Artificial Intelligence

Name: Dariga Kassenova

Task 1: Longest Path

To identify the words for the *print_long_path* function, I used a custom heuristic algorithm.The algorithm began its search from nodes with the lowest degree, since such nodes are located on the outer edges of the graph and have fewer connections to other words. This increases the likelihood of constructing a longer path without cycles, effectively traversing the network from edge to edge. The path was then greedily expanded by selecting unvisited neighbors with similarly low degrees, avoiding loops. After several iterations, the route with the greatest number of unique nodes was selected, and its first and last words were fixed as *start_word* and *end_word* in the *print_long_path* function.

Task 2: Longest Quote *(explanation was written with the help of ChatGPT (GPT-5, November 2025) to improve clarity and wording)*

The helper function performs a linear traversal through the *original_tokens* of the text, the algorithm sequentially checks each word to find the longest continuous chain of words that are connected in the graph and not included in *rare_tokens*. When a connection break or a rare word is encountered, the current chain is reset. Finally, the longest sequence found is selected, and its first and last words are fixed as the *start_word* and *end_word* in the *print_long_quote* function.

Task 3: Most Expensive Path *(explanation was written with the help of ChatGPT (GPT-5, November 2025) to improve clarity and wording)*

To select the words for *print_expensive_path*, I used a greedy search approach. I formed a set of starting nodes from those with the lowest and highest degrees, covering both the edges and the center of the network. For each start node, the algorithm built a path without cycles: at each step, it chose an unvisited neighbor with the maximum edge weight, where the weight was taken from the *distance_matrix*. The search was limited by *max_depth* to prevent infinite growth and revisits. For each path, the total edge cost was accumulated, and the pair with the highest *total_cost* was saved. The final expensive path defined *start_word = path[0]* and *end_word = path[-1]*.

Task 4: Most Expensive Quote

The algorithm uses similar logic to task Longest Quote (6.2). For each continuous sequence, the total edge cost is accumulated from the *distance_matrix*, and the one with the highest *max_cost* is retained.

Task 5: Heuristic Search (Part c. Heuristic Description) *(explanation was written with the help of ChatGPT (GPT-5, November 2025) to improve clarity and wording)*

Both functions *complete_sentence* and *start_sentence* were implemented based on a modified **A*** search algorithm.

In *complete_sentence*, the algorithm starts from the last word of the left context (*start*) and searches for a connection to the first word of the right context (*end*), or to a period if the right side is missing. I used the adjacency data and token frequency counts (*token_counts*) to build an outgoing neighbor map *out_n*, showing which words most commonly follow each other in the text. I then combined several heuristic factors to guide the search. The total cost *f* consists of three components:

1. The ***path cost*** *g*, which adds a small base cost for each step plus an additional penalty for rare word transitions.
2. The ***heuristic estimate*** *h*, computed using a BFS distance from the current node to the target node, representing how close a word is to the goal.
3. A ***length penalty*** that prevents the algorithm from producing sentences that are too short.

The rarity penalty is calculated as

-log((adj[(u,v)] + 1) / (max_count + 1)) − 0.5 · log(freq(v) + 1)

This formula penalizes uncommon transitions and infrequent words. The first part, *-log((adj[(u,v)] + 1)/(max_count + 1))*, decreases the score for bigrams that rarely occur in the text, while the second term, *−0.5·log(freq(v)+1)*, slightly penalizes globally rare words. Together, these components push the search toward statistically probable and natural word sequences. The algorithm skips <RARE> tokens ensuring grammatical and coherent completions. After the search, I merge the left context, the generated segment, and the right context, removing any duplicated boundary words.

For *start_sentence*, I used the same principle but in reverse. I built an inverse neighbor map *in_n* to identify which words typically come before others in the text. The algorithm begins from the first word of the right-hand context and moves backward through the graph, using the same cost function and heuristics as in *complete_sentence*. Once a valid sequence is found, I reverse it and join it with the right context to form a coherent full sentence.

Both heuristics balance statistical plausibility and structural coherence: the rarity penalty ensures semantic naturalness, while the distance heuristic and length control maintain syntactic flow and sentence balance.