When people send messages on their phones, they sometimes modify word spelling by adding extra letters for emphasis. For example, if you want to emphasize hello you might write it “hellloooooooo”. Let's call the “ls” and the “os” word extensions. Regular text contains 2 or fewer of the same character in a row, while word extensions have 3 or more of the same character in a row. Given an input string representing one word, write a method that returns the start and end indices of all extensions in the word.

“Hellooo” -> [[2,4], [5,7]]

“hhhellloloo”

```
Typedef std::pair<int, int> Range;
O(length of str)
Using namespace std;
vector<Range> findWordExtensions(const std::string& str) {
   vector<Range> res;
   Int start = 0;
   While (start + 2 < str.length()) { // at least two char after the current
      Char c0 = str[start];
      Int nextPos = start + 1;
      for(; nextPos < str.length() && str[nextPos] == c0; ++nextPos) {
      }
      // either reaches the end of string or encounter a different char
     // hhhellloloo, start = 0 (index of first h), nextPos = 3 (index of e)
      Int length = nextPos - start;  // 3 - 0 = 3
      If (length >= 3) {
        res.emplace_back(start, nextPos - 1);
      }
      Start = nextPos;
   }
   Return res;
}
```

Now we want to spell-check extended words. You are given a dictionary of words. Implement method isExtendedDictionaryWord that will return:

- true if it is a dictionary word.
- true if you collapse the extensions in the word and it is a dictionary word.
- false otherwise.

[‘hello’, ‘hi’]
‘Hellllooo’ -> ‘hello’

```
// “Helllooo” -> [[2,4], [5,7]]
//  1or2,
// number of combination   2^( length of str / 3)
// first O(1)
//
Struct Trie {
  bool isTerminal;
  Trie* children[26];
}

Bool isExtendedDictionaryWord(const string& str, Trie* root) {
      Return helper(0, str, root);
}

Bool helper(int start, const string& str, Trie* root) {
    If (start == str.length()) {
         Return root != nullptr &&  root->isTerminal;
    } else if (root == nullptr) {
        Return false;
    }
     Char c = str[start];
    // extension case
    If (start + 2 < str.length() && str[start + 1] == c0 && str[start + 2] == c0) {
        Int nextPos = start + 3;
        For (; nextPos < str.length() && str[nextPos] == c0; ++nextPos) {
        }
        // try one or two
        Return
            (helper(nextPos, str, root->children[c - ‘a’]) // one occurence
       ||  (helper(nextPos - 1, str, root->children[c - ‘a’]) // two occurence

    } else {
       Return helper(start + 1, str, root->children[c - ‘a’]);
    }
}
```