

# PRACTICA 03

Alumno: Emiliano Martinez Torres

December 2, 2023



# 1 Introducción

Durante la elaboración de esta práctica se verá el desarrollo e implementación del algoritmo de dijkstra a lo largo de 1 código que contiene 5 ejemplos así como también la explicación del funcionamiento del código sección por sección para mayor comprensión del algoritmo

## 2 Desarrollo

### 2.1 Explicación del código:

Primero iniciamos colocando “import heapq” y “import time” que son las librerías con las que vamos a trabajar, esto lo que hace es que en el caso de heapq es permitirme utilizar la estructura de montículo (heap), en el caso de time me permite medir el tiempo de ejecución del algoritmo.

Después de importar las librerías con las que se va a trabajar se procede a definir la clase Grafo, ya definida la clase de grafo se Inicializa el objeto Grafo con un diccionario vacío llamado vertices para almacenar los vértices y sus conexiones, después colocamos la función: “def addvertex(self, vertice):” la cual nos agrega un vértice al gráfico si no existe previamente en el diccionario vertices.

Una vez realizado esto se procedió a colocar: “def addedge(self, desdevertice, haciavertice, peso):” la cual agrega una arista entre dos vértices con su respectivo peso, en el caso de un grafo no dirigido, se agrega la arista en ambas direcciones.

Una vez listo el funcionamiento de la clase Grafo se procede a definir el algoritmo de dijkstra de la siguiente manera “def dijkstra(self, inicio):” esto lo que hace es implementar el algoritmo de dijkstra para encontrar las distancias más cortas desde un nodo inicial hasta todos los demás nodos del gráfico. Enseguida de eso definimos lo siguiente “distancias = vertice: float('inf') for vertice in self.vertices:” lo cual inicializa un diccionario “distancias” con distancias iniciales infinitas para todos los vértices del gráfico, excepto para el nodo de inicio, al que se le asigna una distancia de 0. Una vez listo eso colocamos “pq = [(0, inicio)]:” el cual crea un montículo (heap) con una tupla que contiene la distancia actual y el nodo actual (es el inicial).

Cuando tenemos listo eso procedemos a colocar “while pq.” el cual nos sirve para comenzar un bucle que se ejecuta mientras existen elementos en el montículo. Después ponemos “distanciaactual, verticeactual = heapq.heappop(pq).” el cual extrae el nudo con la distancia mínima del montículo. Una vez cumplida esa acción se coloca “if distanciaactual < distancias[verticeactual]: continue.” para indicar que si la distancia actual es mayor que la almacenada en el diccionario de distancias, omita el nodo y continúe con el siguiente. Después colocamos “for vecino, peso in self.vertices[verticeactual].items():” el cual itera sobre los vecinos del nodo actual y sus respectivos pesos para ver cual es la ruta más corta. Una vez hecho esto se coloca “distancia = distanciaactual + peso.” el cual sirve para calcular la distancia acumulada desde el nodo inicial al vecino actual. Después se colocó “if distancia < distancias[vecino]:” el cual nos sirve si la distancia calculada es menor que la distancia almacenada en el diccionario para ese vecino. Una vez listo esto colocamos “distancias[vecino] = distancia.” para poder actualizar la distancia más corta hasta el vecino. Una vez hecho esto se colocó “heapq.heappush(pq, (distancia, vecino)).” para agregar la tupla actualizada al montículo para explorar más adelante. Después para finalizar esta parte se colocó “return distancias.” lo cual nos sirve para devolver el diccionario de distancias más cortas desde el nodo inicial a todos los demás nodos.

