CPSC 320 2017W2: Assignment 1

January 13, 2018

Please submit this assignment via GradeScope at `https://gradescope.com`. Be sure to identify everyone in your group if you're making a group submission. (Reminder: groups can include a maximum of three students; we strongly encourage groups of two.)

Submit by the deadline **Monday 22 Jan at 10PM**. For credit, your group must make a **single** submission via one group member's account, marking all other group members in that submission **using GradeScope's interface**. Your group's submission **must**:

- Be on time.
- Consist of a single, clearly legible file uploadable to GradeScope with clearly indicated solutions to the problems. (PDFs produced via LaTeX, Word, Google Docs, or other editing software work well. Scanned documents will likely work well. **High-quality** photographs are OK if we agree they're legible.)
- Include prominent numbering that corresponds to the numbering used in this assignment handout (not the individual quizzes). Put these **in order** starting each problem on a new page, ideally. If not, **very clearly** and prominently indicate which problem is answered where!
- Include at the start of the document the **ugrad.cs.ubc.ca e-mail addresses** of each member of your team. (Please do **NOT** include your name on the assignment, however.[1])
- Include at the start of the document the statement: "All group members have read and followed the guidelines for academic conduct in CPSC 320. As part of those rules, when collaborating with anyone outside my group, (1) I and my collaborators took no record but names (and GradeScope information) away, and (2) after a suitable break, my group created the assignment I am submitting without help from anyone other than the course staff." (Go read those guidelines!)
- Include at the start of the document your outside-group collaborators' ugrad.cs.ubc.ca IDs, but **not** their names. (Be sure to get those IDs when you collaborate!)

1 Looping Back to Asymptotic Analysis

Before we begin, a few notes on pseudocode throughout CPSC 320: Your pseudocode need not compile in any language, but it must communicate your algorithm clearly, concisely, correctly, and without irrelevant detail. Reasonable use of plain English is fine in such pseudocode. You should envision your audience as a capable CPSC 320 student unfamiliar with the problem you are solving. If you choose to use actual code, note that you may **neither** include what we consider to be irrelevant detail **nor** assume that we understand the particular language you chose. (So, for example, do not write `#include <iostream>` at the start of your pseudocode, and avoid idiosyncratic features of your language like Java's ternary (question-mark-colon) operator.)

[1] If you don't mind private information being stored outside Canada and want an extra double-check on your identity, include your student number rather than your name.

In this problem, if you need to work with indexes, assume 0-based indexing, i.e., the first element of an array A of length n is $A[0]$ while the last is $A[n-1]$.

Answer each of the following:

1. What common algorithm would you use to find "a given target value" in an unordered array?
2. What common algorithm would you use to find "a given target value" in a sorted array?
3. **NOT GRADED** (just for practice): Assume we are totalling the values in the array (to find the average) and that the values are integers. Assume that it takes $\Theta(\lg x)$ bits to represent an integer x and that the largest value in the array m is big, i.e., $n \leq m$. Give a good asymptotic big-O bound on the number of bits needed to represent the total. *Show your work.*
4. Complete the pseudocode function below for an algorithm that runs in worst-case linear time and finds "a pair of values summing to a given target value" in a sorted array. You may assume as a precondition that such a pair must exist in the array.

   READ THE NEXT TWO PARTS and ensure your algorithm will work correctly with them.

   ```
   // Given a sorted array A containing at least one pair of values
   // v1 and v2 such that v1 + v2 = target, returns such a pair (v1, v2).
   FindPair(A, target):
   ```

5. Describe how if given a length n, you could produce an input array `A` of that length and a target value `target` such that your `FindPair` algorithm will run in constant time, independent of n.
6. **NOT GRADED** (just for practice): Describe how if given a length n, you could produce an input array `A` of that length and a target value `target` such that your `FindPair` algorithm will run $\Theta(n)$ time, i.e., describe worst-case input to your algorithm.
7. Complete the pseudocode function below for an algorithm that runs in worst-case linear time and finds "the three smallest values" in an unordered array (of length ≥ 3). Your code **must** be trivial to modify to find the 4, 5, 6, or any other constant number c smallest values in the array.

