jkastra-shortest-path-algorithm-1

June 28, 2024

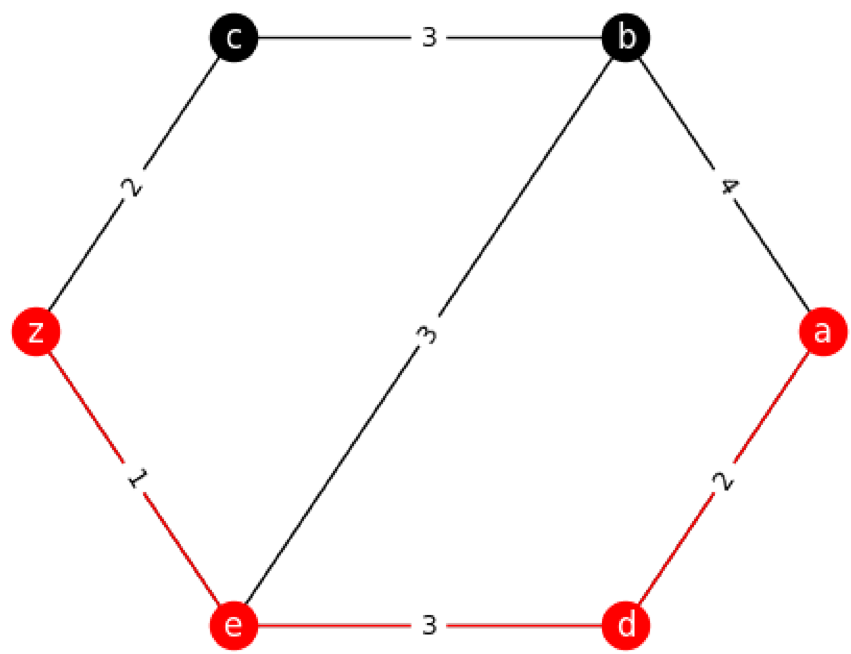
[1]:
```python
import networkx as nx
import matplotlib.pyplot as plt
```

[2]:
```python
# 1 Dijkastra Algorithm for undirected weighted graph
def do_dijkstra(graph,src,dest):
  Shortest_path = nx.
 ↪dijkstra_path(G=graph,source=src,target=dest,weight='weight')
  # Combine the shortest paths as tuple
  path_edges = list(zip(Shortest_path,Shortest_path[1:]))
  # Use f before the opening quotation mark in a print() statement, so that,
  # we can write a Python expression between { } characters that can refer to␣
 ↪variables or literal values.
  print(f"Shortest Path From {src} -> {dest}: {Shortest_path}")
  print("Shortest Path Edges:",path_edges)
  print("length of the Path: ", nx.
 ↪dijkstra_path_length(graph,src,dest,'weight'))
  # To draw the graph and the shortest path
  pos = nx.circular_layout(graph)
  nx.draw_networkx_nodes(Shortest_path, pos, node_color='r')
  nx.draw_networkx_nodes(graph.nodes - Shortest_path, pos,node_color='k')
  nx.draw_networkx_edges(graph, pos, edgelist=graph.edges)
  nx.draw_networkx_edges(graph,pos,edgelist=path_edges,edge_color='r')
  labels = nx.get_edge_attributes(graph,'weight')
  nx.draw_networkx_edge_labels(graph,pos,labels)
  nx.draw_networkx_labels(graph,pos,font_color='w')
  plt.axis("off")
  plt.show()
```

[3]:
```python
D = nx.Graph()
d_edges =␣
 ↪[('a','b',4),('b','c',3),('c','z',2),('z','e',1),('e','d',3),('b','e',3),('a','d',2)]
print(type(d_edges))
D.add_weighted_edges_from(d_edges)
do_dijkstra(D,'a','z')
```

```
<class 'list'>
Shortest Path From a -> z: ['a', 'd', 'e', 'z']
```

```
Shortest Path Edges: [('a', 'd'), ('d', 'e'), ('e', 'z')]
length of the Path:  6
```



