FIND ALL ANAGRAMS OF A GIVEN WORD

FIND ALL ANAGRAMS OF A GIVEN WORD

ANAGRAMS OF A WORD HAVE ALL THE SAME LETTERS OF THE WORD ITSELF IN A DIFFERENT ORDER

CONSIDER THAT THE LETTERS IN THE WORD PASSED IN ARE UNIQUE AND THAT THERE ARE NO SPACES I.E IT'S A WORD NOT A SENTENCE

THE ANAGRAM NEED NOT BE A VALID WORD, WE JUST WANT ALL THE PERMUTATIONS OF THE LETTERS

IF ALL LETTERS ARE UNIQUE THEN THERE ARE N! ANAGRAMS

ANAGRAMS

FOR THE WORD "ROAD"

ANAGRAMS ARE:

"ROAD" "ORAD" "OARD" "RDOA"

"RAOD" "AROD" "AORD" "DAOR" "OADR"

"ARDO" "AODR" "ADRO" "ADOR" "DROA"

"ODAR" "RODA" "ODRA" "RDAO" "DORA"

"RADO"

"ORDA"

"DARO" "DOAR" "DRAO"

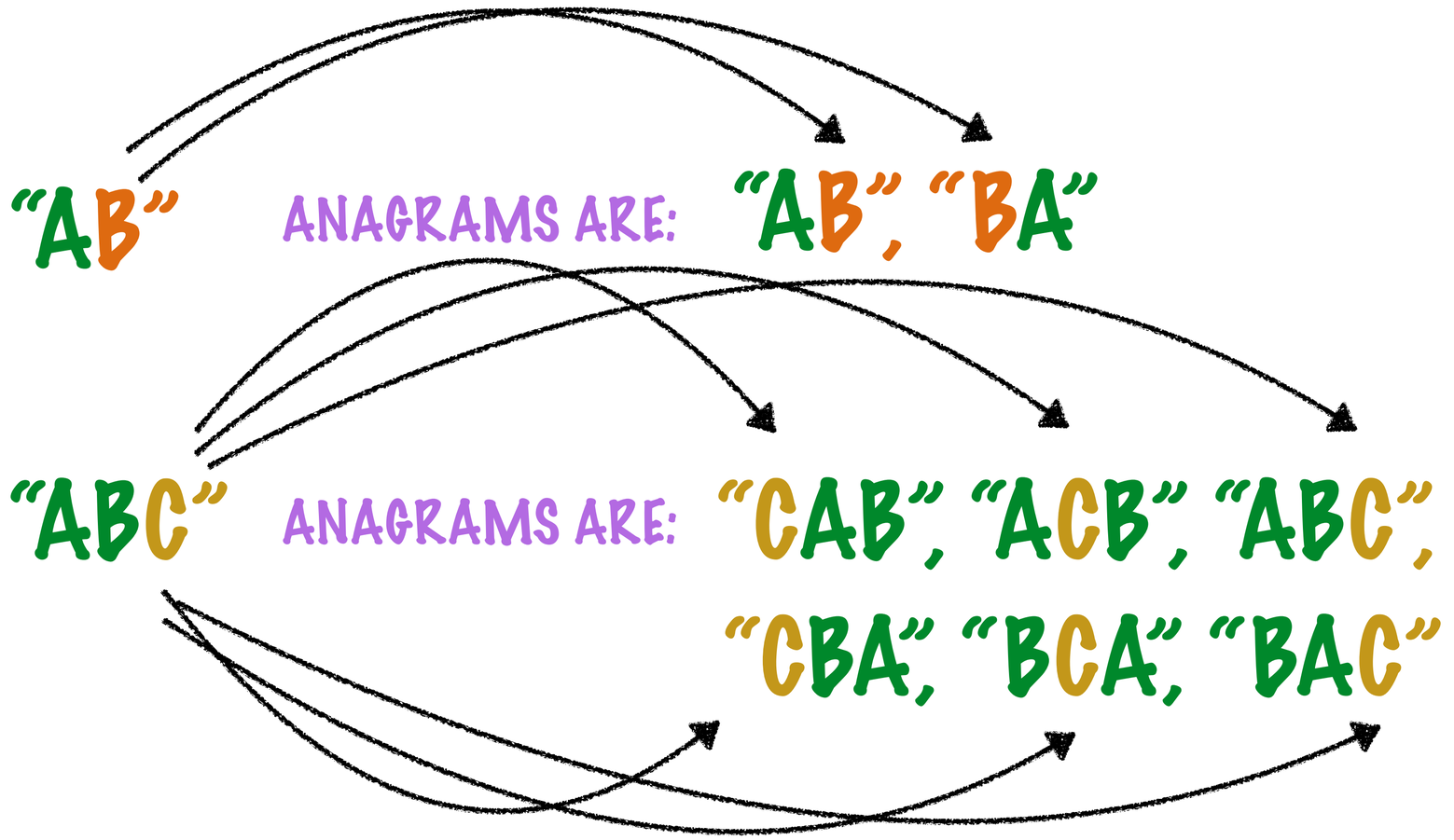
THERE ARE 4! ANAGRAMS OF THIS SET, IF WE HAVE A SET OF N ELEMENTS THE NUMBER OF ANAGRAMS WILL BE N!

