

Code: (BFS)

```
from collections import deque

graph = {
    '5': {
        '3': 2,
        '7': 1
    },
    '3': {
        '2': 3,
        '4': 4
    },
    '7': {
        '8': 5
    },
    '2': {},
    '4': {
        '8': 6
    },
    '8': {}
}

visited = set()
partial_explored = set()

def bfs_with_cost(graph, start_node, goal_node):

    queue = deque([(start_node, 0)])

    while queue:
        node, path_cost = queue.popleft()

        if node not in visited:
            print(f"{node} (Cost: {path_cost})")
            visited.add(node)

            if node == goal_node:
                print(f"Goal node {goal_node} reached!")
                return

            partial_explored.add(node)
```

```

        for neighbour, edge_cost in graph[node].items():
            if neighbour not in visited:
                queue.append((neighbour, path_cost + edge_cost))

        print(
            f"Nodes with partially explored children in current iteration:
{partial_explored}"
        )

goal_node = '8'
print(
    f"Following is the Breadth-First Search with Path Cost to reach
goal node {goal_node}"
)
bfs_with_cost(graph, '5', goal_node)

print("\nNodes with partially explored children in each iteration:")
print(partial_explored)

```

Output:

```

Following is the Breadth-First Search with Path Cost to reach goal node 8
5 (Cost: 0)
Nodes with partially explored children in current iteration: {'5'}
3 (Cost: 2)
Nodes with partially explored children in current iteration: {'3', '5'}
7 (Cost: 1)
Nodes with partially explored children in current iteration: {'7', '3', '5'}
2 (Cost: 5)
Nodes with partially explored children in current iteration: {'7', '3', '2', '5'}
4 (Cost: 6)
Nodes with partially explored children in current iteration: {'4', '3', '7', '2', '5'}
8 (Cost: 6)
Goal node 8 reached!

Nodes with partially explored children in each iteration:
{'4', '3', '7', '2', '5'}

```

Code: (UCS)

```

import heapq

def ucs(graph, start, goal):
    priority_queue = [(0, start, [start])]
    visited = set()

```

```

while priority_queue:
    cost, current_node, path = heapq.heappop(priority_queue)

    if current_node == goal:
        print("Goal reached! Total cost:", cost)
        print("Traversal path:", path)
        return

    if current_node not in visited:
        print("Visiting node:", current_node)
        visited.add(current_node)

        for neighbor, edge_cost in graph[current_node]:
            if neighbor not in visited:
                heapq.heappush(priority_queue,
                               (cost + edge_cost, neighbor, path +
                                [neighbor]))

graph = {
    'A': [('B', 1)],
    'B': [('C', 2), ('A', 1), ('R', 4), ('S', 3)],
    'C': [('B', 2), ('T', 1), ('O', 3), ('E', 2), ('P', 5)],
    'R': [('B', 4)],
    'S': [('B', 3)],
    'T': [('C', 1)],
    'O': [('C', 3), ('I', 1), ('N', 2)],
    'I': [('O', 1), ('Z', 2)],
    'N': [('O', 2)],
    'Z': [('I', 2)],
    'E': [('C', 2), ('G', 5)],
    'G': [('E', 5)],
    'P': [('C', 5), ('L', 5), ('F', 1), ('D', 3)],
    'L': [('P', 5)],
    'F': [('P', 1)],
    'D': [('P', 3)]
}

ucs(graph, 'C', 'Z')

```

Output:

```
Run Ask AI 137ms on 15:10:18, 04/05 ✓  
Visiting node: C  
Visiting node: T  
Visiting node: B  
Visiting node: E  
Visiting node: A  
Visiting node: O  
Visiting node: I  
Visiting node: N  
Visiting node: P  
Visiting node: S  
Visiting node: F  
Visiting node: R  
Goal reached! Total cost: 6  
Traversal path: ['C', 'O', 'I', 'Z']
```