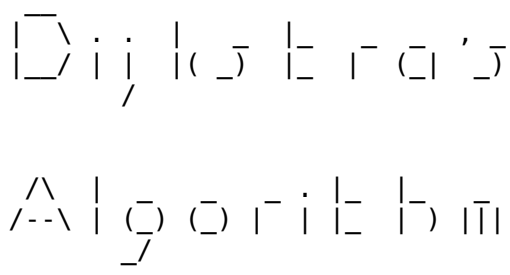
"Análisis de Algoritmos"
José Helo Guzmán



Contenido

▬▬▬▬ 1. Introducción.

Continuamos nuestra discusión sobre grafos. El algoritmo de Dijkstra, nos ayudará a encontrar el camino de menor costo entre dos nodos cualesquiera escogiendo siempre la ruta más corta.

▬▬▬▬ 2. Variantes del Algoritmo de Dijkstra.

Deseamos resolver el problema de la ruta más corta entre dos nodos.

Para resolver este problema existen muchas variantes, cada una con diferentes condiciones iniciales.

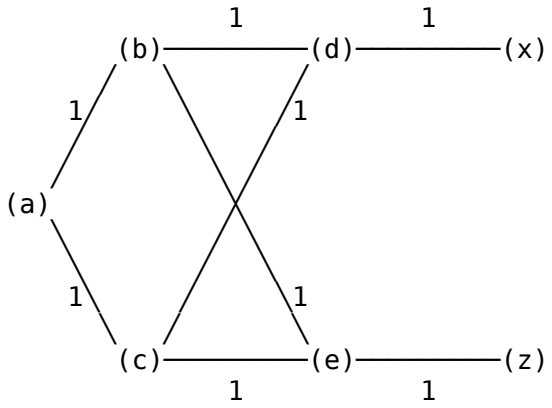
En particular nosotros vamos a resolver el problema para:

• Grafos con pesos mayores o iguales que cero.

• Los arcos pueden ser dirigidos o no dirigidos y pueden contener con ciclos.

▬▬▬ 3. Descripción del Grafo Sin Pesos.

Supongamos que tenemos el siguiente grafo con sus arcos no dirigidos:



Observemos como existe un peso en cada uno de los arcos.
Y existen ciclos en el grafo.

Para describir este grafo se utilizará la siguiente estructura, donde aparece al inicio un nodo, y luego una lista con los nodos con los que se conecta y el costo del pero de cada uno de esos nodos.

V = {a,b,c,d,e,x,z}

E:
[
[a, [[b,1], [c,1]]],
[b, [[a,1], [d,1], [e,1]]],
[c, [[a,1], [d,1], [e,1]]],
[d, [[b,1], [c,1], [x,1]]],
[e, [[b,1], [c,1], [z,1]]],
[x, [[d,1]]],
[z, [[e,1]]]
];

Como todos los pesos de los arcos tienen un valor de “1”, podemos eliminar ese valor de nuestra representación.

V = {a,b,c,d,e,x,z}

E:
[
[a, [b,c]],
[b, [a,d,e]],

```
[c, [a,d,e]],
[d, [b,c,x]],
[e, [b,c,z]],
[x, [d]],
[z, [e]]
];
```

▬▬▬▬ 4. Algoritmo de Dijkstra.

El algoritmo que vamos a realizar es una modificación del BFS. Funciona de la siguiente manera:

1. De la lista de rutas listaRutas Escoger la ruta de menor costo R.

2. Si R es una solución nos detenemos.

3. En caso contrario:

   Extendemos R y agregamos a listaRutas

   Continuamos el proceso.

▬▬▬▬ 5. Una Corrida.