LET'S BREAK THIS DOWN ...

CONSIDER A WORD OF 1 LETTER "A" ANAGRAMS ARE: "A"

CONSIDER A WORD OF 2 LETTERS "AB" ANAGRAMS ARE: "AB", "BA"

CONSIDER A WORD OF 3 LETTERS "ABC" ANAGRAMS ARE: "CAB", "ACB", "ABC", "CBA", "BCA", "BAC"



BREAK THE WORD INTO SMALLER AND SMALLER WORDS, TILL WE FIND THE ANAGRAM OF THE SMALLEST WORD

TO THE SMALLER ANAGRAMS ADD IN LETTER BY LETTER AT EVERY POSSIBLE POSITION

RECURSIVELY DO THIS TILL WE GET ALL THE LETTERS IN AT EVERY POSITION

WHAT IS THE BASE CASE?

1. THE ANAGRAM OF AN EMPTY STRING IS AN EMPTY STRING
2. THE SMALLEST WORD OF ONE LETTER IS THE ANAGRAM ITSELF

WHAT IS THE RECURSIVE CASE?

FIND ANAGRAMS OF SMALLER WORDS BY REMOVING LETTERS FROM THE ORIGINAL WORD

ADD THE LETTERS BACK ONE BY ONE TO EVERY POSITION

INSERT A LETTER AT EVERY INDEX - HELPER METHOD

```
private static void insertCharacterAtEveryIndex(List<String> potentialAnagramList,
                                                char currentChar,
                                                List<String> anagramList) {
    for (String potentialAnagram : potentialAnagramList) {
        // Inserts the current character at every position.
        for (int insertIndex = 0; insertIndex <= potentialAnagram.length(); insertIndex++) {
            StringBuilder sb = new StringBuilder(potentialAnagram);
            if (insertIndex < potentialAnagram.length()) {
                sb.insert(insertIndex, currentChar);
            } else {
                sb.append(currentChar);
            }

            anagramList.add(sb.toString());
        }
    }
}
```

HOLDS A LIST OF SMALLER WORDS WHICH WILL BE BUILT UP INTO ANAGRAMS

THE CURRENT LETTER TO BE INSERTED AT EVERY POSITION

STORE THE ANAGRAMS CREATED AFTER INSERTION

INSERT AT EVERY POSITION, INCLUDING THE VERY LAST ONE

ADD THE STRING WITH THE LETTER ADDED IN TO OUR LIST OF ANAGRAMS

FIND ANAGRAMS

```java
public static List<String> findAnagrams(String word) {
    if (word.length() == 1) {
        List<String> potentialAnagrams = new ArrayList<>();
        potentialAnagrams.add(word);

        return potentialAnagrams;
    }

    List<String> anagramList = new ArrayList<>();
    char currentChar = word.charAt(0);
    String subset = word.substring(1, word.length());

    List<String> potentialAnagramList = findAnagrams(subset);
    insertCharacterAtEveryIndex(potentialAnagramList, currentChar, anagramList);

    return anagramList;
}
```

STRING OF LENGTH 1 HAS A SIMPLE ANAGRAM WHICH IS THE STRING ITSELF, BASE CASE

REMOVE ONE CHARACTER AT A TIME FROM THE ORIGINAL WORD AND FIND THE ANAGRAM OF THE REST OF THE WORD

ONCE YOU GET THE POTENTIAL ANAGRAMS, INSERT THE CURRENT LETTER AT EVERY POSITION TO GET THE ACTUAL ANAGRAMS

AS THE STACK UNWINDS THE FINAL CALL WILL HAVE THE ANAGRAMS OF THE ORIGINAL WORD

THE NUMBER OF ANAGRAMS FOR A WORD WITH UNIQUE LETTERS IS N! WHICH MEANS THE COMPLEXITY OF THIS ALGORITHM IS ALSO O(N!)

THIS CANNOT BE FURTHER OPTIMIZED