Camberos Cordova Carlos Raúl

20310415

Inteligencia Artificial

6E2

Ingeniería en Mecatrónica

Practica 5

Mauricio Alejandro Cabrera Arellano

Marco teórico

Realizar un simulador Árbol de Máximo y Mínimo coste Kruskal.

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?

Agregar al PDF su repositorio de Git

¿Qué es?

El árbol de máximo y mínimo coste (también conocido como el árbol de expansión mínima) es un subgrafo de un grafo no dirigido y ponderado que conecta todos los vértices del grafo con el mínimo peso total posible. El algoritmo de Kruskal es un algoritmo greedy utilizado para encontrar este árbol de expansión mínima. El algoritmo comienza por ordenar todas las aristas del grafo por peso y luego selecciona la arista de menor peso y la agrega al árbol de expansión mínima. A continuación, se repite el proceso, agregando aristas de menor peso y verificando que no se formen ciclos en el árbol de expansión. El algoritmo termina cuando todas las aristas han sido evaluadas o cuando se han agregado n-1 aristas (donde n es el número de vértices del grafo), lo que indica que el árbol de expansión mínima ha sido encontrado.

¿Para qué sirve?

El algoritmo de Kruskal sirve para encontrar el árbol de mínimo costo en un grafo ponderado. Esto puede ser útil en muchas aplicaciones, como redes de comunicación, diseño de circuitos, planificación de rutas, entre otros.

¿Cómo se implementa en el mundo?

El algoritmo de Kruskal se utiliza en muchas aplicaciones del mundo real, como la planificación de rutas de transporte, la construcción de redes de comunicación, la identificación de componentes en sistemas complejos y la optimización de sistemas. Además, se utiliza en la industria del software para la implementación de algoritmos de grafos, como la detección de ciclos y la construcción de árboles de expansión mínima.

¿Cómo lo implementarías en tu vida?

En mi vida personal, el algoritmo de Kruskal podría ser útil en la planificación de rutas de viaje o en la optimización de mi tiempo en la realización de tareas. Por ejemplo, podría utilizar el algoritmo para determinar el camino más corto entre dos lugares o para priorizar las tareas en función de su importancia y urgencia.

¿Cómo lo implementarías en tu trabajo o tu trabajo de ensueño (Mi trabajo en sueño es trabajar en una ensambladora de autos en el área de automatización)?

En mi trabajo de ensueño como ingeniero de automatización en una ensambladora de autos, el algoritmo de Kruskal podría ser útil para optimizar el diseño y la planificación de la producción. Por ejemplo, podría utilizar el algoritmo para

determinar la ruta más eficiente para los robots de ensamblaje o para priorizar las tareas en función de su costo y duración. Además, podría utilizar el algoritmo para identificar componentes críticos en el sistema de producción y para optimizar el mantenimiento preventivo.

Codigo:

```
#Camberos Cordova Carlos Raul 20310415 6E2


import pygame
import networkx as nx
import matplotlib.pyplot as plt
import random
import itertools


ANCHO = 800
ALTO = 600


def dibujar_grafo(grafo, aristas_seleccionadas, color_aristas):
    # Dibujar los nodos del grafo
    pos = nx.get_node_attributes(grafo, 'pos')
    nx.draw_networkx_nodes(grafo, pos, node_size=500, node_color='lightblue', alpha=0.9)


    # Dibujar las aristas del grafo
    nx.draw_networkx_edges(grafo, pos, alpha=0.7, edge_color='black', width=2)


    # Dibujar las aristas seleccionadas en rojo
    nx.draw_networkx_edges(grafo, pos, edgelist=aristas_seleccionadas, edge_color=color_aristas, width=2)
```

```
    # Dibujar las etiquetas de los nodos
    nx.draw_networkx_labels(grafo, pos, font_size=20, font_family='sans-serif')

    # Establecer los límites del eje
    plt.xlim([-1.2,1.2])
    plt.ylim([-1.2,1.2])

    # Actualizar la pantalla
    plt.pause(0.001)
    plt.show()


def crear_grafo(n):
    grafo = nx.Graph()

    # Crear los nodos
    for i in range(n):
        x = random.uniform(-1, 1)
        y = random.uniform(-1, 1)
        grafo.add_node(i, pos=(x, y))

    # Crear las aristas
    combinaciones = itertools.combinations(grafo.nodes(), 2)
    for u, v in combinaciones:
        if random.random() < 0.5:
            continue
        pos1 = grafo.nodes[u]['pos']
        pos2 = grafo.nodes[v]['pos']
```

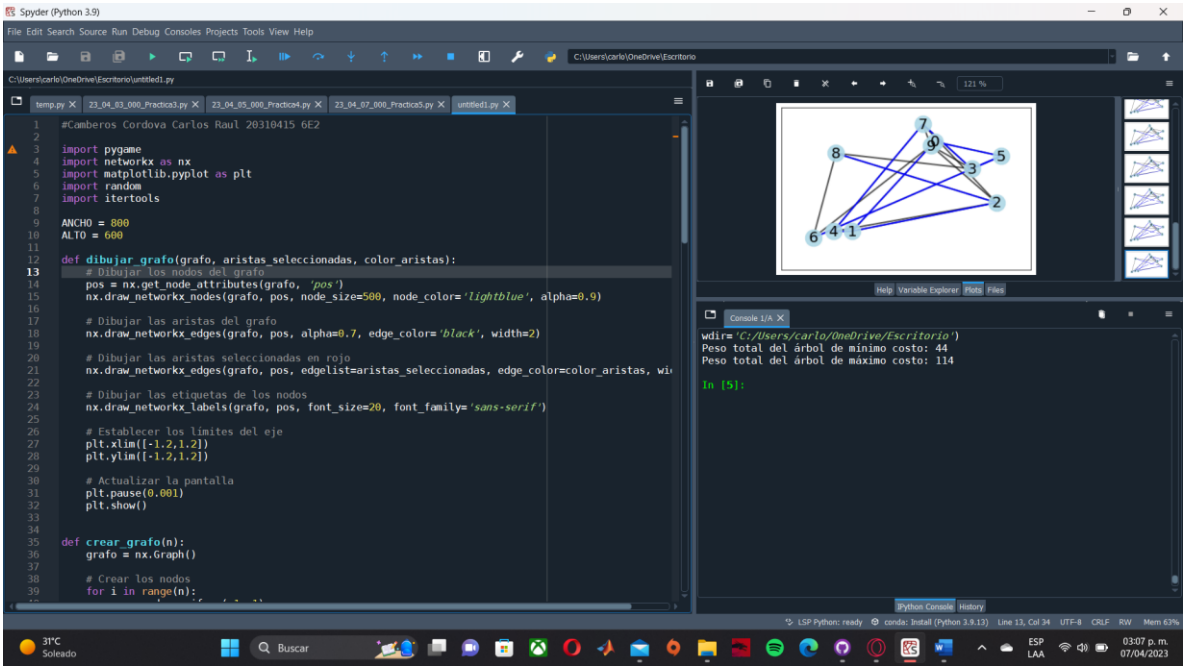
```
            peso = random.randint(1, 20)
            grafo.add_edge(u, v, weight=peso, pos1=pos1, pos2=pos2)

    return grafo

# Función para encontrar el padre de un nodo en el algoritmo de Kruskal
def encontrar_padre(padres, nodo):
    if padres[nodo] != nodo:
        padres[nodo] = encontrar_padre(padres, padres[nodo])
    return padres[nodo]

# Función para unir dos componentes conexas en el grafo
def unir_componentes(padres, rank, nodo1, nodo2):
    padre1 = encontrar_padre(padres, nodo1)
    padre2 = encontrar_padre(padres, nodo2)
    if rank[padre1] > rank[padre2]:
        padres[padre2] = padre1
    else:
        padres[padre1] = padre2
        if rank[padre1] == rank[padre2]:
            rank[padre2] += 1

# Función para ordenar las aristas por peso
def ordenar_aristas_por_peso(aristas):
    return sorted(aristas, key=lambda x: x[2])

def kruskal_minimo_costo(grafo):
    # Inicializar los padres y los rangos de los nodos
```

```
    padres = {nodo: nodo for nodo in grafo.nodes()}
    rank = {nodo: 0 for nodo in grafo.nodes()}

    # Obtener las aristas del grafo y ordenarlas por peso
    aristas = [(u, v, data['weight']) for u, v, data in grafo.edges(data=True)]
    aristas_ordenadas = ordenar_aristas_por_peso(aristas)

    # Inicializar la lista de aristas seleccionadas y el peso total del árbol
    aristas_seleccionadas = []
    peso_total = 0

    # Seleccionar las aristas del árbol de mínimo costo
    for u, v, peso in aristas_ordenadas:
        if encontrar_padre(padres, u) != encontrar_padre(padres, v):
            aristas_seleccionadas.append((u, v))
            peso_total += peso
            unir_componentes(padres, rank, u, v)

            # Dibujar el grafo con las aristas seleccionadas
            dibujar_grafo(grafo, aristas_seleccionadas, 'r')

    # Imprimir el peso total del árbol de mínimo costo
    print(f'Peso total del árbol de mínimo costo: {peso_total}')

def kruskal_maximo_costo(grafo):
    # Inicializar los padres y los rangos de los nodos
    padres = {nodo: nodo for nodo in grafo.nodes()}
    rank = {nodo: 0 for nodo in grafo.nodes()}
```

```
    # Obtener las aristas del grafo y ordenarlas por peso de mayor a menor
    aristas = [(u, v, data['weight']) for u, v, data in grafo.edges(data=True)]
    aristas_ordenadas = sorted(aristas, key=lambda x: x[2], reverse=True)