Después de esto se comienza con la implementación de los ejemplos de uso en donde se crean varios gráficos ( grafo1, grafo2,grafo3,grafo4, grafo5) con diferentes vértices y aristas. Después se realizan llamadas al método dijkstra de cada gráfico con un nodo de inicio específico. Por último calcula el tiempo de ejecución del algoritmo para cada caso y se imprimen las distancias más cortas desde el nodo de inicio a todos los demás nodos junto con el tiempo de ejecución en segundos. En general este es el funcionamiento del código realizado en esta práctica a continuación se muestra el código realizado para que se pueda visualizar de mejor forma la explicación dada:

```

1 ✓ import heapq
2   import time
3
4 ✓ class Grafo:
5   def __init__(self):
6     self.vertices = {}
7
8   def add_vertex(self, vertice):
9     if vertice not in self.vertices:
10       self.vertices[vertice] = []
11
12  def add_edge(self, desde_vertice, hacia_vertice, peso):
13    if desde_vertice in self.vertices and hacia_vertice in self.vertices:
14      self.vertices[desde_vertice].append((hacia_vertice, peso))
15      self.vertices[hacia_vertice].append((desde_vertice, peso)) # En caso de grafo no dirigido
16
17  def dijkstra(self, inicio):
18    distancias = {vertice: float('inf') for vertice in self.vertices}
19    distancias[inicio] = 0
20    pq = [(0, inicio)]
21
22  while pq:
23    distancia_actual, vertice_actual = heapq.heappop(pq)
24
25    if distancia_actual > distancias[vertice_actual]:
26      continue
27
28    for vecino, peso in self.vertices[vertice_actual]:
29      distancia = distancia_actual + peso
30      if distancia < distancias[vecino]:
31        distancias[vecino] = distancia
32        heapq.heappush(pq, (distancia, vecino))
33
34  return distancias
35
36 # Ejemplo de uso 1
37 grafo1 = Grafo()
38 grafo1.add_vertex("A")
39 grafo1.add_vertex("B")
40 grafo1.add_vertex("C")
41 grafo1.add_edge("A", "B", 3)
42 grafo1.add_edge("A", "C", 5)
43 grafo1.add_edge("B", "C", 2)
44
45 nodo_inicio = "A"
46
47 start_time = time.time()
48 result = grafo1.dijkstra(nodo_inicio)
49 end_time = time.time()
50
51 resultados = grafo1.dijkstra(nodo_inicio)
52 print("Distancias más cortas desde el nodo", nodo_inicio, "a los demás nodos en el caso 1:")

```

```

53 ✓ for nodo, distancia in resultados.items():
54     print(f"Nodo: {nodo}, Distancia: {distancia}")
55
56 tiempo_ejecucion = end_time - start_time
57 print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
58
59 # Ejemplo de uso 2
60 grafo2 = Grafo()
61 grafo2.add_vertex("A")
62 grafo2.add_vertex("B")
63 grafo2.add_vertex("C")
64 grafo2.add_vertex("D")
65 grafo2.add_edge("A", "B", 2)
66 grafo2.add_edge("A", "C", 5)
67 grafo2.add_edge("C", "B", 7)
68 grafo2.add_edge("B", "D", 4)
69
70 nodo_inicio = "A"
71
72 start_time = time.time()
73 result = grafo2.dijkstra(nodo_inicio)
74 end_time = time.time()
75
76 resultados = grafo2.dijkstra(nodo_inicio)
77 print("Distancias más cortas desde el nodo", nodo_inicio, "a los demás nodos en el caso 2:")
78 ✓ for nodo, distancia in resultados.items():
79     print(f"Nodo: {nodo}, Distancia: {distancia}")
80
81 tiempo_ejecucion = end_time - start_time
82 print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
83
84 # Ejemplo de uso 3
85 grafo3 = Grafo()
86 grafo3.add_vertex("A")
87 grafo3.add_vertex("B")
88 grafo3.add_vertex("C")
89 grafo3.add_vertex("D")
90 grafo3.add_vertex("E")
91 grafo3.add_edge("A", "B", 8)
92 grafo3.add_edge("A", "C", 9)
93 grafo3.add_edge("B", "D", 10)
94 grafo3.add_edge("C", "E", 2)
95 grafo3.add_edge("D", "C", 5)
96 grafo3.add_edge("E", "B", 13)
97
98
99 nodo_inicio = "A"
100
101 start_time = time.time()
102 result = grafo3.dijkstra(nodo_inicio)
103 end_time = time.time()
104
105 resultados = grafo3.dijkstra(nodo_inicio)
106 print("Distancias más cortas desde el nodo", nodo_inicio, "a los demás nodos en el caso 3:")

