Version 1.1 `tag — inprogress` ***due:*** *Mon, Jan 28 @ 11:59pm*

1 Introduction

This assignment is meant to help you familiarize yourself with the hand-in mechanism for 15-210 and to get you thinking about proofs and coding again. To that end, you will answer some questions about the mechanics and infrastructure of the course, implement two solutions to the parenthesis distance problem, and perform some analysis of your solutions. Finally, you will complete some exercises involving big-O and prove an important identity.

2 Files

After downloading the assignment tarball from Autolab, extract the files from it by running `tar -xvf parenlab-handout.tgz` from a terminal window. You should see the following files:

1. `parenlab.pdf`
2. `Makefile`
3. `sources.cm`
4. `support.cm`
5. `lib/`
6. `PAREN.sig`
7. `ArrayParenPackage.sml`
8. `ParenDistTester.sml`
9. `MkSequentialPD.sml`
10. * `MkBruteForcePD.sml`
11. * `MkDivideAndConquerPD.sml`
12. * `inputs.txt`

You should only modify the last 3 files, denoted by *. Additionally, you should create a file called `parenlab-written.pdf` which contains the answers to the written part of the assignment.

3 Submission

To submit your assignment: open a terminal, `cd` to the `parenlab-handout` folder, and run `make`. This should produce a file called `handin.tgz` which contains the following files: `MkBruteForcePD.sml`, `MkDivideAndConquerPD.sml inputs.txt`, and `parenlab-written.pdf`. Open the Autolab webpage and submit this file via the "Handin your work" link.

4 Mechanics

The following questions are intended to make sure that you have read and understood the various policies (see `http://cs.cmu.edu/~15210/policy.html`) for the course, as well as found the tools that we've set up to communicate with you.

Task 4.1 (1%). Describe the picture posted as an instructor's note on Piazza. Be creative!

Solution 4.1 The picture shows a kitten with a bow tie.

Task 4.2 (3%). In each of the following situations, have the students followed or broken the collaboration policy? Briefly justify your answers with a sentence or two.

1. Ludmilla, Joaquin, and Alexander have lunch together after 210 lecture. Over lunch, they discuss the homework assignment released earlier in the week, including their progress so far. After lunch, all three go to different classes and don't think about the 210 homework until that evening.
2. While working on 213 homework near his friends from 210, Jon has a moment of insight into the 210 homework assignment. He becomes excited, and tells his friends what he thought of. Ishmael hasn't gotten that far in the homework, so he doesn't quite understand what Jon is talking about. Nonetheless, Ishmael copies down what he thinks are the key bits of what he heard to look at when he gets that far.
3. Yelena has been working on the 210 homework but can't figure out why her solution isn't compiling. She asks Abida to work through a couple of toy examples of functor syntax together, and they do so with a text editor and the SML REPL. Afterwards, Yelena gets her homework to compile and keeps working towards a solution.

Solution 4.2

1. The trio has not violated the collaboration policy. They had a casual conversation about the homework, and no notes were taken.
2. Ishamel violated the collaboration policy by writing down notes about Jon's idea. Jon has not violated the collaboration policy.
3. Yelena and Abida have not violated the collaboration policy. By working through a different example of the syntax that Yelana was stuck on, they worked together to help Yelena learn SML syntax without unintentionally trading answers to the homework.

Task 4.3 (1%). If you hand in your homework 25 hours after the posted due date, and you have no late days remaining, what is your maxiumum possible score?

Solution 4.3

- With no more late days, a perfectly correct assignment worth 100 points handed in 25 hours late will receive no more than 50 points.

- With one more late days, a perfectly correct assignment worth 100 points handed in 25 hours late will receive no more than 75 points.
- With two more late days, a perfectly correct assignment worth 100 points handed in 25 hours late will receive no more than 100 points.

5 The Parenthesis Distance Problem

A string s 'closed' if and only if it contains only '(' and ')' characters and is one of the following:

- The concatenation of two closed strings, $s_1 s_2$.
- A single closed string s_0 surrounded by a pair of 'matched' parentheses, (s_0).
- A single pair of matched parentheses, ().

The distance between a pair of matched parentheses is the number of characters between the parentheses (exclusive). The maximum parenthesis distance problem is to find the largest distance between a pair of matched parentheses in a closed string. More formally, for the closed string s, let M_s be the set of index pairs such that for $(i, j) \in M_s$, $i < j$ and (s_i, s_j) is a pair of matched parentheses. The maximum parenthesis distance is

$$\max\{j - i - 1 | (i, j) \in M_s\}$$

For example, the string '**(**()()**)**()', has a maximum parenthesis distance of 4.

