Instituto Tecnológico Superior de Xalapa

MPSCO-0101- Análisis y diseño de algoritmos

Flores Mendez Brian Delfino - 257o02563

Guerrero Villalobos Cristian Gael - 257O02706

Viveros Aguilar Kevin de Jesús - 257O02541

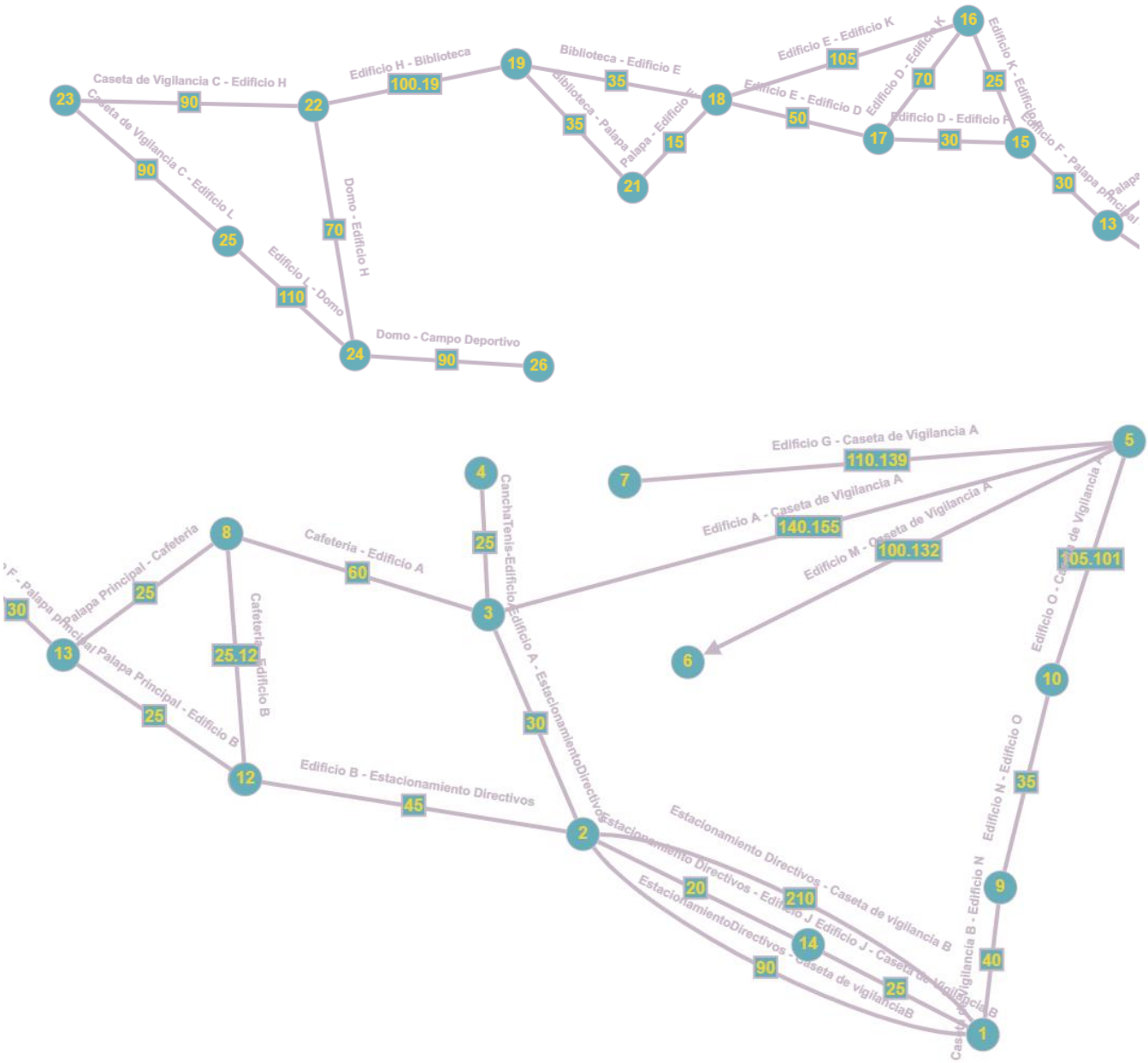
María Angélica Cerdán

PRY03a- Evaluación del caso para el PRY03

Entrega:10 de noviembre de 2025

En este caso, decidimos usar una matriz de adyacencia ya que los nodos no son dinámicos, lo que significa que no van a estar cambiando frecuentemente, además de que se pueden realizar búsquedas rápidas o aplicar algoritmos facilmente, como el de encontrar el camino mas corto entre 2 vertices dados. En esta ocasión, para encontrar el camino mas corto usamos el algoritmo de Dijkstra con una pequeña modificacion, en los pesos de los caminos hay valores que tienen valores decimales donde estos decimales significa que hay escaleras en los caminos y se deberá tratar de evitar usar estos caminos a menos que sea el unico camino disponible para pasar de un punto a otro, está será la modificación del algoritmo para tratar de evitar caminos con escaleras.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	0	90	0	0	0	0	0	0	40	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0
2	90	0	30	0	0	0	0	0	0	0	45	0	0	20	0	0	0	0	0	0	0	0	0	0	0	0
3	0	30	0	25	140.155	0	0	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	140.155	0	0	100.132	110.139	0	0	105.101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	110.139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	60	0	0	0	0	0	0	0	0	25.12	25	0	0	0	0	0	0	0	0	0	0	0	0	0
9	40	0	0	0	0	0	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	105.101	0	0	0	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	25.12	0	0	0	0	25	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	25	0	0	0	25	0	0	30	0	0	0	0	0	0	0	0	0	0	0
14	25	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	30	0	0	25	30	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	25	0	70	105	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	30	70	0	50	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	105	50	0	35	0	15	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0	0	35	100.19	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	15	35	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100.19	0	0	0	90	70	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	0	0	90	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	70	0	0	110	90
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	110	0	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	90	0	0



Este es nuestro grafo que representa el Instituto Tecnológico Superior de Xalapa, en el cual se representan los andadores dentro de toda la institución, el peso de cada camino es la distancia en metros de los andadores, pero en los caminos que tienen decimales, los decimales representan las escaleras que hay en esos caminos.

El código que utilizamos basado en el algoritmo de Dijkstra para encontrar los caminos más cortos es el siguiente:

