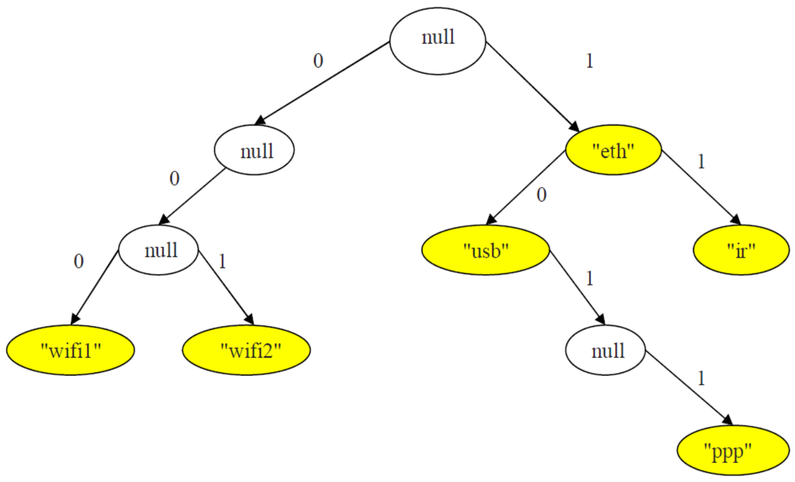
Top-20 Training Program
(Trie & Bloom Filter)

Apply the Trie and BloomFilter data structures discussed in class to solve the following problems.

Problem1: Routing Table Implementation

The figure bellow illustrates a trie that stores the bit string keys1: 1, 11, 10, 1011, 000, 001. In this binary tree a move to the left denotes a 0 and a move to the right denotes a 1. Thus, when searching for a key $a = a_0a_1a_2 \ldots a_n$, we go left at a node of depth i if $a_I = 0$ and right if $a_I = 1$. Not all the nodes in the tree represent key-value pairs. Some nodes represent only unique prefixes common to several keys ("0", "00" and "101") and are essential for establishing a path to other nodes. In the figure below, only the yellow nodes represent key-value pairs. One should note that there is no need to actually store the keys in the nodes, since they can be inferred by traversing the path from the root to the node (but there is a need to define that a node represents a key).



Your assignment is to design a class that implements a routing table using a trie. Specifically, you have to implement following methods:
void add(int address, int n, String value) – adds a key-value pair to the table. The key is a 0-1 string of up to 32 bits, specified using an integer *address* and the key consists of the *n* high-order bits of *address*; the value is a string.
String route(int address) – returns the value associated with the longest prefix in the routing table that matches the argument. If no prefix matches the given address, an exception is thrown.

Example:

```
t = new RoutingTable();
t.add(10000000....b,1,"eth");
t.add(11000000....b,2,"ir");
t.add(10000000....b,2,"usb");
t.add(10110000....b,4,"ppp");
t.add(00000000....b,3,"wifi1");
t.add(00100000....b,3,"wifi2");
```

On this object, the query **`t.route(01000000....b)`** should throw an exception, and the query **`t.route(10101111....b)`** should return **`"usb"`**. Assume that each input argument to add and route methods is 32-bit integer and shown here in binary form for better understanding and while storing the key in trie, you have to store the first n bits of that integer only.

