

**Name: Devanshi Jain**  
**Enrollment-No: 22162101006**  
**Batch: 51**

**Institute of Computer Technology**  
**B. Tech Computer Science and Engineering**

**Sub: Algorithm Analysis and Design**

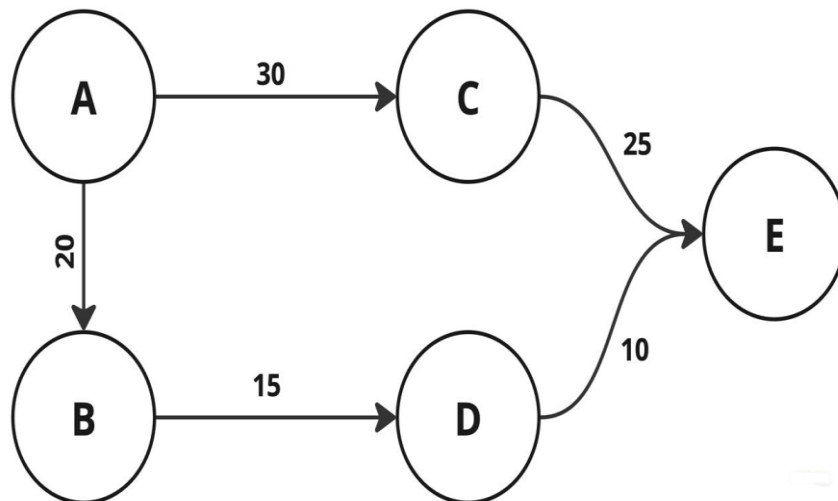
**Practical 11**

**AIM:**

A government official needs to visit several cities within a state. To minimize travel costs, they want to find the shortest path between their starting city and each destination city.

**Task:**

Given a graph representing the cities and their connecting roads, determine the minimum cost path from a given starting city to all other cities.



**Input:**

Enter total number of nodes: 5

Enter the node from where you want to calculate the distance: A

Enter Data (Weight):

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>A</i>	0	20	30	$\infty$	$\infty$
<i>B</i>	$\infty$	0	$\infty$	15	$\infty$
<i>C</i>	$\infty$	$\infty$	0	$\infty$	25
<i>D</i>	$\infty$	$\infty$	$\infty$	0	10
<i>E</i>	$\infty$	$\infty$	$\infty$	$\infty$	0

**Output:**

	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>A</i>	0	20	30	35	45
<i>B</i>	$\infty$	0	$\infty$	15	25
<i>C</i>	$\infty$	$\infty$	0	$\infty$	25
<i>D</i>	$\infty$	$\infty$	$\infty$	0	10
<i>E</i>	$\infty$	$\infty$	$\infty$	$\infty$	0

