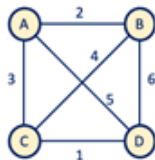
El problema de la Viajero al ser un problema de Selección con Optimización cumple con los requisitos para poder ser resuelto con la técnica de Branch & Bound. El árbol de búsqueda de soluciones tendrá n niveles, y cada nivel se trabajará un nodo a conectar, y cada nodo tendrá $n - i - 1$ hijos donde i es el nivel en que va, diciendo los posibles nodos que falten en esa ruta. Los nodos almacenarán el costo acumulado hasta el momento, así como su costo posible, el último nodo visitado, y el nodo actual. La idea es trabajar en anchura con Best-First, esto es se maneja una fila priorizada con prioridad menor costo posible y se saca al mejor y se generan sus hijos.

El cálculo del costo posible se realiza de la siguiente forma:

- ▶ El costo acumulado, más ...
- ▶ Del último vértice alcanzado, escoger el menor costo con los vértices faltantes, más ...
- ▶ Por cada vértice faltante, escoger el menor costo entre ellos y llegar al vértice inicial.

El criterio de selección del nodo será:

- ▶ Si el vértice en el nivel i del árbol, debe ser adyacente al vértice en el nivel $i - 1$ del camino correspondiente en el árbol.
- ▶ Si el costo posible a acumular al expandir el nodo i , es menor al mejor costo acumulado hasta ese momento.



	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0

Costo óptimo = INF



1



Costo posible.

A -> min {B, C, D} = 2

B -> min {A, C, D} = 2

C -> min {A, B, D} = 1

D -> min {A, B, C} = 1

Total = 6

Adyacencia && Cpos > Costo óptimo

Costo óptimo = INF



2



Costo posible.

A-B = 2

B -> min {C, D} = 4

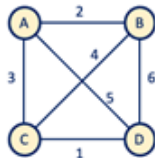
C -> min {A, D} = 1

D -> min {A, C} = 1

Total = 8

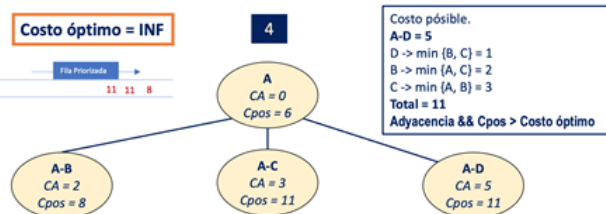
Adyacencia && Cpos > Costo óptimo



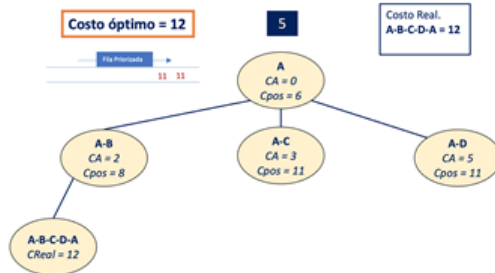


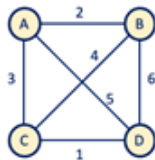
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0

