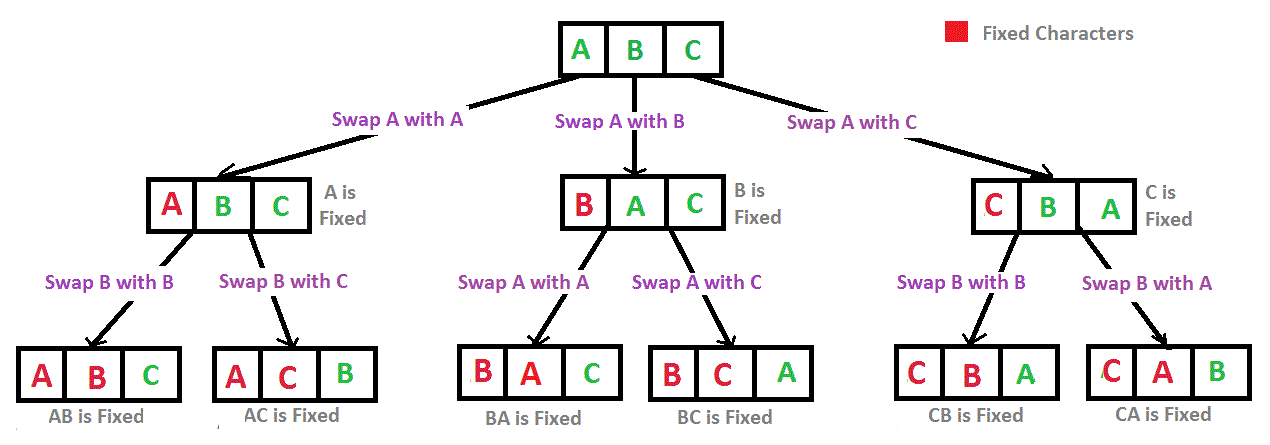
8.7: Write an algorithm to print all permutations of a string.



**High Level Algorithm:** You start off with the original string “ABC” and it is all green. There is a parameter called start index which specifies the first green element and all elements after it are green and all elements before it are red. If you look at the implementation of the permute function with only one parameter, you will see that the program initializes with your string “ABC” and zero for the start index indicating all elements are green.

To create the child nodes, you iterate around each green character in the parent string. For green each character, you swap the character with the first green element in the parent string pointed to by start index. Then you pass in the modified string with the start index increment in the recursive call.

The base is case (leaves in the tree) is when only the last character is green or in implementation terms (the start index is equal to the last index in the string). In this case print the leaf node. The program should print all the leaf nodes in the diagram above.

|  |
| --- |
| **Implementation:**  **public** **static** **void** permute(String string){  *permute*(string, 0);  }  **private** **static** **void** permute(String string, **int** startIndex){  **if** (startIndex == string.length() - 1){  System.*out*.println(string);  **return**;  }  **for** (**int** i = startIndex; i < string.length(); i++){  string = *swapCharacters*(string,startIndex,i);  *permute*(string, startIndex+1);  string = *swapCharacters*(string,startIndex,i);  }  }  **public** **static** String swapCharacters(String string, **int** i, **int** j){} |