



INSTITUTO POLITÉCNICO NACIONAL
Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas.

MATERIA:

- **Álgebra y Diseño de Algoritmos**

Practica 2:

- **Algoritmo de Dijkstra**

DOCENTE:

- **Erika Sánchez Femat**

Alumno:

- **Luis Eduardo Espino Gutierrez**

Introducción

El algoritmo de Dijkstra puede abordarse desde el contexto de la teoría de gráficos y la resolución de problemas relacionados con la búsqueda del camino más corto entre dos puntos en un gráfico ponderado. Desarrollado por el científico de la computación holandés Edsger Dijkstra en 1956, este algoritmo se ha convertido en una herramienta fundamental en el ámbito de la informática y la ingeniería de redes.

En esencia, el algoritmo de Dijkstra se utiliza para encontrar la ruta más corta entre un nodo de inicio y todos los demás nodos en un gráfico dirigido y ponderado.

Desarrollo

Codigo

Aquí estamos creando una clase llamada Graph para representar un gráfico ponderado dirigido.

```
import heapq
import time #libreria para el tiempo
class Graph:
    def __init__(self):
        self.vertices = set()
        self.edges = {}
```

En el constructor, se inicializan dos atributos: vertices(un conjunto para almacenar los vértices) y edges(un diccionario donde las claves son los vértices y los valores son listas de tuplas representando las aristas y sus pesos).

```
def add_vertex(self, value):
    self.vertices.add(value)
    self.edges[value] = []

    def add_edge(self, from_vertex, to_vertex, weight):
        self.edges[from_vertex].append((to_vertex, weight))
```

Este método implementa el algoritmo de Dijkstra para calcular las distancias mínimas desde un vértice de inicio dado a todos los demás vértices. Utiliza un diccionario distancespara almacenar las distancias mínimas y una cola de prioridad para explorar los vértices de manera eficiente.

```
def dijkstra(self, start_vertex):
    # Inicializar diccionario de distancias m nimas
    distances = {vertex: float('infinity') for vertex in self.vertices}
    distances[start_vertex] = 0

    # Inicializar cola de prioridad con la tupla (distancia, v rtice)
    priority_queue = [(0, start_vertex)]

    while priority_queue:
        current_distance, current_vertex = heapq.heappop(priority_queue)

        # Verificar si ya se ha encontrado un camino m s corto
```

```

        if current_distance > distances[current_vertex]:
            continue

        # Explorar los vecinos del v rtice actual
        for neighbor, weight in self.edges[current_vertex]:
            distance = current_distance + weight

            # Actualizar la distancia m nima si se encuentra un camino m s corto
            if distance < distances[neighbor]:
                distances[neighbor] = distance
                heapq.heappush(priority_queue, (distance, neighbor))

    return distances

```

Este bloque de código crea un gráfico de ejemplo, agrega vértices y aristas con pesos, y luego calcula e imprime las distancias mínimas desde el vértice "A" a todos los demás vértices.

```

def casoPrueba():

    print("Caso de prueba ")

    inicio = time.time()

    graph = Graph()

    # Agregar v rtices
    graph.add_vertex("A")
    graph.add_vertex("B")
    graph.add_vertex("C")
    graph.add_vertex("D")
    graph.add_vertex("E")
    graph.add_vertex("F")
    graph.add_vertex("G")
    graph.add_vertex("H")
    graph.add_vertex("I")
    graph.add_vertex("J")

    # Agregar aristas con pesos
    graph.add_edge("A", "D", 2)
    graph.add_edge("A", "C", 4)
    graph.add_edge("A", "B", 2)
    graph.add_edge("B", "E", 5)
    graph.add_edge("B", "F", 7)
    graph.add_edge("C", "F", 9)
    graph.add_edge("D", "G", 3)
    graph.add_edge("D", "H", 1)
    graph.add_edge("E", "I", 2)
    graph.add_edge("F", "I", 9)
    graph.add_edge("F", "J", 8)
    graph.add_edge("G", "J", 7)
    graph.add_edge("H", "J", 4)
    graph.add_edge("I", "J", 1)