[4]:
```python
import heapq
def dijkstra(graph, start):
    # Initialize distances to all vertices as infinity
    distances = {vertex: float('infinity') for vertex in graph}
    # Distance from start vertex to itself is 0
    distances[start] = 0
    # Priority queue to keep track of vertices to visit
    priority_queue = [(0, start)]
    while priority_queue:
        # Pop the vertex with the smallest distance from priority queue
        current_distance, current_vertex = heapq.heappop(priority_queue)

        # Skip if we have already found a shorter distance to this vertex
        if current_distance > distances[current_vertex]:
            continue
        # Explore neighbors of the current vertex
        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight
            # If new distance is shorter than the known distance, update
            if distance < distances[neighbor]:
                distances[neighbor] = distance
```

```
                # Add to priority queue
                heapq.heappush(priority_queue, (distance, neighbor))
    return distances
def shortest_path_length(graph, start, end):
    # Apply Dijkstra's algorithm to find shortest distances from start vertex
    distances = dijkstra(graph, start)
    # Return the distance to the end vertex
    return distances[end]

# Example graph representation (adjacency list)
graph = {
    'A': {'B': 3, 'C': 2},
    'B': {'C': 1, 'D': 5},
    'C': {'D': 7},
    'D': {'E': 2},
    'E': {}
}
start_vertex = 'A'
end_vertex = 'E'
shortest_length = shortest_path_length(graph, start_vertex, end_vertex)
print(f"The length of the shortest path from vertex {start_vertex} to vertex␣
 ↪{end_vertex} is: {shortest_length}")
print("Shortest distances from vertex", start_vertex, ":", dijkstra(graph,␣
 ↪start_vertex))
# To draw the graph and the shortest path
```

```
The length of the shortest path from vertex A to vertex E is: 10
Shortest distances from vertex A : {'A': 0, 'B': 3, 'C': 2, 'D': 8, 'E': 10}
```

[5]:

```
import heapq

def dijkstra(graph, start):
    # Initialize distances to all vertices as infinity
    distances = {vertex: float('infinity') for vertex in graph}
    # Distance from start vertex to itself is 0
    distances[start] = 0
    # Priority queue to keep track of vertices to visit
    priority_queue = [(0, start)]

    while priority_queue:
        # Pop the vertex with the smallest distance from priority queue
        current_distance, current_vertex = heapq.heappop(priority_queue)

        # Skip if we have already found a shorter distance to this vertex
        if current_distance > distances[current_vertex]:
            continue
```

```
        # Explore neighbors of the current vertex
        for neighbor, weight in graph[current_vertex].items():
            distance = current_distance + weight
            # If new distance is shorter than the known distance, update
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                # Add to priority queue
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances
def shortest_path_length(graph, start, end):
    # Apply Dijkstra's algorithm to find shortest distances from start vertex
    distances = dijkstra(graph, start)
    # Return the distance to the end vertex
    return distances[end]
# Example graph representation (adjacency list)
graph = {
    'A': {'B': 3, 'C': -2},
    'B': {'C': 1, 'D': -5},
    'C': {'D': 7},
    'D': {'E': 2},
    'E': {}
}

start_vertex = 'A'
end_vertex = 'E'
print("Shortest distances from vertex", start_vertex, ":", dijkstra(graph,␣
 ↪start_vertex))
shortest_length = shortest_path_length(graph, start_vertex, end_vertex)
print(f"The length of the shortest path from vertex {start_vertex} to vertex␣
 ↪{end_vertex} is: {shortest_length}")
```

```
Shortest distances from vertex A : {'A': 0, 'B': 3, 'C': -2, 'D': -2, 'E': 0}
The length of the shortest path from vertex A to vertex E is: 0
```

[ ]:

```
```