```

```

107     for nodo, distancia in resultados.items():
108         print(f"Nodo: {nodo}, Distancia: {distancia}")
109
110     tiempo_ejecucion = end_time - start_time
111     print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
112
113     # Ejemplo de uso 4
114     grafo4 = Grafo()
115     grafo4.add_vertex("A")
116     grafo4.add_vertex("B")
117     grafo4.add_vertex("C")
118     grafo4.add_vertex("D")
119     grafo4.add_vertex("E")
120     grafo4.add_edge("A", "D", 4)
121     grafo4.add_edge("D", "C", 5)
122     grafo4.add_edge("C", "B", 8)
123     grafo4.add_edge("B", "E", 9)
124     grafo4.add_edge("E", "A", 15)
125
126     nodo_inicio = "A"
127
128     start_time = time.time()
129     result = grafo4.dijkstra(nodo_inicio)
130     end_time = time.time()
131
132     resultados = grafo4.dijkstra(nodo_inicio)
133     print("Distancias más cortas desde el nodo", nodo_inicio, "a los demás nodos en el caso 4:")
134
135     for nodo, distancia in resultados.items():
136         print(f"Nodo: {nodo}, Distancia: {distancia}")
137
138     tiempo_ejecucion = end_time - start_time
139     print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
140
141     # Ejemplo de uso 5
142     grafo5 = Grafo()
143     grafo5.add_vertex("A")
144     grafo5.add_vertex("B")
145     grafo5.add_vertex("C")
146     grafo5.add_vertex("D")
147     grafo5.add_vertex("E")
148     grafo5.add_vertex("F")
149     grafo5.add_edge("A", "B", 5)
150     grafo5.add_edge("A", "C", 7)
151     grafo5.add_edge("B", "C", 9)
152     grafo5.add_edge("C", "D", 12)
153     grafo5.add_edge("C", "E", 15)
154     grafo5.add_edge("D", "F", 8)
155     grafo5.add_edge("E", "F", 11)
156
157     nodo_inicio = "A"
158
159     start_time = time.time()
160     result = grafo5.dijkstra(nodo_inicio)

```

```

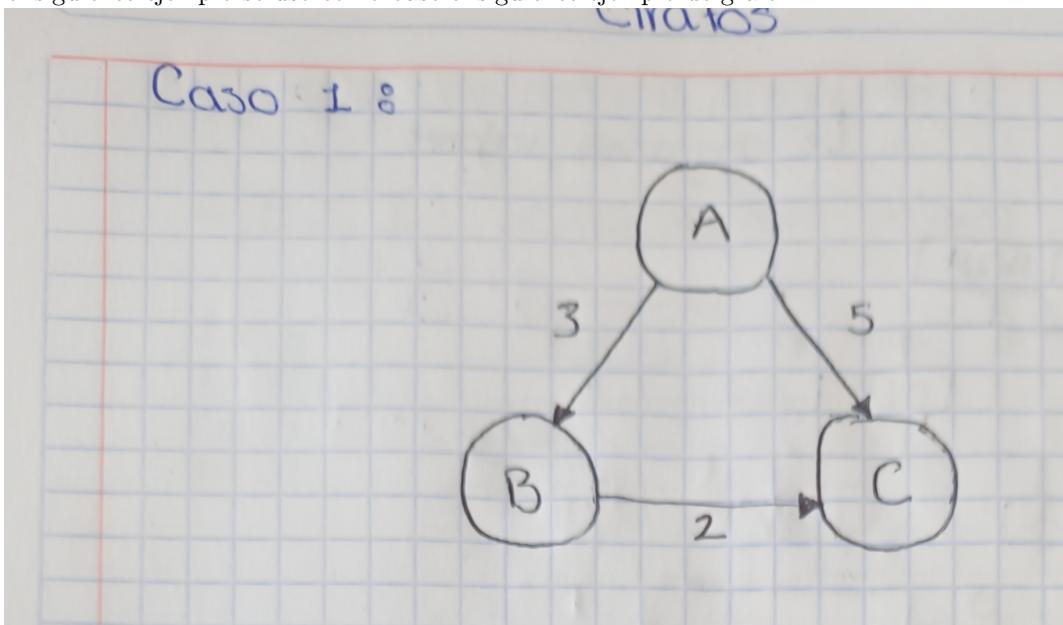
159     result = grafo5.dijkstra(nodo_inicio)
160     end_time = time.time()
161
162     resultados = grafo5.dijkstra(nodo_inicio)
163     print("Distancias más cortas desde el nodo", nodo_inicio, "a los demás nodos en el caso 5:")
164     for nodo, distancia in resultados.items():
165         print(F"Nuevo: {nodo}, Distancia: {distancia}")
166
167     tiempo_ejecucion = end_time - start_time
168     print(F"Tiempo de ejecución: {tiempo_ejecucion} segundos")

```

### 3 Resultados

#### 3.1 Ejemplo de uso 1:

En el siguiente ejemplo se usó como base el siguiente ejemplo de grafo:



