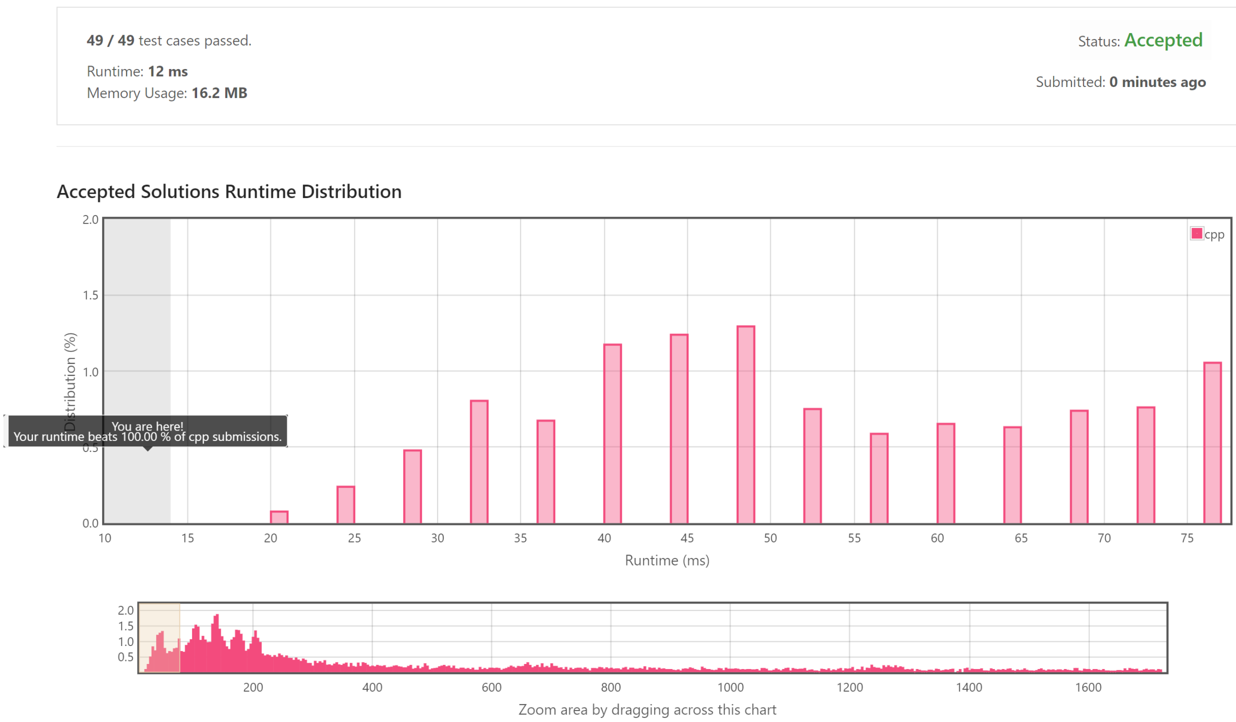
1. use dijkstra (which is bfs in the unweighted case), there are n vertices and $L \cdot |\Sigma|$ edges (which is tight in the worst case). we use hash table to store the strings, if a string changes a letter, its hash value can be re-computed in $O(1)$. total time $O(L \cdot |\Sigma|)$.
2. deterministic $O(L \cdot |\Sigma|)$, avoid hash table: we can map a substring to a node in the suffix tree of the concatenation of all strings (constructing suffix tree is $O(L)$). for a string s of length m, store it as a triple $(map(s[1..i-1]), s[i], map(s[i+1..m]))$, which can be viewed as an integer $\leq \text{poly}(L)$. the queries for edges are also triples. We can then perform the queries offline, by radix sort.
3. we can further get deterministic $O(L)$, by implicitly storing the edges. string s is stored in each "edge group" $(map(s[1..i-1]), map(s[i+1..m]))$, for each i. all edge groups contain L strings in total. when we visit s during BFS, delete it from all edge groups (assume edge groups are represented as doubly linked list). when we add the neighbors of s during BFS, all strings in the edge groups related to s haven't been visited at that time, so the total running time is $O(L)$.



Remark. the problem allows that beginWord not be contained in wordList, but endWord must be in wordList.

References