Project 1: implementing algorithms

CPSC 335 - Algorithm Engineering
Spring 2020
Instructors: Doina Bein (dbein@fullerton.edu)

Abstract

In this project you will set up an environment for implementing algorithms in C++, and use that to implement two algorithms that solve the same problem. The first step is for you to either create your own Tuffix Linux install, or arrange to use a Tuffix environment on campus. The second step is to become familiar with GitHub and makefiles. We will use the Tuffix environment to run code and GitHub to submit and grade projects all semester. The third step is for you to translate descriptions of two algorithms into pseudocode; analyze your pseudocode mathematically; implement each algorithm in C++; test your implementation; and describe your results.

Installing Tuffix

As described in the syllabus, "Tuffix" is the Computer Science Department's official Linux development environment. Instructions on how to install Tuffix or a Tuffix based VM are online at «http://csufcs.com/tuffixinstall». The Tuffix Titanium Community for Students, https://communities.fullerton.edu/course/view.php?id=1547 is the best venue to find help with Tuffix.

The first part of this project is installing Tuffix, either using a virtual machine or a bare-metal install. You should follow the instructions in those documents, and the documents they link to, to build a working Tuffix environment.

Alternatively, you may opt to use the Tuffix VMs available in the ECS Open Lab in the CS building. In this case, you will need to make arrangements to be able to complete your projects during the lab's open hours.

The Alternating Disk Problem

The problem below is slightly changed from the one presented in Levitin's textbook as Ex. 14 on page 103:

> You have a row of $2n$ disks of two colors, n light and n dark. They alternate: light, dark, light, dark, and so on. You want to get all the light disks to the left-hand end, and all the dark disks to the right-hand end. The only moves you are allowed to make are those that interchange the positions of two adjacent disks (i.e. they touch each other). No other moves are allowed. Design an algorithm for solving this puzzle and determine the number of moves it takes.

The *alternating disks problem* is:

Input: a positive integer `n` and a list of `2n` disks of alternating colors light-dark, starting with light
Output: a list of `2n` disks, the first `n` disks are light, the next `n` disks are dark, and an integer `m` representing the number of swaps to move the dark ones after the light ones.

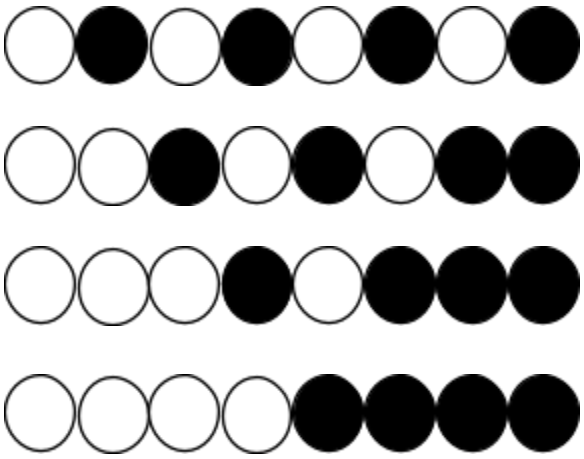
There are two algorithms, presented below, that solve this problem in $O(n^2)$ time. Some improvement can be obtained by not going all the way to the left or to the right, since some disks at the ends are already in the correct position. You need to translate the descriptions of the two algorithms into clear pseudocode. You are allowed to do the improvements as long as it does not change the description of the algorithm.

The first algorithm starts with the leftmost disk and proceeds to the right until it reaches the rightmost disk: compares every two adjacent disks and swaps them only if necessary. Now we have two light disks at the left-hand end and two dark disks at the right-hand end. Once it reaches the right-hand end, it goes back to the leftmost disk and proceeds to the right, doing the swaps as necessary. It repeats `n` times.

The second algorithm proceeds like a lawnmower: starts with the leftmost disk and proceeds to the right until it reaches the rightmost disk: compares every two adjacent disks and swaps them only if necessary. Now we have two light disks at the left-hand end and two dark disks at the right-hand end. Once it reaches the right-hand end, it starts with the rightmost disk, compares every two adjacent disks and proceeds to the left until it reaches the leftmost disk, doing the swaps only if necessary. The lawnmower movement is repeated $\lceil n/2 \rceil$ times.