    # Verificar si hay al menos una arista con costo mayor a cero
    if len(aristas_ordenadas) == 0 or aristas_ordenadas[0][2] == 0:
        print('El grafo no tiene aristas con costo mayor a cero')
        return

    # Inicializar la lista de aristas seleccionadas y el peso total del árbol
    aristas_seleccionadas = []
    peso_total = 0

    # Seleccionar las aristas del árbol de máximo costo
    for u, v, peso in aristas_ordenadas:
        if encontrar_padre(padres, u) != encontrar_padre(padres, v):
            aristas_seleccionadas.append((u, v))
            peso_total += peso
            unir_componentes(padres, rank, u, v)

            # Dibujar el grafo con las aristas seleccionadas
            dibujar_grafo(grafo, aristas_seleccionadas, 'b')

    # Imprimir el peso total del árbol de máximo costo
    print(f'Peso total del árbol de máximo costo: {peso_total}')

def main():
```

```
    # Crear el grafo
    grafo = crear_grafo(10)
    # Dibujar el grafo vacío
    plt.figure(figsize=(8, 6))
    plt.axis('off')
    dibujar_grafo(grafo, [], 'none')

    # Ejecutar el algoritmo de Kruskal para encontrar el árbol de mínimo costo
    kruskal_minimo_costo(grafo)

    # Ejecutar el algoritmo de Kruskal para encontrar el árbol de máximo costo
    kruskal_maximo_costo(grafo)

    # Mostrar la ventana con la animación
    plt.show()

if __name__ == "__main__":
    main()
```

Evidencia:



Desarrollo:

Este es un código en Python que implementa el algoritmo de Kruskal para encontrar el árbol de mínimo costo en un grafo no dirigido y conexo. El código utiliza la biblioteca de Python NetworkX para trabajar con grafos y la biblioteca de Python Pygame para dibujar el grafo en una ventana.

En la función dibujar_grafo, se dibuja el grafo y se resaltan en rojo las aristas seleccionadas. La función crear_grafo crea un grafo aleatorio con n nodos y aristas aleatorias con un peso entre 1 y 20. La función encontrar_padre encuentra el padre de un nodo en el algoritmo de Kruskal. La función unir_componentes une dos componentes conexas en el grafo. La función ordenar_aristas_por_peso ordena las aristas del grafo por peso. La función kruskal_minimo_costo implementa el algoritmo de Kruskal para encontrar el árbol de mínimo costo en el grafo. La función kruskal_maximo_costo implementa el algoritmo de Kruskal para encontrar el árbol de máximo costo en el grafo.