**OR**

Source	Destination	Cost
A	A	0
	B	20
	C	30
	D	35
	E	45

## Code:

```
import sys

def dijkstra(graph, start_node):
    n = len(graph)
    visited = [False] * n
    distance = [sys.maxsize] * n
    distance[start_node] = 0

    for _ in range(n):
        min_distance = sys.maxsize
        min_index = -1

        for i in range(n):
            if not visited[i] and distance[i] < min_distance:
                min_distance = distance[i]
                min_index = i

        visited[min_index] = True

        for j in range(n):
            if graph[min_index][j] != float('inf') and not visited[j]:
                new_dist = distance[min_index] + graph[min_index][j]
                if new_dist < distance[j]:
                    distance[j] = new_dist

    return distance

def print_distances(distance, cities):
    print("Source\tDestination\tCost")
    for i in range(len(cities)):
        if distance[i] == sys.maxsize:
            print(f"{cities[0]}\t{cities[i]}\t\t∞")
        else:
            print(f"{cities[0]}\t{cities[i]}\t\t{distance[i]}")

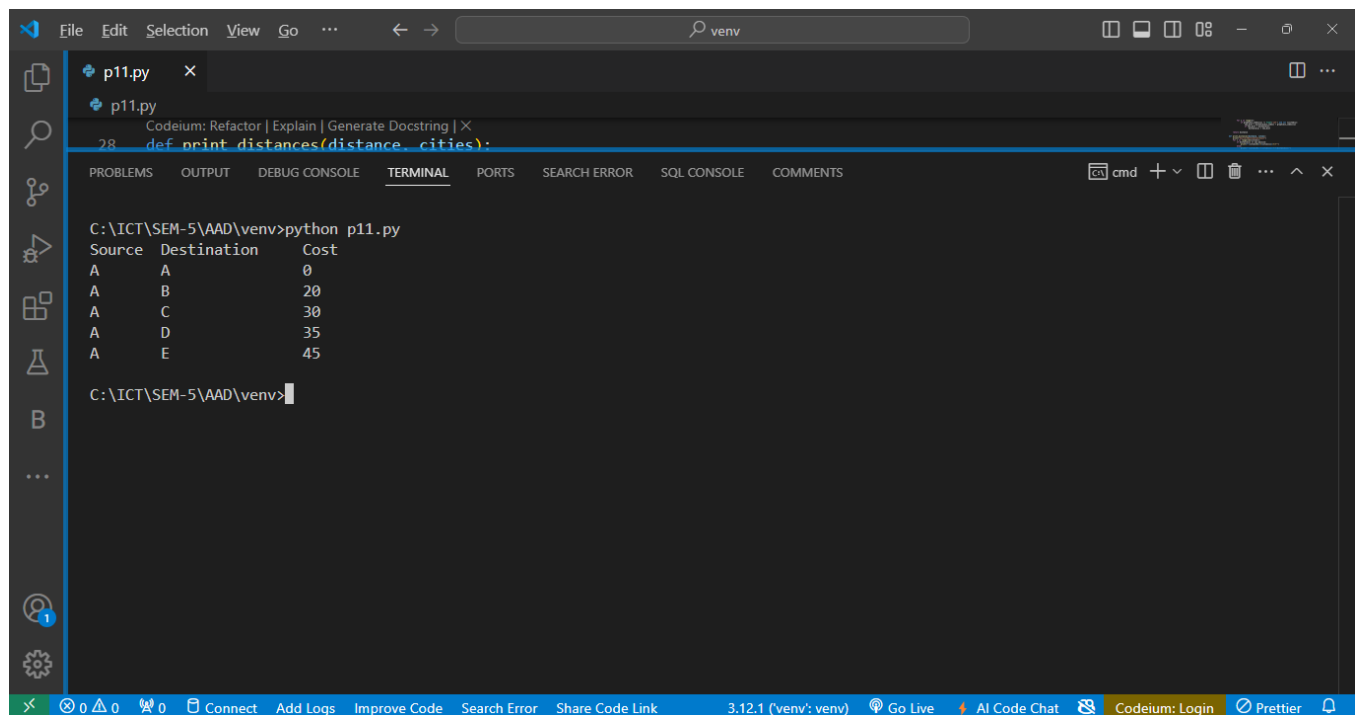
# Define the cities and graph as an adjacency matrix
cities = ['A', 'B', 'C', 'D', 'E']
graph = [
    [0, 20, 30, float('inf'), float('inf')],
    [float('inf'), 0, float('inf'), 15, 25],
    [float('inf'), float('inf'), 0, float('inf'), 25],
    [float('inf'), float('inf'), float('inf'), 0, 10],
    [float('inf'), float('inf'), float('inf'), float('inf'), 0]
```

```
]

# Set the starting city
start_city = 'A'
start_node = cities.index(start_city)

# Run Dijkstra's algorithm and print the distances
distances = dijkstra(graph, start_node)
print_distances(distances, cities)
```

## Output:



The screenshot shows a Visual Studio Code editor with a file named `p11.py` open. The terminal window at the bottom displays the command `C:\ICT\SEM-5\AAD\venv>python p11.py` and its output. The output is a table showing the source city (A) and the destination cities (A, B, C, D, E) along with their respective costs (0, 20, 30, 35, 45). The terminal also shows the current directory `C:\ICT\SEM-5\AAD\venv>`.

Source	Destination	Cost
A	A	0
A	B	20
A	C	30
A	D	35
A	E	45