Assignment #3

Greg Plaxton

October 3, 2012

NOTE ADDED 10/7/12: The original version of this handout incorrectly referred to Case 2, rather than Case 3, in the statement of Lemma 2. End of Note.

There are two parts to this assignment. The first part is described in Section 1 below, and may be done with a partner. If you are having trouble finding a partner, send me an email and I will match you up with someone else in the same situation.

The second part, described in Section 2 below, consists of textbook exercises, and is due at the beginning of class on Wednesday, October 17. Each student should work on this part separately.

Any corrections or clarifications related to the assignment will be posted in the *Assignments* section of the class website.

1 Programming & Problem Solving: Mooov Around

This part of the assignment has two subparts: a paper-and-pencil subpart involving proofs (see Section 1.2) and a programming subpart (see Section 1.3).

In Assignment 2 you developed a polynomial-time algorithm for solving the special case of the Mooov Around problem in which the student preferences are strict (i.e., there are no ties). In the present assignment, we develop an $O(n^2)$-time algorithm for exactly the same computational task as we addressed in Assignment 2. The algorithm that you will implement is a specific determinization of the algorithm described in the Assignment 2 handout. Thus all of the properties we established in Assignment 2 (see Lemmas 1 to 8 in the Assignment 2 handout) hold for the algorithm that you will implement in Assignment 3.

1.1 The Algorithm

After setting up the initial configuration as in Assignment 2, for each student i running from 0 to $n-1$, we will perform a "search from student i", as described below.

Fix an arbitrary student i. The process of searching from student i involves updating the configuration by performing a sequence of revelation and trading actions. We also maintain an ordered list of vertices that starts at student i. When the search from student i is initiated, the list is initialized to contain only the student i vertex. At a general step of the search

from student i, we maintain the following invariant: either the list is empty, or it contains the sequence of vertices visited by some simple directed path in the configuration starting at student i. The configuration and list are iteratively updated as follows.

Case 1: The list is empty. In this case, the search from student i terminates.

Case 2: The list ends with some student vertex i' (possibly equal to i). Let the unique outgoing edge of student i' go to space j.

Case 2.1: Space j belongs to the list. In this case, the directed path corresponding to the suffix of the list beginning at space j, plus the edge (i', j), forms a cycle C of length $2k$ for some positive integer k. We update the list by removing the last $2k$ vertices (i.e., the vertices of C). We update the configuration as follows.

Case 2.1.1: $k = 1$. Let student i'' denote the predecessor of space j in the list. Then a revelation action is enabled for student i'' (see Lemma 1 in Section 1.2), and we perform this action.

Case 2.1.2: $k > 1$. We perform the trading action that is enabled for C.

Case 2.2: Space j does not belong to the list. We leave the configuration unchanged and append space j to the list.

Case 3: The list ends with some space vertex j. Let the unique outgoing edge of space j go to some student i'. One of the following two conditions holds (see Lemma 2 in Section 1.2).

Case 3.1: $i' = i$. In this case the directed path corresponding to the list, plus the edge (j, i), forms a cycle C of length $2k$ for some positive integer k. We empty the list, and if $k > 1$ then we perform the trading action that is enabled for C.

Case 3.2: Student i' does not belong to the list. We leave the configuration unchanged and append student i' to the list.

1.2 Proofs

In this subpart of the assignment you will prove a subset of the seven lemmas stated below. When proving a given lemma, you are allowed to make use of any lower-numbered lemma, but you are not allowed to make use of a higher-numbered lemma. You can also make use of any of the Assignment 2 lemmas.

Each team is required to turn in a proof of Lemma 7, along with proofs of any three of Lemmas 1 through 6.

Your solutions to this subpart are due at the beginning of class on Wednesday, October 10.

Lemma 1: Prove that in Case 2.1.1, a revelation action is enabled for student i''.

Lemma 2: Prove that in Case 3, either $i' = i$ or student i' does not belong to the list.

Lemma 3: Prove that the total number of iterations (across all n searches) reaching Case 1 is n.

Lemma 4: Prove that the total number of iterations (across all n searches) reaching Case 2.1.1 is $O(n^2)$.

Lemma 5: Prove that the sum, over all iterations (across all n searches) reaching either Case 2.1.2 or 3.1, of the length of the associated cycle C, is $O(n)$. Hint: How many times

can a given vertex participate in a trading action?

Lemma 6: Prove that the sum, over all iterations (across all n searches), of the change in the length of the list is $-n$.

Lemma 7: Prove that the total number of iterations (across all n searches) reaching either Case 2.2 or 3.2 is $O(n^2)$. Hint: Make use of the three previous lemmas.

1.3 Programming Task

Implement the algorithm described in Section 1.1. Your program is due by 8pm on Wednesday, October 10. For any iteration reaching Case 2.1.2 or 3.1, your implementation is required to perform the iteration in $O(k)$ time, where k is the length of the associated cycle C. For any other iteration, your implementation is required to perform the iteration in $O(1)$ time. Assuming that your implementation meets these performance bounds, it spends $O(n)$ total time on Case 1 iterations (by Lemma 3), $O(n^2)$ time on Case 2.1.1, 2.2, and 3.2 iterations (by Lemmas 4 and 7), and $O(n)$ time on Case 2.1.2 and 3.1 iterations (by Lemma 5). Hence the total running time for the n search operations is $O(n^2)$. Subject to the following minor caveats, this implies the desired $O(n^2)$ overall time bound for your implementation.

1. The above analysis assumes that certain simple operations on $O(\log n)$-bit integers can be performed in constant time. For example, we assume that an $O(\log n)$-bit integer can be stored in a single "word" of storage, or can be used to index into an array in constant time. This is a common assumption in the design and analysis of algorithms. Computers with N "words" of memory tend to have a word size on the order of $\log_2 N$ bits, so that the address of an arbitrary memory location can be stored in a single word. Such computers also tend to have CPUs that perform basic indexing, arithmetic, and logic operations "quickly" on word-sized operands. Under the so-called "sequential RAM model" — a widely used model for analyzing the time complexity of sequential algorithms, and the model that we adopt throughout most of this course — such operations are assumed to take constant time. This is an idealization, since the circuitry required to perform most "basic" operations (e.g., addition of two k-bit words) has nonconstant depth. That is, the number of gate delays required by these families of circuits tends to infinity as the word size tends to infinity.

2. The total running time of your implementation includes not only the cost of performing the search operations, but also the cost of reading the input, initializing data structures, and writing the output. For the present assignment, only the cost of reading the input merits further discussion, since the cost of initializing the data structures and writing the output is minimal. Since the input specifies $O(n^2)$ $O(\log n)$-bit integers, we view the time required to read the input as $O(n^2)$ steps in the RAM model. In the present assignment, one could quibble with this view because we have chosen to represent the input values as strings of decimal digits. In an execution of your program, these digits are read sequentially, and therefore $\Omega(n^2 \log n)$ time steps are used to read the input. Had we made the (computationally more natural) assumption that the input

is provided as $O(n^2)$ $O(\log n)$-bit binary words, then the time required to read the input would truly be $O(n^2)$ in the RAM model. In summary, for an assignment like the present one, we sometimes adopt a decimal representation of the input for the sake of human readability, but when we discuss time complexity, we generally assume an efficient binary representation of the input.

2 Textbook Exercises

1. Problem 4.15, page 196.
2. Problem 4.28, page 203.
3. Problem 5.2, page 246.
4. Problem 5.6, page 248.