CSCI323 Lab 2 Assignment (2025)

Name: Jeslyn Ho Ka Yan

7-digit UOW ID:8535383

🔧 Problem Setup

We define a class ElevatorProble to model the actions available at each level (walk to s+1, elevator to 2*s).

```
# ------------------ util.py ------------------
import heapq

class PriorityQueue:
    def  __init__(self):
        self.DONE = -100000
        self.heap = []
        self.priorities = {}

    def update(self, state, newPriority):
        oldPriority = self.priorities.get(state)
        if oldPriority is None or newPriority < oldPriority:
            self.priorities[state] = newPriority
            heapq.heappush(self.heap, (newPriority, state))
            return True
        return False

    def removeMin(self):
        while self.heap:
            priority, state = heapq.heappop(self.heap)
            if self.priorities[state] == self.DONE: continue
            self.priorities[state] = self.DONE
            return (state, priority)
        return (None, None)

# ------------------ elevator.py ------------------

class ElevatorProblem:
    def __init__(self, N):
        self.N = N

    def startState(self):
        return 1

    def isEnd(self, state):
        return state == self.N

    def succAndCost(self, state):
        result = []
        if state + 1 <= self.N:
            result.append(('walk', state + 1, 1.0))
        if state * 2 <= self.N:
            result.append(('elevator', state * 2, 1.5))
        return result

```

🔎 Uniform Cost Search Algorithm

We apply UCS to find the cheapest path. UCS always chooses the lowest-cost unvisited state.

```
def uniformCostSearch(problem):
    frontier = PriorityQueue()
    frontier.update(problem.startState(), 0)
    backpointers = {}
    costSoFar = {problem.startState(): 0}

    while True:
        state, currentCost = frontier.removeMin()
        if state is None:
            break
        if problem.isEnd(state):
            actions = []
            while state != problem.startState():
                action, prev, stepCost = backpointers[state]
                actions.append((action, state, stepCost))
                state = prev
            actions.reverse()
            return (currentCost, actions)

        for action, newState, stepCost in problem.succAndCost(state):
            newCost = currentCost + stepCost
            if newState not in costSoFar or newCost < costSoFar[newState]:
                costSoFar[newState] = newCost
                frontier.update(newState, newCost)
                backpointers[newState] = (action, state, stepCost)

    return (float('inf'), [])

def printSolution(solution):
    totalCost, history = solution
    print('Total cost:', totalCost)
    for step in history:
        print(step)

```

Sample Output Explanation

If n = 10, the output might be:

```
# Test the search for a building with 10 levels
problem = ElevatorProblem(10)
solution = uniformCostSearch(problem)
printSolution(solution)

```

```
Total cost: 5.0
('walk', 2, 1.0)
('elevator', 4, 1.5)
('walk', 5, 1.0)
('elevator', 10, 1.5)
```

```
N = int(input("Enter the number of levels (n): "))
problem = ElevatorProblem(N)
solution = uniformCostSearch(problem)
printSolution(solution)

```

```
Enter the number of levels (n): 10
Total cost: 5.0
('walk', 2, 1.0)
('elevator', 4, 1.5)
('walk', 5, 1.0)
('elevator', 10, 1.5)
```