Recursion Problems

Warm-Ups

1. Write a recursive implementation of the factorial function. Recall that $n! = 1 \times 2 \times \ldots \times n$, with the special case that $0! = 1$.
2. Write a recursive function that, given a number n, returns the sum of the digits of the number n.
3. Write a recursive function that, given a string s, prints the characters of s in reverse order.
4. Write a recursive function that checks whether a string is a palindrome (a ***palindrome*** is a string that's the same when reads forwards and backwards.)
5. Write a recursive function that, given a pointer to the root of a binary search tree, prints out the elements in that tree in sorted order.
6. Write a recursive function that, given two strings, returns whether the first string is a subsequence of the second. For example, given `hac` and `cathartic`, you should return true, but given `bat` and `table`, you should return false.

Enumeration

7. Given a list of n distinct elements, write a function that lists all subsets of those elements.
8. Given a list of n distinct elements, write a function that lists all permutations of that list.
9. Given a list of n distinct elements and a number k, write a function that lists all k-element subsets of that list. Make sure not to output the same subset multiple times.
10. Given a list of n distinct elements and a number k, write a function that lists all k-element permutations of that list.
11. Given a list of n distinct elements and a number k, write a function that lists all possible ways of assigning those elements into k (possibly empty) groups.
12. Given a nonnegative integer n, write a function that lists all strings formed from exactly n pairs of balanced parentheses. For example, given $n = 3$, you'd list these five strings:

    `((()))` `(()())` `(())()` `()(())` `()()()`

    As a hint, given any string of $n \geq 1$ balanced parentheses, focus on the first open parentheses and the close parentheses it matches. What can you say about the string inside those parentheses? What about the string that follows those parentheses?
13. Given a number n, generate all n-character passwords, subject to the restriction that every password must have a number, a lower-case letter, an upper-case letter, and a symbol.
14. Given a number n, generate all distinct ways to write n as the sum of positive integers. For example, with $n = 4$, the options are 4, 3 + 1, 2 + 2, 2 + 1 + 1, and 1 + 1 + 1 + 1.

Backtracking

15. Given a list of n integers and a number U, write a function that returns whether there is a subset of those n integers that adds up to exactly U. *(This is a famous problem called the "subset sum problem," by the way.)*

16. A ***shrinkable word*** is a word that can be reduced down to the empty string by deleting one letter at a time such that, at each stage, the remaining string is a word. For example, the word "startling" is shirnkable because of this sequence of words:

    startling → starting → staring → string → sting → sing → sin → in → i → *(empty)*

    Write a function that accepts as input a string and a set of all the words in English, then reports whether the input word is shrinkable.

17. Some words can be spelled out using only element symbols from the periodic table. For example, the world "canine" can be spelled as `CaNiNe` (calcium, nickel, neon). Some words, like "element," cannot. Write a function that accepts as input a string and a set of all the element symbols in the periodic table, then returns whether the word can or cannot be spelled using element symbols. Even better, if the string can be spelled this way, return the string with the capitalization changed to reflect that spelling.

18. Suppose you have n doctors, each of which are free for a certain number of hours per day, and m patients, each of whom needs to be seen for a certain number of hours. Write a function that determines whether it's possible for all the patients to be scheduled so that none of the doctors spends more time than they have available. Better yet, tell us which people should see which doctors.

19. You have a collection of cell phone towers, each represented by an (x, y) coordinate in the plane. Each cell phone tower needs to be assigned a transmitting frequency chosen from a list of k different frequencies. The only requirement is that no two towers within distance d of one another can be assigned the same frequency, since that would cause interference. Determine whether it's possible to assign the cell towers the frequencies so that no two towers overlap. If you can, tell us what frequencies get assigned to which towers.

20. You are given a grid of nonnegative integers. You begin in the upper-left corner, and your goal is to get to the lower-right corner. However, at each point in time, you can only move up, down, left, or right exactly the number of squares given by the number you're standing on. For example, if you were standing on a 3, you could move exactly three spots left, exactly three spots right, exactly three spots up, or exactly three spots down. You can't walk off the board. Determine if there's a path that gets you from the upper-left corner of the grid to the lower-right corner.

