COMP20007 Assignment 2: Spelling correction

Name: Renjie(Rudy) Meng ID: 877396

Analysis of task 3 – Spelling checking

- **Applied approach**
  - *Data structure*

    I use hash table with move to front to create a dictionary. The hash table contains two data type, int "size" used to store the size of hash table and another is Bucket "**buckets" which is an array of linked lists. The size of the hash table is 4/3 times the dictionary size to reduce collision. The hash table use separate chaining to hold collision. Each node of each linked list is called a bucket. In each bucket, there are two data type, one is a "data" pointer to a piece of data used to store word in dictionary and another is "next" which is a pointer point to another bucket.
  - *Algorithm*
  - My algorithm is based on hash table. First create a hash table of dictionary. I use the function "hash_table_has" to check the existence of word. For each word in the document, I use hash function to hash it then get the hash value. Then, access the linked list with this value and check if the word in this linked list. If yes, then the word is in the dictionary, otherwise, it is not in the dictionary.
  - *Asymptotic time complexity*

    Assume there are n words in document and m words in dictionary. In average, it takes O(1) to insert each word to hash table and O(1) time to check each word in document. In the worst case, it takes O(m) to insert each word to hash table and the worst case for check is that all word hash to the same bucket. And we have a m long linked list to traverse. The cost of checking each word is O(m). Therefore, in average case, the complexity is O(1*m+n*1) -> O(m+n). In worst case, the complexity is O(m*m + n*m) -> $O(m^2+mn)$.

- **Alternative approach**
  - *Data structure*

    The data structure of dictionary I used for alternative approach is the linked list. In this linked list, there are a "head" pointer, a "last" pointer as well as an int type data called "size" to store the size of linked list. In each node of linked list, there are two data type, one is a "data" pointer to a piece of data and another is a "next" pointer which points to another node.
  - *Algorithm*

    Linear searching in the linked list. For each word in the document, I traverse the linked list to access data in each node one by one to check if the word in document exists in the dictionary. If find the word in the dictionary then finish linear search for this word and turn to next word in document, otherwise we will reach the end of the dictionary which is a NULL pointer. If we reach the NULL pointer, print "word?" and turn to next word in document.
  - *Asymptotic time complexity*

    Assume there are n words in the document ang m words in the dictionary. In both average case and worst case, checking each word in dictionary cost O(m) times. Therefore, in both average and worst case, linear searching has a complexity in O(m*n).
  - *Reason of discarding*

    The asymptotic time complexity of this approach is in O(mn). In average case, the asymptotic time complexity for that hash table based algorithm is O(n+m). Since O(mn) > O(n)+m, I choose the hash table based algorithm. Furthermore, the worst case of hash based algorithm is hard to occur, which means all item has same hash value is almost impossible.

Analysis of task 4 – Spelling correction

- **Applied approach**
  - *Data structure*

    I use both hash table and linked list data structure for the dictionary. There two structures are the two data structure mention in analysis of task 3 respectively.
  - *Algorithm*

    First create a hash table of dictionary. For each word in the document. I use hash function "hash_table_has" to check if the word exists in dictionary. If yes print it out, else go to linear searching of edit distance checking in the linked list. For the edit distance checking, we find words with edit distance 1, 2 and 3 at same time by traversing the linked list once. First, if we find word with edit distance 1, then print it out and break; Then, we also store first word with edit distance 2 and 3. If we do not find word with edit distance 1, we go to check word with edit distance 2. If it exists print it out and check next word in document. If not, go to check word with edit distance 3. If it exists then print it out and stop. If not exist, print out "word?" and check next word in document.
  - *Asymptotic time complexity*

    Assume there are n words in document and the average length is a as well as m words in dictionary and the average length is b. As discussed in analysis of task 3. In average, the cost to create a hash table and checking existence of word is O(m+n). And, in worst case, it costs $O(m^2+mn)$. For edit distance, it cost O(ab) for each computation. And two if condition clause as well as O(m) for traversing the whole linked list. For linear searching, both average case and worst case costs O(m). Therefore, for each word in document, we need O(m*ab+3*m+m) -> O(m(ab+4)**)** in both average and worst case. Therefore, the average total cost is O(m+n+nm(ab+4)) -> **O(n+m+nm(ab+4))**. In worst case, the total cost is **$O(m^2 + nm + nm(ab+4))$**.

- **Alternative approach**
  - *Data structure*

    Only use linked list data structure for dictionary. The linked list data mentioned in analysis of task 3 in alternative approach.
  - *Algorithm*

    Linear searching in the linked list. For each word in document, we traverse the linked list to check if the word exists in dictionary. If not we traverse the linked list to check if there are word in dictionary has edit distance 1 print it out. If not, check for edit distance 2, if there are word with edit distance 2 print it out. If not, check for edit distance 3, if there are word with edit distance 3 print it out. Otherwise, we cannot find the correct version within edit distance 3 of word in dictionary, then print "word?".
  - *Asymptotic time complexity*

    Assume there are n words in document and the average length is a as well as m words in dictionary and the average length is b. For edit distance, it cost O(ab) for each computation. In both average case and worst case, the cost of traversing linked list once is O(m). Thus, in worst case, for each word we check for all three edit distance and original, we need go through linked list 4 times and the cost is O(4m). As well as 4 conditional clauses to decide whether true or false and cost(4m). And compute edit distance cost O(ab). Thus, checking each word costs O(4m + 4m + mab) -> O(m(ab+8)). Therefore, in worst case the total cost is O(nm(ab+8));
  - *Reason of discarding*

    The applied algorithm runs faster than the alternative method. Because in the applied method, I only need to traverse linked list once but in alternative method I usually need to traverse for four times. From complexity, the applied approach's worst case is hard to occur with a good hash function. Thus, I compute the average case of applied approach – O(m + n + nm(ab+4)) with the average case of alternative case – O(nm(ab+8)). O(m + n + nm(ab+4)) and O(nm(ab+8)) both in O(mn), but O(nm(ab+8)) is larger since ab+8 > ab+4.