**Project Specification**

**1) Problem to solve**

Given a keypad, where number 2 stands for letter **a** or **b** or **c**, and number 3 stands for **d** or **e** or **f** , etc, as defined on keypad of a old cell phone.



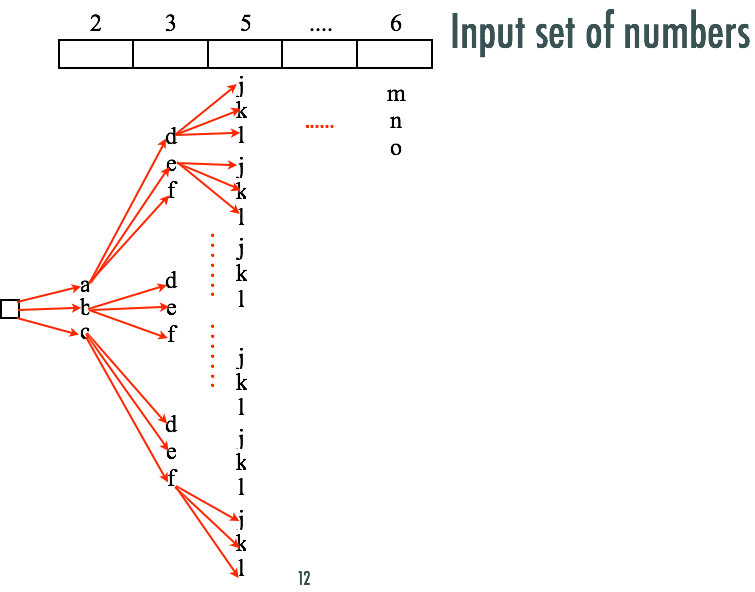
Given a standard dictionary **D**, after **D** is constructed using its constructor and an insert() method, **D** provides only **ONE** extra service(method) -- where you can check whether a string of letters comprise a valid English word or not, with signature specified **boolean isWord(String str).**

**Input:** A sequence **S** contains **N** **integer** numbers, such as **S**= {2, 3, 4} with **N** = 3, And the dictionary **D** and the keypad that shows the letters each integer stands for.

**Output**: all possible valid English words represented by the input sequence of digits **S**, that appear in the provided dictionary **D**.

**2) Specifics**

1. You have to construct a tree (called the solver tree) in the memory of your computer as we learned in class, as described below. You have to create the solver tree on the fly for each input sequence **S,** each time when you run your program. Each run of your program will handle a different input sequence **S**. For example, the tree below is built for input sequence S={2, 3, 5, …. 6}. So that the original problem will be transformed into another problem where you have to first find all paths in the solver tree below. Then you check the string associated with each path to see whether it appears in the dictionary **D** as a valid English word or not**.**



1. Given a list of English words in a text file (attached in the package and used for dictionary D), in the dictionary.txt, please **ignore** the integer number to the right of each word. You have to implement the dictionary **D** in two ways, in order to fast index a word in a dictionary, namely by using Hashing and Prefix Tree. In your implementation, it is required to have two different Java classes, **DictHashTable** and **DictPrefixTree**, corresponding to the two different dictionary implementations. You are allowed to use the built-in HashTable class in your code. You have to test your project against both dictionary implementations.
2. Pleas analyze the time complexity of your algorithm if users enter **N** digits. Specifically, the time complexity for constructing the solver tree, the time complexity for dictionary searching, and the time complexity for solving the entire problem of N digits of input.
3. You have to implement a user interface that allows users to enter some digits on the **standard input**, then your program outputs all possible English words on **standard output** for the input sequence. A GUI is **not** required.
4. Write a project report in PDF format. The report includes:

1) Your answers to question (c) above.

2) Please search online references regarding the concept of **Exhaustive Search**. What is Exhaustive Search? Which part of this project falls into Exhaustive Search?

3) Please search online references regarding the concept of **Branch and Bound**. What is Branch and bound?

4) If you will use a Prefix Tree to implement the dictionary, can you use the prefix tree property to accelerate your algorithm together with Branch and Bound? and How to do that? please illustrate with a specific example and a diagram if necessary.

5) Please include at least 5 different runs of your program with different input sequences. Please catch any errors so that your program will not crash.

6) At the end of the report, please list the name of the members in your team and indicate whether you have attempted the extra credit portion or not.

1. Please organize your source code, so that the users are able to compile all your source code on the command line using **javac \*.java** and Run your program with a command **java Tester**
2. **If you successfully implemented both the Exhaustive Search solution and the Branch and Bound solution, as we discussed in the classroom, (you compared the time cost of these two solutions and your two solutions produced correct answers) you will get 5% bonus on top of 100%. In this case, in order to implement the Branch-and-Bound solution, your dictionary D is allowed to have another method boolean hasWordPrefixed(String s), which checks whether or not there are English words exist in D that have a prefix of s. But please keep in mind, in the Exhaustive Search solution your dictionary provides only one method boolean isWord(String str), in addition to its constructors and insert().**

**It is noteworthy that it might exist better solution to this problem, for example, sending the sequence of N digits one digit after another to the prefix tree-based dictionary. However, here we purposely constrained the services that the dictionary can provide, in order to introduce the Exhaustive Search Concept and Brand-and-Bound Concept.**

1. **Please carefully follow the requirements specified above. If you do not follow the guidelines, you will lose a lot of points. Please finish this project independently within your team.**

**3) To turn in**: Please wrap up all your source code and the project report into a single zip file, and turn in the single zip on Canvas, under the Assignment section. Please name your zip file using the convention we used before.