I don't explicitly use the terms "Mapper" and "Reducer" as I would in a traditional MapReduce framework like Hadoop. However, I can map the concepts of Mapper and Reducer to the steps performed in the code.

1. **Mapper Equivalent**:
   * In my case, the "Mapper" equivalent is represented by the part of the code where I process each input path and generate key-value pairs. This is similar to the mapping phase in MapReduce.
   * Specifically, the mapping functionality is achieved through the list comprehension:

[(path.split("/"), len(path.split("/"))) for path in input\_data]

* + Here, I iterate over each input path (**path**), split it by "/" to obtain its components, and calculate the length of the path. Then, I create a tuple containing the path components as the key and the length of the path as the value. This is analogous to emitting key-value pairs in the mapping phase of MapReduce.

1. **Reducer Equivalent**:
   * In my code, the "Reducer" equivalent is represented by the part where I find the longest path(s) among the mapped data. This is similar to the reducing phase in MapReduce.
   * Specifically, the reducing functionality is achieved through the **max** function:

longest\_path = max(mapped\_data, key=lambda x: x[1])

* + Here, I use the **max** function to find the tuple with the maximum value in the mapped data based on the length of the path (which is the second element of each tuple). This operation effectively aggregates the mapped data by selecting the tuple with the longest path length.