CMPS 2200 Assignment 3

In this assignment we'll explore further sequence, map, reduce, scan, and divide and conquer algorithms.

To make grading easier, please place all written solutions directly in `answers.md`, rather than scanning in handwritten work or editing this file.

All coding portions should go in `main.py` as usual.

Parenthesis Matching

A common task of compilers is to ensure that parentheses are matched. That is, each open parenthesis is followed at some point by a closed parenthesis. Furthermore, a closed parenthesis can only appear if there is a corresponding open parenthesis before it. So, the following are valid:

- `( ( a ) b )`
- `a () b ( c ( d ) )`

but these are invalid:

- `( ( a )`
- `(a ) ) b (`

Below, we'll solve this problem three different ways, using iterate, scan, and divide and conquer.

a. (8pts) iterative solution Implement `parens_match_iterative`, a solution to this problem using the `iterate` function. **Hint**: consider using a single counter variable to keep track of whether there are more open or closed parentheses. How can you update this value while iterating from left to right through the input? What must be true of this value at each step for the parentheses to be matched? To complete this, complete the `parens_update` function and the `parens_match_iterative` function. The `parens_update` function will be called in combination with `iterate` inside `parens_match_iterative`. Test your implementation with `test_parens_match_iterative`.

.
.

b. (8pts) What are the recurrences for the Work and Span of this solution? What are their Big Oh solutions?

enter answer in `answers.md`

.
.

c. (8pts) scan solution Implement `parens_match_scan` a solution to this problem using `scan`. **Hint**: We have given you the function `paren_map` which

maps `(` to `1`, `)` to `-1` and everything else to `0`. How can you pass this function to `scan` to solve the problem? You may also find the `min_f` function useful here. Implement `parens_match_scan` and test with `test_parens_match_scan`

.
.

d. (8pts) Assume that any `map`s are done in parallel, and that we use the efficient implementation of `scan` from class. What are the recurrences for the Work and Span of this solution?

enter answer in `answers.md`

.
.

e. (10pts) divide and conquer solution Implement `parens_match_dc_helper`, a divide and conquer solution to the problem. A key observation is that we *cannot* simply solve each subproblem using the above solutions and combine the results. E.g., consider '((()))', which would be split into '(((' and ')))', neither of which is matched. Yet, the whole input is matched. Instead, we'll have to keep track of two numbers: the number of unmatched right parentheses (R), and the number of unmatched left parentheses (L). `parens_match_dc_helper` returns a tuple (R,L). So, if the input is just '(', then `parens_match_dc_helper` returns (0,1), indicating that there is 1 unmatched left parens and 0 unmatched right parens. Analogously, if the input is just ')', then the result should be (1,0). The main difficulty is deciding how to merge the returned values for the two recursive calls. E.g., if (i,j) is the result for the left half of the list, and (k,l) is the output of the right half of the list, how can we compute the proper return value (R,L) using only i,j,k,l? Try a few example inputs to guide your solution, then test with `test_parens_match_dc_helper`.

.
.

f. (8pts) Assuming any recursive calls are done in parallel, what are the recurrences for the Work and Span of this solution? What are their Big Oh solutions?

enter answer in `answers.md`

.
.