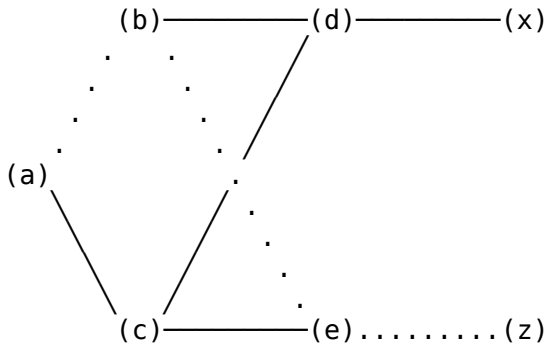
Veamos una corrida del algoritmo.
Estamos buscando la ruta de “a” hacia “z”.

```
(%i) dijkstra(a,z,grafo);
-----

[
[a]
]

-----

[
[a,b]
[a,c]
]

-----

[
[a,b,d]
[a,b,e]
[a,c]
]

-----

[
[a,c,d]
[a,c,e]
```

```
[a,b,d]
[a,b,e]
]

-----

[
[a,c,d,b]
[a,c,d,x]
[a,c,e]
[a,b,d]
[a,b,e]
]

-----

[
[a,c,e,b]
[a,c,e,z]
[a,c,d,b]
[a,c,d,x]
[a,b,d]
[a,b,e]
]

-----

[
[a,b,d,c]
[a,b,d,x]
[a,c,e,b]
[a,c,e,z]
[a,c,d,b]
[a,c,d,x]
[a,b,e]
]

-----

[
[a,b,e,c]
[a,b,e,z]
[a,b,d,c]
[a,b,d,x]
[a,c,e,b]
[a,c,e,z]
[a,c,d,b]
[a,c,d,x]
]

-----

[
[a,b,e,c,d]
[a,b,e,z]
[a,b,d,c]
[a,b,d,x]
[a,c,e,b]
[a,c,e,z]
[a,c,d,b]
[a,c,d,x]
]

-----

(%o) [a,b,e,z]
```

▬▬▬▬ 6. Solución Encontrada.

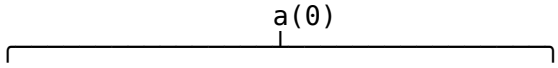
Observe como se obtiene una solución con un largo de 4.



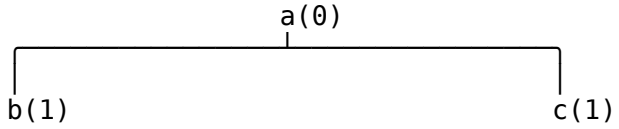
▬▬▬▬ 7. Construcción del Árbol de Soluciones.

Podemos ver como el árbol de soluciones se construye desarrollando siempre la rama con el menor valor. Iniciamos con el nodo “a”.

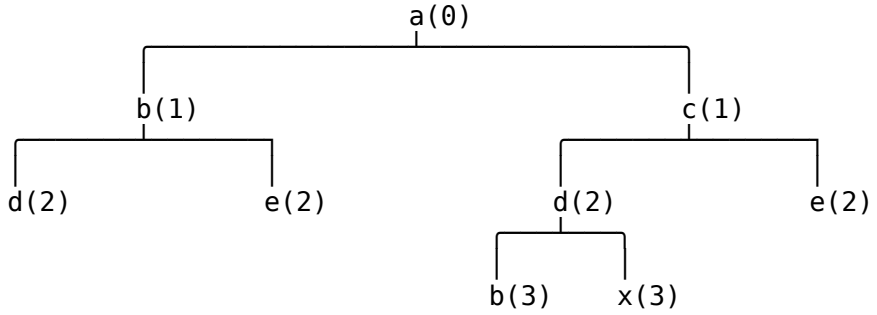
>>> Inicmos con “a”:



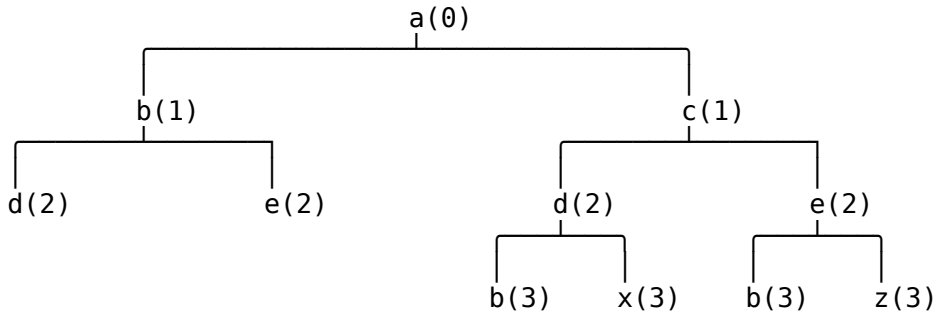
>>> Se desarrolla el nodo:



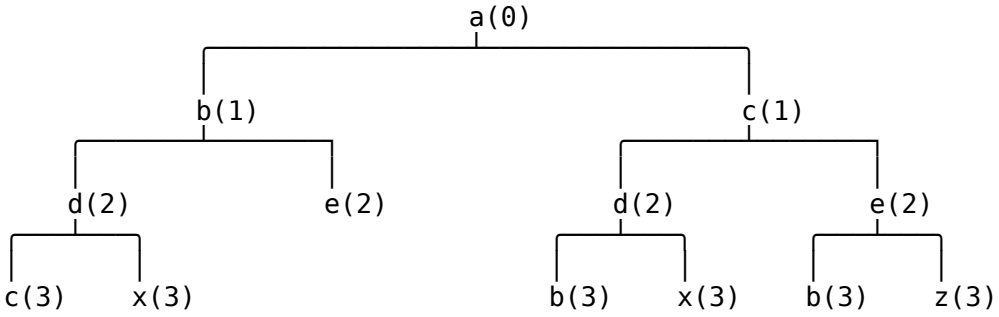
>>> Se desarrolla el nodo menor:



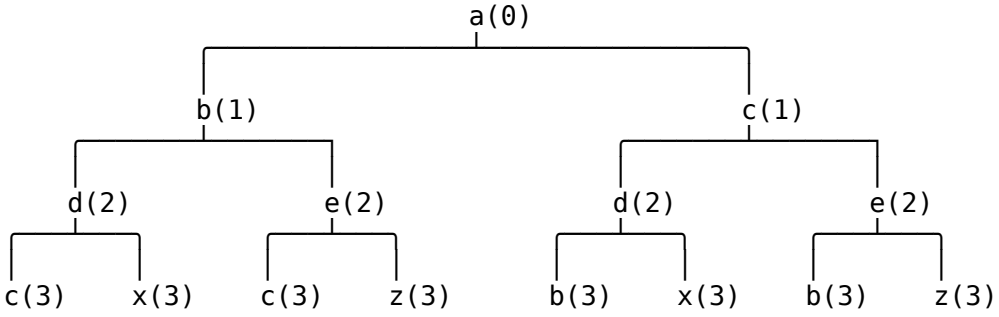
>>> Se desarrolla el nodo menor:



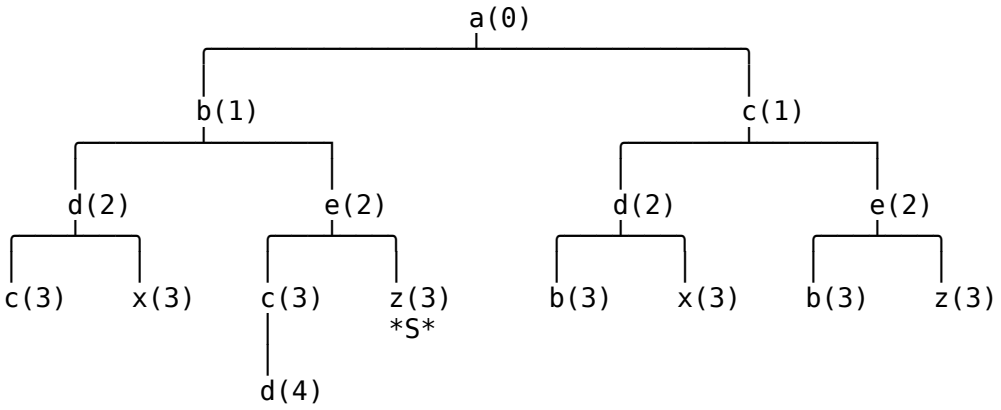
>>> Se desarrolla el nodo menor:



>>> Se desarrolla el nodo menor:



>>> Se desarrolla el nodo menor:

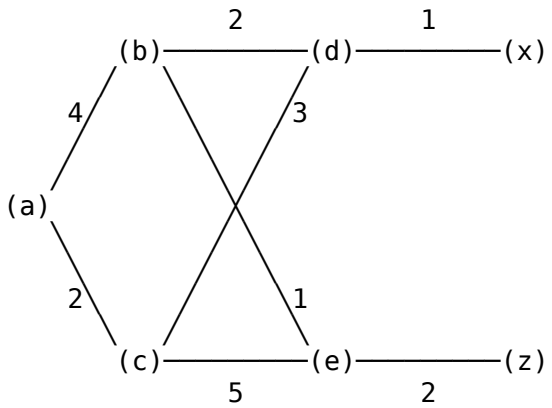


>>> Y así se obtiene la solución.