5.1 Representation

When solving this problem, instead of interacting with strings, you will work with `paren` sequences, where the type `paren` is defined in a structure that ascribes to `PAREN_PACKAGE` and is given by

```
datatype paren = OPAREN | CPAREN
```

with `OPAREN` corresponding to a left parenthesis and `CPAREN` corresponding to a right parenthesis.

5.2 Implementation

You will implement two solutions of the function `parenDist : paren seq -> int option` such that `parenDist` $S \Rightarrow$ `NONE` if S is not closed and `parenDist` $S \Rightarrow$ `SOME` max if S is closed, where max is the maximum parenthesis distance in S.

Each solution will be a functor ascribing to the signature `PAREN_DIST`, defined in `PAREN.sig`. Each functor will take a structure ascribing to `PAREN_PACKAGE` as its only argument. The signature `PAREN` is defined in terms of the `SEQUENCE` signature from our library. For testing, you should use the structure `ArrayParenPackage`, which uses the `ArraySequence` implementation of `SEQUENCE`.

How to indicate parallelism? As seen in recitation, you should use the function `par` (inside the structure `Primitives`) to express parallel evaluation. Parallel operations can also be expressed in terms of operations on sequences such as `map` or `reduce`. *You must be explicit about what calls are being made in parallel to receive full credit.*

5.3 Brute-Force Algorithm

It is possible to give a brute-force algorithm by generating all possible solutions and picking the best. Note that this is different from the sequential solution which is provided for you in `MkSequentialPD.sml`.

Task 5.1 (9%). Complete the functor `MkBruteForcePD` in the file `MkBruteForcePD.sml` with a brute-force solution to the maximum parenthesis distance problem. You may use the the solution to the parenthesis matching problem from recitation 1. You may also find `Seq.subseq` to be useful for your solution.

Task 5.2 (5%). What is the work and span of your brute-force solution? You should assume `subseq` has $O(1)$ work and span, where m is the length of the resulting subsequence, and `parenMatch` has $O(n)$ work and $O(\log^2 n)$ span where n is the length of the sequence.

5.4 Divide-and-Conquer Algorithm

You will now implement a solution to the maximum parenthesis distance problem using divide-and-conquer recursive programming. If the work of showing the `treeview` of a sequence is denoted W_{showt}, then the work of your solution must be expressed by the recurrence

$$W_{parenDist}(n) = 2 \cdot W_{parenDist}\left(\frac{n}{2}\right) + W_{showt} + O(1)$$

with

$$W_{parenDist}(0) \in O(1)$$

A solution with correct input-output behavior but with work that is not described by this recurrence will not receive full credit.

Task 5.3 (24%). Complete the functor `MkDivideAndConquerPD` in the file `MkDivideAndConquerPD.sml` with a divide-and-conquer solution as described above. For this assignment, you are not required to submit a proof of correctness of your implementation. However, we advise that you work out a proof by mathematical induction for your solution as an exercise.

Task 5.4 (20%). The specification in Task 5.3 stated that the work of your solution must follow a recurrence that was parametric in the work it takes to view a sequence as a tree. Naturally, this changes with the sequence implementation.

1. Solve the recurrence with the assumption that $W_{showt} \in O\left(\lg n\right)$. where n is the length of the input sequence.
2. Solve the recurrence with the assumption that $W_{showt} \in O(1)$
3. In two or three sentences, describe a data structure to implement the sequence α `seq` that requires `showt` to have $O(\lg n)$ work.
4. In two or three sentences, describe a data structure to implement the sequence α `seq` that requires `showt` to have $O(1)$ work.

Solution 5.4

1. If $W_{showt} \in O\left(\lg n\right)$, then we have the recurrence

$$W_{parenDist}(n) = 2 \cdot W_{parenDist}\left(\frac{n}{2}\right) + O(\lg n) + O(1)$$