Trees

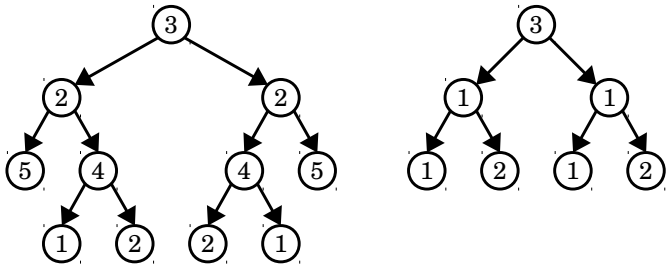
21. Write a function that takes as input a pointer to the root of a binary tree, then returns whether it's a valid binary search tree.

22. Write a function that takes in a pointer to the root of a binary search tree, then deallocates all the nodes in that binary search tree.

23. In a binary tree, a *common value subtree* is a complete subtree (that is, either the empty tree or a node and all its descendants) where every node has the same value. Given a binary tree, return a pointer to the largest common value subtree in that tree.

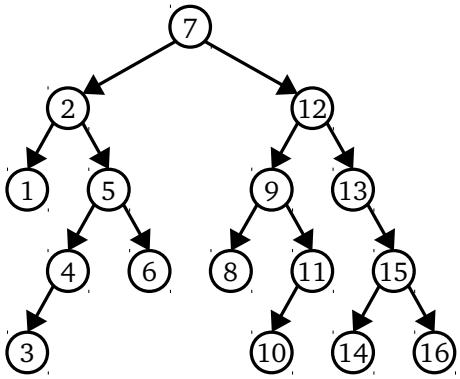
24. Write a function that, given a sorted array, builds a balanced binary search tree with the same values.

25. A *palindromic tree* is a tree that is the same when it's mirrored around the root. For example, the left tree below is a palindromic tree and the right tree below is not:



Given a binary tree, determine whether it is a palindromic tree. Then try generalizing this to arbitrary multiway trees.

26. In a binary search tree, a node u is called an *ancestor* of a node v if the path from v back up to the root of the tree passes through u. For example, consider this BST:



Here, the ancestors of 6 are 6, 5, 2, and 7; the ancestors of 10 are 10, 11, 9, 12, and 7; and the only ancestor of 7 is 7 itself. (Note that every node is always an ancestor of itself).

A *common ancestor* of two nodes v_1 and v_2 is a node u such that u is an ancestor of both v_1 and v_2. For example, in the above BST, 3 and 6 have 5 as a common ancestor, 9 and 16 have 7 as a common ancestor, and 2 and 15 have 7 as a common ancestor. The common ancestor of two nodes might be one of those nodes: for example, 12 is a common ancestor of 8 and 12.

Finally, the *lowest common ancestor* of two nodes v_1 and v_2 is the node u such that u is a common ancestor of v_1 and v_2 that is as deep as possible. For example, the common ancestors of 14 and 16 include 7, 12, 13, and 15, of which 15 is their lowest common ancestor. The lowest common ancestor of 1 and 6 is 2, and the lowest common ancestor of 10 and 16 is 12. The lowest common ancestor of two nodes might be one of those nodes: the lowest common ancestor of 13 and 16 is 13, for example.

Write a function that takes in a binary search tree and two values known to be present in that binary search tree, then returns their lowest common ancestor.

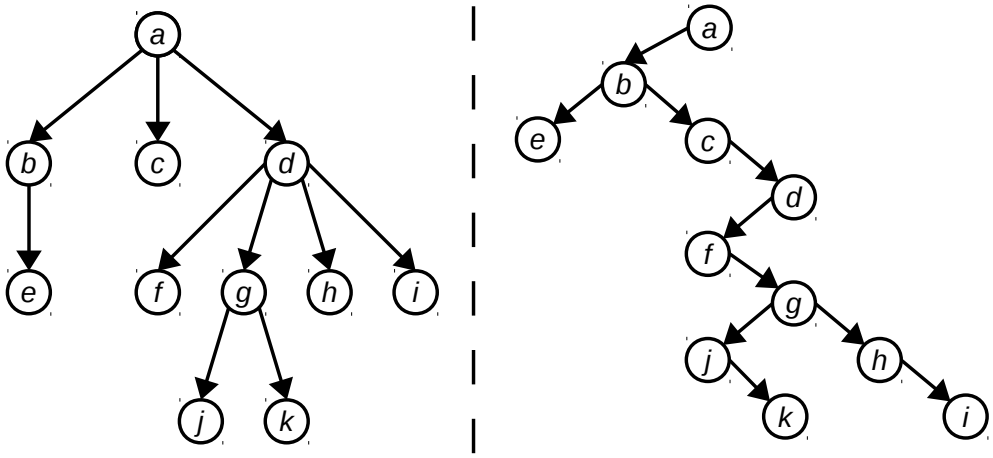
27. Suppose you have a multiway tree where each node has an associated integer value. Find a set of nodes with the maximum possible sum, subject to the constraint that you cannot choose a node and any of its children at the same time.

