3805ICT Advanced Algorithms – Assignment 1

Chris Vuong – s5193954

**Question 5:** The goal of a ladder-gram is to transform a source word into the target

word on the bottom rung in the least number of steps. During each step, you must

replace one letter in the previous word so that a new word is formed, but without

changing the positions of the other letters. All words must exist in the supplied

dictionary (dictionary.txt). For example, we can achieve the alchemist's dream of

changing LEAD to GOLD in 3 steps or HIDE to SEEK in 6 steps. Minimise the number of

steps.

**Idea:**

Let call the first word is the source and the last word is the target, we treat each word as a node of a graph. At each source word, we have e edges where e is the permutation that will make up n-1 characters. For example, we have the word “LEAD”, then at the node “LEAD”, we have 4 edges as “\_EAD”, “L\_AD”, “LE\_D”, and “LEA\_”. From each edge, it will connect to a word or a node that will share the same feature. Because each edge is treated equally, the Breadth-first search algorithm will be used to find the closes target from this graph.

**Data Structure**

* We need a set of strings that store the dictionary words.
  + Set<string> dict
* We need a map with a key as a string (edge) and a value as an array of strings (vertex). This map is the graph that will check edges (normally a graph will check vertices)
  + Map<string , vector<string> graph;
* We need a queue structure that will stack all vertices as first in first out. It also has an array of string for storing the path.
  + Queue <pair < string, vector<string> >> queue;

**Pseudo-Code**

|  |
| --- |
| Function generating\_dict(size s): => set<string>  set<string> dict;  Loop to all word in dictionary: // O(n), n is the number of word in dictionary.txt  If word.size == s :  Push the word in dict  Return dict;  Function generating\_graph(set<string> dict): => map<string, vector<string> >  map<string, vector<string> > graph;  loop to all word in dict:  loop from i=0 to all word.size:  graph[word.substring(0,i) + “\_” + word.substring(i+1)].push\_back(word)  return graph  Function FindShortestChain(string source, string target, set<string> dict) => vector<string>  map<string, vector<string> > graph = generating\_graph(dict);  queue< pair<string, vector<string> > q = {make\_pair(source, {source} )} ; // create the queue and add the source in  map<string, bool> visited;  while q is not empty:  current = q.front();  q.pop();  if current.first == target:  return current.second;  word = current.first;  loop from i=0 to word.size:  edge = word.substring(0,i) + “\_” + word.substring(i+1);  loop through all the node of graph[edge]:  if ( node is visited):  visit[node] = true;  q.push( make\_pair(node, new vector(current.sencond.add(node);  return empty vector if cannot find the target. |

**Performance:**

|  |  |  |
| --- | --- | --- |
| Generating dictionary | O(n) | The n value is the number of words in the dictionary.txt |
| Generating graph | O(n\* s) | The n value is the number of words that are generated after being sorted out from generating dictionary. |
| Finding the shortest chain | O(n \* s) | The worst case is that there is no result of the chain can be found. The loop will go through all the possible nodes, and these nodes are the number of nodes stored in the graph. So, this function’s worst-case scenario will be depended on the number of nodes from the graph, which is O(n\*s) |

**OUTPUT:**

|  |
| --- |
| ./a lead gold  lead load goad gold  Total of 3 steps |
| ./a hide seek  hide bide bids beds bees sees seek  Total of 6 steps |
| ./a lead get  target and source must have the same size |
| ./a  please giving the source and target |