In [ ]:
```
import graphviz as gv
```

Recorrido en Profundidad Limitada (DLS)

Mediante este tipo de recorrido logramos:

- Limitar la profundidad máxima de una ruta para la búsqueda por profundidad (DFS)
- Establecer un límite sin conocer mucho sobre el espacio de estados.
- Obtener un resultado que puede no ser completa ni óptima:

  > - Un límite muy pequeño puede no contener la solución.
  > - Un limite muy grande puede contener muchas soluciones no óptimas.

Funcion DLS

In [ ]:
```
def dls(G, s, L):
  n = len(G)
  visited = [False]*n
  path = [-1]*n

  def _dls(u, L):
    if L > 0 and not visited[u]:
      visited[u] = True
      for v in G[u]:
        if not visited[v]:
          path[v] = u
          _dls(v, L - 1)

  _dls(s, L)
  return path
```

Creamos una función para dibujar el grafo utilizando la libreria gv, indicando si ser un grafo dirigido o no. Si le pasamos una lista con una ruta, debera colorear dicha ruta (path).

In [ ]:
```
def drawG_al(G, directed=False, weighted=False, path=[], layout="sfdp"):
  graph = gv.Digraph("di-anyname") if directed else gv.Graph("anyname")
  graph.graph_attr["layout"] = layout
  graph.edge_attr["color"] = "gray"
  graph.node_attr["color"] = "orangered"
  graph.node_attr["width"] = "0.1"
  graph.node_attr["height"] = "0.1"
  graph.node_attr["fontsize"] = "8"
  graph.node_attr["fontcolor"] = "mediumslateblue"
  graph.node_attr["fontname"] = "monospace"
  graph.edge_attr["fontsize"] = "8"
  graph.edge_attr["fontname"] = "monospace"
  n = len(G)
  added = set()
  for v, u in enumerate(path):
    if u != -1:
      if weighted:
```

```
        for vi, w in G[u]:
          if vi == v:
            break
        graph.edge(str(u), str(v), str(w), dir="forward", penwidth="2", color="orange"
      else:
        graph.edge(str(u), str(v), dir="forward", penwidth="2", color="orange")
      added.add(f"{u},{v}")
      added.add(f"{v},{u}")
  for u in range(n):
    for edge in G[u]:
      if weighted:
        v, w = edge
      else:
        v = edge
      draw = False
      if not directed and not f"{u},{v}" in added:
        added.add(f"{u},{v}")
        added.add(f"{v},{u}")
        draw = True
      elif directed and not f"{u},{v}" in added:
        added.add(f"{u},{v}")
        draw = True
      if draw:
        if weighted:
          graph.edge(str(u), str(v), str(w))
        else:
          graph.edge(str(u), str(v))
  return graph
```

Generamos una LA segun el siguiente texto:

In [ ]:
```
%%file 04a.la
1 4
-
-
2 6 17
7
2
10
0 8
4
8
-
8 14
16
9 17
15
11 17
15 17
13
```

```
Writing 04a.la
```

Cargamos la LA generada desde el archivo 04a.la a un arreglo tipo numpy en la variable G

In [ ]:
```
with open("04a.la") as f:
  G = []
  for line in f:
    if line == "-\n":
```

