University of Waterloo
CS240 Winter 2025
Programming Question 2

Due Date: Tuesday, March 11 at 5:00pm

Please read `https://student.cs.uwaterloo.ca/~cs240/w25/assignments.phtml#guidelines` for guidelines on submission. Submit the file pq2.cpp electronically to Marmoset.
Grace period: submissions made before 11:59PM on March 11 will be accepted without penalty. Your last submission will be graded. Please note that submissions made after 11:59PM **will not be graded** and may only be reviewed for feedback.

Question 1 [20 marks]

Design and implement `PrunedTrie` over alphabet $\Sigma = \{a, e, o\}$. Your pruned trie should be implemented exactly as the pruned trie in Module06, except it is over alphabet with three letters (a, e, o). Provide an implementation of the class below. The running time for all methods should be $O(|w|)$, where $|w|$ is the word length. All printing should be done to the standard output.

Your program will read words from the standard input. Words are guaranteed to be over alphabet $\Sigma = \{a, e, o\}$. For the first four functions you are required to implement, words will end with the special end of word character '$'. For each word w, we define *goodness* function, which adds up 100 for each character 'o', 10 for each character 'e', and 1 for each character 'a'. For example, $goodness(aoaaee\$) = 1 \cdot 3 + 10 \cdot 2 + 100 \cdot 1 = 123$. We define goodness of an empty string as -1, i.e. $goodness(\$) = -1$.

Important: If your trie is not pruned, i.e. the standard trie, you will receive at most 5/20 marks.

```
PrunedTrie{
// add any fields and methods, as necessary
public:
  int size();                     // returns total number of keys stored in the trie
  void insert(const string& w);  // inserts word w; word w ends with $
  void remove(const string& w);  //  removes  word w; word w ends with $
  bool search(const string& w);  //  returns true if word w is in the trie
                                  // false otherwise; word w ends with $
  int prefix_search(const string& p); // finds the word w in the trie with
                                       // prefix p and returns the goodness of w.
                                       // If there are multiple w matching prefix p
                                       // returns w with largest goodness
                                       // p ends with $$
                                       // if prefix p not found, returns -1
  int num_nodes(const string& w);// First search for w in the trie. Search stops
                                 // at node v. Return the number of nodes in subtree
                                 // of v, including v.
                                 // Note that w may or may  not end with $
  int num_keys(const string& w); // First search for w in the trie. Search stops
                                 // at node v. Return the number of keys in subtree
                                 // of v, including v.
                                 // Note that w may or may  not end with $
```

You may use C++ vector, stack, string, and pair data structures and smart pointers. No other data structures/algorithms are allowed.

Place your program in file pq2.cpp. We provide you with a starter code that has the main function that accepts commands from the standard input. You may assume all inputs are valid, i.e. we will never input a word having characters outside the alphabet. See the starter code for the description of the commands. You are not allowed to modify the main function.

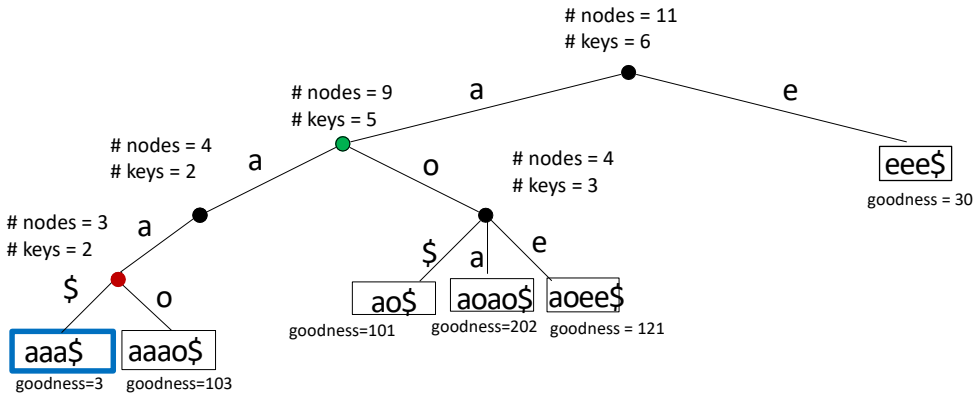

Figure 1: `PrunedTrie`

Below we provide several example outputs for the Pruned trie in Fig. 1.

- `prefix_search(e$) = 30.`
- `prefix_search(a$) = 202.`
- `prefix_search(aaa$) = 103.`
- `prefix_search($) = 202.`
- `num_nodes(aaa) = 3.` (search stops at the red node)
- `num_keys(aaa) = 2.` (search stops at the red node)
- `num_nodes(aaa$) = 1.` (search stops at the blue leaf)
- `num_keys(aaa$) = 1.` (search stops at the blue leaf)
- `num_nodes(a) = 9.` (search stops at the green node)
- `num_keys(a) = 5.` (search stops at the green node)
- `num_nodes(o) = 11.` (search stops at the root)
- `num_keys(o) = 6.` (search stops at the root)

We also provide several sample inputs and outputs.