02-201 Programming Practice Problems #1

Carl Kingsford

These are some programming problems you can try on your own. They are not required, but a good starting point to test your knowledge.

1. Write a function `merge(L1, L2 []int) []int` that takes two sorted lists and returns a merged sorted list.

2. Write a function `countPartitions(L []int, k int) int` that counts the number of ways that integers in `L` can be summed together to equal `k`. For example:

   ```
   countPartitions([]int{2,3,4}, 6)
   ```

   should return 3 since $6 = 2 + 4$, $6 = 3 + 3$, and $6 = 2 + 2 + 2$. Hint: use recursion.

3. Connect four: write a function `connectFour(board [][]int) bool` that returns true if there are 4 consecutive `true` values in a row, column, or diagonal of the matrix `board`.

4. Write a function `secondSmallest(L []int) int` that returns the *second smallest* item in a list of integers.

5. Write a function `sample(dist []float64) int` that returns a random integer between 0 and `len(dist)-1`, inclusive, where integer i is chosen with probability `dist[i]`. You can assume the the sum of the elements of `dist` is 1, and that you have a function `random()` that returns a random `float64` in $[0, 1)$.

6. Write a function `wordHistogram(filename string) map[string]int` that reads every space-separated word from the given file and returns a map that says how many times each word occurred.

7. Write a function `isPermutation(a, b []int) bool` that returns `true` if the integers in `b` are a permutation of the integers in `a`.

8. (a) Write a function `intPatternReplace(long, pattern, newpat []int)` that searches `long` for the first occurrence of the sequence of integers in `pattern`, and replaces it with `newpat`. For example:

   ```
   L := []int{1,2,3,2,3}
   intPatternReplace(L, []int{2,3}, []int{5,7})
   ```

   should change `L` to `[]int{1,5,7,2,3}`. You can assume `len(pattern) == len(newpat)`.

   (b) Remove the assumption that `len(pattern) == len(newpat)` and modify your function to return the new list (which may be of different size than the original `long` list).

   (c) Change your function of part (b) to replace *all* occurrences of `pattern`. Among overlapping occurrences, replace the first. Be careful of the situation:

   ```
   L := []int{1,2,3,3,3}
   intPatternReplace(L, []int{2,3}, []int{2})
   ```

   which should return `[]int{1,2,3,3}` and not `[]int{1,2,2,2}`.

9. Write a function `maze(m [][]bool, startx, start, finishx, finishy int) []string` that returns a slice of commands `"left"`, `"right"`, `"forward"` that indicate a path to move from position $(startx, starty)$ to $(finishx, finishy)$ without stepping on any squares in `m`

that contain `false`. You can assume that all the walls of the maze (the `false` entries in `m`) are connected together. Hint: hug the left wall.

10. **(Harder)** Write a function

    ```
    wordChain(dict map[string]bool, start, end string) []string)
    ```

    that takes as parameters two strings and a set of words. Here `dict` is a map of strings to bools, where we don't care about the bools: we say a string `w` is in the dictionary if the entry `dict[w]` exists. Your function should return a list of words that are in the dictionary, and where `start` is the first word, `end` is the last word, and each word differs from its neighbors by 1 letter. For example: `["cat", "bat", "bet", "get"]`. Hint: use a stack.