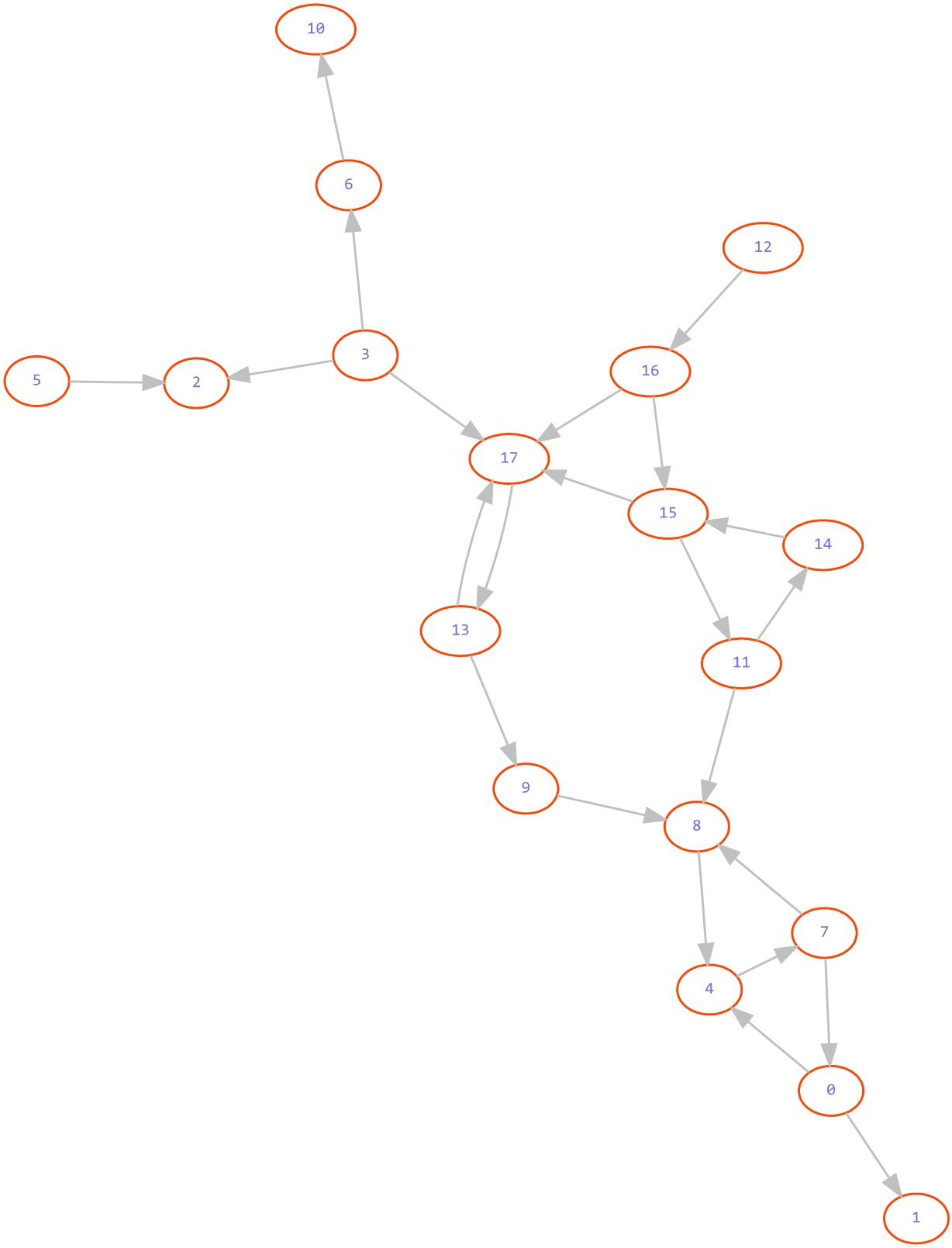
```
      G.append([])
    else:
      G.append([int(x) for x in line.split()])

for x in G:
  print(x)
```

```
[1, 4]
[]
[]
[2, 6, 17]
[7]
[2]
[10]
[0, 8]
[4]
[8]
[]
[8, 14]
[16]
[9, 17]
[15]
[11, 17]
[15, 17]
[13]
```