28. *(The Great Tree-List Recursion Problem, by Nick Parlante)* A cell in a circularly, doubly-linked list essentially looks the same as a node in a binary search tree – each stores a value and two pointers. The only difference is that in a doubly-linked list the pointers stand for "previous" and "next," while in a binary search tree the pointers stand for "left subtree" and "right subtree." Write a function that, given a pointer to the root of a binary search tree, converts that binary search tree into a sorted, doubly-linked list by rewiring the pointers so that "left" and "right" now stand for "previous" and "next."

29. Write a function that takes as input a pointer to the root of a binary tree, then deletes all the leaves from that binary tree.

30. Write a function that takes as input a pointer to the root of a binary tree, then returns whether that binary tree is a binary max-heap (that is, whether it's a complete binary tree and whether each element is greater than its children).

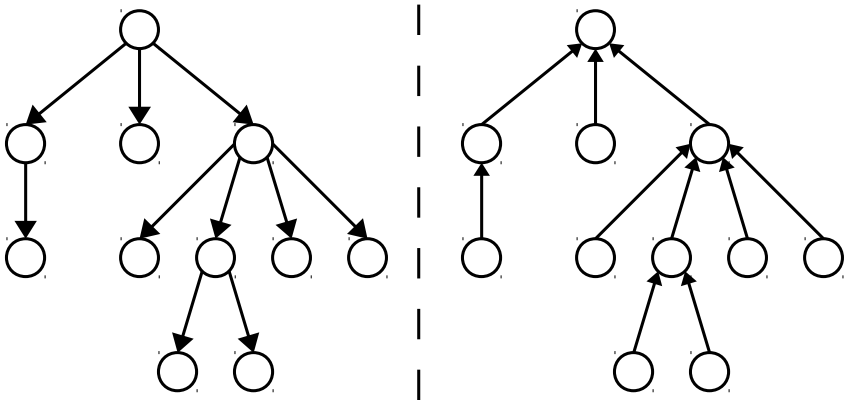
31. A typical multiway tree is represented by having each node store pointers to each of its children. There's an alternative representation of a multiway tree as a binary tree called the ***left-child, right-sibling*** (LCRS) representation. In this representation, every node stores two pointers: a left pointer pointing to its first (leftmost) child, and a right pointer pointing to its next sibling. For example, here is a multiway tree and its LCRS representation:



For example, focus on the node *b*. Its left pointer points to *e*, its first child. Its right pointer points to *c*, its next sibling (the next node that's a child of the same parent). Node *c* has no children, so its left pointer is null. Its right pointer points to its next sibling, *d*, whose right pointer points to null because it has no more siblings. Notice that node *e* has a null right pointer – even though in the multiway tree the node *f* is to its right, *f* is not a sibling of node *e*.

Write a pair of functions, one that converts a multiway tree into its LCRS representation, and one that converts an LCRS representation into a multiway tree.

32. Another way of representing multiway trees is as a ***spaghetti stack***, where instead of nodes storing multiple pointers, one per child, the nodes store a single pointer that points back up to the parent. For example, below is a multiway tree and its representation as a spaghetti stack:



When working with a spaghetti stack, instead of storing a single pointer to the root, you store a list of pointers, one for each leaf. Write three functions: one that converts a multiway tree to a spaghetti stack, one that converts a spaghetti stack into a multiway tree, and one that deallocates all the nodes in a spaghetti stack.

Tries

33. Write a function that, given a pointer to the root of a trie, prints out all words in that trie.

34. Write a function that, given pointers to the roots of two tries, returns a pointer to the root of a new trie that contains all the words present in both tries.

35. Solve the previous problem, but do so in a way that doesn't allocate any new memory and instead destructively modifies the two input tries to combine them together into a single trie.

36. Write a function that, given a pattern string consisting of a series of letters intermixed with question marks (for example, a string like `do?ble`) and a pointer to the root of a trie, returns all words in the trie that match the pattern, where a question mark matches any single character. If given the pattern `do?ble`, for example, and a trie of all words in English, your code should return `doable` and `double`. Given the pattern `??`, it would return a list of all two-letter words in English.

