Strategy

The approximation algorithm uses a simple greedy approach. For every vertex in the graph, the algorithm starts a path from that vertex and repeatedly chooses the unvisited neighbor connected by the highest-weight edge. This produces one greedy path per starting vertex. After trying all starting vertices, the algorithm returns the path with the greatest total weight. Tie-breaking is deterministic based on Python's dictionary iteration order (no randomness is used).

Analytical Runtime

The greedy walk from any starting vertex can visit at most nnn vertices, and at each step it may scan up to nnn neighbors. This gives a worst-case cost of $O(n^2)$ per start. Since the algorithm tries all nnn starting vertices, the total runtime is:

$O(n) \times O(n^2) = O(n^3)$

This runtime is polynomial, so the approximation method scales well to larger graphs and meets the project requirement for polynomial efficiency.

Performance

The greedy algorithm performs well on most test cases. It typically finds paths close to the optimal solution and runs very quickly, even on larger graphs. On structured graphs where heavier edges tend to lead toward longer paths, greedy often achieves the exact optimal value. The implementation successfully handles large problem sizes and behaves efficiently on Gradescope tests.

Non-Optimal Case

We include a test case where the greedy algorithm does not achieve the optimal answer. In this example, the algorithm chooses a locally heavy edge early in the path, which prevents it from reaching another region of the graph that contains a sequence of moderately weighted edges whose total would be larger. This demonstrates the main limitation of greedy: it makes decisions based only on the next step and cannot look ahead to consider how early choices affect the overall path. The run_nonopt_cases.sh script includes this example.