Visualizamos el grafo dirigido de la lista G

In [ ]:
```
drawG_al(G, directed=True, layout="neato")
```

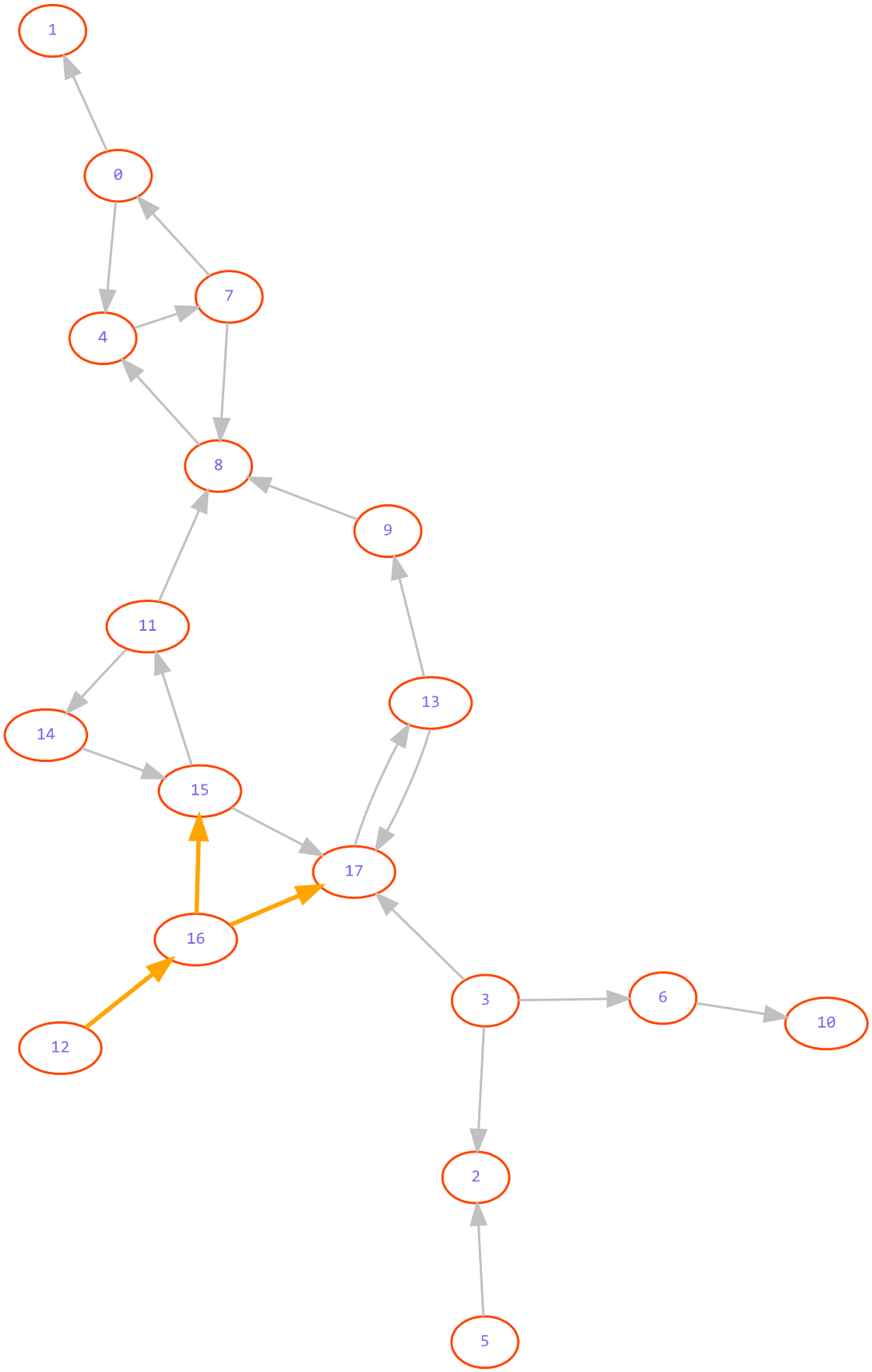
Out[ ]:



Hacemos una busqueda en profundidad limitada (DLS) del grafo dirigido G partiendo del nodo 12 y 2 niveles de profundidad.

