CPSC 320 2018W1: Assignment 1

September 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 **Saturday 22 September 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!)

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.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
For license purposes, the author is the University of British Columbia.

1 Looping Back to Asymptotic Analysis

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 determine if no one was born in 1980?
2. Complete the pseudocode function below for an algorithm that runs in worst-case linear time and returns the number of dates of birth that are repeated (appear more than once) in a sorted array:

```
// Given a sorted array A containing dates of birth, return the number of
// dates of birth that appear more than once
CountRepeatedBirthDates(A):
```

3. Suppose that instead of having each array element be a birth date, the **positions** in (i.e., the indexes to) the array are the dates, and that an array element is a possibly empty list of famous people born on that day (you can assume that the length of such a list is available in constant time). Show how, after constructing another array of the same length in $O(n)$ time, you can then determine how many people were born between date x and date y in $O(1)$ time.

2 Knowing Your Structures

1. Describe an algorithm with $O(n \lg n)$ worst case running time to insert the elements 1 to n in a binary search tree that is not self-balancing (that is, no special care is taken by the `insert` and `delete` operations to keep the tree balanced).
2. Let T be a binary search tree with n keys. Counting the number of keys x of T such that $k_1 < x < k_2$ for some given parameters k_1 and k_2 requires $\Omega(n)$ time in the worst case, even if T is balanced (like an AVL tree). Suppose that we store in each node N of the tree the size of the subtree rooted at N. Precisely describe an algorithm that uses this information to count in $O(\log n)$ time the number of keys x of T such that $k_1 < x < k_2$ for some given parameters k_1 and k_2. Hint: how would you find the number of elements of T that are smaller than k_1?
3. If we were to store in each node N of the tree from question 2 the number of elements of T that are smaller than the key stored at N, we could find even more easily the number of keys x of T such that $k_1 < x < k_2$ for some given parameters k_1 and k_2. Explain why this is **not** a good solution (the reasons are not directly related to the counting problem from question 2).

3 Incomplete ranking

The SMPI (the Stable Matching Problem with Incomplete Lists) was defined in Quiz 1. SMPI is a natural variant of SMP in which there are n hospitals and n residents but preference lists may be incomplete—that is, hospitals need not rank all residents, and residents need not rank all hospitals. An hospital cannot be matched to a resident that is not on its preference list, nor can a resident be matched with a hospital not on the resident's preference list. Like on the quiz, **we assume that each hospital has exactly one slot available**.

An instance of SMPI is a tuple (n, P_H, P_R) where

- n is the number of hospitals and also the number residents,

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
For license purposes, the author is the University of British Columbia.

- P_H is the set of preference lists of the hospitals: for each hospital h, $P_H[h]$ is a list that ranks a subset of the residents, and
- P_R is the set of preference lists of the residents: for each resident r, $P_R[r]$ is a list that ranks a subset of the hospitals.

Let's define a **valid solution** of SMPI to be a matching S such that for each pair (h, r) in S, r is in the list $P_H[h]$ and h is in the list $P_R[r]$.

1. Suppose that each preference list of an hospital or resident has size just 2. In this case, does the number of valid solutions scale polynomially or exponentially with n in the worst case? (Don't forget to justify your answer.)
2. Describe how to construct an instance with n hospitals and residents, for any n, for which each preference list of an hospital or resident has size 2 but there is only one valid solution, namely the empty matching.
3. Given an instance I of SMPI, a pre-processing step of an algorithm could be to repeat the following until neither of the rules apply:

   Rule 1 If for some pair h and r, h is on r's list but r is not on h's list, then remove h from r's list

   Rule 2 If for some pair h and r, r is on h's list but h is not on r's list, then remove r from h's list

   Let I' be the instance after this pre-processing step. Fill in the circle with the best answer for the following statements, where I is some instance of SMPI and I' is the result of the pre-processing step applied to I.

   The set of stable matchings of I and I' are identical.

   ○ Yes ○ No

   The set of valid matchings of I and I' are identical.

   ○ Yes ○ No
4. Generalize the Gale-Shapley Algorithm so that it solves SMPI.
5. Explain briefly why your algorithm always terminates and outputs a stable matching.
6. Show that for any instance of SMPI, all stable matchings have the same size and involve exactly the same hospitals and exactly the same residents (note that they may however be matched differently).

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
For license purposes, the author is the University of British Columbia.