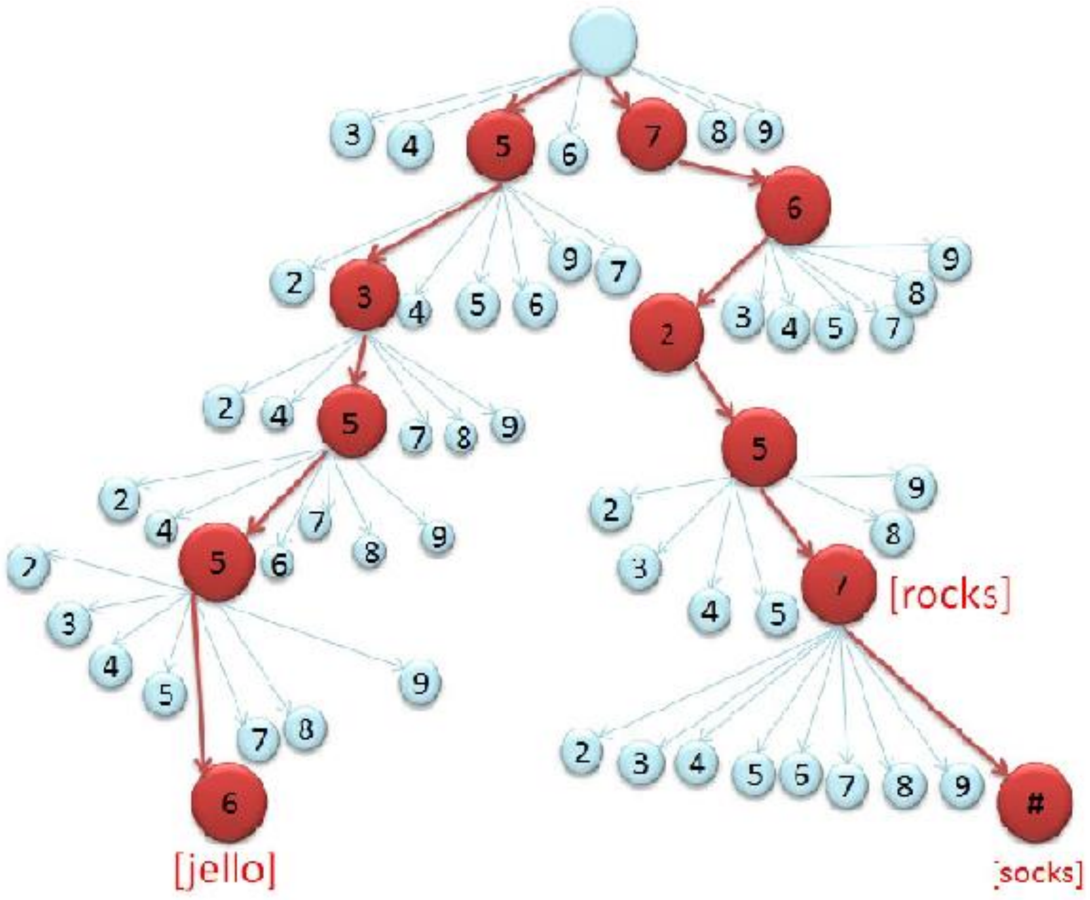
Problem2: T9 Predictive Text Implementation

In this problem, you have to implement T9 predictive text, a text input mode available on cellphones. On a cellphone, each number from 2-9 on the keypad represent three or four letters, the number 0 represents a space, and 1 represents a set of symbols such as { , . ! ? } etc. The numbers from 2-9 represent letters as follows:

2	ABC
3	DEF
4	GHI
5	JKL
6	MNO
7	PQRS
8	TUV
9	WXYZ

Since multiple letters map to a single number, many key sequences represent multiple words. For example, the input 2665 represents "book" and "cool", among other possibilities.

Write a class T9Prediction class with the constructor doing the following functionality: It should read in a dictionary file that contains a list of words and translate each word in the dictionary into its numeric key sequence, then add the key sequence to your trie, with the word at the end of the path corresponding to the digits. If a word with the same numeric sequence already exists in the trie, add the new word as part of linked list at that node. You can download the dictionary file named "dictionary.txt" from algorithmica github repository. If the program reads the set of words "jello","rocks", and "socks" from the dictionary and adds it to an empty trie, the resulting trie should look like this:



Implement a method List<String> getAllWords(String number) which should return all the words corresponding to the sequence of numbers you passed as input. The sequences below can be used to validate your program:
22737: acres, bards, barer,bares,barfs,baser,bases,caper,capes,cards,carer,cares,cases
46637: goner,goods,goofs,homer,homes,honer,hones,hoods,hoofs,inner
2273: acre, bard,bare,barf,base,cape,card,care,case
729: paw,pax,pay,raw,rax,ray,saw,sax,say
76737: popes,pores,poser,poses,roper,roses,sords,sorer,sores

Top-20 Training Program (Trie & Bloom Filter)

Problem3: URL Filter

Your URL filter will take in URL(s) as input parameters and determine whether they match a blacklist of bad URLs. Your web filter should implement the URLFilter interface that we provide. You should assume that the input source for blacklists is a newline-separated file of URLs.

- **void clearFilter():** empty the web filter blacklist
- **void addBlacklist(<blacklist file>):** add the URLs from the file specified to the URL filter.
- **void filter(<input urls>, <filtered_urls>):** read URLs from the input urls file. For each URL that is in the web filter, add it to a newline-separated output file specified by filtered urls.
- **void perf(<input urls>,<n>):** read up to n URLs from the input urls file. Determine whether each URL is in the filter and report how many passed the filter, along with the total time taken in milliseconds. If n is larger than the number of URLs in the input, the existing URLs are reused repeatedly until n total URLs have been tested.

Problem4: Book Index

Write a program to determine, given a large text file, all of the line numbers (zero-based) on which each word appears. Specifically, write a class Index, the constructor of which builds a trie associating each word with a Queue of the line numbers (0-based) on which it appears. The get() method returns the Queue associated with a particular word (or null if that word never appears).

Problem5: Infix Search

We've discussed about using TRIE for prefix queries in class. Find an efficient strategy to allow infix based search suggestions over list of words given in the file.