   ```
   // Return the three smallest values in A as (v1, v2, v3), where v1
   // is the smallest, v2 the 2nd smallest, and v3 the 3rd smallest.
   FindSmallestThree(A):
   ```

8. **NOT GRADED** (just for practice): Describe briefly in English a worst-case linear-time algorithm to determine whether any value is repeated in a sorted array.
9. Briefly justify why no algorithm for finding the smallest three elements in an unordered array can perform better in the case where the input happens to be (but is not known to be) sorted.

2 All Tied Up

Reminders: STP is SMP except where ties are allowed in preference lists.

A *strong instability* in a perfect matching consists of a woman w and a man m such that w and m both (strictly) prefer each other to their current partner.

A *weak instability* in STP is one of:

- a strong instability (i.e., every strong instability is also a weak instability),

- a woman w and man m such that w prefers m to her partner and m is indifferent between w and his partner, or
- a man m and woman w such that m prefers w to his partner and w is indifferent between m and her partner.

Now, solve these problems:

1. Give a good reduction from STP to SMP.

   In particular, you should identify an algorithm `TransformInstance` that takes an instance of SMP and transforms it into an instance of SMP, an algorithm `TransformSolution` that transforms the solution of such an SMP instance back into a solution to the original STP instance. (You may assume `TransformSolution` has access to the original STP instance and any data it needs from `TransformInstance`.)

   You're not supplying an algorithm to solve SMP. That's not part of a reduction from STP to SMP!

   BE SURE that your solution works for the next part.

2. Prove that your reduction (combined with an arbitrary algorithm—which is **not** necessarily Gale-Shapley—that produces stable SMP solutions) produces STP solutions with no strong instabilities.

   As is often the case, you will likely want to approach the proof via the contrapositive: a strong instability in the STP solution implies an instability in the SMP solution.

3. **NOT GRADED** (just for practice): Describe a way to create an STP instance of an arbitrary size n that has $n!$ solutions without strong instabilities.

4. **NOT GRADED** (just for practice): Give a **small** instance I of STP and a solution P to I in which: every woman is indifferent between m_1 and m_2; m_1 ranks w_1 first (tied with no one); m_2 ranks w_1 last (tied with no one); P has no strong instabilities; and m_2 marries w_1 in P.

5. Give a **small** instance I of STP for which every valid solution has a weak instability. **Briefly** explain why every valid solution has a weak instability.

6. We might try to modify the Gale-Shapley algorithm to alternate proposals between men and women. Give a **small** instance I of SMP (**not** STP), trace a run of this modified Gale-Shapley algorithm on it up to its solution, and identify an instability in that solution.

3 To Re Mi Pa So Ti La Do!

Recall that a solution to a problem is **not** "strong Pareto optimal" if a single step change to that solution can make at least one person better off while making *no one* worse off.

A solution to a problem is **not** "weak Pareto optimal" if a single step change to that solution can make **everyone** better off.

We decided that a "single step" in SMP is to take any two married pairs (m, w) and (m', w') and swap them to get the two married pairs (m, w') and (m', w). "Better off" and "worse off" are defined naturally in terms of the preference lists.

You may find it useful to say that a "good" step that makes a solution not Pareto optimal is a "Pareto inefficiency". A "strong Pareto inefficiency" is a single step that shows that the solution is not "strong Pareto optimal". A "weak Pareto inefficiency" is such a step for "weak Pareto optimality".

(Reminder: we're talking in this problem about standard SMP, not SMP with ties, unequal numbers of men and women, or any other previously discussed variant.)

Solve these problems under those definitions:

1. Consider the following scenario and statement. Indicate whether the statement is **always**, **sometimes**, or **never** true in any situation matching the scenario. Then, justify your answer as follows:

   - Justify an **ALWAYS** answer by giving a small instance that fits the scenario for which the statement is true and then briefly sketching the key points in a proof that the statement is true for all instances that fit the scenario.
   - Justify a **NEVER** answer by giving a small instance that fits the scenario for which the statement is **false** and then briefly sketching the key points in a proof that the statement is **false** for all instances that fit the scenario.
   - Justify a **SOMETIMES** answer by giving **two** small instances that fit the scenario: one for which the statement is true and one for which the statement is false.

   Scenario: P is a valid solution to SMP instance I in which four (different) people are in two married couples as follows $(m, w), (m', w')$.

   Statement: A single step that swaps (m, w) and (m', w') makes some but not all of the four people better off while making none of the four worse off.

2. Prove that any solution to SMP that is stable is also strong Pareto optimal. (You may want to use the contrapositive!)

