Word ladders report

Otto Holmström

April 8, 2020

1 Results

The script shows all results as correct. On the largest input (6huge2.in), it takes the program between 5 to 10 seconds to run and using a time counter in the program as well as using the terminal as stdout, the calls to the BFS-function take up from 30ms to 70ms of the total execution time, showing that something else is the bottleneck in the program.

2 Implementation details

The solution was implemented using lookup tables using two maps, to convert the strings into integers for use internally in the program. Each input is given an integer depending on when it is read; the first string will correspond to: 0, the second: 1 and so on. With this, we can represent the graph as an array of (adjacency) lists, giving us constant time complexity to access these lists in the BFS algorithm, compared to logarithmic time if we had used a map to map the input words to their adjacency lists.

In the BFS function, we use a list where we store the nodes to visit, and an array to keep track of the visited nodes. We also use an array to store the predecessor to a node which we can later use to find the path length.

Because we use a list and arrays, all the container operations of access and removal have $O(1)$ time complexity. If we define n to be the number of nodes and m to be the number of edges, we have at least $O(n)$ time complexity in the BFS algorithm. However, for each node we also have to check all the neighbours, called degree(n). The total time complexity thus becomes $O(n + \sum_{k=1}^{n} \text{degree}(k))$.

The sum of all neighbours, $\sum_{k=1}^{n} \text{degree}(k)$ is however simply the number of edges m in the graph; this gives that $\sum_{k=1}^{n} \text{degree}(k) = m$ (not $2m$ because the graph in this lab is directed) and the BFS algorithm time complexity is thus $O(n + m)$.

To check which words have an edge between eachother however, we need to compare each word against every other word; this is a typical $O(n^2)$ operation which means that the total running time is $O(n^2)$, since $m \leq n$.