CS358: Applied Algorithms

Assignment 8: The SA-IS Algorithm (due 12/09/21 10PM EDT)

Instructor: Sam McCauley

Instructions

All submissions are to be done through github. This process is detailed in the handout "Handing In Assignments" on the course website. Answers to the questions below should be submitted by editing this document. All places where you are expected to fill in solution are marked in comments with "FILL IN."

Please contact me at srm2@cs.williams.edu if you have any questions or find any problems with the assignment materials.

We will have a leaderboard for this assignment. This will allow some automatic testing and feedback of your code, as well as giving some opportunity for friendly competition and extra credit. Leaderboard testing will start Friday evening; I will push a `largeInput.txt` to help you time your code before then.

Note that this assignment is due on **Thursday**.

Problem Description

In this assignment, you will be obtaining the suffix array of a given string using the SA-IS algorithm.

You will be given the string as an array of unsigned 32-bit integers,[1] along with the length of the string, and the number of unique characters in the string.

Running the Executable: There are several flags to change the behavior of the program (`-vc` is default if there are no flags):

- `-v` is verbose mode: the original string and the final suffix array will both be output to standard input.
- `-t` times the suffix array creation and outputs the result to the screen
- `-c` means that the test program will compare the result of your SA-IS implementation to the result of a slower suffix array implementation to make sure your algorithm is correct.

[1] Ideally, the algorithm would take characters as input, and transform them into larger characters if necessary before making a recursive call. I don't want to get too bogged down in that level of detail however. Your algorithm will, therefore, receive unsigned 32-bit integers, and you should copy over the string if you'd like to use another data type.

For example, to check if your program gives the correct answer on `banana.txt`, outputting the string and the final suffix array to the screen, you would run:

```
./test.out -vc banana.txt
```

To compare the running time of your SA-IS implementation with the slower suffix array implementation (and also check for correctness) on input file `largeInput.txt`, you would run:

```
./test.out -ct largeInput.txt
```

To time the program on `largeInput.txt` without output or a correctness check, you would run:

```
./test.out -t largeInput.txt
```

Starter Code: I have given considerably more scaffolding for this assignment in the starter code compared to previous assignments, mostly in the form of comments and some filled-in functions. You may change any function in `sais.c` in any way you'd like—what I included is only there to (hopefully) save you some time.

(As with previous assignments, however, you should not change `test.c`.)

There is an implementation of an alternate method to calculate the suffix array given in `suffixArraySlow.c`; this is only there to allow you to test your code.

Data files: Currently, I have included the following data files.

- `notesExample.txt` is the example from the lecture notes and slides. (This is a particularly useful file for debugging.)
- `banana.txt` contains the string `banana`
- `chrom10ln.txt` contains 10 lines (of 80 characters each) of DNA
- `singleRecurse.txt` contains an instance that recurses exactly once (can be useful for catching some corner cases)

Problems

Code (80 points). Implement the SA-IS algorithm to find the suffix array of a string. You do not need to describe your implementation.

Problem 1 (20 points)**.** Analyze the number of cache misses incurred by the SA-IS algorithm in the external memory model. Explain your answer.

Solution.