En el cual dio el resultado esperado a la hora de correr el código y ver el resultado, cabe aclarar que se toma el nodo A como punto de partida dicho esto es el resultado obtenido:

```

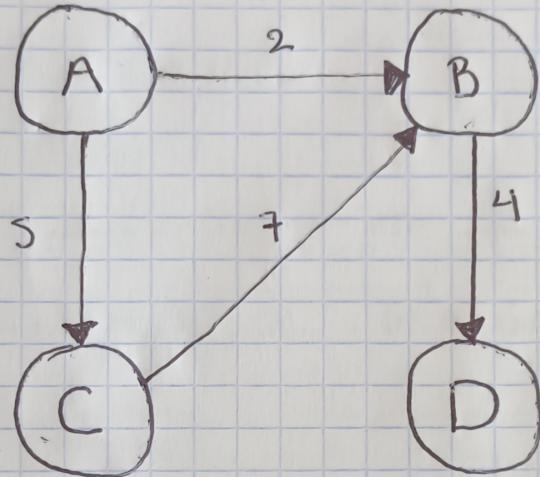
Distancias más cortas desde el nodo A a los demás nodos en el caso 1:
Nuevo: A, Distancia: 0
Nuevo: B, Distancia: 3
Nuevo: C, Distancia: 5
Tiempo de ejecución: 0.0 segundos

```

#### 3.2 Ejemplo de uso 2:

En el siguiente ejemplo se usó como base el siguiente ejemplo de grafo:

Caso 2 :



En el cual dio el resultado esperado a la hora de correr el código y ver el resultado, cabe aclarar que se toma el nodo A como punto de partida dicho esto es el resultado obtenido:

**Distancias más cortas desde el nodo A a los demás nodos en el caso 2:**

Nodo: A, Distancia: 0

Nodo: B, Distancia: 2

Nodo: C, Distancia: 5

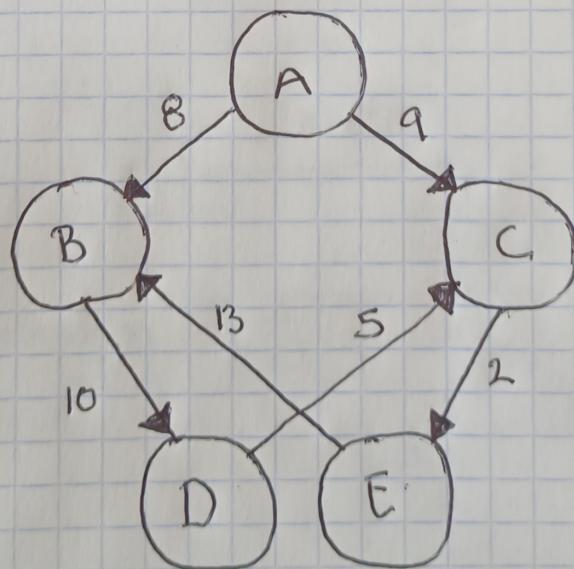
Nodo: D, Distancia: 6

Tiempo de ejecución: 0.0 segundos

### 3.3 Ejemplo de uso 3:

En el siguiente ejemplo se usó como base el siguiente ejemplo de grafo:

Caso 3 :



En el cual dio el resultado esperado a la hora de correr el código y ver el resultado, cabe aclarar que se toma el nodo A como punto de partida dicho esto es el resultado obtenido:

Distancias más cortas desde el nodo A a los demás nodos en el caso 3:

Nodo: A, Distancia: 0

Nodo: B, Distancia: 8

Nodo: C, Distancia: 9

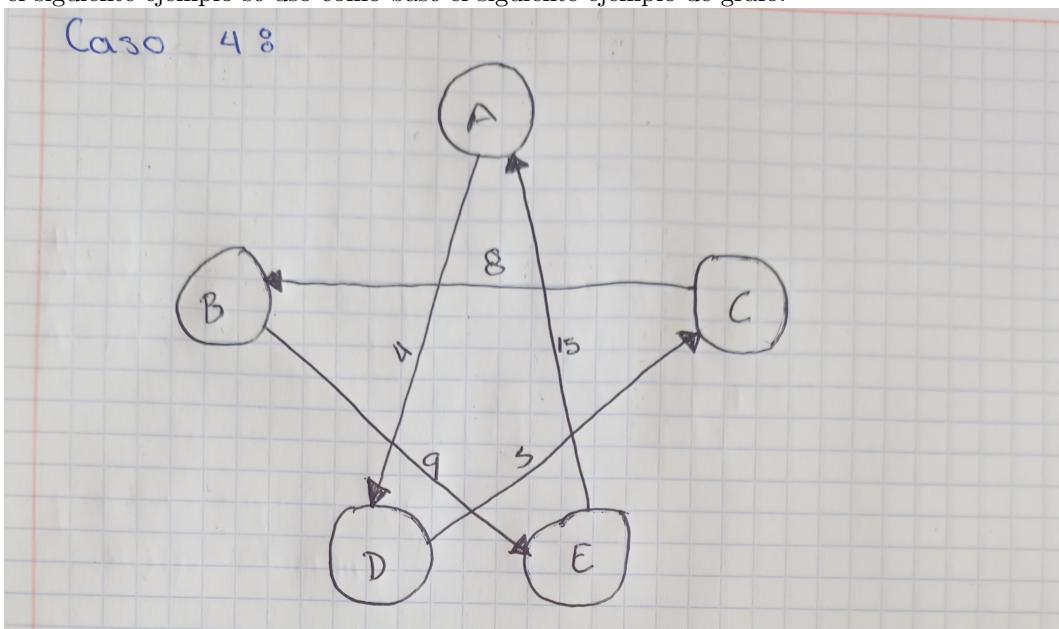
Nodo: D, Distancia: 14

Nodo: E, Distancia: 11

Tiempo de ejecución: 0.0 segundos

### 3.4 Ejemplo de uso 4:

En el siguiente ejemplo se usó como base el siguiente ejemplo de grafo:



En el cual dio el resultado esperado a la hora de correr el código y ver el resultado, cabe aclarar que se toma el nodo A como punto de partida dicho esto es el resultado obtenido:

Distancias más cortas desde el nodo A a los demás nodos en el caso 4:

Nodo: A, Distancia: 0

Nodo: B, Distancia: 17

Nodo: C, Distancia: 9

Nodo: D, Distancia: 4

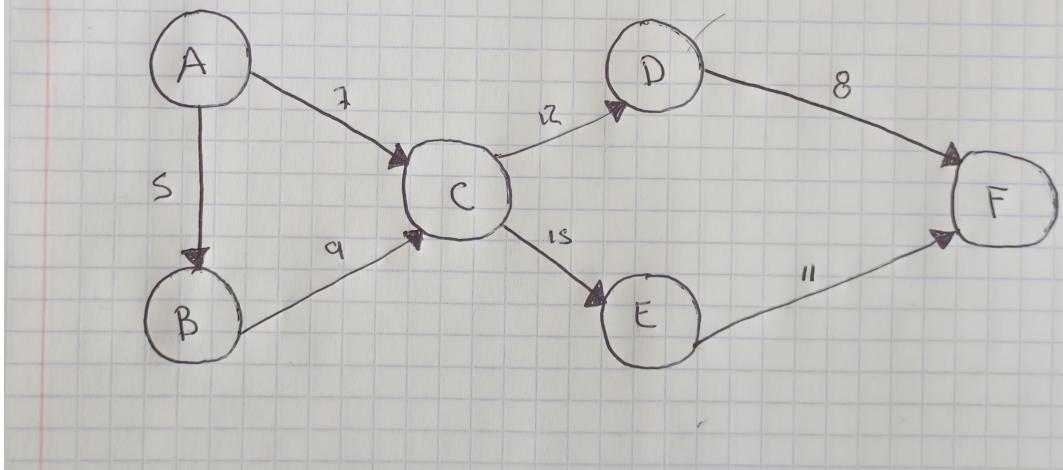
Nodo: E, Distancia: 15

Tiempo de ejecución: 0.0 segundos

### 3.5 Ejemplo de uso 5:

En el siguiente ejemplo se usó como base el siguiente ejemplo de grafo:

Caso 5 :



En el cual dio el resultado esperado a la hora de correr el código y ver el resultado, cabe aclarar que se toma el nodo A como punto de partida dicho esto es el resultado obtenido:

**Distancias más cortas desde el nodo A a los demás nodos en el caso 5:**

Nodo: A, Distancia: 0

Nodo: B, Distancia: 5

Nodo: C, Distancia: 7

Nodo: D, Distancia: 19

Nodo: E, Distancia: 22

Nodo: F, Distancia: 27

Tiempo de ejecución: 0.0 segundos

## 4 Conclusiones

En resumen y como conclusión de esta práctica, este código implementa la clase Grafo con métodos para agregar vértices, aristas y encontrar las distancias más cortas entre nodos utilizando el algoritmo de Dijkstra, luego realiza ejemplos de uso con diferentes gráficos y muestra los resultados obtenidos demostrando su aplicación en diferentes gráficos para encontrar las distancias más cortas entre nodos.