### Algorithm: Dijkstra's Algorithm

In the kingdom of Algoria, Sir Cedric and Ember faced the Crystal Caverns, a place of shimmering beauty but treacherous paths. Princess Elara was hidden deep within, and the shortest path to her was through the caverns.

#### Initialize Data Structures:

* Sir Cedric used a bag of enchanted pebbles (priority queue) to track the shortest paths.
* He carried a scroll (dictionary) to record the shortest distance to each chamber.

#### Find the Shortest Path:

* Sir Cedric began at the entrance, placing an enchanted pebble there with a cost of 0.
* He moved through the chambers, always choosing the one with the smallest cost first.

#### Repeat:

* He continued until he found the chamber where Princess Elara was held.

#### Implementation:

| **import** **heapq**  **def** dijkstra(caverns: Dict[int, List[Tuple[int, int]]], start: int) -> Dict[int, int]:  priority\_queue = [(0, start)]  distances = {chamber: float('inf') **for** chamber **in** caverns}  distances[start] = 0  **while** priority\_queue:  current\_distance, chamber = heapq.heappop(priority\_queue)  **if** current\_distance > distances[chamber]:  **continue**  **for** next\_chamber, weight **in** caverns[chamber]:  distance = current\_distance + weight  **if** distance < distances[next\_chamber]:  distances[next\_chamber] = distance  heapq.heappush(priority\_queue, (distance, next\_chamber))  **return** distances  *# Example usage:*  caverns = {  1: [(2, 2), (3, 4)],  2: [(3, 1), (4, 7)],  3: [(4, 3)],  4: []  }  start = 1  print(dijkstra(caverns, start)) *# Output: Shortest path distances* |
| --- |

#### Explanation:

Initialize:

* priority\_queue: A bag of enchanted pebbles to track the shortest paths.
* distances: A scroll to record the shortest distance to each chamber.

Find the Shortest Path:

* Sir Cedric moved through the chambers, always choosing the path with the smallest cost first.

Repeat:

### He continued until he found the chamber where Princess Elara was held.