CPSC 320 2018W1: Assignment 3

October 18, 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 October 27, 2018 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! When uploading assignments to gradescope, marks will be docked if pages are not properly assigned to each question.
- 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.

1 Hi Ho, hi ho, we are digging the road

Recall that the city of Vancouver needs to dig up many roads during the summer in order to replace ageing water pipes. There are pipes along n road segments that need to be replaced in a given summer. The process of replacing a pipe consists in

- first digging a hole
- then performing the replacement
- and finally filling in the hole and repaving

We will assume all holes have been dug by the beginning of the summer. The third part of the job can be done by any one of a large number of teams, and can be performed fully in parallel. However the task of actually replacing the pipes requires a high degree of specialization, and hence there is only one team in the city that is capable of performing it.

Let us call the road segments $R_1, \ldots R_n$, and suppose the job on road R_i requires s_i hours of time for the first task, and t_i hours of time for the second task. Since there is only one team capable of performing the replacement of the old water pipes, city managers need to work out an order in which this team will replace the pipes on the n road segments. As soon as the replacement on one road segment is complete, a team in charge of refilling the hole and repaving the road can be called in.

Let us say that a *schedule* is an ordering of the jobs for the pipe replacement team, and the *completion time* of the schedule is the earliest time at which all pipes have been replaced and all roads restored to their initial condition (or better). This is an important quantity to minimize, since users tend to get upset when roads have large holes in them.

In quiz 3, you saw that we can obtain an optimal schedule by ordering the jobs by decreasing t_i value (that is, starting with the job whose t_i value is the largest, etc). To simplify notation, let us assume that R_i is the job with the i^{th} largest t_i value. That is, assume that $t_1 \geq t_2 \geq \cdots \geq t_{n-1} \geq t_n$.

1. Prove that there is an optimal schedule whose first job **is** job R_1. Hint: consider what happens if you have an optimal schedule whose first job is not R_1, and then you swap two (carefully chosen) jobs.
2. Using your result from question 1 and mathematical induction, prove that ordering the job by decreasing t_i value is guaranteed to return an optimal solution.

2 Lowest common ancestor

Recall that in a tree rooted at a node r, a node v is an *ancestor* of node i (where v and i may be equal) if v is on the path from r to i. Node v is a *common ancestor* of nodes i and j if v is an ancestor of i and v is also an ancestor of j. The *lowest common ancestor* of i and j is the common ancestor of i and j with the largest depth. Biologists are interested in finding the lowest common ancestor of pairs of nodes in trees that model evolutionary relationships among organisms.

On the quiz, you derived a $\Theta(n)$ algorithm that takes two node i and j of a tree, and finds their lowest common ancestor. Unfortunately, calling the `Lowest-Common-Ancestor` algorithm repeatedly on different pairs of nodes will quickly become inefficient, because a lot of the work of traversing the tree is repeated each time.

It is possible to pre-process the tree so that, after $\Theta(n)$ time pre-processing to build a data structure that uses $\Theta(n)$ space, we can find the lowest common ancestor of two given nodes i and j in $\Theta(d)$ time in the worst case, where d is the depth of the tree.

1. Describe the data structure you will build, and show why it can be constructed in $\Theta(n)$ time. Hint: parent pointers may be helpful, but not sufficient.
2. Now rewrite the `Lowest-Common-Ancestor` algorithm to run in $\Theta(d)$ time in the worst case, given the pre-processed tree data structure..

3 Hamming distance

Let x and y be binary strings of length n. The *Hamming Distance* between x and y, denoted by $H(x, y)$, is the number of positions where x and y disagree. That is, if $x = x_1x_2 \dots x_n$ and $y = y_1y_2 \dots y_n$ then $H(x, y)$ is the number of positions i for which $x_i \neq y_i$. Sets of strings that have high Hamming distance from each other are useful as a means for information encoding in a situation where bits can get corrupted.

A S set of binary strings of length n is an (n, d) *Hamming set* if the Hamming distance between any pair of strings in S is at least d.

1. Describe how to construct a $(n, 2)$ Hamming set of size $\Omega(4^{n/3})$. (Hint: there is a construction that is similar to, but better than, that given in the quiz solution.)

2. For $0 \leq i \leq 2^n - 1$, let s_i be the binary string of length n which encodes the nonnegative number i in binary. For example, if $n = 4$ then $s_0 = 0000$, $s_1 = 0001$, $s_2 = 0010$, and so on.

   Let Set(n, i, d) be the set of i' in the range 0 to $2^n - 1$ such that $H(s_i, s_{i'}) < d$. Explain why the size of Set(n, i, d) is independent of i.

3. The following variant of Algorithm Hamming-Set from the quiz is essentially the same as an algorithm described in a 1997 U.S. patent by biologist and Nobel Prize winner Sydney Brenner (patent number 5,604,097, "Methods for Sorting Polynucleotides Using Oligonucleotide Tags").

```
**Algorithm** Brenner-Hamming-Set($n$,$d$)

    Initialize $M$ to be a list of the $2^n$ binary strings of length n (in arbitrary order)
    $i = 1$
    Repeat
        Set $M$-old to be equal to the list $M$
        Let $s$ be the $i$th string in $M$
        For each string $s'$ occurring after string $s$ in $M$
            If $H(s, s') < d$ then remove $s'$ from $M$
        $i$++
        Until ($M$-old = $M$) or ($i$ is greater than the size of $M$)
        Return $M$
```

   Describe an input (n, d) and an initial list M on which Algorithm Brenner-Hamming-Set(n,d) produces an output that is *not* a (n, d)-Hamming set. Show what the output is. (Brenner would have benefited from having a CPSC 320 student in his lab!).

4 Pell numbers

The infinite sequence of rational approximations to the square root of 2 starts with the numbers 1/1, 3/2, 7/5, 17/12, and 41/29. The infinite sequence of denominators of this sequence is called the Pell sequence, so the Pell number sequence begins with 1, 2, 5, 12, and 29. The Pell numbers are also defined by the following recurrence relation:

$$P_n = \begin{cases} 0, & if n = 0, \\ 1, & if n = 1, \\ 2P_{n-1} + P_{n-2}, & \text{otherwise.} \end{cases}$$

1. Use induction to show that

$$P_n = \frac{(1+\sqrt{2})^n - (1-\sqrt{2})^n}{2\sqrt{2}}.$$

2. Write a recursive algorithm Pell(n) that returns the n th Pell number.
3. Write a recurrence relation that describes the running time of your algorithm as a function of n.
4. Describe the solution to the recurrence from part (3) in terms of another sequence of numbers you are almost certainly already familiar with (a closed form formula for the n^{th} term of that sequence is easily found online).