3. **NOT GRADED** (just for practice): Give a **small** instance of SMP and a valid (but not necessarily stable) solution to that instance that is **not** weak Pareto optimal.

4. Briefly explain why for every instance of SMP with $n \geq 3$, any valid solution is also weak Pareto optimal.

4 Knowing Your Structures

1. Complete the following algorithm with pseudocode to efficiently find the successor key value of a key value present in a BST (not necessarily an AVL) that does have a successor in the tree, given the key value of the node and the root of the tree.

```
// precondition: key is present in the tree rooted at root,
//               and key has a successor also present in the tree
FindSuccessor(key, root):
```

   It will likely be helpful to also complete the following algorithm:

```
// precondition: the tree rooted at root is non-empty
// postcondition: produces the key with minimum value in the tree
FindMin(root):
```

2. **NOT GRADED** (just for practice): The same code runs correctly on both a BST and an AVL to find the successor. Briefly explain why it takes $O(\lg n)$ time in the worst case on an AVL tree but not on a general BST.

3. **NOT GRADED** (just for practice): What key values present in the tree could you use for k_1 and k_2 to get worst-case performance from an efficient algorithm that counts the number of keys between k_1 and k_2 in a balanced BST?

4. **NOT GRADED** (just for practice): Propose an additional variable (beyond the tree's size) that would be useful to characterize the asymptotic performance of an efficient algorithm that counts the number of keys between k_1 and k_2 in a balanced BST. **Briefly justify** your proposal.

5. Here is pseudocode to check if a binary tree is a maxHeap. (That is, it has the heap order property. It need not have the heap shape property.) The pseudocode only compares a given node's key against its direct children's keys, not against all descendents' keys. Either (a) clearly but concisely prove this pseudocode is correct or (b) give and explain a counterexample to its correctness.

```
// root is the root of a binary tree that may be empty
IsMaxHeap(root):
    if root is empty:
        return TRUE
    else:
        if root.left is not empty and root.left.key > root.key:
            return FALSE
        if root.right is not empty and root.right.key > root.key:
            return FALSE
        return IsMaxHeap(root.left) and IsMaxHeap(root.right)
```

6. Give the runtime of an asymptotically efficient algorithm to find the first 150 keys in an AVL tree (i.e., the 150 smallest keys) and **briefly justify** your answer.

5 Choosing Your Structures

Reminder: We state below the data structure that most efficiently supports a solution to several problems of the options: **array**, **stack**, **queue**, **priority queue** (implemented as a binary heap), **balanced BST** (balanced binary search tree implemented as an AVL tree) and **dictionary** (dictionary/map implemented as a hash table).

1. **NOT GRADED** (just for practice): Determine the next collision that will happen in a game where balls are bouncing around on a pool table. You may assume that a function to determine when two balls will collide (assuming they do not change direction beforehand) has already been written.

   Priority queue

2. Given a map of a cave (represented as a graph), a starting location (a vertex), and a magic spell that will affect all locations within a given distance from the starting location, determine which locations will be affected by the spell. Note that spells do not penetrate walls.

   Queue

3. Given the same cave map inputs as above, and a set of exits (vertices), find the path containing the least number of dangerous creatures from the start to an exit.

   Priority queue

4. **NOT GRADED** (just for practice): You are building an interpreter for a version of the C language that does not have pointers, and you need to keep track (by name) of the current value of each global and local variable.

   Dictionary

5. **NOT GRADED** (just for practice): You are a teaching assistant who is maintaining the current projected final grades of the students in the course. The projections are updated every time a new assigment, quiz or exam grades becomes known (whether a single updated or a group of simultaneous ones), and you want to be able to return efficiently a list of all students whose projected final grade lies within a given range (this range is not fixed, but varies with every query).

   Balanced binary search tree

6. Given a string containing only lowercase alphabetic characters (a–z), determine which character occurs the most often in the string.

   Array

7. **NOT GRADED** ("spoiled" below): Given a mathematical expression, determine if it is parenthesized properly. For example, "(((2+3)*5))" is parenthesized properly, but "(2+4)) + (5" is not.

   Stack

Now, justify the answers briefly. Specifically: Give a high-level description of an algorithm to solve the problem (pseudocode is unnecessary for the level of detail we want) that includes an explanation of how and why the correct data structure fits into the algorithm.