Arguing informally with the tree method, we do $O(\lg n) + O(1)$ work at every level of the recursion tree. Each call makes two subcalls with exactly half the data, giving us $\lg n$ levels where the i^{th} level has 2^i nodes. This gives the following sum:

$$\begin{aligned}
&\sum_{i=0}^{\lg n}\left(2^i\left(k_1 \lg\left(\frac{n}{2^i}\right) + k_2\right)\right) \\
=&\sum_{i=0}^{\lg n}\left(k_1 2^i \lg\left(\frac{n}{2^i}\right)\right) + \sum_{i=0}^{\lg n}\left(k_2 2^i\right) \\
=&k_1 \sum_{i=0}^{\lg n}\left(2^i\left(\lg n - \lg 2^i\right)\right) + k_2 \sum_{i=0}^{\lg n}\left(2^i\right) \\
=&k_1 \sum_{i=0}^{\lg n}\left(2^i\left(\lg n - i\right)\right) + k_2 \sum_{i=0}^{\lg n}\left(2^i\right) \\
=&k_1 \sum_{i=0}^{\lg n}\left(2^i \lg n\right) - k_1 \sum_{i=0}^{\lg n}\left(i 2^i\right) + k_2 \sum_{i=0}^{\lg n}\left(2^i\right) \\
=&k_1 \lg n \sum_{i=0}^{\lg n}\left(2^i\right) - k_1 \sum_{i=0}^{\lg n}\left(i 2^i\right) + k_2 \sum_{i=0}^{\lg n}\left(2^i\right) \\
=&k_1 \lg n(2n-1) - k_1 \sum_{i=0}^{\lg n}\left(i 2^i\right) + k_2(2n-1) \\
=&k_1 \lg n(2n-1) - k_1((2n \lg n) - (2n-2)) + k_2(2n-1)
\end{aligned}$$

which is in $O(n)$.

2. If $W_{showt} \in O\left(1\right)$, then we have the recurrence

$$W_{parenDist}(n) = 2 \cdot W_{parenDist}\left(\frac{n}{2}\right) + O(1)$$

Arguing informally with the tree method, we do $O(1)$ work at every level of the recursion tree. Each call makes two subcalls with exactly half the data, giving us $\lg n$ levels

where the i^{th} level has 2^i nodes. This gives the following sum:

$$\begin{aligned}
&\sum_{i=0}^{\lg n} 2^i \\
=&n + \frac{n}{2} + \frac{n}{4} + \ldots + \frac{n}{n} \\
=&n \sum_{i=0}^{\lg n} \frac{1}{2^i} \\
=&2n
\end{aligned}$$

which is in $O(n)$

3. If we implement the abstract sequence type as a balanced tree, we can find the middle element of the tree in $O(\log(n))$ work, and then collect the left and right halves of the tree and rebalance in $O(\log(n))$ work as well. Note that it is not sufficient to split at the root and rebalance, because showt requires that the two halves have almost exactly equal sizes, while most balanced trees do not.
4. If we implement the abstract sequence type in an array paired with two integers, one indicating the index that the represented array begins at and the other representing the length of the array, then we can split the array into halves by using the same array but changing the lengths to be half and adding the new length to the right array's starting index. Note that in this implementation the array itself is not changing and only the integers paired with it are modified.

5.5 Testing Your Code

Task 5.5 (2%). You will receive 1 point for each implementation that passes the very basic tests that we provided for you. You should, however, add more tests for your code as described in the next task.

Note: This is the only part of the assignment you will get immediate feedback on AutoLab for.

Task 5.6 (5%). In this course you will be expected to test your code extensively. For this assignment you should make sure you thoroughly and carefully test both of your implementations of the `PAREN_DIST` signature. Your tests should include both edge cases and more general test cases on specific sequences.

To aid with testing we have provided a testing structure, `ParenDistTester`, which should simplify the testing process. `ParenDistTester` will look at a file, `inputs.txt`, in which you should put your test input. `ParenDistTester` parses the file line by line making each line a new test case. Each line should consist of the characters '(' and ')' only, which the tester will translate into a `paren seq`. You should be careful not to have white-space within or at the end of a line; white-space will cause the tester to raise a `FailInputOutputFileFormat` exception. Finally, you should document your test cases in the `inputs.txt` file. Any line in `inputs.txt` that begins with `//` will not be treated as a test case and can be used for comments.

At submission time there must not be any testing code in the same file as your various `PAREN_DIST` implementations, as it can make it difficult for us to test your code. Additionally, unlike the 15-150 testing methodology, our testing structure does not test your code at compile time. In order to test

your code, after running `CM.make`, you will need to run `ParenDistTester.testMyCodeBF ()` to test your Brute Force implementation, and `ParenDistTester.testMyCodeDC ()` to test your Divide-and-Conquer implementation.