[a,b,e,z]

▬▬▬ 8. Descripción del Grafo con Pesos.

Supongamos que tenemos el siguiente grafo:



Observemos como existe un peso en cada uno de los arcos.
Y existen ciclos en el grafo.

Para describir este grafo se utilizará la siguiente estructura, donde aparece al inicio un nodo, y luego una lista con los nodos con los que se conecta y el costo del pero de cada uno de esos nodos.

```
V = {a,b,c,d,e,x,z}

E:
[
[a, [ [b,4], [c,2] ]],
[b, [ [a,4], [d,2], [e,1] ]],
[c, [ [a,2], [d,3], [e,5] ]],
[d, [ [b,2], [c,3], [x,1] ]],
[e, [ [b,1], [c,5], [z,2] ]],
[x, [ [d,1] ]],
[z, [ [e,2] ]]
];
```

▬▬▬ 9. Una Corrida.

Veamos una corrida del algoritmo.
Estamos buscando la ruta de “a” hacia “z”.

```
-----

[
[[a],0]
]

-----

[
[[a,b],4]
[[a,c],2]
]

-----

[
[[a,c,d],5]
[[a,c,e],7]
[[a,b],4]
]

-----

[
[[a,b,d],6]
[[a,b,e],5]
[[a,c,d],5]
[[a,c,e],7]
]

-----

[
[[a,b,e,c],10]
[[a,b,e,z],7]
[[a,b,d],6]
[[a,c,d],5]
[[a,c,e],7]
]
```

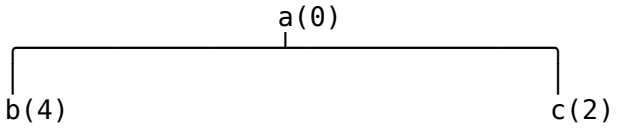
```
-----

[
[[a,c,d,b],7]
[[a,c,d,x],6]   --> Se elimina!
[[a,b,e,c],10]
[[a,b,e,z],7]
[[a,b,d],6]
[[a,c,e],7]
]

-----

[
[[a,c,d,b],7]
[[a,b,e,c],10]
[[a,b,e,z],7]
[[a,b,d],6]
[[a,c,e],7]

]

-----

[
[[a,b,d,c],9]
[[a,b,d,x],7]  --> Se elimina!
[[a,c,d,b],7]
[[a,b,e,c],10]
[[a,b,e,z],7]
[[a,c,e],7]

]

-----

[
[[a,b,d,c],9]
[[a,c,d,b],7]
[[a,b,e,c],10]
[[a,b,e,z],7]
[[a,c,e],7]
]

-----

[
[[a,c,d,b,e],8]
[[a,b,d,c],9]
[[a,b,e,c],10]
[[a,b,e,z],7]
[[a,c,e],7]
]

-----

(%o) [[a,b,e,z],7]
```

▬▬▬ 10. Construcción del Árbol de Soluciones.

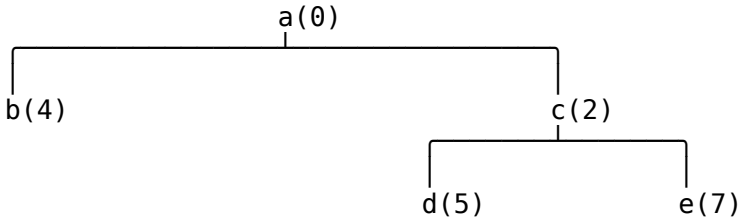
Podemos ver como el árbol de soluciones se construye desarrollando siempre la rama con el menor valor. Iniciamos con el nodo “a”.