In [ ]:
```
path = dls(G, 12, 2)
drawG_al(G, directed=True, path=path, layout="neato")
```

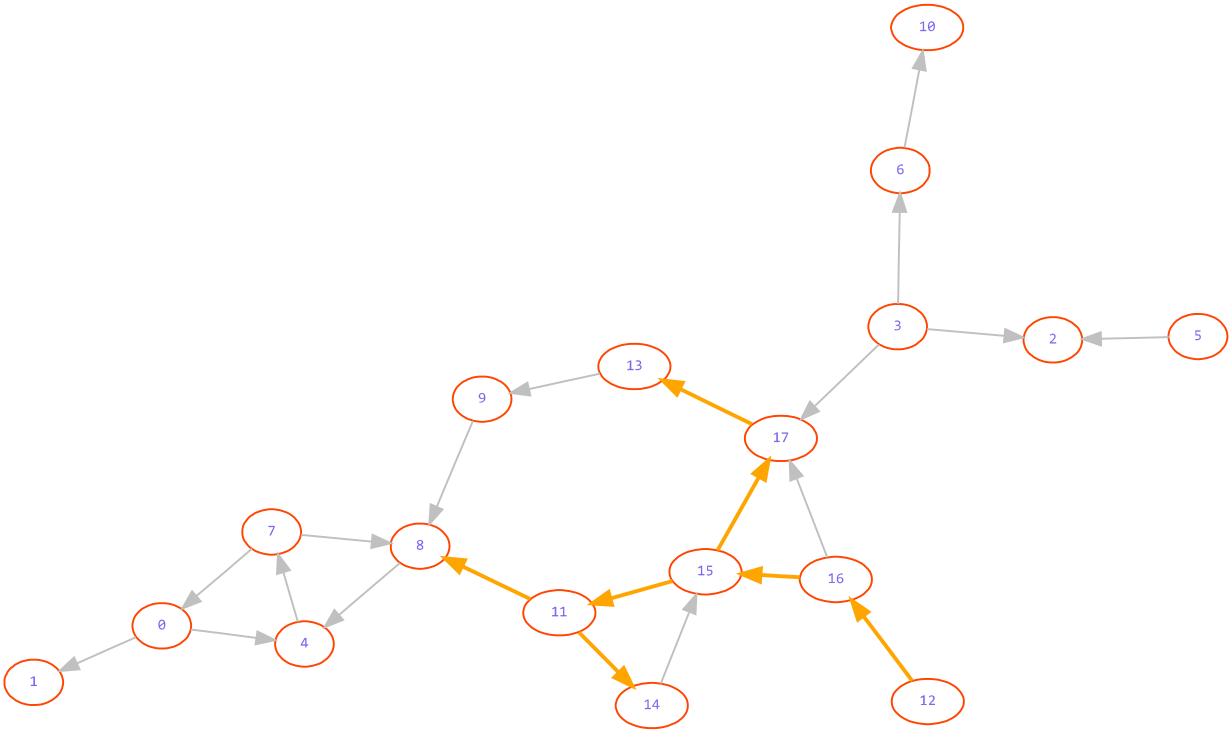
Out[ ]:



In [ ]:

```
path = dls(G, 12, 4)
drawG_al(G, directed=True, path=path, layout="neato")
```