For example, on the seventh problem (which you should not submit), we might say:

> Scan the string, pushing opening parentheses onto the stack. At each closing parenthesis, ensure the stack is non-empty (or the string was unbalanced) and pop. If the stack is empty at the end, the string was balanced. The stack ensures we always compare the most recently encountered, unbalanced opening parenthesis against the next closing parenthesis.

Side note: you could solve this problem with just a counter (increment on open, decrement on close, fail if the counter is ever negative, fail if the counter ends up positive... essentially tracking only the size of the stack). However, a wide variety of slight variations on the problem (e.g., having two different types of brackets) makes a stack appropriate.

6 Tangling the Knot

Reminder: In the mentor/mentee problem (MMP), an instance is a set of (at least two) people, each with two preference lists over everyone but themselves: one ranking others as potential mentors, one ranking them as potential mentees. A solution should make mentor/mentee pairs so that each person has exactly one mentor and exactly one mentee.

1. A friend proposes solving MMP via the following reduction, assuming each employee has a unique employee ID:

   > Create an SMP instance as follows: Let the set of men M be the first half of the employees by ID. Let the set of women W be the second half of the employees by ID. Let the preference lists of men match the first half of employees' mentee preferences over the second half of employees. Let the preference lists of women match the second half of employees' mentor preferences over the first half of employees.
   >
   > Given a solution to the SMP instance, create the solution to MMP as follows: For a pair (m, w), let the "first-half" employee corresponding to m be a mentor to the "second-half" employee corresponding to w (the mentee).

   Give a small, **complete** instance of MMP such that this reduction fails, and briefly explain how and why it fails.

2. **NOT GRADED** (just for practice): Follow these steps to establish that this reduction can produce an MMP solution in which some employee has no mentor:

   (a) Give a small, **complete** instance of MMP such that this reduction succeeds.

   (b) Write out the SMP instance produced by the reduction.

   (c) Write out a stable solution to this SMP instance (arrived it however you like).

   (d) Write out the MMP solution produced by the reduction given this stable solution to the SMP instance.

   (e) Identifiy an employee who received no mentor in the MMP solution.

3. **NOT GRADED** (just for practice): Prove that in any valid solution to an instance of MMP, there is a path going from mentor to mentee in P (i.e., from a person to the person who is their mentee and then optionally continuing from that person to their mentee and so on) that is a cycle (starts and ends on the same person). (Hint: the Pigeonhole Principle can help you make short work of this proof, but be sure you establish specifically that some path starts and ends at the same person, not just that the path *contains* a cycle.)

4. A friend proposes solving MMP via this alternate reduction, slightly different from the one you saw on the quiz:

   > Create an SMP instance as follows: Let the set of men M be the set of employees. Let the set of women W be a second copy of the set of employees. Let the preference list of each man m_e (a copy of employee e) match the **mentee** preference list of e but with e themselves (as the copy w_e of e) added to the end of the list. Similarly let the preference lists of women match the employees' **mentor** preferences but with each employee added to the end of their own preference list.
   >
   > Given a solution to the SMP instance, create the solution to MMP as follows: For a pair (m, w), let the employee corresponding to m be a mentor to the employee corresponding to w (the mentee).

   Now, follow these steps to establish that this reduction can produce an MMP solution in which some employee is their own mentor:

   (a) Give a small, **complete** instance of MMP such that this reduction succeeds.

   (b) Write out the SMP instance produced by the reduction.

   (c) Write out a stable solution to this SMP instance (arrived it however you like).

   (d) Write out the MMP solution produced by the reduction given this stable solution to the SMP instance.

   (e) Identifiy an employee who received themselves as mentor in the MMP solution.

5. Prove that using the reduction from the previous problem, no two employees can both be assigned themselves as mentors. (You may not **assume** that in SMP no two people can both receive their last choices on their preference lists, although this is true. You can prove it and then use the fact if you like, but there's an easier and shorter proof!)

6. For one **COURSE BONUS POINT**, write a **short**, **clear**, **correct** proof that no two women can both receive their least-preferred partner in any stable solution to any SMP instance.

   WARNING: We'll be rapid and harsh in grading these (e.g., if an answer is too long for our liking, we won't read it; if an answer makes an irrelevant or false statement, we won't read further), and it doesn't affect your assignment grade. So, skip it unless you have time and motivation on your hands.