>>> Inicmos con “a”:

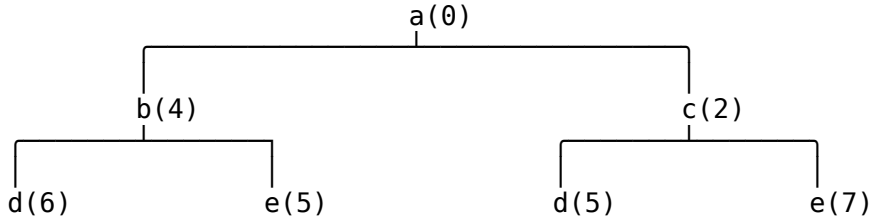
>>> Se desarrolla el nodo:



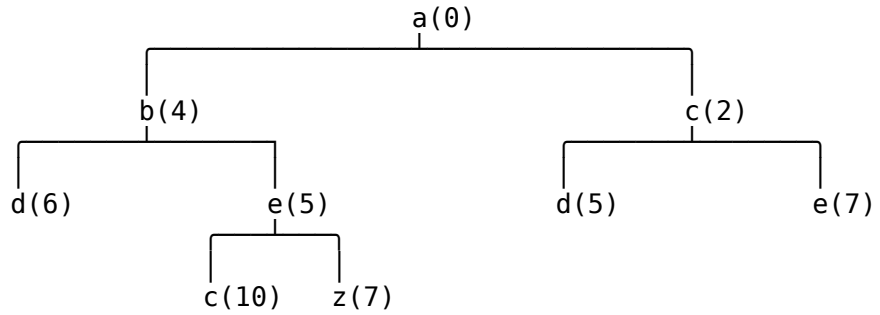
>>> Se desarrolla el nodo menor:



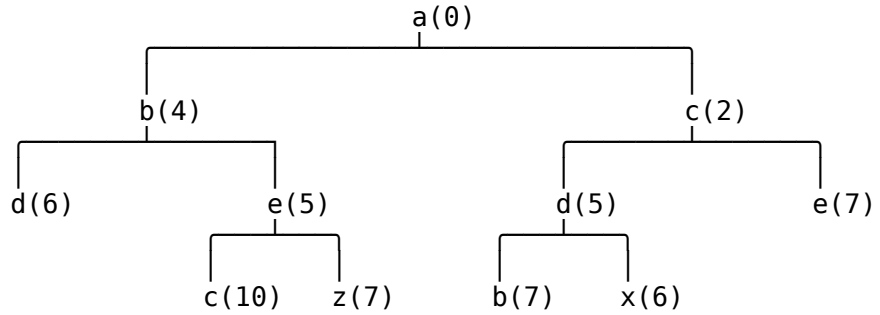
>>> Se vuelve a desarrollar el nodo menor:



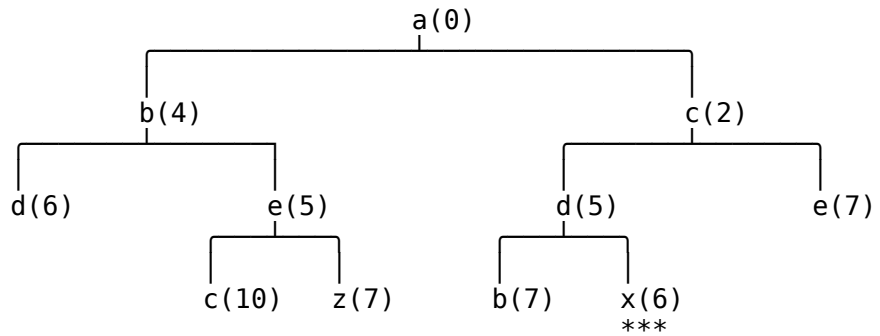
>>> Se desarrolla el menor:



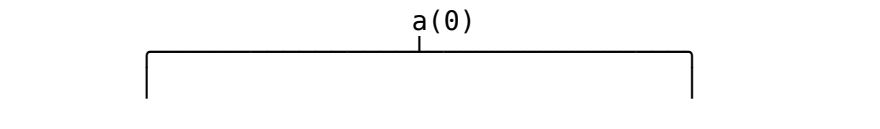
>>>> Una iteración más:

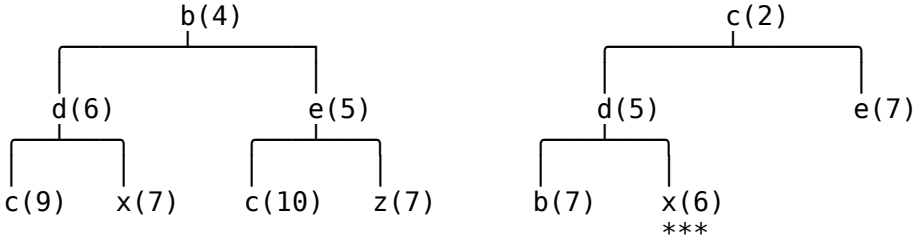


>>>> Una iteración más:

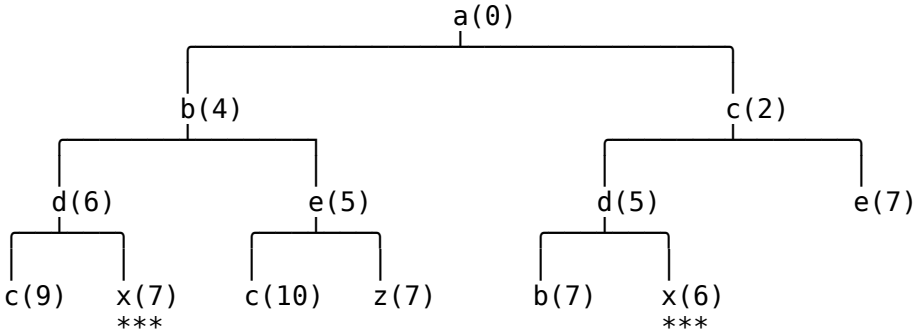


>>>> Una iteración más:

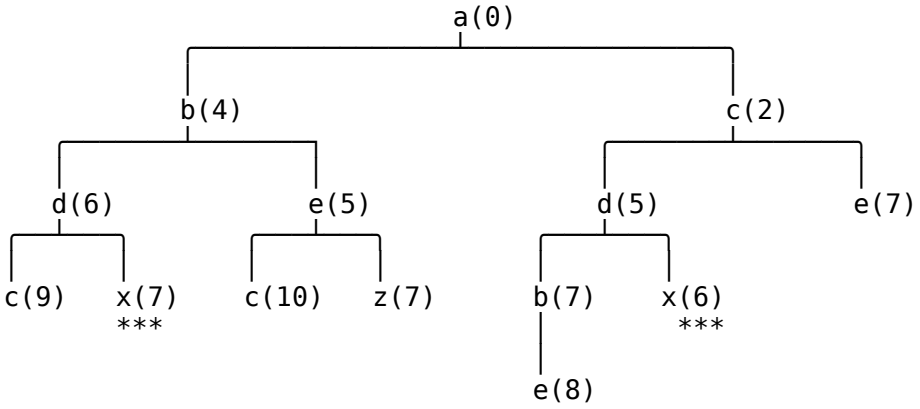




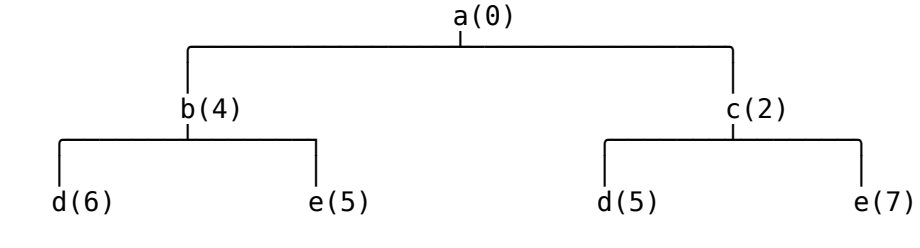
>>>> Una iteración más:

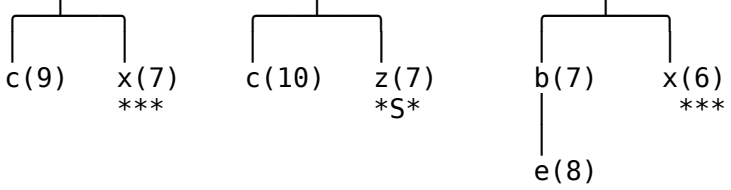


>>>> Una iteración más:



>>> Se encuentra el valor óptimo:





>>> Por lo tanto, la solución final es:

[[a,b,e,z],7]

▬▬▬ 11. Análisis de la Complejidad.