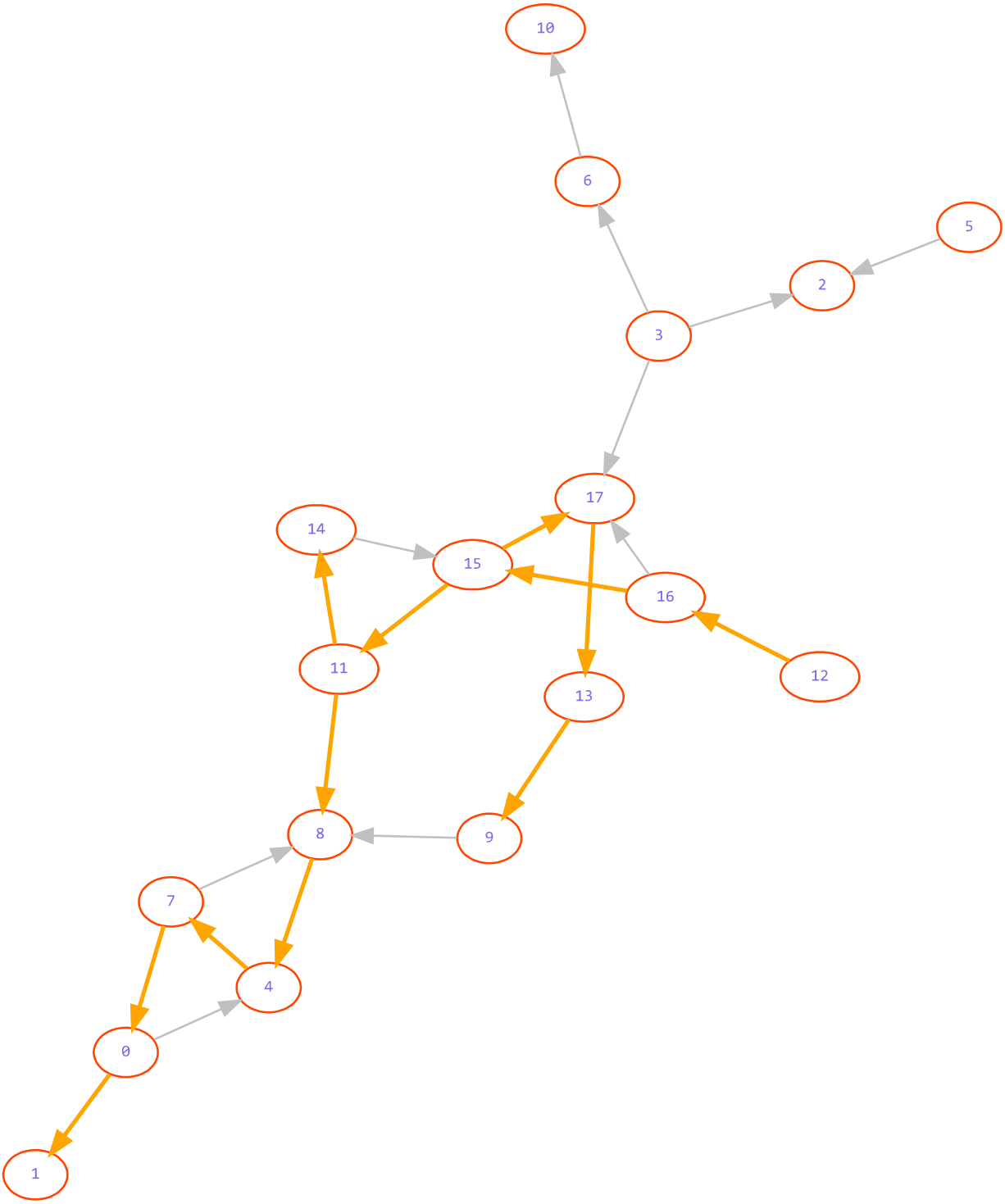
Out[ ]:



In [ ]:

```
path = dls(G, 12, 100)
drawG_al(G, directed=True, path=path, layout="neato")
```

Out[ ]:



Recorrido en Profundidad Iteractiva (IDS)

Mediante este tipo de recorrido logramos:

- Eliminar la dificultad de elegir un límite adecuado de profundidad en el DLS.
- Probar todos los límites de profundidad posibles, primero la profundidad 0, luego la 1, luego la 2, etc.
- Combinar las ventajas de las búsquedas por profundidad y por amplitud.

