CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Problem Set 4

Harvard SEAS - Fall 2021 *Due: Wed Oct. 6, 2021 (5pm)*

Your name:
Collaborators:
No. of late days used on previous psets:
No. of late days used after including this pset:

The purpose of this problem set is to solidify your understanding of the RAM model (and variants), and the relations between the RAM model, the Word-RAM model, Python programs, and variants. In particular, you will build skills in simulating one computational model by another and in evaluating the runtime of the simulations (both in theory and in practice).

1. (Simulation in practice: RAMs on Python) In the Github repository, we have given you a partially written Python implementation of a RAM Model simulator. Your task is to fill in the missing parts of the code to obtain a complete RAM simulator. Your simulator should take as input a RAM Program P and an input x, and simulate the execution of P on x, and return whatever output P produces (if it halts). The RAM Program P is given as a Python list $[v, C_0, C_1, \ldots, C_{\ell-1}]$, where v is the number of variables used by P. For simplicity, we assume that the variables are numbers $0, \ldots, v-1$, but you can introduce constants to give names to the variables. The 0^{th} variable will always be `input_len`, the 1^{st} variable `output_ptr`, and the 2^{nd} variable `output_len`. A command C is given in the form of a list of the form [`cmd`], [`cmd`, i], [`cmd`, i, j], or [`cmd`, i, j, k], where `cmd` is the name of the command and i, j, k are the indices of the variables or line numbers used in the command. For example, the command $\texttt{var}_i = M[\texttt{var}_j]$ would be represented as (`read`, i, j). See the Github repository for the precise syntax as well as some RAM programs you can use to test your simulator.

2. (Empirically evaluating simulation runtimes and explaining them theoretically) Consider the

following two RAM programs:

```
Input     : A single natural number N (as an array of length 1)
Output    : 7^(2^N) (as an array of length 1)
Variables: input_len, output_ptr, output_len, counter, result
 1 zero = 0;
 2 one = 1;
 3 output_len = 1;
 4 output_ptr = 0;
 5 result = 7;
 6 counter = M[zero];
 7     IF counter == 0 GOTO 11;
 8     result = result * result;
 9     counter = counter − one;
10     IF zero == 0 GOTO 7;
11 M[output_ptr] = result;
12 HALT;
```

```
Input     : A single natural number N (as an array of length 1)
Output    : 7^(2^N) mod 2^32 (as an array of length 1)
Variables: input_len, output_ptr, output_len, counter, result, temp, W
 1 zero = 0;
 2 one = 1;
 3 output_len = 1;
 4 output_ptr = 0;
 5 result = 7;
 6 W = 2^32;
 7 counter = M[zero];
 8     IF counter == 0 GOTO 15;
 9     result = result * result;
10     temp = result/W;
11     temp = temp × W;
12     result = result − temp;
13     counter = counter − one;
14     IF zero == 0 GOTO 8;
15 M[output_ptr] = result;
16 HALT ;
```

(a) Exactly calculate (without asymptotic notation) the RAM-model running times of the above algorithms as a function of N. Which one is faster?

(b) Using your RAM Simulator, run both RAM programs on inputs $N = 0, 1, 2, \ldots, 15$ and graph the actual running times (in clock time, not RAM steps). (We have provided you with some timing and graphing code in the Github repository.) Which one is faster?

(c) Explain the discrepancies you see between Parts 2a and 2b.

(d) (challenge*) Give a theoretical explanation (using asymptotic estimates) of the shapes of the runtime curves you see in Part 2b. You may need to do some research online and/or make guesses about how Python operations are implemented to come up with your estimates.

3. (Simulation in theory: Word-RAM vs. variants) One common assembly language operation that we did not include in our Word-RAM model is *bitwise-XOR*. Given w-bit numbers x and y with binary representations $x = x_{w-1}x_{w-2}\cdots x_0$ and $y = y_{w-1}y_{w-2}\cdots w_0$, their bitwise XOR $z = x \oplus y$ is the number whose binary representation $z = z_{w-1}z_{w-2}\cdots z_0$ satisfies $z_i = x_i \oplus y_i$ for $i = 0, \ldots, w-1$.

   (a) Prove that when the current word size is w, a bitwise XOR operation $z = x \oplus y$ can be computed in our Word-RAM model (without an $\oplus$ operation) in time $O(w)$. Here you are given x, y, z as individual variables in the Word-RAM program (no need to read from or write to memory), and the current word size is also given as a variable `word_len`. You may may introduce additional temporary variables if useful.

   (b) Using Part 3a, show that for every XOR-extended Word-RAM program P, there is an ordinary Word-RAM Program P' such that for every input x, $P'(x) = P(x)$. As above, you may assume that the current word size is given as a variable `word_len`, which is automatically updated as the word size increases.

   In addition, argue that when the input $x = (x[0], \ldots, x[n-1])$ satisfies $x[0], x[1], \ldots, x[n-1] \leq n$ (i.e. the input numbers are not too big relative to the length) and $\text{Time}_P(x) \geq n$ (i.e. P runs in enough time to read the entire input), then

$$\text{Time}_{P'}(x) = O\left(\text{Time}_P(x) \cdot \log \text{Time}_P(x)\right).$$

   (Hint: First show that the maximum memory size used by P on x is at most $n + \text{Time}_P(x)$.)

The take-away point is that while the exact choice of which operations to include in a RAM model may affect the asymptotic running time, it typically affects it by logarithmic factors (so it might make the difference between $O(n)$ and $O(n \log n)$, but not between $O(n)$ and $O(n^2)$).