Trie, trie again

Let’s revisit the idea of tries, since I’m sure a lot of you were preoccupied by the midterm when we went over them on Thursday.

The basic idea of a trie is that for some purposes, like spell checking and boggle, it often makes sense to see if a string is the prefix of some word in a dictionary.

To do this we keep a tree that contains letters representing prefixes. This is a bit hard to describe, so we use this diagram instead.

This trie contains the words “be”, “bed”, and the start of “baccalaureate”. Image credit: Frank Pfenning.

The boolean flags in the trie represent ends of words. (It’s useful to know when we’ve reached the end of a word so we can know that, for instance, “b” and “bacca” are not words.)

This approach is really great, since it lets us look up whether a word of length $k$ is in the trie in $O(k)$ time.

However, it has some major problems. In particular, we’ll be wasting a lot of space, since we have an array of 26 elements at each level, of which we’ll only need a few the majority of the time.

We don’t have unlimited memory, so this is bad.

So, we’ll use a slightly more complicated but far smaller structure, a ternary search trie.

In a ternary search trie node, we’ll store a character $c$ and three pointers, left, right, and middle. Similarly to a binary search tree, the left subtree stores words starting with characters alphabetically less than $c$ and the right subtree stores words starting with characters alphabetically greater than $c$. The middle subtree stores a subtrie with all words that start with $c$, starting from their next character (there’s no reason to store $c$ twice).
Here’s another image, shamelessly stolen lovingly borrowed from Professor Pfenning’s notes. It contains the same words as before (with one extra ‘c’ in “baccalaureate”), and has dots (actually ‘.’) instead of boolean values to indicate the end of strings.

**Trie interface?!?!**

This is the interface of tries as we defined them in class.

tagdef struct trie_header *trie;

trie trie_new(void (*elem_free)(void *e));
void *trie_lookup(trie TR, char *s);
void trie_insert(trie TR, char *s, void *e);
void trie_free(trie TR);

Recall our goal of being able to see if something was a prefix of a word.
This does not do that.
This is … problematic.
Let’s spend some time talking about how we can fix this interface problem.
We’ll discuss it more in lecture tomorrow, but coming up with interfaces is a good skill to have, so we’ll brainstorm some.