Practice design decision problems

For each of the following scenarios:

- What data structures would you use in the following scenarios?
- Why? (Note that *how* you would use it does not answer *why*)
- If there is no clear answer (or for practice, even if there is), what questions would you ask to inform your design decisions?

All the practice problems are on one page to save paper/space. You may want to use more pages if you want to practice writing solutions.

1. Keeping track of someone's actions in a program to let them undo/redo.

2. Store the genealogy of different species of animals, and quickly find which animals are most genetically similar to another animal.

3. Store friendship information on a social networking site, in which you can find someone's friends, or friends of friend's.

4. Keep track of possible moves in a game of chess to know what to play next.

5. Find the minimum length of extension cords to connect twinkle lights around the house with one power outlet and you know where you want to hang the lights. Note that you can connect the lights to each other to power them.

6. Store an interactive table of contents for a book that automatically updates when you write new chapters, sub chapters, and sub sub-chapters, and you can click on part of the table of contents to immediately get to the right part of the book.

7. Manage threads for concurrency, where some threads are more critical than others and should access resources first no matter what order the threads started.

SPOILER ALERT:
Answers are on the next page!

Suggested Answers

1. Keeping track of someone's actions in a program to let them undo/redo.
   Stack (either linked list or array, because it can undo/redo with pop/push *in O(1)* time)
   (Linked list version for speed, array for space)

2. Store the genealogy of different species of animals, and quickly find which animals are most genetically similar to another animal.
   Tree (does not need to be a BST. If you have a reference to the animal in question, local traversals in the subtree with that animal can quickly find similar animals in O(n) where n is the number of similar animals. Also stores animals in a genealogically logical order, which makes the code simpler to understand [also important in decisions!])

   (If you want to get fancy and fast look-up for where that animal is in the tree and space is not a constraint, you can additionally store a HashTable with animal names as keys and references to their respective nodes in the tree as values to look up that initial animal in O(1))

3. Store friendship information on a social networking site, in which you can find someone's friends, or friends of friend's.
   Graph using adjacency list
   (Space: assuming people on the site don't know most of the other people, it's a sparse graph so adjacency matrices waste a lot of space)
   (Efficiency: adjacency lists are faster for finding neighbors (i.e. friends) -- especially for sparse graphs!)

4. Keep track of possible moves in a game of chess to know what to play next.
   Tree

5. Find the minimum length of extension cords to connect twinkle lights around the house with one power outlet and you know where you want to hang the lights. Note that you can connect the lights to each other to power them.
   Graph for MST

6. Store an interactive table of contents for a book that automatically updates when you write new chapters, sub chapters, and sub sub-chapters, and you can click on part of the table of contents to immediately get to the right part of the book.
   Search Tree (can represent hierarchy and have fast search. Need not be binary)

7. Manage threads for concurrency, where some threads are more critical than others and should access resources first no matter what order the threads started.
   Heap