Costo óptimo = INF



Costo óptimo = 12





	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0

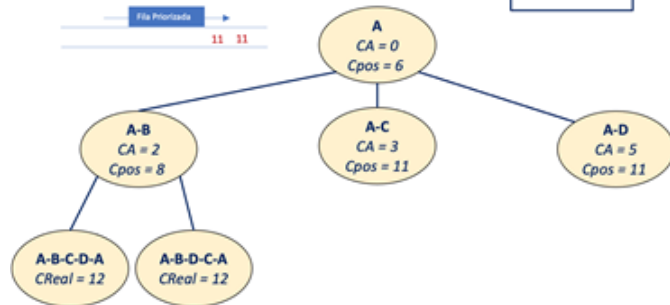
Costo óptimo = 12

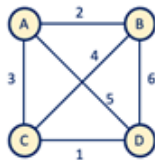
Fila Priorizada

11 11

6

Costo Real.
A-B-D-C-A = 12





	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0

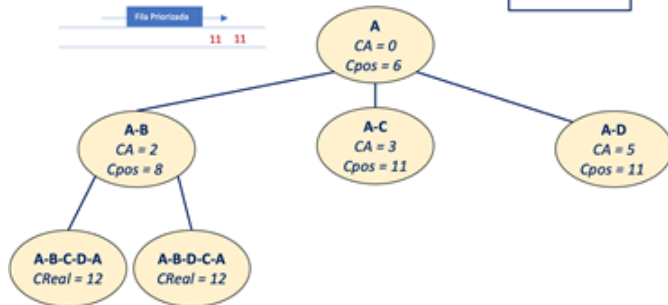
Costo óptimo = 12

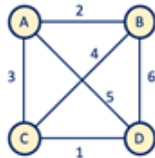
Fila Priorizada

11 11

6

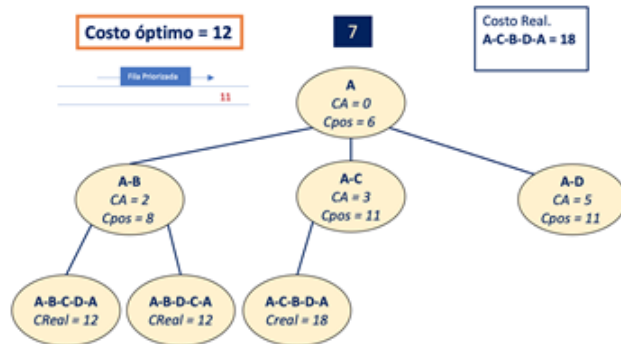
Costo Real.
A-B-D-C-A = 12

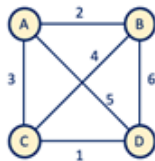




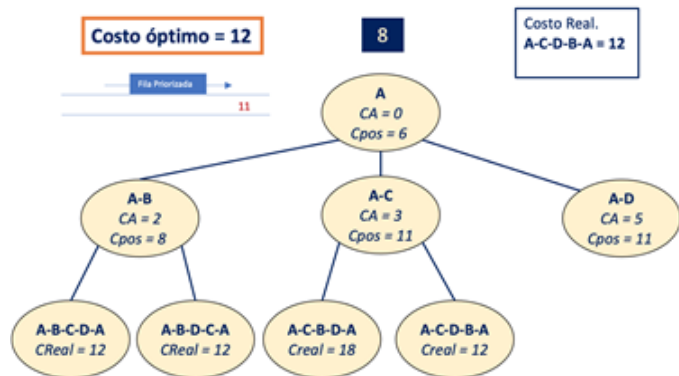
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0

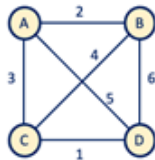
Costo óptimo = 12





	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



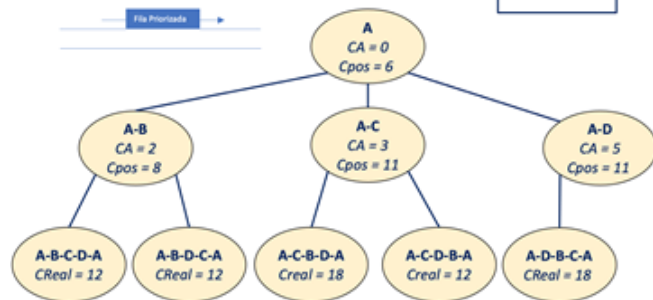


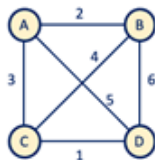
	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0

