Camberos Cordova Carlos Raúl

20310415

Inteligencia Artificial

6E2

Ingeniería en Mecatrónica

Practica 4

Mauricio Alejandro Cabrera Arellano

Marco Teórico

Realizar un simulador Árbol Parcial mínimo de Prim.

En consola que muestre paso a paso es lo mínimo, si logran parte gráfica puntitos extras.

Parte Teórica

¿Qué es?
¿Para qué sirve?
¿Cómo se implementa en el mundo?
¿Cómo lo implementarías en tu vida?
¿Cómo lo implementarías en tu trabajo o tu trabajo de ensueño?

Que es

El código es una implementación del algoritmo de Prim para encontrar el árbol de expansión mínima de un grafo no dirigido y conexo. Este algoritmo es útil en el campo de la teoría de grafos y es ampliamente utilizado en problemas de optimización en redes, en particular para la planificación de redes de transporte.

Para que sirve

El algoritmo de Prim funciona al comenzar desde un nodo arbitrario y agregando iterativamente la arista de menor peso conectada al conjunto de nodos visitados. En cada iteración, se selecciona la arista de menor peso que conecta un vértice visitado con uno no visitado y se agrega al conjunto de aristas del árbol. El proceso continúa hasta que todos los vértices del grafo han sido visitados.

Como se implementa en el mundo

El algoritmo del árbol parcial mínimo de Prim se utiliza en muchas aplicaciones en el mundo real, especialmente en problemas de optimización de redes. Algunos ejemplos incluyen:

Redes de transporte: El algoritmo de Prim se utiliza para encontrar la red de transporte óptima en la que se minimiza la distancia total recorrida. Esto se aplica en la construcción de carreteras, ferrocarriles y sistemas de transporte público.

Redes de telecomunicaciones: El algoritmo de Prim se utiliza para encontrar la red de telecomunicaciones más eficiente en la que se minimiza la cantidad de hardware y la longitud total de los cables. Esto se aplica en la construcción de redes de fibra óptica y sistemas de telefonía.

Sistemas de suministro de energía: El algoritmo de Prim se utiliza para encontrar la red de suministro de energía óptima en la que se minimiza la cantidad de energía perdida y se maximiza la eficiencia de la red. Esto se aplica en la construcción de redes de suministro de energía eléctrica.

En general, el algoritmo de Prim se utiliza en cualquier situación en la que se deba encontrar un subconjunto de aristas de un grafo ponderado que conecten todos los vértices de manera que se minimice el peso total de las aristas.

Para implementarlo en el mundo, es común utilizar simuladores y herramientas informáticas que permitan aplicar el algoritmo de manera rápida y eficiente en grandes conjuntos de datos y redes complejas. También se puede implementar manualmente en proyectos de ingeniería y construcción que involucren la planificación de redes.

¿Cómo lo implementarías en tu vida?

Para implementar el algoritmo en la vida real, se puede utilizar para la planificación de redes de transporte, la asignación de recursos en la industria, la planificación de rutas en sistemas de transporte público, entre otros casos. En la vida personal, podría utilizarse para planificar la ruta más eficiente para visitar varios lugares turísticos en una ciudad determinada.

¿Cómo lo implementaría en mi trabajo o mi trabajo de ensueño?

Yo quiero trabajar en un lugar en el área de automatización más especifico con los robots en una ensambladora de autos

El algoritmo del árbol parcial mínimo de Prim se utiliza comúnmente en la optimización de rutas y la planificación de trayectorias para robots en la industria de la automatización.

En el contexto de una ensambladora de automóviles, este algoritmo puede ser utilizado para planificar la ruta más eficiente para un robot en el ensamblaje de un automóvil. Por ejemplo, se pueden modelar los diferentes componentes del automóvil como vértices en un grafo, y las rutas entre los componentes como aristas con pesos que representan la distancia o el tiempo necesario para que el robot se mueva de un componente a otro.

El algoritmo del árbol parcial mínimo de Prim se puede utilizar entonces para encontrar el subconjunto mínimo de aristas que conecta todos los componentes del automóvil con la menor distancia total, lo que resulta en una ruta óptima para el robot en la ensambladora.

En cuanto a la implementación práctica, sería necesario contar con un software de simulación que permita modelar el proceso de ensamblaje del automóvil y la trayectoria del robot, y que incorpore el algoritmo del árbol parcial mínimo de Prim

para la planificación de rutas. También sería importante contar con un equipo de ingenieros capacitados en la programación y configuración de robots industriales, así como en el diseño y optimización de procesos de ensamblaje de automóviles.

Código

```
#Camberos Cordova Carlos Raul 20310415
import heapq
import networkx as nx
import matplotlib.pyplot as plt



#La clase Arista define una arista en el grafo, con un destino y un peso.
class Arista:
    def __init__(self, destino, peso):
        self.destino = destino
        self.peso = peso
#La clase Grafo define un grafo, con una lista de vértices y una lista de aristas.
class Grafo:
    def __init__(self):
        self.vertices = {}
#El método agregar_vertice agrega un vértice al grafo
    def agregar_vertice(self, v):
        self.vertices[v] = []


#mientras que agregar_arista agrega una arista con un peso determinado entre dos vértices.
    def agregar_arista(self, v1, v2, peso):
        self.vertices[v1].append((v2, peso))
        self.vertices[v2].append((v1, peso))
```

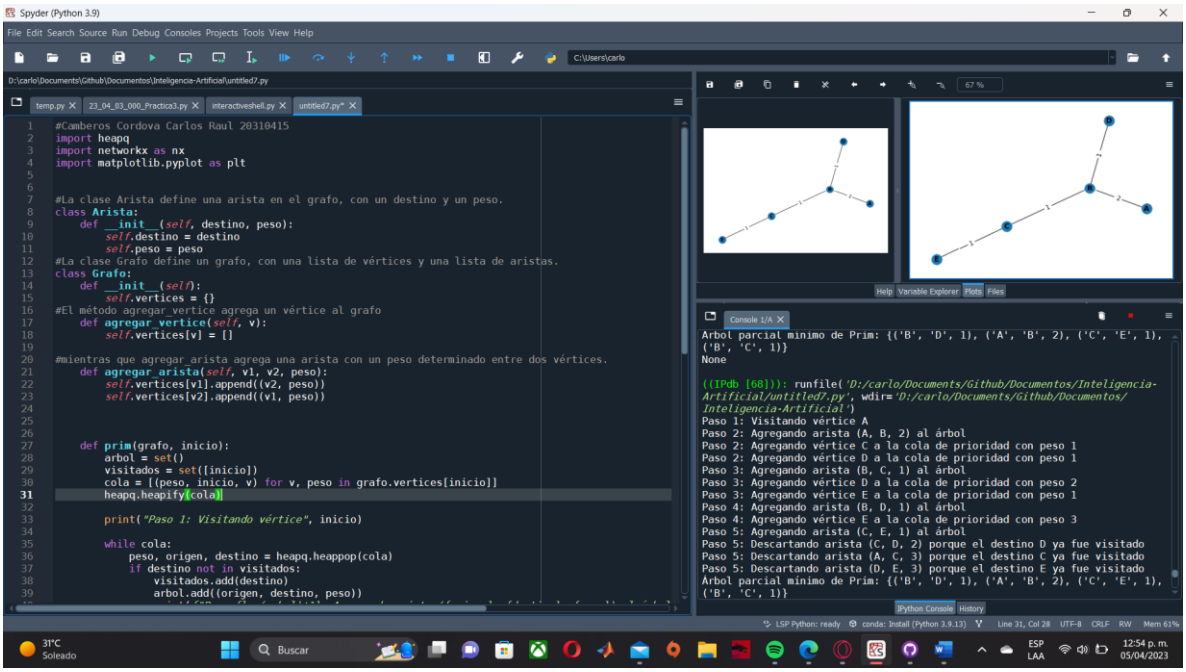
```
    def prim(grafo, inicio):
        arbol = set()
        visitados = set([inicio])
        cola = [(peso, inicio, v) for v, peso in grafo.vertices[inicio]]
        heapq.heapify(cola)

        print("Paso 1: Visitando vértice", inicio)

        while cola:
            peso, origen, destino = heapq.heappop(cola)
            if destino not in visitados:
                visitados.add(destino)
                arbol.add((origen, destino, peso))
                print(f"Paso {len(arbol)+1}: Agregando arista ({origen}, {destino}, {peso}) al árbol")
                for v, peso in grafo.vertices[destino]:
                    if v not in visitados:
                        heapq.heappush(cola, (peso, destino, v))
                        print(f"Paso {len(arbol)+1}: Agregando vértice {v} a la cola de prioridad con peso {peso}")
            else:
                print(f"Paso {len(arbol)+1}: Descartando arista ({origen}, {destino}, {peso}) porque el destino {destino} ya fue visitado")

        return arbol
```

```
# Ejemplo de uso
g = Grafo()
g.agregar_vertice('A')
g.agregar_vertice('B')
g.agregar_vertice('C')
g.agregar_vertice('D')
g.agregar_vertice('E')
g.agregar_arista('A', 'B', 2)
g.agregar_arista('A', 'C', 3)
g.agregar_arista('B', 'C', 1)
g.agregar_arista('B', 'D', 1)
g.agregar_arista('C', 'D', 2)
g.agregar_arista('C', 'E', 1)
g.agregar_arista('D', 'E', 3)
arbol = g.prim('A')
print("Árbol parcial mínimo de Prim:", arbol)

# Crear el grafo visual
G = nx.Graph()
for v in g.vertices.keys():
    G.add_node(v)
for a in arbol:
    G.add_edge(a[0], a[1], weight=a[2])

# Dibujar el grafo visual
pos = nx.spring_layout(G)
nx.draw(G, pos, with_labels=True, font_weight='bold')
labels = nx.get_edge_attributes(G,'weight')
```

nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

plt.show()

Evidencia:



Desarrollo

Este código es una implementación del algoritmo de Prim para encontrar el árbol parcial mínimo de un grafo ponderado. A continuación, se describen las diferentes partes del código:

La clase Arista define una arista en el grafo, con un destino y un peso.

La clase Grafo define un grafo, con una lista de vértices y una lista de aristas. El método agregar vértice agrega un vértice al grafo, mientras que agregar arista agrega una arista con un peso determinado entre dos vértices.

La función prim recibe como parámetros el grafo y un vértice de inicio. Utiliza un conjunto arbol para almacenar las aristas del árbol parcial mínimo encontrado, un conjunto visitado para almacenar los vértices ya visitados, y una cola de prioridad cola para almacenar las aristas candidatas a ser agregadas al árbol. El algoritmo comienza visitando el vértice de inicio y agregando las aristas adyacentes a la cola de prioridad. Luego, mientras hay aristas en la cola, se extrae la de menor peso y se verifica si su destino ha sido visitado. Si no ha sido visitado, se agrega al árbol y se agregan sus aristas adyacentes a la cola de prioridad. Si ya ha sido visitado, se descarta la arista. El algoritmo termina cuando no hay más aristas en la cola de prioridad.

El ejemplo de uso crea un grafo con cinco vértices y siete aristas, y aplica el algoritmo de Prim para encontrar el árbol parcial mínimo con origen en el vértice 'A'.

Finalmente, se crea un grafo visual utilizando la librería NetworkX de Python, y se dibuja utilizando la librería Matplotlib.