DATE:27/06/25

EXP: 01

UNINFORMED SEARCH ALGORITHMS

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. Breadth-first Search
2. Depth-first Search
3. Depth-limited Search
4. Iterative deepening depth-first search
5. Uniform cost search
6. Bidirectional Search

Problem statement

You are given a list of floor requests for an elevator in a building. The elevator starts at floor 1. Your task is to determine the order in which the elevator should visit all requested floors **exactly once** such that the **total wait time** is minimized.

- **Total wait time** is defined as the sum of the individual wait times for each request.
- A request's wait time is the time taken to reach that floor starting from floor 1 and visiting each floor in sequence (one floor unit = 1 unit of time).

AIM:

Find the **optimal order** to serve all requests such that the **sum of waiting times** is minimized.

ALGORITHM:

- **Initialize BFS:**
  - Create an empty queue (deque).
  - For each floor in requests, create an initial state:
    - Starting from that floor
    - Path so far: a tuple with just that floor
    - Initial time = distance from start_floor to that floor
    - Enqueue the tuple (floor, path, time_so_far)
- **Start BFS Loop:**
  - While the queue is not empty:
    - Dequeue the front state: (current_floor, path, time_so_far)
    - If path contains all n floors:
      - Compute **total wait time** using total_wait_time(path, start_floor)
      - Update min_total_wait and best_order if current path is better
    - Else:
      - For every floor **not yet in path**:
        - Calculate move cost
        - Create a new state with:
          - Updated current floor
          - New path (append the next floor)
          - Updated time so far
        - Enqueue the new state
- **Return the best path and its total wait time**

Time complexity: O(n!n)

Space complexity O(n!n)

CODE:

```
from collections import deque
def total_wait_time(path, start_floor):
    time = 0
    current_floor = start_floor
    total_time = 0
    for floor in path:
        time += abs(floor - current_floor)
        total_time += time
        current_floor = floor
    return total_time
def bfs_min_wait(requests, start_floor=1):
    n = len(requests)
    queue = deque()
    min_total_wait = float('inf')
    best_order = []
    print("\nExploring State Space with BFS:")
    # Initial states: starting from each floor
    for floor in requests:
        path = [floor]
        initial_wait = abs(start_floor - floor)
        queue.append((floor, tuple(path), initial_wait))
    while queue:
        current_floor, path, time_so_far = queue.popleft()
        print(f"  Visiting Path: {path}, Wait So Far: {total_wait_time(path, start_floor)}")
        if len(path) == n:
            total_wait = total_wait_time(path, start_floor)
            if total_wait < min_total_wait:
                min_total_wait = total_wait
                best_order = path
            continue
```

```
        for next_floor in requests:
            if next_floor not in path:
                new_path = path + (next_floor,)
                move_cost = abs(current_floor - next_floor)
                queue.append((next_floor, new_path, time_so_far + move_cost))
    return best_order, min_total_wait
# Example usage
if __name__ == "__main__":
    user_input = input("Enter floor requests separated by spaces: ")
    requests = list(map(int, user_input.strip().split()))
    order, min_wait = bfs_min_wait(requests)
    print("\nOptimal Floor Visit Order Found:")
    print(f"  Order: {order}")
    print(f"  Minimum Total Wait Time: {min_wait}")
```

OUTPUT:

```
Visiting Path: (8, 2), Wait So Far: 20
Visiting Path: (8, 5), Wait So Far: 17
Visiting Path: (8, 9), Wait So Far: 15
Visiting Path: (9, 2), Wait So Far: 23
Visiting Path: (9, 5), Wait So Far: 20
Visiting Path: (9, 8), Wait So Far: 17
Visiting Path: (2, 5, 8), Wait So Far: 12
Visiting Path: (2, 5, 9), Wait So Far: 13
Visiting Path: (2, 8, 5), Wait So Far: 18
Visiting Path: (2, 8, 9), Wait So Far: 16
Visiting Path: (2, 9, 5), Wait So Far: 21
Visiting Path: (2, 9, 8), Wait So Far: 18
Visiting Path: (5, 2, 8), Wait So Far: 24
Visiting Path: (5, 2, 9), Wait So Far: 25
Visiting Path: (5, 8, 2), Wait So Far: 24
Visiting Path: (5, 8, 9), Wait So Far: 19
Visiting Path: (5, 9, 2), Wait So Far: 27
Visiting Path: (5, 9, 8), Wait So Far: 21
Visiting Path: (8, 2, 5), Wait So Far: 36
Visiting Path: (8, 2, 9), Wait So Far: 40
Visiting Path: (8, 5, 2), Wait So Far: 30
Visiting Path: (8, 5, 9), Wait So Far: 31
Visiting Path: (8, 9, 2), Wait So Far: 30
Visiting Path: (8, 9, 5), Wait So Far: 27
Visiting Path: (9, 2, 5), Wait So Far: 41
Visiting Path: (9, 2, 8), Wait So Far: 44
Visiting Path: (9, 5, 2), Wait So Far: 35
Visiting Path: (9, 5, 8), Wait So Far: 35
Visiting Path: (9, 8, 2), Wait So Far: 32
Visiting Path: (9, 8, 5), Wait So Far: 29
Visiting Path: (2, 5, 8, 9), Wait So Far: 20
Visiting Path: (2, 5, 9, 8), Wait So Far: 22
Visiting Path: (2, 8, 5, 9), Wait So Far: 32
Visiting Path: (2, 8, 9, 5), Wait So Far: 28
Visiting Path: (2, 9, 5, 8), Wait So Far: 36
Visiting Path: (2, 9, 8, 5), Wait So Far: 30
```

```
  Visiting Path: (5, 2, 8, 9), Wait So Far: 38
  Visiting Path: (5, 2, 9, 8), Wait So Far: 40
  Visiting Path: (5, 8, 2, 9), Wait So Far: 44
  Visiting Path: (5, 8, 9, 2), Wait So Far: 34
  Visiting Path: (5, 9, 2, 8), Wait So Far: 48
  Visiting Path: (5, 9, 8, 2), Wait So Far: 36
  Visiting Path: (8, 2, 5, 9), Wait So Far: 56
  Visiting Path: (8, 2, 9, 5), Wait So Far: 64
  Visiting Path: (8, 5, 2, 9), Wait So Far: 50
  Visiting Path: (8, 5, 9, 2), Wait So Far: 52
  Visiting Path: (8, 9, 2, 5), Wait So Far: 48
  Visiting Path: (8, 9, 5, 2), Wait So Far: 42
  Visiting Path: (9, 2, 5, 8), Wait So Far: 62
  Visiting Path: (9, 2, 8, 5), Wait So Far: 68
  Visiting Path: (9, 5, 2, 8), Wait So Far: 56
  Visiting Path: (9, 5, 8, 2), Wait So Far: 56
  Visiting Path: (9, 8, 2, 5), Wait So Far: 50
  Visiting Path: (9, 8, 5, 2), Wait So Far: 44

Optimal Floor Visit Order Found:
  Order: (2, 5, 8, 9)
  Minimum Total Wait Time: 20
```

RESULT:

The algorithm finds the optimal order visit elevator floor requests and minimises total cumulative wait time and the output has been verified