Costo óptimo = 12

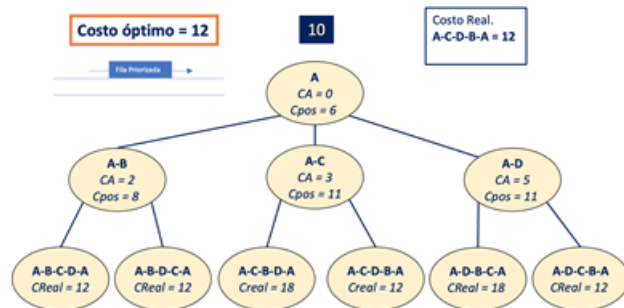
9

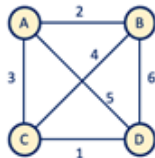
Costo Real.
A-D-B-C-A = 18





	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0



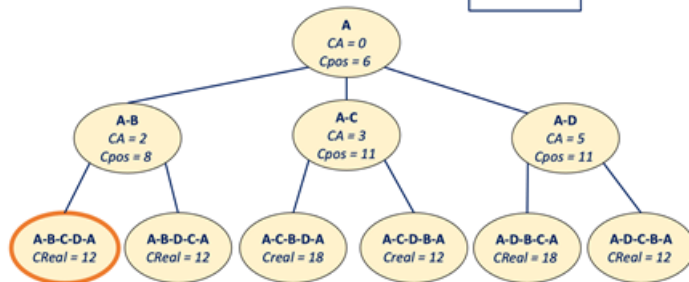


	A	B	C	D
A	0	2	3	5
B	2	0	4	6
C	3	4	0	1
D	5	6	1	0

Costo óptimo = 12

FINAL

Costo Real.
A-B-C-D-A = 12



Procedure 1 TSP

Input: $G : \text{Graph}$

Create a priority queue of nodes, where the priority is lowest possible cost.

Calculate the possible initial cost and put it in the priority queue.

while the priority queue is not empty **do**

if is the possible cost less than optimal cost? **then**

for each edge(u, v) coming out of v **do**

if u is not visited **then**

if is it the last connection? **then**

 Calculate actual accumulated value

 If the new real cost is better than the optimal cost, it is updated.

else

 Put a new node with u in the priority queue.

end if

end if

end for

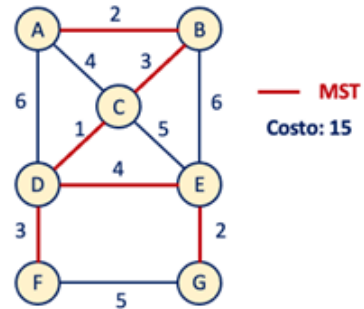
end if

end while

return the optimal value

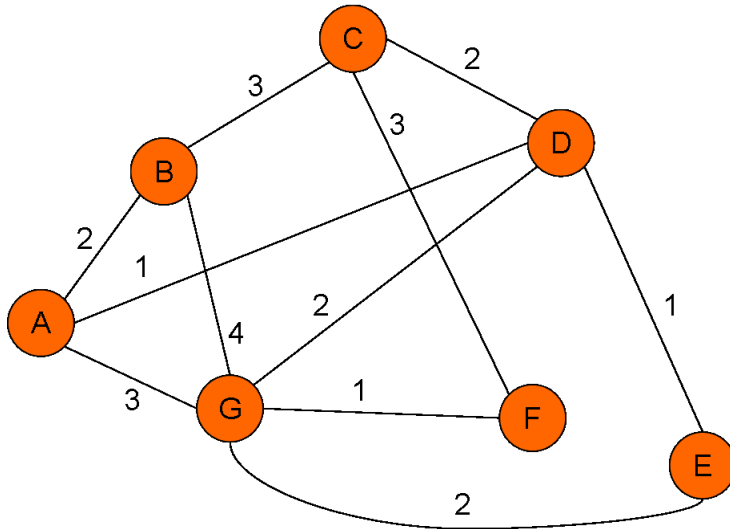
Minimum Spanning Tree

El problema del árbol de mínima expansión es un conjunto de arcos (la cantidad de vértices menos 1 arcos) de un grafo no dirigido, ponderado y conectado, que conecta a todos los vértices juntos, sin ciclos y con el mínimo suma de pesos de los arcos.



Algoritmo de Prim

- ▶ Este algoritmo es un algoritmo voraz que fue diseñado en 1930 por Vojtech Jarník, luego de forma independiente fue diseñado en 1957 por Robert Prim y redescubierto por Dijkstra en 1959, es por esta razón que también es conocido como algoritmo DJP o algoritmo de Jarník.
- ▶ El algoritmo empieza por un vértice como parte de un conjunto solución y va seleccionando el arco con menos costo que vaya de uno de los vértices que están en el conjunto solución a algún vértice faltante por conectar almacenando este arco en un conjunto de arcos de la solución. Cuando en el conjunto solución se encuentran los n vértices, se encontró el árbol de mínima expansión (MST).



Algoritmo de Kruskal

- Este algoritmo es un algoritmo voraz que fue publicado en 1956 por Joseph Kruskal. El algoritmo empieza ordenando los arcos en forma ascendente según su costo, y va seleccionando de menor a mayor verificando que al conectarlo no forma ciclo, en este caso lo integra como parte de la solución, en caso contrario lo desecha, cuando logra llegar a todos los vértices termina su ejecución. Utiliza un conjunto disjunto (Union-Find) para comprobar si hay o no ciclo.

