Inteligencia Artificial

Ingenieria en mecatrónica 6E1

Practica #4

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

Árbol Parcial Mínimo de Prim

¿Qué es?

El Árbol Parcial Mínimo de Prim es un algoritmo utilizado para encontrar un árbol de expansión mínima en un grafo con pesos no negativos. Es decir, dado un grafo conexo y ponderado, el algoritmo encuentra un subconjunto de aristas que conectan todos los nodos del grafo sin formar ciclos y con el peso total mínimo posible.

¿Para qué sirve?

Sirve principalmente para optimizar redes de comunicación, como por ejemplo en la planificación de redes de telecomunicaciones, distribución de energía eléctrica, diseño de redes de transporte, entre otros. También se utiliza en la resolución de problemas de optimización en logística y en la planificación de rutas eficientes.

¿Cómo se implementa en el mundo?

En el mundo real, el Árbol Parcial Mínimo de Prim se implementa utilizando computadoras y software especializado en algoritmos de grafos. Por ejemplo, en sistemas de gestión de redes de telecomunicaciones, el algoritmo puede ser utilizado para determinar la disposición más eficiente de torres de comunicación minimizando el costo de infraestructura.

¿Cómo lo implementarías en tu vida?

En mi vida personal, podría utilizar el concepto del Árbol Parcial Mínimo de Prim para planificar viajes o rutas diarias de manera eficiente. Por ejemplo, al organizar un viaje por varias ciudades, podría utilizar el algoritmo para determinar la ruta más corta y económica que visite todas las ciudades deseadas sin hacer recorridos innecesarios.

¿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 Parcial Mínimo de Prim sería fundamental. Podría utilizarlo para diseñar redes de automatización eficientes, determinando la disposición óptima de sensores, actuadores y componentes de control. Por ejemplo, al diseñar un sistema de automatización para una planta industrial, podría emplear el algoritmo para minimizar la longitud de los cables y tuberías necesarios, reduciendo costos y mejorando la eficiencia del sistema.

Algoritmo:

```python
import sys

def prim_mst(graph):
    """
    Función para encontrar el Árbol Parcial Mínimo de Prim en un grafo dado.

    Args:
    - graph: Grafo representado como una matriz de adyacencia.

    Returns:
    - mst: Lista de aristas que forman el Árbol Parcial Mínimo.
    """
    n = len(graph)
    mst = []  # Lista para almacenar el Árbol Parcial Mínimo
    min_weight = [float('inf')] * n  # Inicializa todas las distancias como infinito
    parent = [-1] * n  # Lista para rastrear el nodo padre en el Árbol Parcial Mínimo
    visited = [False] * n  # Lista para rastrear nodos visitados

    # Comienza desde el primer nodo
    min_weight[0] = 0
    parent[0] = -1

    for _ in range(n):
        # Encuentra el nodo no visitado con la mínima distancia actual
        u = min_distance_node(min_weight, visited)
        visited[u] = True

        # Si no es el primer nodo, agrega la arista al MST
        if parent[u] != -1:
            mst.append((parent[u], u))

        # Actualiza las distancias de los nodos adyacentes no visitados
        for v in range(n):
            if not visited[v] and graph[u][v] < min_weight[v]:
                min_weight[v] = graph[u][v]
                parent[v] = u
```

```python
        # Muestra paso a paso el progreso del MST
        print_step_by_step(graph, mst, min_weight, visited, u)

    return mst

def min_distance_node(min_weight, visited):
    """
    Encuentra el nodo no visitado con la mínima distancia actual.

    Args:
    - min_weight: Lista de mínimas distancias a cada nodo.
    - visited: Lista que indica si cada nodo ha sido visitado.

    Returns:
    - El nodo no visitado con la mínima distancia.
    """
    min_dist = float('inf')
    min_node = -1

    for i in range(len(min_weight)):
        if not visited[i] and min_weight[i] < min_dist:
            min_dist = min_weight[i]
            min_node = i

    return min_node

def print_step_by_step(graph, mst, min_weight, visited, u):
    """
    Muestra paso a paso la construcción del MST.

    Args:
    - graph: Grafo representado como una matriz de adyacencia.
    - mst: Lista de aristas que forman el Árbol Parcial Mínimo.
    - min_weight: Lista de mínimas distancias a cada nodo.
```

```
    - visited: Lista que indica si cada nodo ha sido visitado.
    - u: Nodo actual en el paso actual.
    """
    print(f"\nExplorando nodo {u}. Aristas en el Árbol Parcial Mínimo hasta el momento:")
    for edge in mst:
        print(f"Arista: {edge[0]} - {edge[1]}")

    print("\nDistancias mínimas actuales:")
    for i in range(len(min_weight)):
        print(f"Nodo {i}: {min_weight[i]}")

    print("\nNodos visitados:")
    for i in range(len(visited)):
        if visited[i]:
            print(f"Nodo {i}: Visitado")
        else:
            print(f"Nodo {i}: No visitado")

# Ejemplo de uso:
graph = [
    [0, 2, 0, 6, 0],
    [2, 0, 3, 8, 5],
    [0, 3, 0, 0, 7],
    [6, 8, 0, 0, 9],
    [0, 5, 7, 9, 0]
]

print("Simulador de Árbol Parcial Mínimo de Prim")
print("Grafo de ejemplo:")
for row in graph:
    print(row)

print("\nCalculando Árbol Parcial Mínimo de Prim...")
mst = prim_mst(graph)
```

```

 print("\nÁrbol Parcial Mínimo de Prim:")
 for edge in mst:
    print(f"Arista: {edge[0]} - {edge[1]}")

```

Este código nos permitirá visualizar paso a paso cómo se construye el Árbol Parcial Mínimo de Prim en un grafo dado, mostrando las aristas seleccionadas en cada paso y el estado actual de las distancias y nodos visitados.