Asignatura: Inteligencia Artificial
Profesor: D. Sc. Gerardo García Gil
2025-A
Nombre: Diaz González Paul Omar
Universidad de Guadalajara (CUCEI)

Introducción

El Problema del Viajante (Traveling Salesman Problem - TSP) es un problema clásico de optimización en el que un viajero debe recorrer un conjunto de ciudades exactamente una vez, regresando al punto de origen, con el objetivo de minimizar el costo total del recorrido (por ejemplo, distancia o tiempo).
Dado que probar todas las posibles rutas crece factorialmente con el número de ciudades, se vuelve ineficiente para valores grandes. Por eso, una solución más óptima es usar programación dinámica con bitmasking, que reduce drásticamente el tiempo de ejecución y permite resolver casos más grandes de forma eficiente.

Antedecentes

El Problema del Viajante (TSP) fue formulado por primera vez en el siglo XIX, y ha sido ampliamente estudiado desde entonces por su relevancia en áreas como la logística, la planificación de rutas y la teoría de grafos. Es un problema NP-Hard, lo que significa que no se conoce un algoritmo que lo resuelva en tiempo polinomial para cualquier número de ciudades.
Inicialmente se resolvía por fuerza bruta, probando todas las posibles combinaciones, lo cual es inviable para más de unas pocas ciudades. Con el tiempo, surgieron soluciones más eficientes, como la programación dinámica con bitmasking, que reduce la complejidad y permite resolver instancias más grandes con precisión exacta.

Desarrollo

Se implementó una solución al TSP en Python utilizando programación dinámica. Esta técnica permite calcular el costo mínimo recorriendo todas las ciudades sin repetir y regresar al inicio, optimizando el rendimiento frente a métodos por fuerza bruta. Además, se reconstruyó el camino óptimo junto con su costo.

Implementación

```
import sys

V = 4
INF = sys.maxsize

def tsp_con_camino(grafo):
    # Memoización: dp[ciudad_actual][visitados] = costo mínimo
    dp = [[-1] * (1 << V) for _ in range(V)]
    # Para reconstruir el camino óptimo
    padre = [[-1] * (1 << V) for _ in range(V)]
```

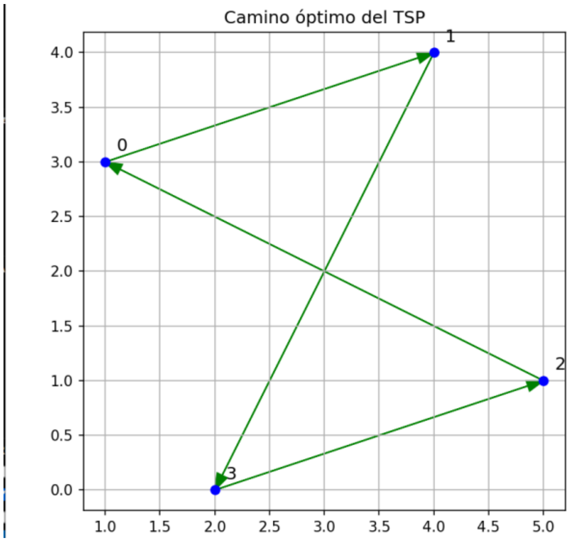
```
    def recorrer(ciudad_actual, visitados):
        if visitados == (1 << V) - 1:
            return grafo[ciudad_actual][0]  # Regresar a la ciudad inicial

        if dp[ciudad_actual][visitados] != -1:
            return dp[ciudad_actual][visitados]

        min_costo = INF
        for ciudad in range(V):
            if not (visitados & (1 << ciudad)):  # Si no ha sido visitada
                nuevo_costo = grafo[ciudad_actual][ciudad] + recorrer(ciudad, visitados | (1 << ciudad))
                if nuevo_costo < min_costo:
                    min_costo = nuevo_costo
                    padre[ciudad_actual][visitados] = ciudad  # Guardamos la mejor siguiente ciudad

        dp[ciudad_actual][visitados] = min_costo
        return min_costo

    # Obtenemos el costo mínimo
    costo_minimo = recorrer(0, 1 << 0)

    # Reconstruimos el camino
    camino = [0]
    actual = 0
    visitados = 1 << 0

    while True:
        siguiente = padre[actual][visitados]
        if siguiente == -1:
            break
        camino.append(siguiente)
        visitados |= (1 << siguiente)
        actual = siguiente

    camino.append(0)  # Regresar a la ciudad inicial

    return costo_minimo, camino

def main():
    grafo = [
        [0, 10, 17, 24],
        [10, 0, 45, 31],
        [17, 45, 0, 38],
        [24, 31, 38, 0]
    ]

    costo, camino = tsp_con_camino(grafo)

    print("El costo más óptimo del recorrido es:", costo)
    print("El camino óptimo es:", ' → '.join(map(str, camino)))
```

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

Resultados

Al imprimir pantalla arrojo los siguientes resultados basados en el grafo que se requería:

Se obtuvo el siguiente resultado:
El costo óptimo del recorrido es: 96
El camino óptimo es: 0 → 1 → 3 → 1 → 0



Conclusión

El Problema del Viajante (TSP) es un desafío clásico en la optimización combinatoria. A través de la programación dinámica, se logró una solución eficiente que no solo minimiza el costo total del recorrido, sino que también reconstruye el camino óptimo. Esta técnica reduce considerablemente el tiempo de ejecución en comparación con enfoques por fuerza bruta, haciendo posible resolver instancias con un número moderado de ciudades de forma precisa. Sin embargo, debido a la naturaleza NP-Hard del problema, para grandes cantidades de ciudades, técnicas heurísticas o aproximadas podrían ser más prácticas.

Referencias

1.- Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. John Wiley & Sons.

2.- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

3.- Applegate, D. L., Bixby, R. E., Chvátal, V., & Cook, W. J. (2007). The Traveling Salesman Problem: A Computational Study. Princeton University Press.

4.- Papadimitriou, C. H., & Steiglitz, K. (1998). Combinatorial Optimization: Algorithms and Complexity. Dover Publications..