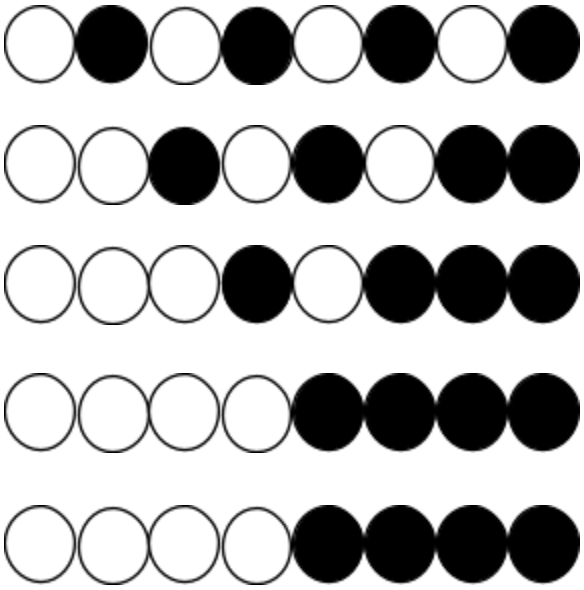
The left-to-right algorithm

It starts with the leftmost disk and proceeds to the right, doing the swaps as necessary. Now we have two lighter disks at the right-hand end and two darker disks at the left-hand end.Now we have two light disks at the left-hand end and two dark disks at the right-hand end. Once it reaches the right-hand end, it goes back to the leftmost disk and proceeds to the right, doing the swaps as necessary. It repeats `n` times.
Consider the example below when n=4, and the first row is the input configuration, the second row is the end of run 1, etc.. The exact list of disks changes as follows at the end of each run (we consider a run to be a check of adjacent disks from left-to-right):



The lawnmower algorithm

It starts with the leftmost disk and proceeds to the right until it reaches the rightmost disk: compares every two adjacent disks and swaps them only if necessary. Now we have two light disks at the left-hand end and two dark disks at the right-hand end. Once it reaches the right-hand end, it starts with the rightmost disk, compares every two adjacent disks and proceeds to the left until it reaches the leftmost disk, doing the swaps only if necessary. The lawnmower movement is repeated $\lceil n/2 \rceil$ times.
Consider the example below when n=4, and the first row is the input configuration, the second row is the end of comparison from left to right, the third row is the end of the first run (round trip that contains left to right followed by right to left), etc.. The exact list of disks changes as follows at the end of each run (we consider a run to be a check of adjacent disks from left-to-right or right-to-left) is shown below:



Algorithm Design

Your first task is to design an algorithm for each of the two problems. Write clear pseudocode for each algorithm. This is not intended to be difficult; the algorithms I have in mind are all relatively simple, involving only familiar string operations and loops (nested loops in the case of oreos and substrings). Do not worry about making these algorithms exceptionally fast; the purpose of this experiment is to see whether observed timings correspond to big-O trends, not to design impressive algorithms.

Mathematical Analysis

Your next task is to analyze each of your two algorithms mathematically. You should prove a specific big-O efficiency class for each algorithm. These analyses should be routine, similar to the ones we have done in class and in the textbook. I expect each algorithm's efficiency class will be one of $O(n)$, $O(n^2)$, $O(n^3)$, or $O(n^4)$.

Obtaining and Submitting Code

This document explains how to obtain and submit your work:

GitHub Education / Tuffix Instructions

Here is the invitation link for this project:

https://classroom.github.com/g/j8wEpFJ2

Implementation

You are provided with the following files.

1. `disks.hpp` is a C++ header that defines functions for the two algorithms described above. There are also classes that represent the input and output of the alternating disk problem. The function definitions are incomplete skeletons; you will need to rewrite them to actually work properly.
2. `disks_test.cpp` is a C++ program with a `main()` function that performs unit tests on the functions defined in `disks.hpp` to see whether they work, prints out the outcome, and calculates a score for the code. You can run this program to see whether your algorithm implementations are working correctly.
3. `rubrictest.hpp` is the unit test library used for the test program; you can ignore this file.
4. `README.md` contains a brief description of the project, and a place to write the names and CSUF email addresses of the group members. You need to modify this file to identify your group members.

What to Do

First, add your group members' names to `README.md`. Then:

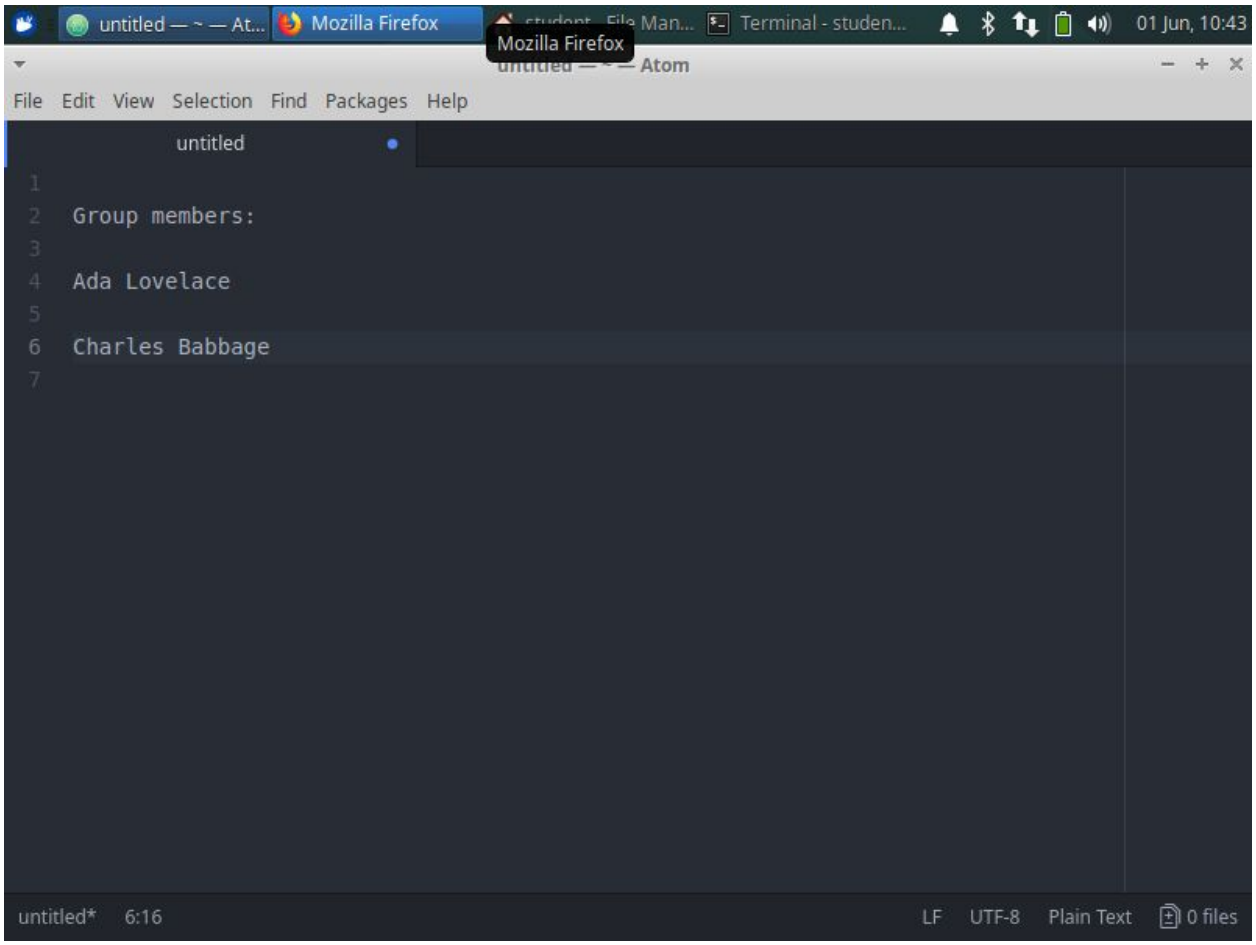
1. Write your own pseudocode for the left-to-right algorithm, and the lawnmower algorithm.
2. Analyze your pseudocode for the algorithm mathematically and prove its efficiency class.

3. Implement all the skeleton functions in `disks.hpp`. Use the `disks_test.cpp` program to test whether your code works.

Finally, produce a brief written project report ***in PDF format***. Submit your PDF by committing it to your GitHub repository along with your code. Your report should include the following:

1. Your names, CSUF-supplied email address(es), and an indication that the submission is for project 1.
2. A full-screen screenshot, inside Tuffix, showing the Atom editor or the editor you used, with your group member names show clearly as below. One way to make your names appear in Atom is to simply open your `README.md`.
3. Two pseudocode listings, for the two algorithms.
4. A brief proof argument for the time complexity of your two algorithms.

Your screenshot might look like this:



Grading Rubric

Your grade will be comprised of three parts: *Form, Function,* and *Analysis.*

Function refers to whether your code works properly as defined by the test program. We will use the score reported by the test program, when run inside the Tuffix environment, as your Function grade.

Form refers to the design, organization, and presentation of your code. A grader will read your code and evaluate these aspects of your submission.

Analysis refers to the correctness of your mathematical and empirical analyses, scatter plots, question answers, and the presentation of your report document.

The grading rubric is below.

1. Function = 14 points, scored by the unit test program
2. Form = 9 points, divided as follows:
    a. README.md completed clearly = 3 points
    b. Style (whitespace, variable names, comments, helper functions, etc.) = 3 points
    c. C++ Craftsmanship (appropriate handling of encapsulation, memory management, avoids gross inefficiency and taboo coding practices, etc.) = 3 points
3. Analysis = 17 points, divided as follows
    a. Report document presentation = 3 points
    b. Screenshot = 3 points
    c. Pseudocode = 3 points
    d. Mathematical analysis = 8 points

Legibility standard: As stated on the syllabus, submissions that cannot compile in the Tuffix environment are considered unacceptable and will be assigned an "F" ~~(50%)~~ grade.

Deadline

The project deadline is Friday, February ~~14~~ **21**, 11:59 pm, on GitHub. Not TITANium.

You will be graded based on what you have pushed to GitHub as of the deadline. Commits made after the deadline will not be considered. Late submissions will not be accepted.