CIS 303 Algorithm Analysis and Design
Assignment 4b: Program and Analysis, Ackermann Function

L. Grabowski

March 6, 2020

Directions: This is an individual assignment. You will submit your source code and analysis/discussion through Moodle. No hard copy is needed.

1. **Background**

   - For this assignment, you will implement Ackermann Function by simulating recursion using a stack. There are different versions of the definition of this function. For this assignment, you will implement the 2-argument version, defined as follows:

$$A(m,n) = \begin{cases} n+1, & \text{if } m = 0; \\ A(m-1,1), & \text{if } m > 0 \text{ and } n = 0; \\ A(m-1, A(m, n-1)), & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

   - You can read about Ackermann Function on this Wikpedia page:
     Ackermann Function NOTE: LaTeX did not let me put the URL for this page outside of the link for some reason. If you have problems with the link, see the Moodle handin assignment for the link.

2. **Part 1: Program.**

   - Here is a straightforward recursive implementation of Ackermann Function in Java:
     Ackermann.java. (URL: https://introcs.cs.princeton.edu/java/53universality/Ackermann.java). You will be using this implementation in your empirical testing in Part 2. I've included a Java source file in the assignment materials.
   - Write a non-recursive implementation for solving Ackermann Function, using a stack data structure to simulate recursion. **You must implement your own stack class. You may NOT use Java's Stack!**
   - You can use the `main` method in the `Ackermann.java` file as the test client program for your program. My test client is very similar to that code. Some testing tips:
     - **Be careful of the values you use as parameters to your function! The value grows very fast.** Be sure you read the background material carefully!
     - Be sure that you use test cases that thoroughly exercise your code.
   - Be sure that your code is correct, well structured, and well commented before moving on to Part 2. Keep in mind that you must follow good implementation practice for all your code. In particular, do NOT hard code parameters values into your test client. See below for details on the required parameter values for experiments.
   - **IMPORTANT NOTE FOR PROGRAMMING ASSIGNMENTS, BEGINNING WITH THIS ASSIGNMENT: YOU MUST TURN IN A README FILE AS DESCRIBED IN THE CS DEPARTMENT CODING STANDARDS**. On this assignment, failure to turn in an appropriate README will result in loss of some points. On future assignments, no README will be cause for me to stop grading your assignment with a score of 0. The README is critical because it includes your description for how you tested to ensure that your code is correct.

3. **Part 2: Empirical Testing and Analysis.**

   - Experiments:
     (a) Before running tests, use the two versions of the code to derive a mathematical analysis of the running time of each implementation (recursive, stack-based). Formulate a hypothesis of which implementation you expect to perform better and a detailed explanation of your hypothesis. As always, this will be the first point of your analysis.
     (b) Run the two solutions using the following 15 sets of parameter values, recording the running time into an output file:

     $$(0,0),(1,0),(1,10),(1,20),(2,0),(2,10),(2,20),(3,1),(3,2),(3,3),(3,4),(3,5),(3,6),(3,7),(3,8)$$

     Run your tests with the parameters in that order, and collect your timing data in the same fashion. For ease of plotting data, we will just consider each set of parameters as "set 1," "set 2," *etc.*.
     (c) Add code to your stack-based solution and to the original recursive version to collect timing data for computing the Ackermann Function.
   - Analysis:
     (a) Plot the timing test data, using the data set number (1 — 15) as the x-axis, and the time value as the y-axis. You will plot 2 different curves, 1 for the recursive solution and 1 for your stack solution. Be sure that your plot clearly identifies the 2 curves (use labels and colors to set them apart). I suggest that you use a simple line plot or scatter plot. Because of the parameter values we're using, curve fitting will not work well here.
     (b) Include the data plot in your analysis document (*i.e.,* generate an image of your plot and insert it into your document).
     (c) Discuss the observed performance of the two implementations of the algorithm. Connect your experimental results to your analysis and hypothesis and explain why you got the results that you did, whether or not they agree with your hypothesis.

4. **Submission.** Submit your completed `Ackermann.java` (and any additional supporting files) electronically to Moodle, along with your analysis/discussion file.

5. **Grading:**

   - Correctness of solution and all necessary files submitted: 20 pts
   - Style and commenting: 10 pts (reminder: follow the code style and commenting document in Moodle)
   - README is submitted and is correct: 5 pts.
   - Discussion: 45 pts
   - **Total: 80 pts**

   Grading form is available in Moodle.