6 Asymptotics

For this problem, let's recall the definition of big-O:

Definition 6.1. A function $f : \mathbb{N} \to \mathbb{R}_+$ is in $O(g)$ if and only if there exist constants $N_0 \in \mathbb{N}$ and $c \in \mathbb{R}_+$ such that for all $n \geq N_0$, $f(n) \leq c \cdot g(n)$.

Task 6.1 (5%). Rearrange the list of functions below so that it is ordered with respect to O—that is, for every index i, all of the functions with index less than i are in big-O of the function at index i. You can just state the ordering; you don't need to prove anything.

1. $f(n) = n^{\lg(n^2)}$
2. $f(n) = 2n^{1.5}$
3. $f(n) = (n^n)!$
4. $f(n) = 43^n$
5. $f(n) = \lg(\lg(\lg(\lg(n))))$
6. $f(n) = 36n^{52} + 15n^{18} + n^2$
7. $f(n) = n^{n!}$

Solution 6.1

1. $f(n) = \lg(\lg(\lg(\lg(n))))$
2. $f(n) = 2n^{1.5}$
3. $f(n) = 36n^{52} + 15n^{18} + n^2$
4. $f(n) = n^{\lg(n^2)}$
5. $f(n) = 43^n$
6. $f(n) = n^{n!}$
7. $f(n) = (n^n)!$

Task 6.2 (15%). Carefully **prove** each of the following statements, or provide a counterexample and **prove** that it is in fact a counterexample. You should refer to the definition of big-O. Remember that verbose proofs are not necessarily careful proofs.

1. O is a transitive relation on functions. That is to say, for any functions f, g, h, if $f \in O(g)$ and $g \in O(h)$, then $f \in O(h)$.

Solution 6.2 This claim is true.
By the definition of $O, \exists N_1, C_1 : \forall n \geq N_1 : f(n) \leq C_1 g(n)$.
By the definition of $O, \exists N_2, C_2 : \forall n \geq N_2 : g(n) \leq C_2 h(n)$.
It follows that $\forall n \geq \max(N_1, N_2) : f(n) \leq C_1 C_2 h(n)$.
This is the definition of $f \in O(h)$ with $N = \max(N_1, N_2)$ and $C = C_1 C_2$.

2. O is a symmetric relation on functions. That is to say, for any functions f and g, if $f \in O\left(g\right)$, then $g \in O\left(f\right)$.

   Solution 6.2 This claim is false. It suffices to provide a counterexample. Let $f(n) = 1$ and $g(n) = n!$. $f \in O\left(g\right)$ is witnessed by $N = 0$, so it suffices to show that it's not the case that $g \in O\left(f\right)$.
   Assume for contradiction that it is the case that $g \in O\left(f\right)$. By definition, then,

   $$\exists N, c : \forall n \geq N, g(n) \leq cf(n)$$

   . Pick n so that $n > N$ and $n! > 1/c$ to show that this yields a contradiction.

3. O is an anti-symmetric relation on functions. That is to say, for any functions f and g, if $f \in O\left(g\right)$ and $g \in O\left(f\right)$, then $f = g$.

   Solution 6.2 This claim is false. It suffice to produce a counterexample. Let $f(n) = n$ and $g(n) = 2n$. Note that $f \neq g$. Not only are they different functions, they differ at every point other than 0: that is to say

   $$\forall x.x \geq 0 \implies f(x) \neq g(x)$$

   We can see that $f \in O\left(g\right)$ by picking $N = 1$ and $c = 1$. Similarly, we can see that $g \in O\left(f\right)$ by picking $N = 1$ and $c = \frac{1}{2}$.

Task 6.3 (10%). Show that for $a \in \mathbb{R}$, $a \neq \pm 1$, $1 + a^2 + a^4 + \cdots + a^{2n} = \frac{a^{2n+2}-1}{a^2-1}$.

Solution 6.3

Lemma 6.2. $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ *for* $x \neq 1$

Proof.

$$\left(\sum_{i=0}^{n} x^i\right)(x-1) = \sum_{i=0}^{n} x^{i+1} - \sum_{i=0}^{n} x^i = x^{n+1} + \left(\sum_{i=1}^{n} x^i\right) - \left(\sum_{i=1}^{n} x^i\right) - 1 = x^{n+1} - 1$$

Since $x \neq 1$, we can divide through by $x - 1$ to get the desired identity.

□

By substituting a^2 for x, we see that $1 + a^2 + a^4 + \ldots a^{2n} = \frac{x^{2n+2}-1}{a^2-1}$.