```
import heapq

def matriz_a_grafo(matriz):
    grafo = {}
    n = len(matriz) - 1

    for i in range(1, n + 1):
        nodo = str(i)
        grafo[nodo] = {}
        for j in range(1, n + 1):
            peso = matriz[i][j]
            if peso != 0:
                grafo[nodo][str(j)] = peso
    return grafo


def dijkstra(grafo, origen):
    dist = {v: float('inf') for v in grafo}
    dist[origen] = 0
    predecesor = {v: None for v in grafo}
    cola = [(0, origen)]

    while cola:
        d_actual, u = heapq.heappop(cola)
        if d_actual > dist[u]:
            continue

        for v, peso in grafo[u].items():
            nueva_dist = dist[u] + peso
            if nueva_dist < dist[v]:
                dist[v] = nueva_dist
                predecesor[v] = u
                heapq.heappush(cola, (nueva_dist, v))
    return dist, predecessor

matriz = [
    # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
20, 21, 22, 23, 24, 25, 26],
    [0, 90, 0, 0, 0, 0, 0, 0, 40, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0],
    [90, 0, 30, 0, 0, 0, 0, 0, 0, 0, 45, 0, 0, 20, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
```

```
    [0, 30, 0, 25, 140.155, 0, 0, 60, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 140.155, 0, 0, 100.132, 110.139, 0, 0, 105.101, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 110.139, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 60, 0, 0, 0, 0, 0, 0, 0, 0, 25.12, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [40, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 105.101, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 45, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 25.12, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 25, 0, 0, 30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [25, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 0, 0, 25, 30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 70, 105, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 70, 0, 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 105, 50, 0, 35, 0, 15, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0, 0, 35, 100.19, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 100.19, 0, 0, 0, 90, 70, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 90, 0, 0, 90, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 70, 0, 0, 110, 90, 0],
```

```
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 90,
110, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
90, 0, 0],
]

# Crear el grafo
grafo = matriz_a_grafo(matriz)

# Algoritmo de Dijkstra ejecutado, vertice 1 como origen
origen = "1"
dist, pred = dijkstra(grafo, origen)

# Mostrar resultados
print(f"Distancias mínimas desde el vértice {origen}:")
for nodo, d in dist.items():
    print(f"{origen} → {nodo}: {d if d != float('inf') else '∞'}")

print("\nPredecesores:")
for nodo, p in pred.items():
    print(f"{nodo}: {p}")
```

