Assignment #1

Greg Plaxton

August 29, 2012

There are two parts to this assignment. The first part, described in Section 1 below, consists of a programming task that is due by 8pm on Friday, September 7. Students are allowed to work on the first part with a partner, and are strongly encouraged to do so. Each team should turn in only one solution for the first part. 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. Please be sure to read the document entitled *Guidelines for Programming Tasks*, which may be found in the *Assignments* section of the class website. This document includes a description of the slack time policy for programming tasks.

The second part, described in Section 2 below, consists of textbook exercises, and is due at the beginning of class on Wednesday, September 12. 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

At the start of the academic year at Cattle U., the university administration generates a "Mooov In" allocation of n dormitory spaces to n students. The spaces are numbered from 0 to $n-1$. For each space i, we refer to the student assigned to space i by the university as student i.

Some of the students decide to set up a website to solicit preference information from the n students, with the hope of using this information to compute an improved allocation. Each student is asked to provide tiered preferences for the set of n spaces: top tier, second tier, third tier, and so on. Given the preferences of a student i, we can determine for any two spaces j and j' whether i prefers j to j', is indifferent between j and j', or prefers j' to j.

Any allocation of the n spaces to the n students can be viewed as a permutation π over $\{0, \ldots, n-1\}$ where $\pi(i)$ denotes the space allocated to student i, for $0 \leq i < n$. (Thus the allocation proposed by the university corresponds to the identity permutation.)

Given the student preferences (i.e., for each student, a partition of the n spaces into tiers), we make the following definitions.

1. An allocation π is *individually rational* if each student i likes space $\pi(i)$ at least as well as space i.

2. Given two allocations π and π', we say that π' *weakly dominates* π if the following two conditions are satisfied: (1) each student i likes space $\pi'(i)$ at least as well as space $\pi(i)$; (2) some student i strictly prefers space $\pi'(i)$ to space $\pi(i)$.

3. An allocation is *strictly Pareto-efficient* if it is not weakly dominated by any other allocation.

Your task is to write a program that takes as input the student preferences and determines whether there exists an allocation that is individually rational and strictly Pareto-efficient. If so, your program should print out such an allocation. If not, your program should indicate that no such allocation exists.

In the present assignment, you are to accomplish this task through direct application of the definitions given above. More specifically, the program that you develop in this assignment should be based on subroutines for the following three subtasks.

1. Given the student preferences and an assignment π, determine whether π is individually rational. Approach: Just apply the definition of an individually rational allocation.

2. Given the student preferences and assignments π and π', determine whether π' weakly dominates π. Approach: Just apply the definition of weak domination.

3. Given the student preferences and an assignment π, determine whether π is strictly Pareto-efficient. Approach: Loop through all assignments π', using the previous subroutine to determine whether π' weakly dominates π.

Having implemented the above subroutines, you can solve the main task by looping through all assignments π and using the first and third subroutines above to check whether π is individually rational and strictly Pareto-efficient.

1.1 Input-Output Format

The first line of the input contains a nonnegative integer k that specifies the number of instances to follow. The integer k is followed by k "input blocks". Your program will produce k "output blocks", one for each input block.

1.1.1 Format of an Input Block

Each input block is formatted as follows. The first line contains a nonnegative integer n specifying the number of students/spaces. Each of the next n lines specifies the preferences of student i, $0 \leq i < n$. The preferences of a student are encoded as a sequence of blank-separated integers on a single line, using the special value -1 to encode tier boundaries. The following example should make this clear. Suppose n is 5 and student 3 has three tiers of preference: the first tier consists of the set of spaces $\{2, 4\}$; the second tier consists of the set of spaces $\{1\}$; the third tier consists of the spaces $\{0, 3\}$. The three lines below give three examples of (equivalent, for our purposes) ways that the preferences of student 3 can be represented in the input.

```
2 4 -1 1 -1 0 3
4 2 -1 1 -1 0 3
-1 4 2 -1 1 -1 -1 0 3 -1
```

1.1.2 Format of an Output Block

Each output block consists of a single line. If the instance specified in the corresponding input block does not admit an individually rational and strongly Pareto-efficient allocation, then your program should print "none" on the output line. Otherwise, the output line encodes an individually rational and strongly Pareto-efficient allocation π as a sequence of n blank-separated integers $\pi(0)$,..., $\pi(n-1)$. For example, for $n = 5$, the output line

```
3 2 4 0 1
```

encodes the allocation that assigns student 0 to space 3, student 1 to space 2, student 2 to space 4, student 3 to space 0, and student 4 to space 1.

1.1.3 Sample Inputs and Outputs

Check the *Assignments* section of the class website for a link to sample inputs and outputs, which will be made available within a couple of days.

2 Textbook Exercises

1. Exercise 2.2, page 67.
2. Exercise 2.3, page 67.
3. Exercise 2.8(a), page 69.
4. Prove that any strategy for solving the 2-jar highest safe rung problem of Exercise 2.8(a) requires $\Omega(\sqrt{n})$ drops in the worst case. Hint: Partition the ladder into a suitable collection of segments, and reason about the behavior of an arbitrary fixed strategy with respect to these segments.