Approximation Algorithm for Longest Path Problem

Author: Patrick Jabalee-Farrell
Course: CS412 / Project

---

1. Overview of Solution

The implemented approximation algorithm seeks a long path in a weighted, undirected graph using a randomized greedy strategy. The key idea is to iteratively construct candidate paths, preferring heavier edges while allowing randomness to escape local optima. This approach is an **anytime algorithm**, meaning it can return a valid solution at any time limit while potentially improving with more runtime.

Algorithm Highlights

1. **Edge Collection:**
   All undirected edges are collected once to allow random selection.
2. **Greedy Random Walk:**
   - Start from a random edge (u, v).
   - Extend the path iteratively by choosing neighbors of the current vertex that haven't been visited yet.
   - Use a **top-k selection** of heaviest edges and pick randomly among them to introduce variability.
3. **Anytime Behavior:**
   - Continue iterating over candidate paths until either the time limit is reached or no better path is found after examining all edges.
   - A final loop introduces additional randomization, sometimes selecting random neighbors instead of the heaviest, further exploring the search space.
4. **Output:**
   Returns the best path found and its total weight.

---

2. Key Design Decisions

- **Top-k selection:** Balances greedy weight maximization and randomness to escape local optima.

- **Randomized choice:** Prevents the algorithm from consistently producing the same path in graphs with multiple high-weight options.
- **Anytime loop:** Allows termination at any moment, providing a usable solution even under strict time constraints and to continue as long as the user wants to keep improving solutions
- **Heap usage:** Efficiently selects top-k neighbors without full sorting.

---

3. Observations and Performance

- The algorithm performs well on sparse and dense graphs with moderate vertex counts.
- Decreasing k favors greedy selection, while increasing it allows more exploration.
- The runtime is controlled by the time limit parameter, and longer limits generally produce heavier paths.

---

4. Development Process

Before reaching the final solution, several strategies were attempted:

1. **Full Greedy Loop:**
   Initially, I tried a full greedy solution before adding in the randomness by always selecting the heaviest unvisited neighbor. This often got stuck in local maxima and failed to explore alternative high-weight paths. It also was never the answer used at the end so it seemed like it was just wasting time.
2. **Look-Ahead Heuristics:**
   Next, I tried adding look-ahead to estimate the benefit of extending each neighbor. While theoretically promising, this increased the runtime significantly, making it impractical for larger graphs and the trade-off of making better choices but increasing runtime proved not to be worth it. Just doing normal greedy picks made the algorithm try so many more possible solutions it almost always resulted in better answers.
3. **Multiple Runs per Vertex:**
   Another attempt involved repeatedly running the code proportionally to the amount of edges. This improved quality but it was similar to just running the anytime but ran way less times.

4. **Anytime Randomized Approach (Final Solution):**
   Based on previous challenges, I adopted an anytime, randomized strategy. By combining random edge starts, top-k greedy selections, and iterative improvement with a stopping criterion based on time, this algorithm was the most efficient and effective that I could find.