Funcion IDS

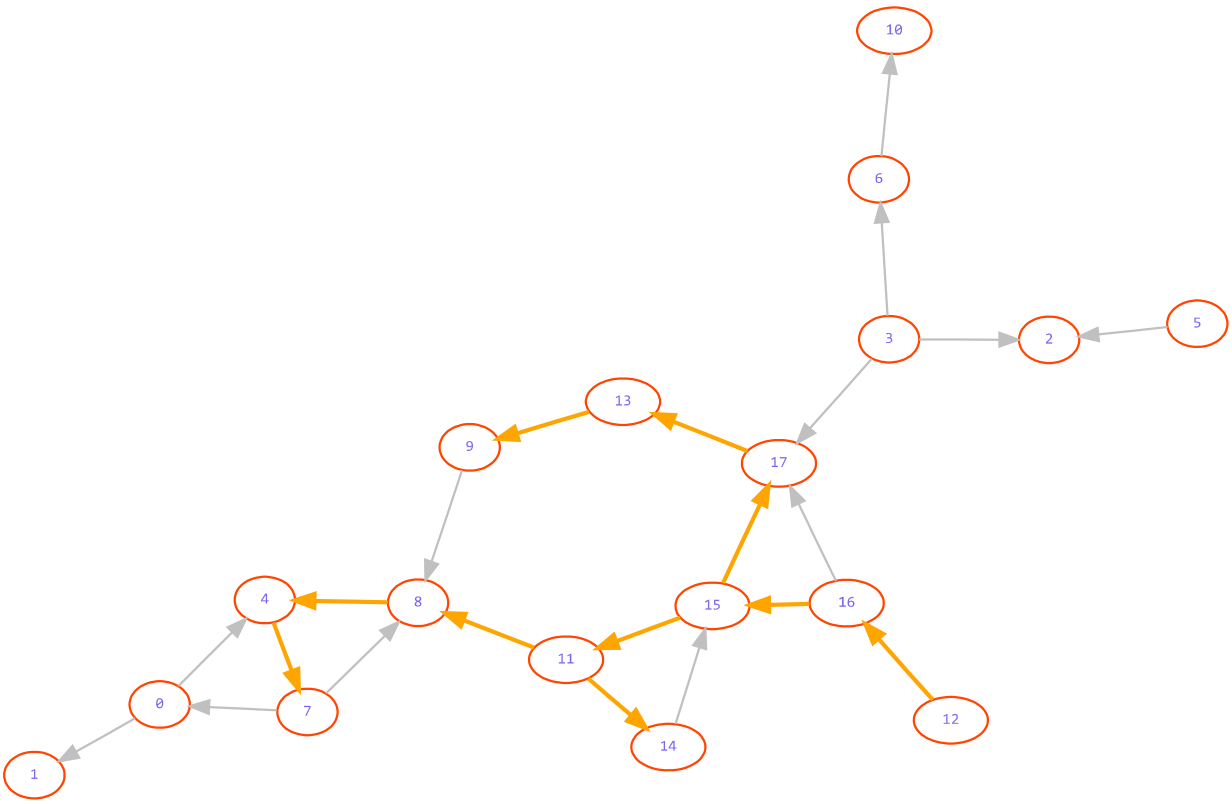
In [ ]:
```
def ids(G, start, target):
  n = len(G)
  for limit in range(n):
    path = dls(G, start, limit)
    if path[target] != -1:
      break
  return path
```

Obtenemos un trayecto (ruta) del grafo dirigido G, iniciando del nodo 12 y finalizando en el 7. Luego visualizamos en color naranja el recorrido iteractivo.

In [ ]:
```
path = ids(G, start=12, target=7)
print(path)
drawG_al(G, directed=True, path=path, layout="neato")
```

```
[-1, -1, -1, -1, 8, -1, -1, 4, 11, 13, -1, 15, -1, 17, 11, 16, 12, 15]
```

Out[ ]:

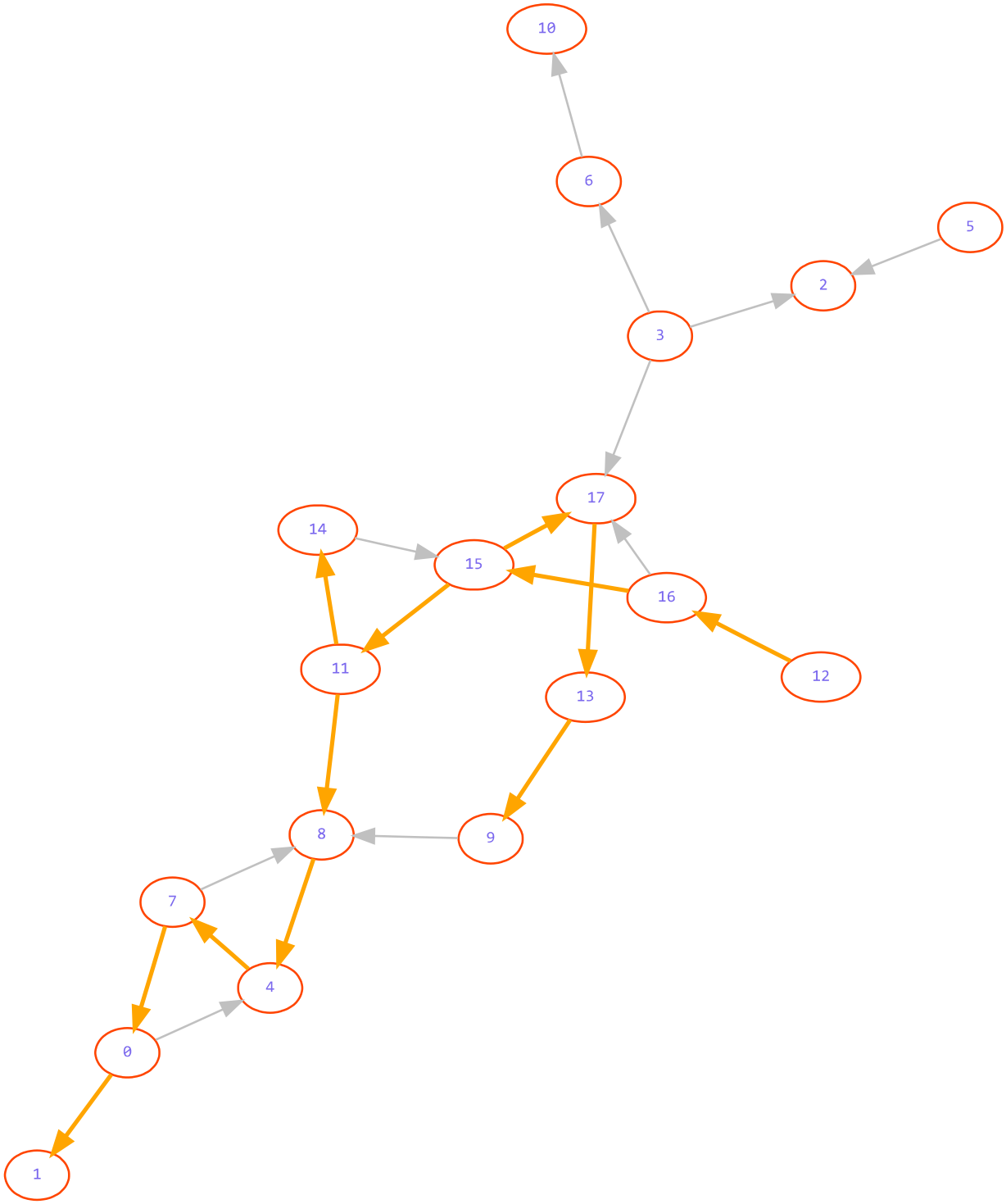


Obtenemos un trayecto (ruta) del grafo dirigido G, iniciando del nodo 12 y finalizando en el 10. Luego visualizamos en color naranja el recorrido iteractivo.

In [ ]:
```
path = ids(G, start=12, target=10)
print(path)
drawG_al(G, directed=True, path=path, layout="neato")
```

```
[7, 0, -1, -1, 8, -1, -1, 4, 11, 13, -1, 15, -1, 17, 11, 16, 12, 15]
```

Out[ ]:



In [ ]: