Camberos Cordova Carlos Raúl

20310415

Inteligencia Artificial

6E2

Ingeniería en Mecatrónica

Practica 3

Mauricio Alejandro Cabrera Arellano

Marco Teórico

Realizar un simulador de Algoritmo de Dijkstra.

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 Algoritmo de Dijkstra es un algoritmo utilizado en teoría de grafos para encontrar el camino más corto entre un nodo inicial y todos los demás nodos de un grafo ponderado con pesos no negativos. Este algoritmo fue propuesto por el matemático y científico de la computación holandés Edsger W. Dijkstra en 1956 y es uno de los algoritmos más utilizados en redes de comunicaciones, sistemas de transporte y en general en cualquier sistema que necesite encontrar rutas óptimas.

¿Para qué sirve?

El algoritmo de Dijkstra se utiliza en muchas aplicaciones, como en el enrutamiento de redes de computadoras y en la planificación de rutas de transporte. También se puede utilizar en sistemas de navegación para encontrar la ruta más corta entre dos ubicaciones.

¿Cómo se implementa en el mundo?

El algoritmo de Dijkstra tiene diversas aplicaciones en el mundo, en particular en la planificación y optimización de rutas en redes de transporte y comunicación, como carreteras, aeropuertos, redes de computadoras, entre otros.

Por ejemplo, en la planificación de rutas de transporte terrestre, el algoritmo de Dijkstra puede ser utilizado para encontrar el camino más corto entre dos puntos, teniendo en cuenta factores como la distancia, el tiempo de viaje, la congestión del tráfico, entre otros.

En el ámbito de las redes de comunicación, el algoritmo de Dijkstra es utilizado para encontrar la ruta más eficiente entre dos nodos en una red, teniendo en cuenta factores como el ancho de banda, la congestión de la red, entre otros.

En resumen, el algoritmo de Dijkstra es ampliamente utilizado en la planificación y optimización de rutas y comunicaciones en diversos ámbitos, y su implementación puede mejorar significativamente la eficiencia y eficacia de los sistemas involucrados.

¿Cómo lo implementarías en tu vida?

En mi vida personal, podría utilizar el algoritmo de Dijkstra para planificar rutas de viaje en mis vacaciones. Podría ingresar los lugares que quiero visitar y el algoritmo podría encontrar la ruta más corta para visitar todos los lugares. También podría utilizarlo para planificar una ruta de manejo más corta entre varios destinos

¿Cómo lo implementarías 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 de Dijkstra puede ser útil en la automatización de la planificación de rutas para los robots en una ensambladora de autos. Los robots pueden tener diferentes tareas asignadas, como mover piezas o herramientas, soldar o pintar partes del automóvil, entre otras. Para realizar estas tareas, los robots necesitan moverse por el espacio de trabajo y llegar a diferentes ubicaciones en el tiempo adecuado.

El Algoritmo de Dijkstra puede ayudar a planificar las rutas óptimas para los robots. El grafo utilizado para el algoritmo puede representar el mapa del espacio de trabajo de la ensambladora, donde cada nodo representa una ubicación en el mapa y cada borde representa la distancia entre dos ubicaciones.

Para aplicar el algoritmo en este contexto, primero se deben identificar los nodos y los bordes del grafo. Los nodos pueden representar las diferentes estaciones de trabajo, las posiciones de las piezas o herramientas, o cualquier otra ubicación relevante en el espacio de trabajo. Los bordes pueden representar las distancias entre dos ubicaciones, el tiempo que tarda un robot en moverse entre dos ubicaciones, o cualquier otra medida relevante para la planificación de rutas.

Una vez que se tiene el grafo, se puede utilizar el Algoritmo de Dijkstra para encontrar la ruta óptima para que un robot se mueva de un nodo a otro. El algoritmo puede ser utilizado para planificar rutas para un solo robot o para múltiples robots que trabajan en conjunto.

En general, el uso del Algoritmo de Dijkstra en la automatización de la planificación de rutas para robots en una ensambladora de autos puede ayudar a mejorar la

eficiencia y la precisión de los procesos de producción, lo que puede conducir a una mayor productividad y una mejor calidad de los productos.

Código:

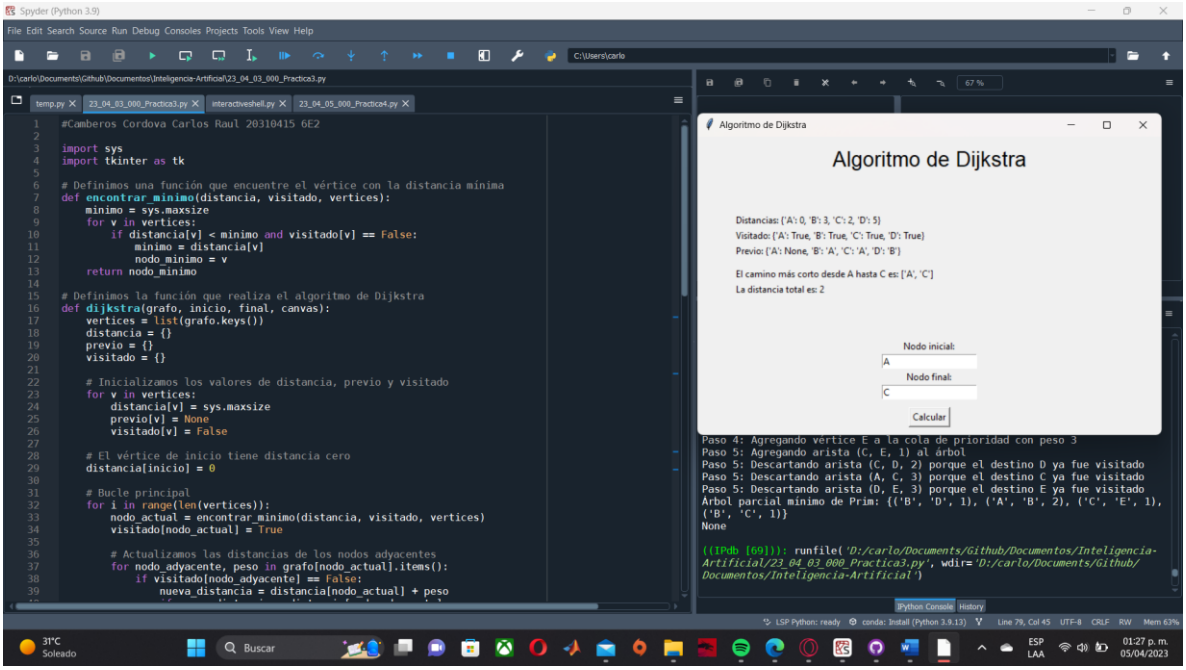
```
#Camberos Cordova Carlos Raul 20310415 6E2


import sys
import tkinter as tk


# Definimos una función que encuentre el vértice con la distancia mínima
def encontrar_minimo(distancia, visitado, vertices):
    minimo = sys.maxsize
    for v in vertices:
        if distancia[v] < minimo and visitado[v] == False:
            minimo = distancia[v]
            nodo_minimo = v
    return nodo_minimo


# Definimos la función que realiza el algoritmo de Dijkstra
def dijkstra(grafo, inicio, final, canvas):
    vertices = list(grafo.keys())
    distancia = {}
    previo = {}
    visitado = {}


    # Inicializamos los valores de distancia, previo y visitado
    for v in vertices:
        distancia[v] = sys.maxsize
        previo[v] = None
```

```
    visitado[v] = False

# El vértice de inicio tiene distancia cero
distancia[inicio] = 0

# Bucle principal
for i in range(len(vertices)):
    nodo_actual = encontrar_minimo(distancia, visitado, vertices)
    visitado[nodo_actual] = True

    # Actualizamos las distancias de los nodos adyacentes
    for nodo_adyacente, peso in grafo[nodo_actual].items():
        if visitado[nodo_adyacente] == False:
            nueva_distancia = distancia[nodo_actual] + peso
            if nueva_distancia < distancia[nodo_adyacente]:
                distancia[nodo_adyacente] = nueva_distancia
                previo[nodo_adyacente] = nodo_actual

    # Mostramos el estado actual del algoritmo en la interfaz gráfica
    canvas.itemconfigure(distancia_text, text=f"Distancias: {distancia}")
    canvas.itemconfigure(visitado_text, text=f"Visitado: {visitado}")
    canvas.itemconfigure(previo_text, text=f"Previo: {previo}")
    canvas.update()

# Construimos el camino desde el nodo final al nodo inicial
camino = []
nodo = final
while nodo != None:
```

```
        camino.insert(0, nodo)
        nodo = previo[nodo]

    # Mostramos el resultado final en la interfaz gráfica
    canvas.itemconfigure(resultado_text, text=f"El camino más corto desde {inicio} hasta {final} es: {camino}")
    canvas.itemconfigure(distancia_final_text, text=f"La distancia total es: {distancia[final]}")
    canvas.update()

# Función que se ejecuta cuando se presiona el botón "Calcular"
def calcular():
    grafo = {
        'A': {'B': 3, 'C': 2},
        'B': {'D': 2},
        'C': {'B': 1, 'D': 4},
        'D': {}
    }
    inicio = nodo_inicio.get()
    final = nodo_final.get()

    dijkstra(grafo, inicio, final, canvas)

# Creamos la interfaz gráfica
ventana = tk.Tk()
ventana.title("Algoritmo de Dijkstra")

# Creamos los widgets de la interfaz gráfica
titulo = tk.Label(ventana, text="Algoritmo de Dijkstra", font=("Arial", 20))
```

```
titulo.pack(pady=10)

canvas = tk.Canvas(ventana, width=600, height=200)
canvas.pack()

distancia_text = canvas.create_text(50, 50, text="Distancias: ", anchor="w")
visitado_text = canvas.create_text(50, 70, text="Visitado: ", anchor="w")
previo_text = canvas.create_text(50, 90, text="Previo: ", anchor="w")
resultado_text = canvas.create_text(50, 120, text="Resultado: ", anchor="w")
distancia_final_text = canvas.create_text(50, 140, text="Distancia total: ", anchor="w")

nodo_inicio_label = tk.Label(ventana, text="Nodo inicial:")
nodo_inicio_label.pack()
nodo_inicio = tk.Entry(ventana)
nodo_inicio.pack()

nodo_final_label = tk.Label(ventana, text="Nodo final:")
nodo_final_label.pack()
nodo_final = tk.Entry(ventana)
nodo_final.pack()

calcular_boton = tk.Button(ventana, text="Calcular", command=calcular)
calcular_boton.pack(pady=10)

ventana.mainloop()
```

