**For Trie’s find() method:**

1. If there is no child in root, return an empty vector
2. If the first character in *key* matches with any label in root’s children,
   1. use helper function findMatch to find any path that would match with *key*
   2. and store the corresponding value in *result*, return *result*
3. If no label in root’s pointers matches with the first character in *key*,
   1. do not call the helper function and return an empty vector

**For find()’s helper function void findMatch:**

1. Base case: if the current node’s has no child, return;
2. If key’s length is 0 (have examined through all characters in key)
   1. Push back current node’s value into *result*, return
3. For every child in current node:
   1. If its pointer’s label is same with *key*,
      1. recursively call findMatch to examine the next character in *key*
      2. and set the current node to this pointer
   2. Else if *exactMatchOnly* is false (allows Snips)
      1. recursively call findMatch to examine the next character in *key*
      2. and set the current node to this pointer
      3. and set *exactMatchOnly* to false

(does not allow any mismatching character after this mismatch)

**For GenomeMatcher’s findGenomesWithThisDna() method:**

1. Clear all elements in *matches*
2. If *minimumLength* is less than the minimum search length passed to constructor,

return false

1. If cannot find any matching prefix of minimum search length in this class’s genome,

return false

1. Store the value of all the matching prefix in a vector *prefix*

//The value is in the form of pair<int, int>,

//where the first int is the prefix’s position in this class’s genome vector

//and the second int is the start position of this prefix segment

1. Create a vector called *result* with this class’s genome vector’s size and fill it with DNAMatch objects with length 0 and position -1

//to store the greatest DNAMatch for the corresponding gene in genome vector

1. For every *prefix* stored by the find() function:
   1. For every character in the string *fragment*:
      1. Starting from the *prefix’s* start position stored in its value (the *second parameter* in pair),

Extract the one gene character from the corresponding genome in this class’s genome’s vector, (use the *first parameter* in pair)

* + 1. Compare this character with the corresponding character in *fragment*:
       1. If equal, increase the matching *length* by 1
       2. If not equal, if it is not an *exact* match, ignore this mismatch and set bool *exact* to false (no other mismatch after this)
       3. Else, (the characters do not match) break this loop
  1. If the matching *length* is greater than *minimumLength*,
     1. If this one’s matching *length* is greater than *the stored one’s* for this gene in result,

Or if the matching *length* is the same and this one’s matching *position* is less than *the stored one’s,*

Set the corresponding DNAMatch object in *result* with this one’s name, position and length

* 1. Set *length* back to 0 and *exact* back to *exactMatchOnly* to examine the next prefix

1. Push back all values in *result* that has a length greater than 0 (being inserted by previous loop) to *matches*
2. If *matches* is empty (no value found), return false
3. Else, return true