Algoritmo Dijstrka - Applicación 2

David Corzo & Jean Pierre Mejicanos

15 de noviembre 2020

1. Implementación de algoritmo Dijkstra

En el método `dijktra` se encuentra la implementación del algoritmo Dijsktra. En el método `dibujar_grafica` se encuentra la implementación para graficar el grafo ingresado.

```python
import networkx as nx
import matplotlib.pyplot as plt
import os

def make_new_name(filename:str):
    i = 0
    while os.path.exists(f'{filename}{i}.png'):
        i += 1
    return f'{filename}{i}.png'

class Grafo:
    def __init__(self, grafo, nodo_origen='1'):
        self.grafo = grafo
        self.nodo_origen = nodo_origen
        self.pesos = dict()
        self.camino = dict()
        self.nodos_restantes = list()

    def dibujar_grafica(self, forma="circular"):
        graph = nx.Graph()
        for nodo in grafo:
            grafo[nodo]
            for nodo2 in grafo[nodo]:
                print(f"{nodo} : {nodo2}")
                graph.add_edge(nodo, nodo2, weight=grafo[nodo][nodo2])
        if forma == "circular":
            pos = nx.circular_layout(graph)
        else:
            pos = nx.planar_layout(graph)
        nx.draw(graph, pos, with_labels=True)
        labels = nx.get_edge_attributes(graph, 'weight')
        nx.draw_networkx_edge_labels(graph, pos,edge_labels=labels)
        plt.savefig(make_new_name("grafo"))


    def dijsktra(self, nodo_destino):
        """ Encontrar matriz de adyacencia de Dijsktra. """
        for node in self.grafo: # n
```

```python
            self.pesos[node] = float("inf")
            self.camino[node] = None
            self.nodos_restantes.append(node)

        self.pesos[self.nodo_origen] = 0
        while len(self.nodos_restantes) != 0:
            llave_minima = min(self.nodos_restantes)
            nodo_actual = llave_minima
            self.nodos_restantes.remove(nodo_actual)

            for nodo in self.grafo[nodo_actual]:
                nuevo_peso = self.grafo[nodo_actual][nodo] + self.pesos[nodo_actual]
                if self.pesos[nodo] > nuevo_peso:
                    self.pesos[nodo] = nuevo_peso
                    self.camino[nodo] = nodo_actual

        print(f'The path between {self.nodo_origen} to {nodo_destino}')
        order = list()
        order.append(nodo_destino)
        while True:
            nodo_destino = self.camino[nodo_destino]
            if nodo_destino is None:
                break
            order.insert(0,nodo_destino)
        print("->".join(order))

if __name__ == "__main__":
    grafo = {
        '1': {'2':2, '3':4},
        '2': {'3':1, '4':7},
        '3': {'5':3},
        '4': {'6':1},
        '5': {'4':2, '6':5},
        '6': {}
    }
    g = Grafo(grafo, '1')
    g.dijsktra('6')
    g.dibujar_grafica("plano")
```

```
# Output:
#     The path between 1 to 6
#     1->2->3->5->4->6
```

Y el output gráfico que se aprecia a continuación:

