Inteligencia Artificial

Ingenieria en mecatrónica 6E1

Practica #5

Alumno(a): Alma Paola Garcia Landeros
Maestro(a): Mauricio Alejandro Cabrera Arellano

Árbol de Máximo y Mínimo Coste Kruskal

¿Qué es?

El Árbol de Máximo y Mínimo Coste Kruskal es un algoritmo que se utiliza para encontrar dos árboles de expansión mínima y máxima en un grafo ponderado y conexo. Estos árboles conectan todos los nodos del grafo sin formar ciclos y tienen el costo total mínimo y máximo posible, respectivamente.

¿Para qué sirve?

Sirve para optimizar la conexión de redes, rutas de transporte, y cualquier escenario donde se necesite encontrar tanto la ruta más corta como la más larga entre puntos específicos en un sistema con múltiples nodos y conexiones.

¿Cómo se implementa en el mundo?

En el mundo real, el Árbol de Máximo y Mínimo Coste Kruskal se implementa en aplicaciones como la planificación de redes de telecomunicaciones, la distribución de energía eléctrica, el diseño de rutas de transporte eficientes, y en la optimización de sistemas logísticos. Por ejemplo, en el diseño de redes de telecomunicaciones, se utiliza para garantizar que la red tenga tanto la ruta más corta como la ruta más larga posible entre puntos clave.

¿Cómo lo implementarías en tu vida?

En la vida personal, podría usar el concepto del Árbol de Máximo y Mínimo Coste Kruskal para planificar mis viajes de manera eficiente. Por ejemplo, al organizar un viaje por múltiples ciudades, podría usar el algoritmo para encontrar tanto la ruta más corta (menor tiempo de viaje) como la ruta más larga (más lugares visitados) para maximizar la experiencia de viaje.

¿Cómo lo implementarías en tu trabajo o tu trabajo de ensueño?

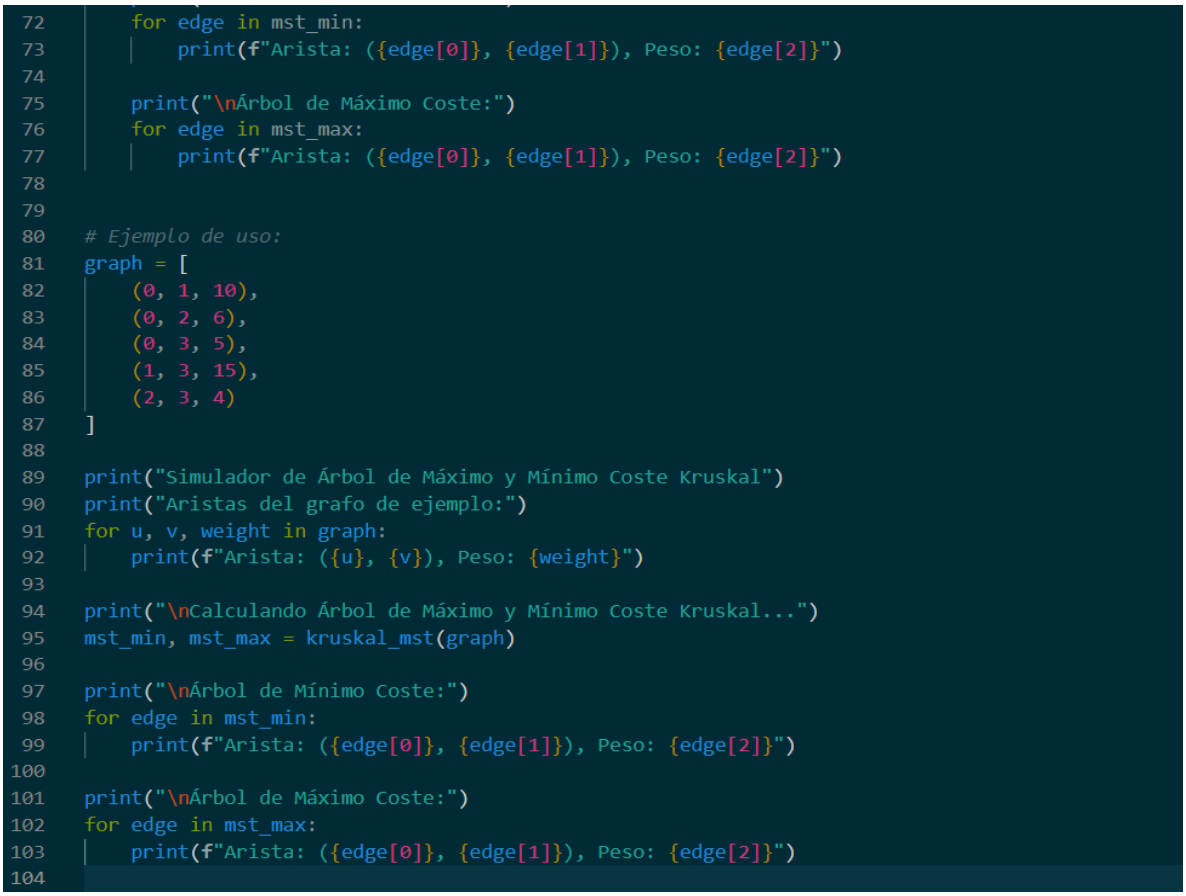
Como diseñador en el área de diseño asistido por computadora para proyectos de automatización, el Árbol de Máximo y Mínimo Coste Kruskal sería invaluable. Podría utilizarlo para optimizar la disposición de componentes y sistemas dentro de una planta industrial o de producción, asegurando tanto la eficiencia en el transporte de materiales y productos (ruta más corta) como la redundancia en la conexión de sistemas críticos (ruta más larga).