```

```

# Calcular caminos minimos desde el vertice "A"
start_vertex = "A"
min_distances = graph.dijkstra(start_vertex)
fin= time.time()
tiempo_ejecutado = fin-inicio

# Mostrar resultados
for vertex, distance in min_distances.items():
    print(f"Distancia minima desde {start_vertex} a {vertex}: {distance}")

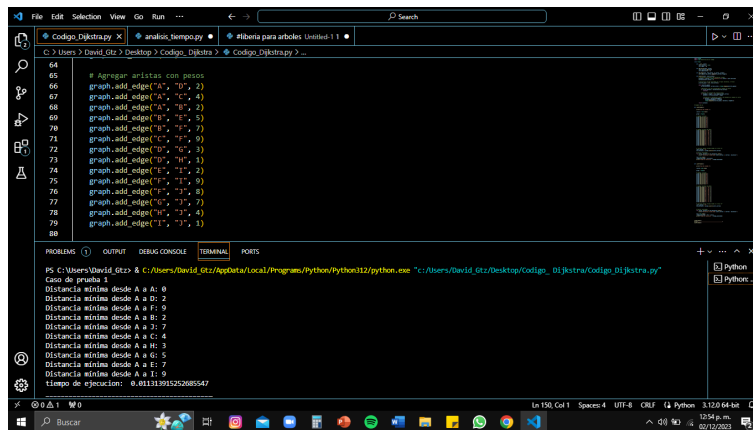
print("tiempo de ejecucion: ",tiempo_ejecutado)

casoPrueba()

```

Resultados

Resultado 1



The screenshot shows a code editor with a Python script and its execution output. The script defines a graph with 10 vertices (A-J) and 10 edges with weights. It then uses Dijkstra's algorithm to find the shortest path from vertex 'A' to all other vertices. The output in the terminal shows the minimum distances from 'A' to each vertex: A: 0, B: 2, C: 4, D: 7, E: 4, F: 5, G: 3, H: 7, I: 9, and J: 0. The total execution time is 0.011313915252685547 seconds.

```

# Agregar aristas con pesos
graph.add_edge("A", "B", 2)
graph.add_edge("A", "C", 4)
graph.add_edge("A", "D", 7)
graph.add_edge("A", "E", 5)
graph.add_edge("A", "F", 7)
graph.add_edge("C", "G", 0)
graph.add_edge("D", "G", 3)
graph.add_edge("D", "H", 1)
graph.add_edge("E", "H", 2)
graph.add_edge("F", "I", 0)
graph.add_edge("F", "J", 4)
graph.add_edge("H", "I", 7)
graph.add_edge("H", "J", 1)

```

```

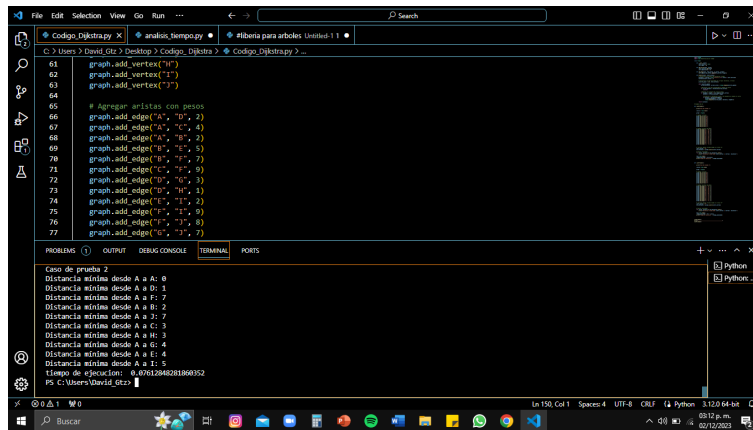
PS C:\Users\David_G12> & C:\Users\David_G12\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\David_G12\Desktop\Codigo_01jstra\Codigo_01jstra.py"
Caso de prueba 1
Distancia minima desde A a A: 0
Distancia minima desde A a B: 2
Distancia minima desde A a C: 4
Distancia minima desde A a D: 7
Distancia minima desde A a E: 5
Distancia minima desde A a F: 7
Distancia minima desde A a G: 3
Distancia minima desde A a H: 7
Distancia minima desde A a I: 9
Distancia minima desde A a J: 0
tiempo de ejecucion: 0.011313915252685547

```

Figure 1: imagen1, prueba1

En la imagen 1, observamos la primer prueba del codigo de dijkstra, añadimos 10 vertices y 10 edges, donde cada edges tiene un valor definido, nuestro vertice mas lejano de nuestro vertice inicial "A" fue "J", donde su camino mas corto para llegar del vertice "A" al vertice "J" fue de "7", ademas su tiempo de ejecucion de esta prueba en total, fue de 0.011313915252685547 segundos.

Resultado 2



```
61 graph.add_vertex("A")
62 graph.add_vertex("I")
63 graph.add_vertex("J")
64
65 # Agregar aristas con pesos
66 graph.add_edge("A", "D", 2)
67 graph.add_edge("A", "C", 4)
68 graph.add_edge("A", "B", 2)
69 graph.add_edge("B", "I", 5)
70 graph.add_edge("B", "J", 7)
71 graph.add_edge("C", "I", 9)
72 graph.add_edge("D", "I", 3)
73 graph.add_edge("D", "J", 2)
74 graph.add_edge("E", "I", 2)
75 graph.add_edge("E", "J", 8)
76 graph.add_edge("F", "I", 3)
77 graph.add_edge("F", "J", 7)
```

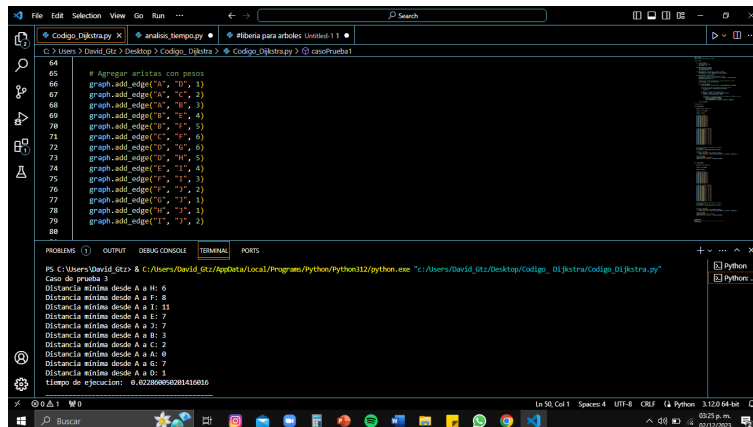
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Caso de prueba 2
Distancia mínima desde A a A: 0
Distancia mínima desde A a D: 1
Distancia mínima desde A a F: 7
Distancia mínima desde A a B: 2
Distancia mínima desde A a J: 7
Distancia mínima desde A a C: 3
Distancia mínima desde A a H: 3
Distancia mínima desde A a G: 4
Distancia mínima desde A a E: 4
Distancia mínima desde A a I: 5
Tiempo de ejecución: 0.07612848281860352
PS C:\Users\David_Gtz>

Figure 2: prueba 2

En la imagen 2, observamos la segunda prueba del código de dijkstra, añadimos 10 vértices y 10 edges, donde cada edge tiene un valor definido diferente a los de la prueba 1, nuestro vértice más lejano de nuestro vértice inicial "A" fue "J", donde su camino más corto para llegar del vértice "A" al vértice "J" fue de "7", además su tiempo de ejecución de esta prueba en total, fue de 0.07612848281860352 segundos.

Resultado 3



```
64
65 # Agregar aristas con pesos
66 graph.add_edge("A", "D", 3)
67 graph.add_edge("A", "C", 2)
68 graph.add_edge("A", "B", 3)
69 graph.add_edge("B", "I", 4)
70 graph.add_edge("B", "J", 5)
71 graph.add_edge("C", "I", 6)
72 graph.add_edge("D", "I", 6)
73 graph.add_edge("D", "H", 5)
74 graph.add_edge("E", "I", 4)
75 graph.add_edge("E", "J", 3)
76 graph.add_edge("F", "I", 2)
77 graph.add_edge("F", "J", 1)
78 graph.add_edge("G", "I", 3)
79 graph.add_edge("G", "J", 2)
80
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

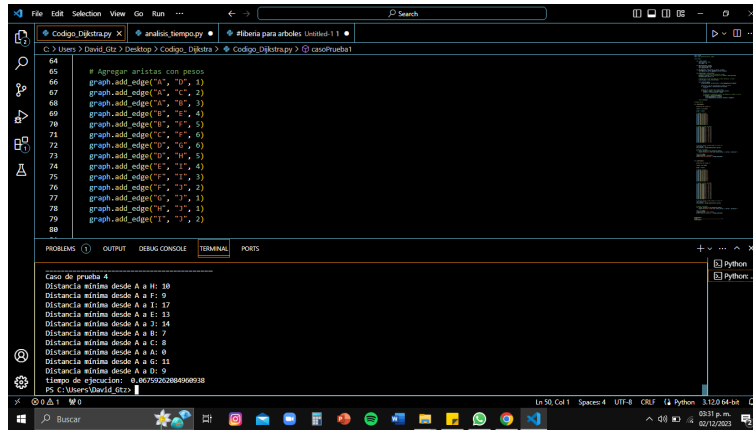
PS C:\Users\David_Gtz> C:\Users\David_Gtz\AppData\Local\Programs\Python\Python312\python.exe "C:\Users\David_Gtz\Desktop\Codigo_01\kstra/Codigo_01\kstra.py"

Caso de prueba 3
Distancia mínima desde A a H: 6
Distancia mínima desde A a F: 8
Distancia mínima desde A a I: 11
Distancia mínima desde A a E: 7
Distancia mínima desde A a J: 7
Distancia mínima desde A a B: 3
Distancia mínima desde A a C: 2
Distancia mínima desde A a A: 0
Distancia mínima desde A a G: 7
Distancia mínima desde A a D: 4
Tiempo de ejecución: 0.022860050201416016

Figure 3: prueba 3

En la imagen 3, observamos la tercera prueba del código de dijkstra, añadimos 10 vértices y 10 edges, donde cada edge tiene un valor definido diferente a los de la prueba 1 y 2, nuestro vértice más lejano de nuestro vértice inicial "A" fue "J", donde su camino más corto para llegar del vértice "A" al vértice "J" fue de "7", además su tiempo de ejecución de esta prueba en total, fue de 0.022860050201416016 segundos.

Resultado 4



```
64
65 # Agregar aristas con pesos
66 graph.add_edge("A", "D", 1)
67 graph.add_edge("A", "C", 2)
68 graph.add_edge("A", "B", 3)
69 graph.add_edge("B", "E", 4)
70 graph.add_edge("B", "F", 5)
71 graph.add_edge("C", "I", 6)
72 graph.add_edge("D", "G", 6)
73 graph.add_edge("E", "H", 5)
74 graph.add_edge("F", "I", 4)
75 graph.add_edge("F", "J", 3)
76 graph.add_edge("G", "J", 2)
77 graph.add_edge("H", "I", 1)
78 graph.add_edge("I", "J", 4)
79 graph.add_edge("I", "G", 2)
80
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Caso de prueba 4

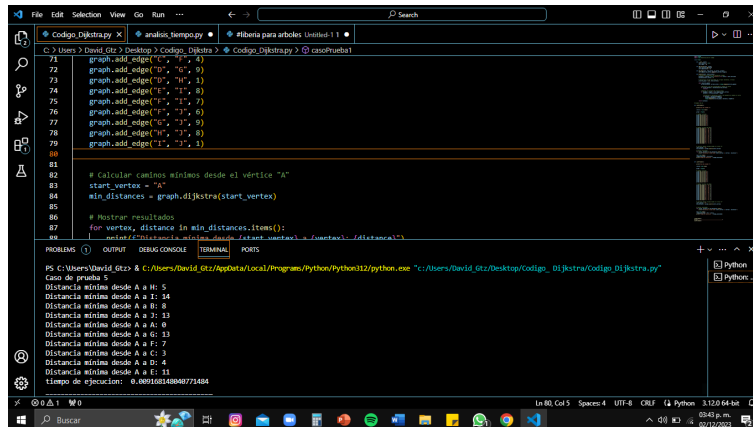
Distancia minima desde A a H: 10
Distancia minima desde A a F: 9
Distancia minima desde A a I: 13
Distancia minima desde A a J: 14
Distancia minima desde A a B: 3
Distancia minima desde A a C: 2
Distancia minima desde A a D: 1
Distancia minima desde A a G: 11
Distancia minima desde A a E: 7
Distancia minima desde A a J: 14
Tiempo de ejecucion: 0.06759262084960938

PS C:\Users\David> g++

Figure 4: prueba 4

En la imagen 4, observamos la cuarta prueba del codigo de dijkstra, añadimos 10 vertices y 10 edges, donde cada edges tiene un valor definido diferentes a los de la prueba 1,2 y 3, nuestro vertice mas lejano de nuestro vertice inicial "A" fue "J", donde su camino mas corto para llegar del vertice "A" al vertice "J" fue de "14", ademas su tiempo de ejecucion de esta prueba en total, fue de 0.06759262084960938 segundos.

Resultado 5



```
71 graph.add_edge("C", "I", 5)
72 graph.add_edge("D", "G", 6)
73 graph.add_edge("E", "H", 5)
74 graph.add_edge("F", "I", 4)
75 graph.add_edge("F", "J", 3)
76 graph.add_edge("G", "J", 2)
77 graph.add_edge("H", "I", 1)
78 graph.add_edge("I", "J", 4)
79 graph.add_edge("I", "G", 2)
80
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\David> g++ & C:\Users\David> g++ -std=c++11 -c C:\Users\David\Desktop\Codigo_01\kstra\Codigo_01\kstra.py

Caso de prueba 5

Distancia minima desde A a H: 5
Distancia minima desde A a J: 14
Distancia minima desde A a B: 3
Distancia minima desde A a C: 13
Distancia minima desde A a D: 0
Distancia minima desde A a F: 7
Distancia minima desde A a G: 3
Distancia minima desde A a I: 4
Distancia minima desde A a J: 11
Tiempo de ejecucion: 0.009168148040771484

Figure 5: prueba 5

En la imagen 5, observamos la quinta prueba del codigo de dijkstra, añadimos 10 vertices y 10 edges, donde cada edges tiene un valor definido diferentes a los de la prueba 1,2,3 y 4, nuestro vertice mas lejano de nuestro vertice inicial "A" fue "J", donde su camino mas corto para llegar del vertice "A" al vertice "J" fue de "13", ademas su tiempo de ejecucion de esta prueba en total, fue de 0.009168148040771484 segundos.

Complejidad

El algoritmo de Dijkstra es un algoritmo eficiente de complejidad $O(n^2)$, donde “n” es el número de vértices, en este caso “n” tiene un valor de 10, lo que resulta en una complejidad de $O(10^2)$ que sirve para encontrar el camino de coste mínimo desde un nodo origen a todos los demás nodos del grafo.

Conclusión

El algoritmo de Dijkstra tiene aplicaciones significativas en la vida diaria y se utiliza en una variedad de contextos, algunas aplicaciones incluyen redes de transporte, redes de computadoras, en la planificación urbana, además de poder dar más optimización a los procesos y algo muy importante en los sistemas de información geográfica.

En general, la aplicación del algoritmo de Dijkstra en la vida diaria demuestra cómo las técnicas algorítmicas pueden tener un impacto sustancial en la eficiencia y la optimización de procesos en diversos campos. La capacidad de encontrar las rutas más cortas en un gráfico ponderado es una herramienta valiosa en la resolución de problemas prácticos en la sociedad contemporánea.