Evidencia:


Spyder (Python 3.9)
#Camberos Cordova Carlos Raul 20310415 6E2
import sys
import tkinter as tk
# Definimos una función que encuentre el vértice con la distancia mínima
def encontrar_minimo(distancia, visitado, vertices):
minimo = sys.maxsize
for v in vertices:
if distancia[v] < minimo and visitado[v] == False:
minimo = distancia[v]
nodo_minimo = v
return nodo_minimo
# Definimos la función que realiza el algoritmo de Dijkstra
def dijkstra(grafo, inicio, final, canvas):
vertices = list(grafo.keys())
distancia = {}
previo = {}
visitado = {}
# Inicializamos los valores de distancia, previo y visitado
for v in vertices:
distancia[v] = sys.maxsize
previo[v] = None
visitado[v] = False
# El vértice de inicio tiene distancia cero
distancia[inicio] = 0
# Bucle principal
for i in range(len(vertices)):
nodo_actual = encontrar_minimo(distancia, visitado, vertices)
visitado[nodo_actual] = True
# Actualizamos las distancias de los nodos adyacentes
for nodo_adyacente, peso in grafo[nodo_actual].items():
if visitado[nodo_adyacente] == False:
nueva_distancia = distancia[nodo_actual] + peso
Algoritmo de Dijkstra
Algoritmo de Dijkstra
Distancias: {'A': 0, 'B': 3, 'C': 2, 'D': 5}
Visitado: {'A': True, 'B': True, 'C': True, 'D': True}
Previo: {'A': None, 'B': 'A', 'C': 'A', 'D': 'B'}
El camino más corto desde A hasta C es: ['A', 'C']
La distancia total es: 2
Nodo inicial:
A
Nodo final:
C
Calcular
Paso 4: Agregando vértice E a la cola de prioridad con peso 3
Paso 5: Agregando arista (C, E, 1) al árbol
Paso 5: Descartando arista (C, D, 2) porque el destino D ya fue visitado
Paso 5: Descartando arista (A, C, 3) porque el destino C ya fue visitado
Paso 5: Descartando arista (D, E, 3) porque el destino E ya fue visitado
Árbol parcial mínimo de Prim: {('B', 'D', 1), ('A', 'B', 2), ('C', 'E', 1), ('B', 'C', 1)}
None
runfile('D:/carlo/Documents/Github/Documentos/Inteligencia-Artificial/23_04_03_000_Practica3.py', wdir='D:/carlo/Documents/Github/Documentos/Inteligencia-Artificial')
31°C
Soleado
Buscar
ESP
LAA
01:27 p. m.
05/04/2023

Desarrollo

Este código es una implementación del algoritmo de Dijkstra en Python 3 utilizando la biblioteca tkinter para crear una interfaz gráfica.

El código comienza importando los módulos necesarios, como el módulo sys para obtener un valor máximo para la distancia inicial de los vértices y el módulo tkinter para crear la interfaz gráfica.

La función encontrar_minimo() encuentra el vértice no visitado con la distancia mínima desde el vértice de inicio.

La función principal dijkstra() recibe el grafo como un diccionario, el nodo inicial, el nodo final y el objeto canvas de tkinter. Primero inicializa la distancia, previo y visitado para cada vértice. Luego, establece la distancia del vértice de inicio a cero y comienza un bucle que se repite una vez por cada vértice en el grafo. En cada iteración, encuentra el vértice no visitado con la distancia mínima, lo marca como visitado y actualiza las distancias de los nodos adyacentes si es necesario. También actualiza la interfaz gráfica con los valores actuales de distancia, visitado y previo. Una vez que se han visitado todos los vértices, construye el camino más corto desde el nodo final hasta el nodo inicial y actualiza la interfaz gráfica con el resultado.

La función calcular() es la que se llama cuando se hace clic en el botón "Calcular". Aquí se crea el grafo de prueba y se obtienen los nodos de inicio y final de la interfaz gráfica.

Finalmente, se crea la interfaz gráfica utilizando la biblioteca tkinter. Se crea un objeto de ventana, se establece un título y se crean los widgets, como etiquetas, entradas de texto y botones. Cuando se hace clic en el botón "Calcular", se llama a la función calcular() para realizar el algoritmo de Dijkstra y actualizar la interfaz gráfica con los resultados.