Por ejemplo, al diseñar un sistema de automatización para una fábrica, podría emplear el algoritmo para determinar la disposición óptima de sensores, actuadores y rutas de comunicación dentro del entorno de producción. Esto no solo reduciría los costos operativos optimizando las distancias y tiempos de respuesta, sino que también garantizaría una mayor fiabilidad y capacidad de respuesta del sistema ante posibles fallos.

Algoritmo:

```python
class UnionFind:
    def __init__(self, n):
        self.parent = list(range(n))
        self.rank = [1] * n

    def find(self, u):
        if self.parent[u] != u:
            self.parent[u] = self.find(self.parent[u])
        return self.parent[u]

    def union(self, u, v):
        root_u = self.find(u)
        root_v = self.find(v)

        if root_u != root_v:
            if self.rank[root_u] > self.rank[root_v]:
                self.parent[root_v] = root_u
            elif self.rank[root_u] < self.rank[root_v]:
                self.parent[root_u] = root_v
            else:
                self.parent[root_v] = root_u
                self.rank[root_u] += 1
            return True
        return False

def kruskal_mst(graph):
    """
    Función para encontrar el Árbol de Mínimo y Máximo Coste usando el algoritmo de Kruskal.

    Args:
    - graph: Lista de tuplas (u, v, peso) representando las aristas del grafo ponderado.

    Returns:
    - mst_min: Lista de aristas que forman el Árbol de Mínimo Coste.
    - mst_max: Lista de aristas que forman el Árbol de Máximo Coste.
    """
    n = len(graph)
```

```python
    mst_min = []
    mst_max = []

    # Ordena las aristas por peso
    graph.sort(key=lambda x: x[2])

    uf = UnionFind(n)

    for u, v, weight in graph:
        if uf.union(u, v):
            mst_min.append((u, v, weight))
            mst_max.append((u, v, weight))
        else:
            mst_max.pop()  # Elimina la última arista añadida en mst_max

        # Muestra paso a paso el progreso del algoritmo
        print_step_by_step(mst_min, mst_max, u, v, weight)

    return mst_min, mst_max

def print_step_by_step(mst_min, mst_max, u, v, weight):
    """
    Muestra paso a paso la construcción del Árbol de Mínimo y Máximo Coste.

    Args:
    - mst_min: Lista de aristas del Árbol de Mínimo Coste.
    - mst_max: Lista de aristas del Árbol de Máximo Coste.
    - u: Nodo u de la arista añadida.
    - v: Nodo v de la arista añadida.
    - weight: Peso de la arista añadida.
    """
    print(f"\nArista añadida: ({u}, {v}), Peso: {weight}")

    print("\nÁrbol de Mínimo Coste:")
```



```
    for edge in mst_min:
        print(f"Arista: ({edge[0]}, {edge[1]}), Peso: {edge[2]}")

    print("\nÁrbol de Máximo Coste:")
    for edge in mst_max:
        print(f"Arista: ({edge[0]}, {edge[1]}), Peso: {edge[2]}")


# Ejemplo de uso:
graph = [
    (0, 1, 10),
    (0, 2, 6),
    (0, 3, 5),
    (1, 3, 15),
    (2, 3, 4)
]

print("Simulador de Árbol de Máximo y Mínimo Coste Kruskal")
print("Aristas del grafo de ejemplo:")
for u, v, weight in graph:
    print(f"Arista: ({u}, {v}), Peso: {weight}")

print("\nCalculando Árbol de Máximo y Mínimo Coste Kruskal...")
mst_min, mst_max = kruskal_mst(graph)

print("\nÁrbol de Mínimo Coste:")
for edge in mst_min:
    print(f"Arista: ({edge[0]}, {edge[1]}), Peso: {edge[2]}")

print("\nÁrbol de Máximo Coste:")
for edge in mst_max:
    print(f"Arista: ({edge[0]}, {edge[1]}), Peso: {edge[2]}")
```

Este código nos permitirá visualizar paso a paso cómo se construyen tanto el Árbol de Mínimo Coste como el Árbol de Máximo Coste utilizando el algoritmo de Kruskal en un grafo dado, mostrando las aristas seleccionadas en cada paso y los árboles resultantes al final.