Homework 10

Due: Friday, December 8th, 2017 23:59
You can work in your assigned teams.

As discussed in class, a TCAM can improve significantly the performance of a search. However, semiconductor manufacturers are investing in both algorithmic and CAM-based solutions. In this assignment, we will become familiar with a data structure that improves lookups (searches) and can be stored in a SRAM.

For our purposes, a *prefix* is a set of binary sequences that all start with the same pattern. For example, `010*` is a prefix denoting all binary sequences whose first digit is zero, the second is one and the third is one. Sequences `01010011` and `01001110` can both be described through prefix `010*`. However, only the first sequence can be described by prefix `0101*`. We say that prefix `0101*` is the longest match for sequence `01010011`.

In this assignment, we are given a list of prefixes of variable lengths and asked to construct a data structure, which will be described later, so that given a binary sequence we can efficiently find the longest prefix match.

Lulea-Compressed Tries

A *unibit trie* (commonly pronounced like "try") is a special kind of a tree, which is particularly designed for efficient search (re**trie**val). Each node in the tree is an array containing a 0-pointer and a 1-pointer. At the root, all prefixes that start with 0 are stored in the subtrie pointed to by the 0-pointer and all prefixes that start with 1 are stored in the subtrie pointed by the 1-pointer. The trie is constructed by recursively applying this rule. For more details, refer to online resources such as:
https://raminaji.wordpress.com/unibit-tries/

For a 32-bit binary sequence, a unibit trie may have to make 32 node accesses. This observation motivates the use of a *multibit trie*. If we want to search a trie in *strides* of, say, 6 bits, the main problem is dealing with prefixes like `11001*` whose lengths are not a multiple of the selected stride. The solution is to expand such prefixes (`110010*` and `110011*`):
https://raminaji.wordpress.com/multibit-tries/

A *Lulea-compressed trie* is a multibit trie with fixed-stride tie nodes that uses bitmap compression to reduce storage significantly.
https://raminaji.wordpress.com/lulea-tries/

In this assignment, you are to implement a Lulea trie in the C programming language. Your code should read a file that contains a list of prefixes, one per line. You can assume that every prefix will end with a star character (*). Given this list of prefixes, the code should build a Lulea trie.

You do not have to handle insertions of new prefixes (or deletions of current prefixes) after the original trie has been built. After reading the input file, your code should print the trie, breadth first.

Your code should include a `lookup` function that can be provided with a string and will return the longest prefix match for that string.

Submit via *turnin* two directories, `code` and `tests`, that include your implementation and your testcases.