Y nos da como resultado:
Distancias mínimas desde el vértice 1:

1 → 1: 0
1 → 2: 30
1 → 3: 55
1 → 4: 170.155
1 → 5: 270.287000003
1 → 6: 280.294
1 → 7: 90
1 → 8: 310.256
1 → 9: 275.256
1 → 10: 45
1 → 11: 115.12
1 → 12: 115
1 → 13: 20
1 → 14: 145
1 → 15: 170
1 → 16: 175
1 → 17: 225
1 → 18: 260
1 → 19: ∞
1 → 20: 240
1 → 21: 360.19
1 → 22: 450.19
1 → 23: 430.19
1 → 24: 540.19
1 → 25: 520.19

Además de tener la información de los predecesores:

1: None	10: 1	19: None
2: 1	11: 7	20: 17
3: 2	12: 7	21: 18
4: 2	13: 1	22: 21
5: 4	14: 12	23: 21
6: 4	15: 14	24: 23
7: 2	16: 14	25: 23
8: 9	17: 16	
9: 4	18: 17	

El código que muestra la distancia mas corta al igual que el camino es el siguiente:

```python
import heapq

def matriz_a_grafo(matriz):
    grafo = {}
    n = len(matriz)

    for i in range(n):
        nodo = str(i + 1)
        grafo[nodo] = {}
        for j in range(n):
            peso = matriz[i][j]
            if peso != 0:
                grafo[nodo][str(j + 1)] = peso
    return grafo

def dijkstra(grafo, origen):
    dist = {v: float('inf') for v in grafo}
    dist[origen] = 0
    predecesor = {v: None for v in grafo}
    cola = [(0, origen)]

    while cola:
        d_actual, u = heapq.heappop(cola)
        if d_actual > dist[u]:
            continue
```

```
        for v, peso in grafo[u].items():
            nueva_dist = dist[u] + peso
            if nueva_dist < dist[v]:
                dist[v] = nueva_dist
                predecesor[v] = u
                heapq.heappush(cola, (nueva_dist, v))
    return dist, predecessor

def reconstruir_camino(predecesor, origen, destino):
    camino = []
    actual = destino
    while actual is not None:
        camino.insert(0, actual)
        actual = predecesor[actual]
    if camino[0] != origen:
        return None  # No hay camino
    return camino

matriz = [
    [0, 90, 0, 0, 0, 0, 0, 0, 40, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0],
    [90, 0, 30, 0, 0, 0, 0, 0, 0, 0, 45, 0, 0, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
    [0, 30, 0, 25, 140.155, 0, 0, 60, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0],
    [0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0],
    [0, 0, 140.155, 0, 0, 100.132, 110.139, 0, 0, 105.101, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0],
    [0, 0, 0, 0, 110.139, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
    [0, 0, 60, 0, 0, 0, 0, 0, 0, 0, 0, 25.12, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
    [40, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0],
    [0, 0, 0, 0, 105.101, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
    [0, 45, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 25.12, 0, 0, 0, 0, 25, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 25, 0, 0, 0, 25, 0, 0, 30, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0],
```

```
    [25, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 0, 0, 25, 30, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 25, 0, 70, 105, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 30, 70, 0, 50, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 105, 50, 0, 35, 0, 15, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 35, 0, 0, 35, 100.19, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 15, 35, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 100.19, 0, 0, 0, 90, 70, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 90, 0, 0, 90, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 70, 0, 0, 110, 90],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 90, 110, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 90, 0, 0],
]

grafo = matriz_a_grafo(matriz)
origen = input("Ingresa el vértice de origen (1-26): ").strip()
destino = input("Ingresa el vértice destino (1-26): ").strip()

dist, pred = dijkstra(grafo, origen)
camino = reconstruir_camino(pred, origen, destino)

print(f"\nDistancia mínima desde {origen} hasta {destino}: {dist[destino] if dist[destino] != float('inf') else '∞'}")

if camino:
    print(f"Camino más corto: {' → '.join(camino)}")
else:
    print(f"No existe camino entre {origen} y {destino}")
```

Y tomando un ejemplo, iniciando en el vértice 2 con dirección al vértice 24, el camino mas corto es el siguiente:
Ingresa el vértice de origen (1-26): 2

Ingresa el vértice destino (1-26): 24

Distancia mínima desde 2 hasta 24: 430.19

Camino más corto: 2 → 3 → 8 → 13 → 15 → 17 → 18 → 19 → 22 → 24