Refactor code for PA2 DictionaryTrie.cpp

Cheng Qian
A53209561

1. Optimize the insert function.

Originally I updated the “success” flag every time for a new node on the path, now I found that whether the insertion is successful is only related to the “is end” flag of the last node. Because in MWT, there is only one way from the root to the node that represent the end of the word. If it is the “isEnd” flag is true, that means that word has been inserted before, which make the insertion unsuccessful. This optimization only reduces some of the constant time update operations in the while loop, which does not affect the time complexity, but makes the code more understandable.

2. Separate code into different module according to their functionality.

In “predictCompletions” method, I modularized the input validation part and sub-trie root localization part. I used to think that combining as much as possible code together will make the code shorter and reduce the traverse time of a prefix. However, I found that this kind of “shorter” code is hard to maintain if one of the part arises bug. Moreover, the two parts have basically different functions, I mean, the input validation part and sub-trie root localization. After separation, it becomes easier for other programmer understand my code. More importantly, re-traverse the string does not significantly do harm to the efficiency of the code, because traverse through the prefix takes linear time. In big-O notation we can combine O(N)+O(N) = O(N).

3. Maintain a capacity of priority queue.

In the predictCompletions function, I implement the method with priority queue. The algorithm is traversing all the “isEnd” node and pushing them into the priority queue. Overwrite the comparator of PQ to assigned the node with high frequency high priority. Finally pop out num_completions nodes. The optimization is to maintain a capacity of the queue. If the qualified nodes number is beyond what we want, we pop out the low priority nodes as soon as possible. Finally we just pop out all the nodes in the priority queue. This optimization will reduce the size of my PQ, especially when the sub-MWT is very large and space consuming. It does not change the time complexity.

Some other small changes:
Removed some test line and clean up the code.
Standardized the BFS part and added comments on how the priority queue was designed.