Section #4

Ashley Taylor and Shreya Shankar (Based on handouts by previous CS106B instructors and TAs)

Recursion and Recursive Backtracking

1. Digit Sum (Recursion)

Write a recursive function named `digitSum` that accepts an integer as a parameter and returns the sum of its digits. For example, calling `digitSum(1729)` should return 1 + 7 + 2 + 9, which is 19. If the number is negative, return the negation of the value. For example, calling `digitSum(-1729)`should return -19.

2. Edit Distance (Exhaustive Search)

Write a recursive function named `editDistance` that accepts string parameters `s1` and `s2` and returns the "edit distance" between the two strings as an integer. Edit distance (also called Levenshtein distance) is defined as the minimum number of "changes" required to get from `s1` to `s2` or vice versa. A "change" can be defined as a) inserting a character, b) deleting a character, or c) changing a character to a different character.

Call	Value Returned
`editDistance("driving", "diving")`	1
`editDistance("debate", "irate")`	3
`editDistance("football", "cookies")`	6

Your solution must not use any loops; it must be recursive. Strings have member functions named `find` and `rfind`, but you should not call them, because they allow you to get around using recursion. Similarly, the `replace` member is forbidden. Do not construct any data structures (no array, vector, set, map, etc.), and do not declare any global variables. You are allowed to define other "helper" functions if you like.

3. List Twiddles (Backtracking)

Write a recursive function named `listTwiddles` that accepts a string `str` and a reference to an English language `Lexicon` and uses exhaustive search and backtracking to print out all those English words that are `str`'s twiddles. Two English words are considered twiddles if the letters at each position are either the same, neighboring letters, or next-to-neighboring letters. For instance, "sparks" and "snarls" are twiddles. Their second and second-to-last characters are different, but 'p' is two past 'n' in the alphabet, and 'k' comes just before 'l'. A more dramatic example: "craggy" and "eschew" are also twiddles. They have no letters in common, but craggy's 'c', 'r', 'a', 'g', 'g', and 'y' are -2, -1, -2, -1, 2, and 2 away from the 'e', 's', 'c', 'h', 'e', and 'w' in "eschew". And just to be clear, 'a' and 'z' are not next to each other in the alphabet; there's no wrapping around at all. *Note: any word is considered to be a twiddle of itself, so it's okay to print str itself.*

4. Longest Common Subsequence (Backtracking)

Write a recursive function named `longestCommonSubsequence` that returns the longest common subsequence of two strings. Recall that if a string is a subsequence of another, each of its letters occurs in the longer string in the same order, but not necessarily consecutively. For example, the following calls should return the following values:

Call	Value Returned
`longestCommonSubsequence("ashley", "shreya")`	”shey”
`longestCommonSubsequence("hannah", "banana")`	”anna”
`longestCommonSubsequence("she sells", "seashells")`	”sesells”
`longestCommonSubsequence("janet", "cs106b")`	””

5. Spellable (Backtracking)

Write a recursive function named `isElementSpellable` that uses backtracking to check whether a given string can be spelled out using just element symbols from the Periodic Table of the Elements. For example, the word ”began” can be spelled out as BeGaN (beryllium, gallium, nitrogen), and the word ”feline” can be spelled out as FeLiNe (iron, lithium, neon). Not all words have this property, though; the word ”interesting” cannot be made out of element letters, nor can the word ”chemistry” (though, interestingly, the word ”physics” can be made as PHYSICS (phosphorous, hydrogen, yttrium, sulfur, iodine carbon, sulfur).

You don’t need to know anything about chemistry or have the periodic table memorized to solve this problem; you are given a `Lexicon` containing all the element symbols in the periodic table. Your function should accept two parameters: a `string` of the text to try to spell out, and a reference to a Lexicon of all the element symbols in the Periodic Table. Your function should return true if that string can be written using only element symbols, and false if it cannot. If you like, you may assume the fact that all element symbols are at most three letters. If passed the empty string, you should return true. You may use a loop in your solution if you like, but the overall algorithm must use recursion and backtracking.

Constraints: Your function should not modify the state of the lexicon passed in.