BUSQUEDA DE COSTO UNIFORME - ALGORITMO DIJKSTRA

In [ ]:
```
import graphviz as gv
import numpy as np
import pandas as pd
import heapq as hq
import math
```

Funciones que permiten leer(cargar) y visualizar una lista de adyacencia.

In [ ]:
```
# fn: Archivo con la Lista de Adyacencia (LA)
def readAdjl(fn, haslabels=False, weighted=False, sep="|"):
  with open(fn) as f: # "f" es alias del Archivo
    labels = None
    if haslabels:
      labels = f.readline().strip().split()
    L = []
    for line in f:
      if weighted:
        L.append([tuple(map(int, p.split(sep))) for p in line.strip().split()])
        # line => "1|3 2|5 4|4" ==> [(1, 3), (2, 5), (4, 4)]
      else:
        L.append(list(map(int, line.strip().split()))) # "1 3 5" => [1, 3, 5]
        # L.append([int(x) for x in line.strip().split()])
  return L, labels


# Genera el Grafo(Grafico) con Graphviz, generando los Vertices y Aristas
def adjlShow(L, labels=None, directed=False, weighted=False, path=[],
             layout="sfdp"):
  g = gv.Digraph("G") if directed else gv.Graph("G")
  g.graph_attr["layout"] = layout
  g.edge_attr["color"] = "gray"
  g.node_attr["color"] = "orangered"
  g.node_attr["width"] = "0.1"
  g.node_attr["height"] = "0.1"
  g.node_attr["fontsize"] = "8"
  g.node_attr["fontcolor"] = "mediumslateblue"
  g.node_attr["fontname"] = "monospace"
  g.edge_attr["fontsize"] = "8"
  g.edge_attr["fontname"] = "monospace"
  n = len(L)
  for u in range(n):
    g.node(str(u), labels[u] if labels else str(u))
  added = set()
  for v, u in enumerate(path):
    if u != None:
      if weighted:
        for vi, w in G[u]:
          if vi == v:
            break
        g.edge(str(u), str(v), str(w), dir="forward", penwidth="2", color="orange")
      else:
        g.edge(str(u), str(v), dir="forward", penwidth="2", color="orange")
```

```
      added.add(f"{u},{v}")
      added.add(f"{v},{u}")
  if weighted:
    for u in range(n):
      for v, w in L[u]:
        if not directed and not f"{u},{v}" in added:
          added.add(f"{u},{v}")
          added.add(f"{v},{u}")
          g.edge(str(u), str(v), str(w))
        elif directed:
          g.edge(str(u), str(v), str(w))
  else:
    for u in range(n):
      for v in L[u]:
        if not directed and not f"{u},{v}" in added:
          added.add(f"{u},{v}")
          added.add(f"{v},{u}")
          g.edge(str(u), str(v))
        elif directed:
          g.edge(str(u), str(v))
  return g
```

Función del Algoritmo Dijkstra

In [ ]:
```
# G:Grafo, s:Nodo Inicial
def dijkstra(G, s):
  n = len(G)
  visited = [False]*n
  path = [None]*n
  cost = [math.inf]*n #math.inf: Punto Flotante Infinito
  cost[s] = 0
  queue = [(0, s)]
  while queue:
    g_u, u = hq.heappop(queue)  #hq=Heap queue(Cola Priorizada o Cola Heap) | remueve
    if not visited[u]:
      visited[u] = True
      for v, w in G[u]:
        f = g_u + w
        if f < cost[v]:
          cost[v] = f
          path[v] = u
          print("v=",v, "path[v]=", path[v], "cost[v]=", cost[v]) #rezc
          hq.heappush(queue, (f, v))  # heappush(heap, ele): inserta el elemento en la

  return path, cost
```

Definición de una LA ponderada. Se lee del nodo 0: hacia nodo 2 existe un peso de 4, del nodo 0: hacia nodo 7 existe un peso de 8 y del nodo 0: hacia nodo 14 existe un peso de 3....