Se tomará el supuesto que tenemos un grafo completamente conectado.

Cada vez que se busca el valor mínimo se tiene un tiempo de O(n).

Una vez que se encuentra ese valor se extiende la ruta de forma recursiva, lo que nos lleva más cerca de la solución con tiempo T(n-1).

Por lo tanto el tiempo de ejecución está dado por:

$T(0) = 1,$

$T(n) = T(n-1) + n$

Al resolver la recurrencia se obtiene:

$T(n) = (1/2)*(n^2 + n + 2)$

Se tiene un tiempo de ejecución es de $O(n^2)$.

Es importante destacar que este algoritmo únicamente sirve para encontrar una ruta dentro de un grafo, no lo podemos usar para otros problemas.

▬▬▬ 12. Resumen.

• DFS va desarrollando la primera lista, lo que produce una búsqueda en profundidad. Devuelve todos los posibles recorridos.

• BFS va desarrollando cada nivel de la lista antes de continuar con el siguiente nivel. Devuelve todos los posibles recorridos.

• El algoritmo de Dijkstra va desarrollando continuamente la ruta con el menor valor posible. Devuelve únicamente el recorrido con el menor valor.

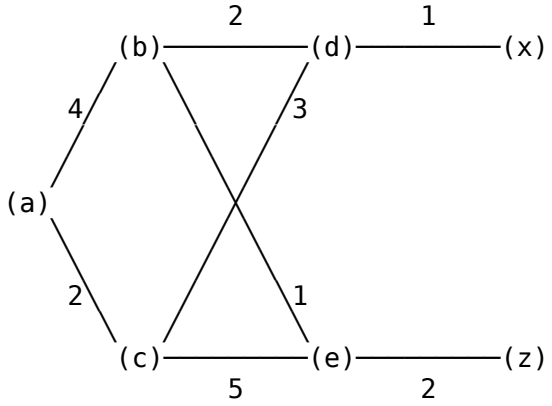
• Existen muchas formulaciones diferentes para el algoritmo de Dijkstra. En nuestro caso, funciona con grafos no dirigidos, con ciclos y con pesos positivos.

• Si existieran pesos negativos, se debe utilizar otra variante conocida como el algoritmo de Bellman-Ford.

▬▬▬ 13. Ejercicios.

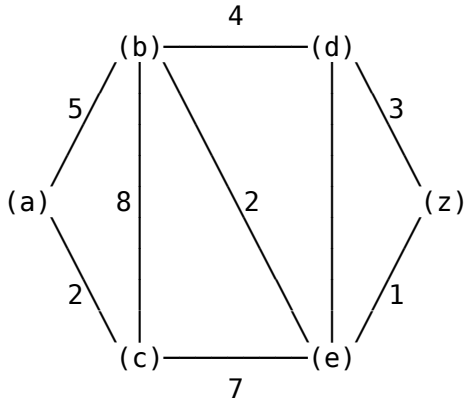
●● Ejercicio 1.

Encuentre la menor ruta para ir de “a” hacia “z” en el siguiente grafo:



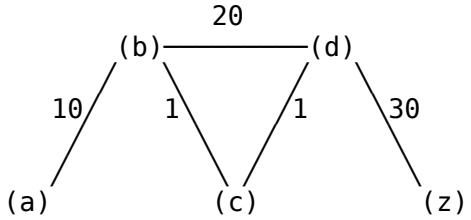
●● Ejercicio 2.

Encuentre la menor ruta para ir de “a” hacia “z” en el siguiente grafo:



●● Ejercicio 3.

Encuentre la menor ruta para ir de “a” hacia “z” en el siguiente grafo:



●● Ejercicio 4.

Encuentre la menor ruta para ir de “a” hacia “z” en el siguiente grafo: