CENTRO DE ENSEÑASA TÉCNICA INDUSTRIAL

INTELIGENCIA ARTIFICIAL

PRÁCTICA 4 SIMULADOR ÁRBOL PARCIAL MÍNIMO DE PRIM

PRECIADO MARTINEZ BRUNO AUGUSTO

N: 21110311

6 E

Parte Teórica

¿Qué ES? Un simulador de Árbol de Expansión Mínima (Minimum Spanning Tree, MST) de Prim es una aplicación o programa que permite visualizar y entender el funcionamiento del algoritmo de Prim para encontrar un MST en un grafo ponderado. El MST de un grafo es un subgrafo que contiene todos los vértices del grafo original y un subconjunto de aristas que conectan estos vértices de manera que se minimice el costo total

¿Para qué sirve? Enseñanza y Aprendizaje: Proporciona una forma efectiva de enseñar y aprender el funcionamiento del algoritmo de Prim para encontrar un MST en un grafo ponderado. Los estudiantes y entusiastas de la informática pueden visualizar cómo se seleccionan las aristas y cómo se construye el MST paso a paso.

Comprensión de Algoritmos: Ayuda a las personas a comprender y apreciar los algoritmos de grafos, en particular el algoritmo de Prim. Al permitir una representación gráfica y paso a paso, facilita la comprensión de los conceptos detrás del MST y cómo se optimiza para encontrar la solución óptima.

¿Cómo se implementa en el mundo? Redes de Comunicación: En la planificación y diseño de redes de comunicación, como redes telefónicas, redes de datos, y redes eléctricas, el algoritmo de Prim se utiliza para encontrar la ruta más eficiente y de menor costo que conecta todos los nodos de la red.

Construcción de Carreteras y Rutas: En ingeniería civil y planificación de transporte, se emplea el MST de Prim para determinar la disposición de carreteras, rutas de ferrocarril y otros medios de transporte para conectar ciudades o ubicaciones de manera eficiente.

¿Cómo lo implementarías en tu vida? Diseño de Circuitos Electrónicos: el algoritmo de Prim puede utilizarse para el diseño de PCB y la colocación de componentes electrónicos de manera que se minimicen las conexiones y se optimice la eficiencia de la placa.

¿Cómo lo implementarías en tu trabajo o tu trabajo de ensueño? Un algoritmo que me diseñe los espacios adecuados en los gabinetes de plc para que queden de la forma más optima y también al optimización de los cableados de las entradas y salidas de los gabinetes de plc

Código

```python
import sys
import matplotlib.pyplot as plt

class PrimMST:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for _ in range(vertices)] for _ in range(vertices)]

    def add_edge(self, u, v, weight):
        self.graph[u][v] = weight
        self.graph[v][u] = weight

    def min_key(self, key, mst_set):
        min_val = sys.maxsize
        min_index = None

        for v in range(self.V):
            if key[v] < min_val and not mst_set[v]:
                min_val = key[v]
                min_index = v

        return min_index

    def prim_mst(self):
        parent = [-1] * self.V
        key = [sys.maxsize] * self.V
        key[0] = 0
        mst_set = [False] * self.V

        for _ in range(self.V):
            u = self.min_key(key, mst_set)
            mst_set[u] = True

            for v in range(self.V):
                if self.graph[u][v] > 0 and not mst_set[v] and key[v] > self.graph[u][v]:
                    key[v] = self.graph[u][v]
                    parent[v] = u

            self.print_mst_step(u, parent, key)

    def print_mst_step(self, u, parent, key):
        print("Edge \tWeight")
        total_weight = 0
        for i in range(1, self.V):
```

```
            for v in range(self.V):
                if self.graph[u][v] > 0 and not mst_set[v] and key[v] > self.graph[u][v]:
                    key[v] = self.graph[u][v]
                    parent[v] = u

            self.print_mst_step(u, parent, key)

    def print_mst_step(self, u, parent, key):
        print("Edge \tWeight")
        total_weight = 0
        for i in range(1, self.V):
            print("{} - {}\t{}".format(parent[i], i, self.graph[i][parent[i]]))
            total_weight += self.graph[i][parent[i]]
        print("Total Weight of MST: {}".format(total_weight))

    def draw_graph(self):
        G = plt.figure()
        pos = {i: (i * 30, 0) for i in range(self.V)}

        for i in range(self.V):
            for j in range(i, self.V):
                if self.graph[i][j] > 0:
                    plt.plot([i, j], [0, 0], 'ro-')
                    plt.text((i + j) / 2, -2, str(self.graph[i][j]))

        plt.title("Minimum Spanning Tree")
        plt.show()

if __name__ == "__main__":
    vertices = 5
    g = PrimMST(vertices)

    g.add_edge(0, 1, 2)
    g.add_edge(0, 3, 6)
    g.add_edge(1, 2, 3)
    g.add_edge(1, 3, 8)
    g.add_edge(1, 4, 5)
    g.add_edge(2, 4, 7)
    g.add_edge(3, 4, 9)

    g.draw_graph()
    g.prim_mst()
```

Resultado

```
Edge    Weight
0 - 1   2
-1 - 2  7
0 - 3   6
-1 - 4  0
Total Weight of MST: 15
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
Total Weight of MST: 16
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
Total Weight of MST: 16
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
Total Weight of MST: 16
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
Total Weight of MST: 16
```