RYAN FILGAS
CS 163
KARLA FANT
03-03-2021

Program 4 Efficiency Writeup

1. How well did the data structure selected perform for the assigned application?

   A binary tree worked well for this application; however it would have been useful to randomize the data order (maybe in a specific way, not random, because we want the tree to be balanced). I wrote a function to monitor chain length, and as I modified my external data file, I noticed that the way I entered the information, some of it was in, or close to alphabetical order, causing a lopsided tree in some cases. One example of this was a tree of size 10 with a height of 5. If evenly distributed, my tree height should have been 3. The algorithm was efficient in some ways, and not so in others. If for some reason I wanted to find matches of some sort for data other than a name, this is a worst case scenario of searching every single node. In this case it would be beneficial to supplement the tree with additional data structures.

2. Would a different data structure work better? If so, which one and why...
   I don't know if using a different data structure on its own is necessarily helpful, but adding additional structures to point to the same data would be. Implementing a circular linked list to hold all homes with the same type of animal could be useful. Progress the rear pointer every time the list is displayed, and keep the alphabetical part unordered, so that no home is prioritized over another. It could serve a seemingly random home to show the user, or all homes with a match. The program wouldn't have to traverse every node, because every node is the desired data. In this case, you could also use a hash table with a randomly changing key. If you're going through the trouble of displaying random data, it might as well be fast.

3. What was efficient about your design and use of the data structure?
   The data structure itself is to 'blame' for most of the efficiency, however a few small things go a long way. There are the usual benefits of not dragging previous pointers, using dynamically allocated memory etc. For my program specifically, I was faced with the decision of how to implement multiple animal types inside of a struct, when I didn't know how many there were going to be. To address this, I used a dynamically allocated array of animal type structs, each containing a dynamically allocated char array to hold animal type. This saves a lot of memory. The downside to using a struct to package that data is the extra '.' Operation, however in this case it's a small price to pay for easily understood and maintainable code. I imagine there are some data sets where a char ** would be preferable, and in fact, our hash tables use a similar structure, a node **. I suspect this efficiency is one of the reasons for this structure, as the purpose of a hash

table is brutal efficiency. The other place where there was a clear choice on how to access the data, was searching for name matches after the information was stored. Searching for information in the same way it was stored is very important to the efficiency of a binary tree. Otherwise we would be searching every node for the very data we organized according to. Worst case scenario!

4. What was not efficient?
   Searching for animal type matches, or any information other than name wasn't efficient. It's very important to know that the primary search method used to organize is the same as the search method that will be most used to retrieve data. Otherwise you might as well keep extra data structures to organize it in whatever way is needed. Use pointers to the same data!

5. What would you do differently if you had more time to solve the problem?

   If I had more time, I would implement a function to monitor where in the tree each piece of data is, not just the height. A good function to implement would tell me how many recursive calls it took to retrieve a particular piece of data, and store that number in the list, so I could compile that data into an average traversal / stack size, that if it went over a certain number, I would have a function to rebalance the tree, or inform the way data is being added to the tree, and what order adjustments would be best for new input.