In [ ]:
```
# Ejecucion Principal, empieza acá

# Generamos la LA en un archivo
%%file 1.in
2|4 7|8 14|3
2|7 5|7
0|4 1|7 3|5 6|1
2|5
```

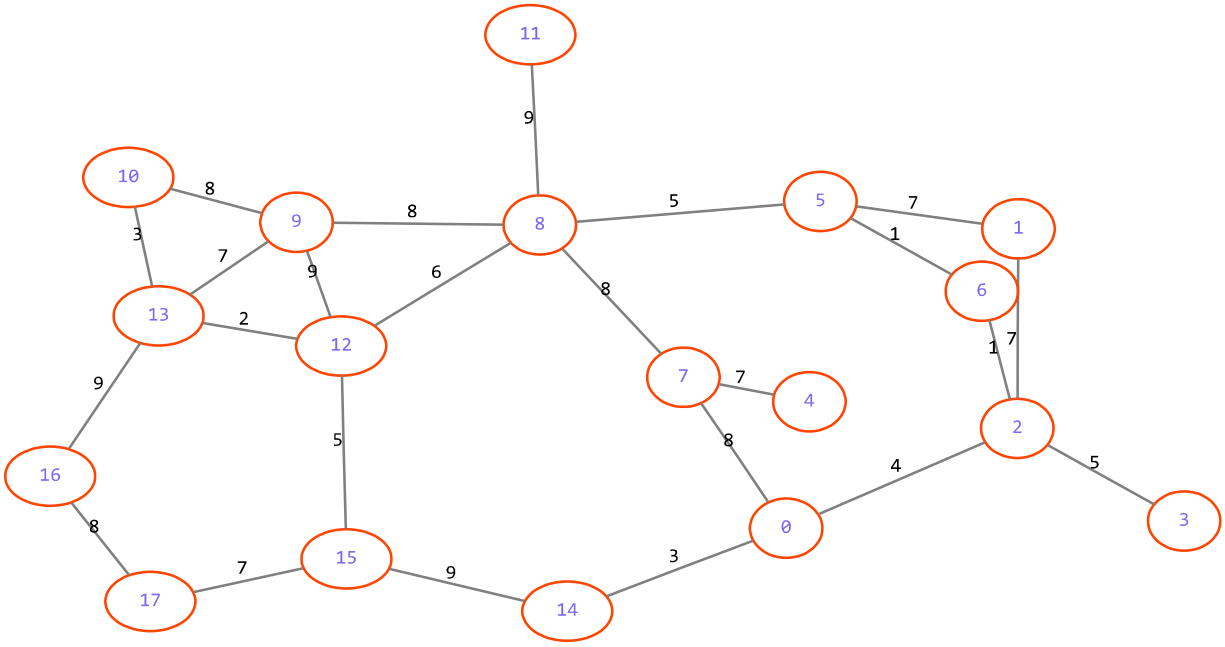
```
7|7
1|7 6|1 8|5
2|1 5|1
0|8 4|7 8|8
5|5 7|8 9|8 11|9 12|6
8|8 10|8 12|9 13|7
9|8 13|3
8|9
8|6 9|9 13|2 15|5
9|7 10|13 12|2 16|9
0|3 15|9
12|5 14|9 17|7
13|9 17|8
15|7 16|8
```

Writing 1.in

In [ ]:
```
# Convertimos la Lista de Adyacencia(LA) en Grafo
G, _ = readAdjl("1.in", weighted=True) # Geramos la Lista de Adyacencia en G
for i, edges in enumerate(G): #Retorna enumeracion:  0: [(2, 4), (7, 8), (14, 3)] | 1:
  print(f"{i:2}: {edges}")  #Imprime las lineas de la LA
adjlShow(G, weighted=True)  #Genera el Grafo(Grafico)
```

```
 0: [(2, 4), (7, 8), (14, 3)]
 1: [(2, 7), (5, 7)]
 2: [(0, 4), (1, 7), (3, 5), (6, 1)]
 3: [(2, 5)]
 4: [(7, 7)]
 5: [(1, 7), (6, 1), (8, 5)]
 6: [(2, 1), (5, 1)]
 7: [(0, 8), (4, 7), (8, 8)]
 8: [(5, 5), (7, 8), (9, 8), (11, 9), (12, 6)]
 9: [(8, 8), (10, 8), (12, 9), (13, 7)]
10: [(9, 8), (13, 3)]
11: [(8, 9)]
12: [(8, 6), (9, 9), (13, 2), (15, 5)]
13: [(9, 7), (10, 13), (12, 2), (16, 9)]
14: [(0, 3), (15, 9)]
15: [(12, 5), (14, 9), (17, 7)]
16: [(13, 9), (17, 8)]
17: [(15, 7), (16, 8)]
```

Out[ ]:

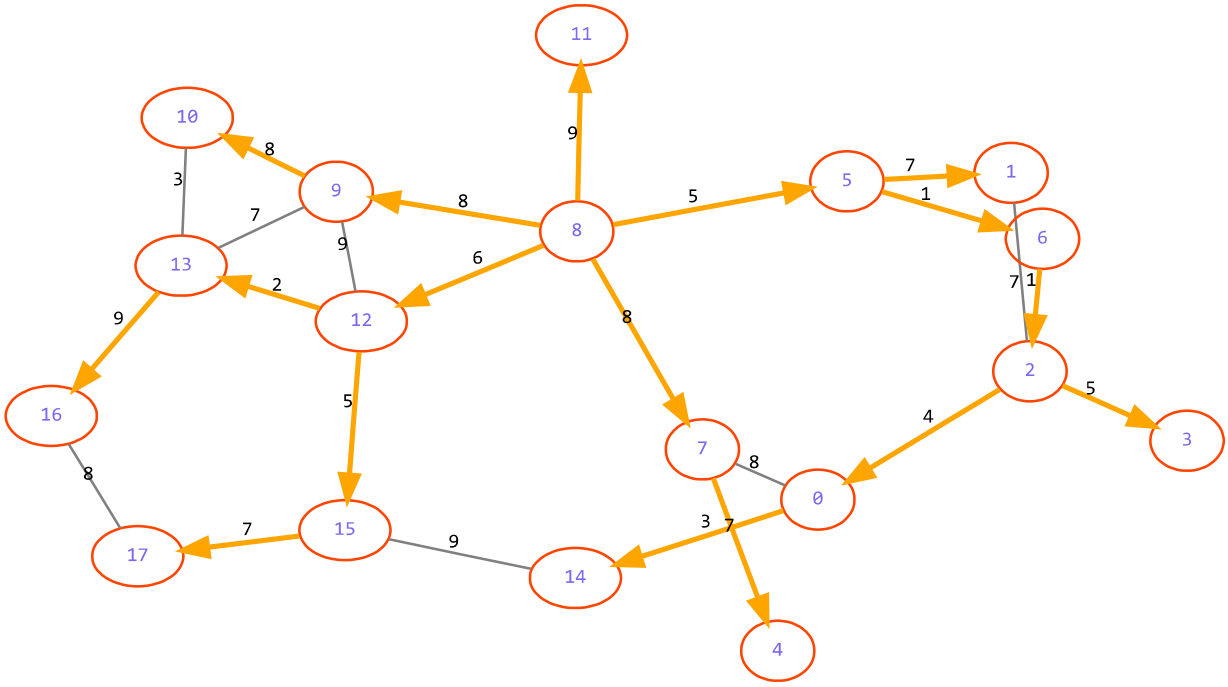


In [ ]:
```
# Ejecución Principal (main)
path, cost = dijkstra(G, 8) #path: Ultimo Nodo Predecesor del Destino | cost: Costo Ac
print()
print("path=", path)  #Imprime Nodos Predecesor al Nodo Destino
print("cost=", cost)  #Imprime Costo Total
adjlShow(G, weighted=True, path=path)  #Genera el Grafo(Gráfico)
```

```
v= 5 path[v]= 8 cost[v]= 5
v= 7 path[v]= 8 cost[v]= 8
v= 9 path[v]= 8 cost[v]= 8
v= 11 path[v]= 8 cost[v]= 9
v= 12 path[v]= 8 cost[v]= 6
v= 1 path[v]= 5 cost[v]= 12
v= 6 path[v]= 5 cost[v]= 6
v= 2 path[v]= 6 cost[v]= 7
v= 13 path[v]= 12 cost[v]= 8
v= 15 path[v]= 12 cost[v]= 11
v= 0 path[v]= 2 cost[v]= 11
v= 3 path[v]= 2 cost[v]= 12
v= 4 path[v]= 7 cost[v]= 15
v= 10 path[v]= 9 cost[v]= 16
v= 16 path[v]= 13 cost[v]= 17
v= 14 path[v]= 0 cost[v]= 14
v= 17 path[v]= 15 cost[v]= 18

path= [2, 5, 6, 2, 7, 8, 5, 8, None, 8, 9, 8, 8, 12, 0, 12, 13, 15]
cost= [11, 12, 7, 12, 15, 5, 6, 8, 0, 8, 16, 9, 6, 8, 14, 11, 17, 18]
```

Out[ ]:



OUTPUT [rezc]:

Ejemplo:

node= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

path= [2, 5, 6, 2, 7, 8, 5, 8, None, 8, 9, 8, 8, 12, 0, 12, 13, 15]

cost= [11, 12, 7, 12, 15, 5, 6, 8, 0, 8, 16, 9, 6, 8, 14, 11, 17, 18]

Interpretación:

Desde el **Origen: Nodo 8**, hacia:

Destino Nodo 0: El predecesor de 0 es el Nodo 2 | Costo Total = 11

Destino Nodo 1: El predecesor de 1 es el Nodo 5 | Costo Total = 12

Destino Nodo 2: El predecesor de 2 es el Nodo 6 | Costo Total = 7