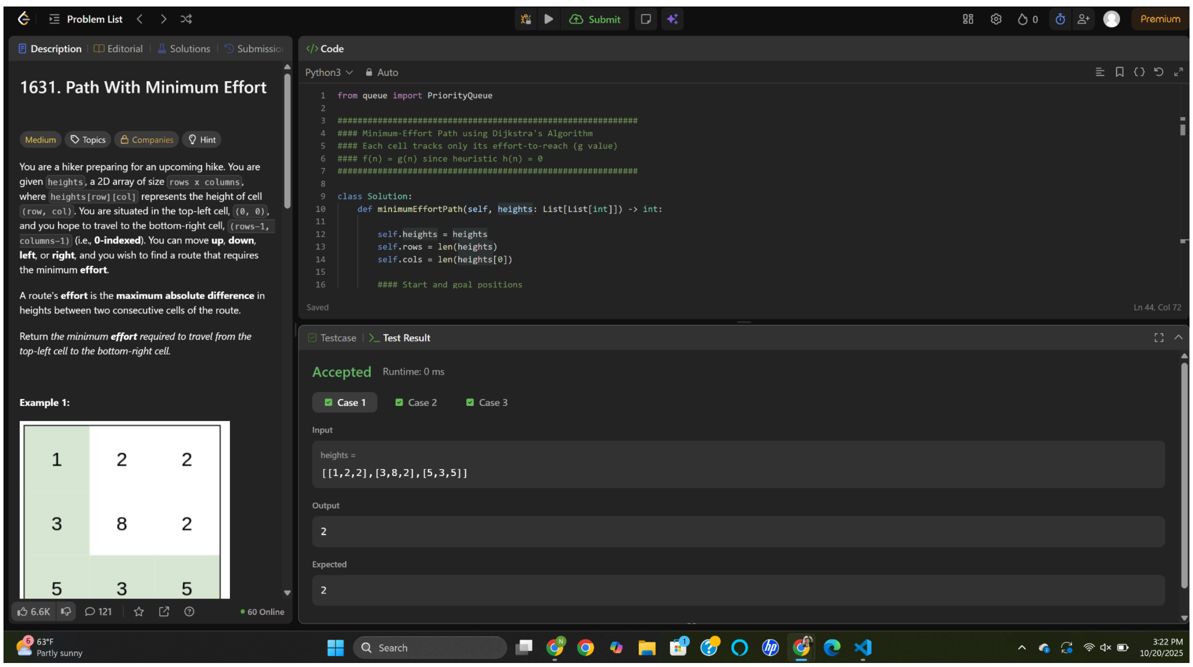
Leetcode Problem:
https://leetcode.com/problems/path-with-minimum-effort/

Explanation of Approach:
The goal of this problem was to find a path from the top-left to the bottom-right of a grid where the cost of each path is equal to the absolute difference in height between adjacent cells. The goal was to minimize the maximum effort, meaning that the optimal path would be the one that has the lowest maximum height difference. I decided to use Dijkstra's algorithm to solve this problem because of the fact that each edge cost was different, and the A* algorithm would rely on having a heuristic. Due to the fact that an A* algorithm would rely on a heuristic to guide the search and that there does not seem to be a meaningful heuristic to use, I opted for Dijkstra's algorithm instead. By using Dijkstra's algorithm, correctness and optimality are guaranteed, even though there is no heuristic that narrows down the search space. Since the grid/search space is relatively small, the absence of a heuristic does not significantly impact the performance of the algorithm.

Screenshot of Accepted Solution:


Code:

```
from queue import PriorityQueue

#################################################################
#### Minimum-Effort Path using Dijkstra's Algorithm
#### Each cell tracks only its effort-to-reach (g value)
#### f(n) = g(n) since heuristic h(n) = 0
#################################################################
```

```
class Solution:
   def minimumEffortPath(self, heights: List[List[int]]) -> int:

      self.heights = heights
      self.rows = len(heights)
      self.cols = len(heights[0])

      #### Start and goal positions
      self.start = (0, 0)
      self.goal = (self.rows - 1, self.cols - 1)

      #### Initialize effort grid (g values)
      g = [[float('inf')] * self.cols for _ in range(self.rows)]
      g[self.start[0]][self.start[1]] = 0

      #### Priority queue for open set
      open_set = PriorityQueue()
      open_set.put((0, self.start))  # (effort, position)

      #### Movement directions: down, up, right, left
      directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]

      while not open_set.empty():
         current_f, (x, y) = open_set.get()

         #### Stop if goal is reached
         if (x, y) == self.goal:
            return g[x][y]

         for dx, dy in directions:
            nx, ny = x + dx, y + dy

            if 0 <= nx < self.rows and 0 <= ny < self.cols:

               #### Step cost = height difference
               step_cost = abs(self.heights[nx][ny] - self.heights[x][y])

               #### Path effort = max step seen so far
               new_g = max(g[x][y], step_cost)

               #### Update if better path found
               if new_g < g[nx][ny]:
                  g[nx][ny] = new_g
                  open_set.put((new_g, (nx, ny)))
```