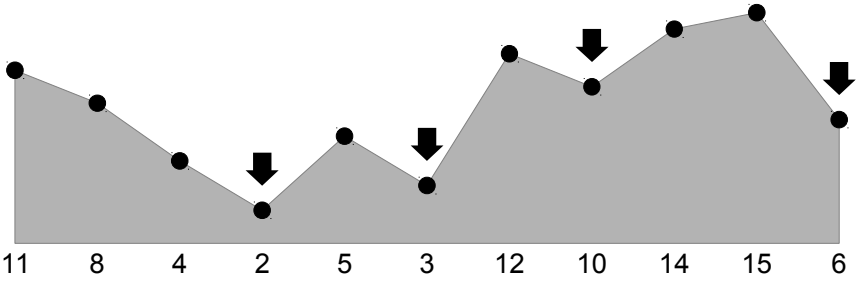
37. A ***Patricia trie*** (sometimes called a ***radix trie***) is a variation on a trie. We'll say that a trie node is a "silly node" if it doesn't represent a word and has exactly one child. A Patricia trie is a trie formed by removing all the silly nodes from a trie and allowing edges to be labeled with arbitrary strings, not just individual characters. Write a function that, given a trie, converts it to a Patricia trie, and a reverse function that converts a Patricia trie to a regular trie.

General Recursion

38. A ***rotated array*** is an array formed by taking a sorted array, splitting it into two pieces, then interchanging the order of those pieces. For example, the array [6, 7, 9, 0, 1, 2, 3, 4] is a rotated array, as is [0, 1, 2, 3] (to see why it's rotated, split the array [0, 1, 2, 3] into the pieces [] and [0, 1, 2, 3], then interchange their order). Design an efficient algorithm that, given a rotated array and a key k, determines whether k is present in the array.

39. Suppose that you are interested in setting up a collection point to funnel rainwater into a town's water supply. The town is next to a ridge, which for simplicity we will assume is represented as a 1D array of the elevations of different points along the ridge.

    When rain falls on the ridge, it will roll downhill along the ridge. We'll call a point where water naturally accumulates (that is, a point lower than all neighboring points) a "good collection point." For example, here is a possible ridge and its good collection points:



    You are given an array of n elements representing the altitudes of points along the ridge, all of which are distinct. Design an efficient algorithm for finding a good collection point.

40. The Fibonacci strings are a series of recursively-defined strings. F_0 is the string **a**, F_1 is the string **bc**, and F_{n+2} is the concatenation of F_n and F_{n+1}. For example, F_2 is **abc**, F_3 is **bcabc**, F_4 is **abcbcabc**, etc. Given a number n and an index k, return the kth character of the string F_n.

Memoization and Dynamic Programming

41. Write a function that uses memoization to compute the nth Fibonacci number without re-computing any intermediate values.

42. You are given a pyramid of numbers like the one shown here:

```
          137
        42   -15
      -4    13    45
    21   14  -92   33
```

    Values in the pyramid can be both positive or negative. A path from the top of the pyramid to the bottom consists of starting at the top of the pyramid and taking steps diagonally left or diagonally right down to the bottom of the pyramid. The cost of a path is the sum of all the values in the pyramid. Find the path from the top of the pyramid to the bottom with the lowest total cost. Use memoization or dynamic programming to dramatically speed up your solution over a naïve recursive algorithm.

43. Let's consider a pattern-matching problem. You're given as input two strings, a text string and a pattern string. The text string consists purely of lower-case letters. The pattern string contains lower-case letters (which only match the given character), plus the ? character (which means "match anything") and the * character (which means "match zero or more characters.") For example, the word `apple` matches the patterns `apple`, `ap?le`, `a*e`, and `*`, but not the patterns `a?`, `*p`, or `apple?`. Write a function that takes in a text and a pattern and determines whether the text matches the pattern. Use memoization or dynamic programming to dramatically speed up your solution over a naïve recursive algorithm.

44. Imagine you have a strip of highway connecting n towns in a line. Each town is being evaluated as a possible location for building a cellular phone tower. Each cell tower will provide coverage to the town it's built in, but no other towns. However, due to the way that cell towers interfere with one another, it's not possible to build cell phone towers in two adjacent cities. Design an algorithm that, given the populations of each of the towns, reports the maximum number of people you can provide cell phone coverage to, subject to the restriction that no two adjacent towns can each have cell towers in them. Use memoization or dynamic programming to dramatically speed up your solution over a naïve recursive algorithm.