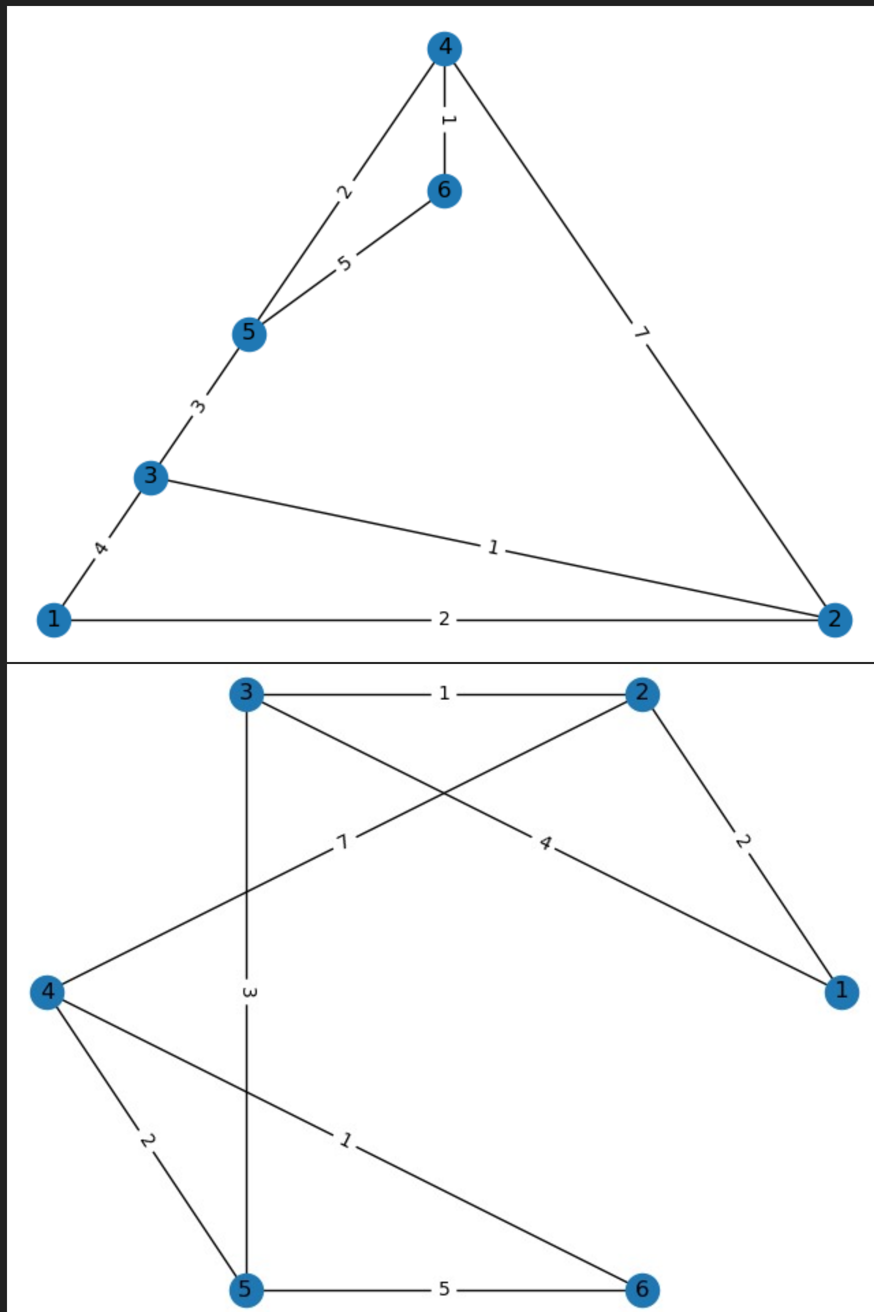

Figura 1: Grafo ingresado

2. Análisis de complejidad del algoritmo

A continuación presentamos los casos de atención en el algoritmo.

- Para preparar los nodos tomamos n, se puede ver en el fragmento de algoritmo a continuación.

```
for node in self.grafo: # n
    self.pesos[node] = float("inf") # 1*n
    self.camino[node] = None # 1*n
```

```
    self.nodos_restantes.append(node) # 1*n
```

- Observamos que la complejidad del peor escenario de este pedazo de código es $O(n)$

- Esta sección de código a continuación fabrica la matriz de adyacencia pertinente al algoritmo Dijkstra.

```
while len(self.nodos_restantes) != 0: # n
    llave_minima = min(self.nodos_restantes) # n*n
    nodo_actual = llave_minima # 1*n
    self.nodos_restantes.remove(nodo_actual) # 1*n

    for nodo in self.grafo[nodo_actual]: # n*n
        alternativa = self.grafo[nodo_actual][nodo] + self.pesos[nodo_actual] # 1*n*n
        if self.pesos[nodo] > alternativa: # 1*n*n
            self.pesos[nodo] = alternativa
            self.camino[nodo] = nodo_actual
```

- Observamos que la complejidad del peor escenario de este pedazo código es $O(n^2)$

- Para determinar qué camino tomar (ya fabricada la matriz de adyacencia) tomamos este código a continuación:

```
while True: # n
    nodo_destino = self.camino[nodo_destino] # 1*n
    if nodo_destino is None:
        break
    order.insert(0,nodo_destino) # 1*n
```

- Observamos que la complejidad del peor caso de este código es: $O(n)$

$\therefore$ Por lo anterior se puede afirmar que la complejidad de la implementación de este código Dijkstra es $O(n^2)$.