CMPSCI 311: Introduction to Algorithms **Spring 2017**

Homework 1 Solutions

You make work in groups, but you must write solutions yourself. List collaborators on your submission.

If you are asked to design an algorithm, please provide: (a) the pseudocode for the algorithm, (b) an explanation of the intuition for the algorithm, (c) a proof of correctness, (d) the running time of your algorithm and (e) justification for your running time analysis.

Submission instructions. This assignment is due by 8:00pm on Friday, Feb 10 in Gradescope (as a pdf file). Please review this updated course policy about Gradescope submissions:

Gradescope. You may type up your answers in LaTeX and compile to pdf (preferred) or hand-write your homework and scan to pdf. If scanned, the pdf must be rotated correctly, scanned at high quality, and readable at a standard letter size. If the problem set is not submitted in the correct format, you will not receive credit. Unreadable work, scratching out, etc. will not be graded. You should view the pdf once you submit to ensure it meets these minimum standards. Review the gradescope submission workflow in order to make sure you upload and annotate your problem set correctly `https://youtu.be/-wemznvGPfg` and `http://goo.gl/uwM1N2`.

1. **(15 points) Stable Matching Running Time.** In class, we saw that the Propose-and-reject algorithm terminates in at most n^2 iterations, when there are n students and n colleges.

   (a) Construct an instance so that the algorithm requires only $O(n)$ iterations, and prove this fact. Your construction should be possible for all values of n.

   (b) Construct an instance so that the algorithm requires $\Omega(n^2)$ iterations (that is, it requires at least cn^2 iterations for some constant $0 < c \leq 1$), and prove this fact. Your construction should be possible for all values of n.

   Solution:

   (a) Construct an instance where all preferences are 100% compatible with each other. Each college c has a first choice student s whom also ranks c first on their preference list. In each iteration of the first n iterations, a college c will propose to their first choice student s, that student will accept, and then (c, s) will remain matched. Thus all colleges are matched after n iterations. In particular, a set of n colleges, $\{c_1, ..., c_n\}$, and students, $\{s_1, ..., s_n\}$, where c_i has preference $s_i > s_{i+1} > ... > s_n > s_1 > ... > s_{i-1}$ and s_i has preference $c_i > c_{i+1} > ... > c_n > c_1 > ... > c_{i-1}$, would follow this scheme and be solved in $O(n)$ iterations.

   (b) Construct an instance of size n with "universal" preferences–all colleges have the same preference list, say $s_1 > s_2 > ... > s_n$. Regardless of student preferences, after running the algorithm each college will be matched to some student. Based on the preference lists, the college that is matched to student s_j will propose to the first j students, for all $j = 1, \dots, n$. Thus, the total number of proposals is $T(n) = 1 + 2 + ... + n = n(n+1)/2 \geq n^2/2 = \Omega(n^2)$, as we argued in class.

   Note: it is also possible to give specific student preferences, say, that all students have the same preference list, say $c_1 > c_2 > ... > c_n$. In this case, one can argue that the unique stable matching is the one where c_i is matched to s_i for all i. However, given the argument above, it is not necessary to explicitly give the student preferences.

2. **(20 points) Stable Matchings: K&T Ch 1, Ex 5.** Consider a version of the stable matching problem where there are n students and n colleges as before. Assume each student ranks the colleges (and vice versa), but now we allow ties in the ranking. In other words, we could have a school that is indifferent two students s_1 and s_2, but prefers either of them over some other student s_3 (and vice versa). We say a student s *prefers* college c_1 to c_2 if c_1 is ranked higher on the s's preference list and c_1 and c_2 are not tied.

   (a) **Strong Instability.** A strong instability in a matching is a student-college pair, each of which prefer each other to their current pairing. In other words, neither is indifferent about the switch. Does there always exist a matching with no strong instability? Either give an example instance for which all matchings have a strong instability (and prove it), or give and analyze an algorithm that is guaranteed to find a matching with no strong instabilities.

   (b) **Weak Instability.** A weak instability in a matching is a student-college pair where one party prefers the other, and the other may be indifferent. Formally, a student s and a college c with pairs c' and s' form a weak instability if either

   - s prefers c to c' and c either prefers s to s' or is indifferent between s and s'.
   - c prefers s to s' and s either prefers c to c' or is indifferent between c and c'.

   Does there always exist a perfect matching with no weak instability? Either give an instance with a weak instability or an algorithm that is guaranteed to find one.

   Solution:

   (a) Yes, there is always a matching with no strong instability. A simple way to find this matching is to design a way to break ties, then run the stable matching algorithm. Consider breaking ties like this: if man m_i is indifferent to women w_i and w_j and $i < j$, then w_i is ranked higher on m_i's preference list. Otherwise w_j is ranked higher. Similarly, if woman w_i is indifferent to men m_i and m_j and $i < j$ then w_i ranks m_i higher, otherwise she ranks m_j higher. This establishes an ordered preference list for every man and woman. Running the stable matching algorithm will produce a stable matching M, which by definition has no instabilities with respect to the fully ordered preference lists. Then, note that a *strong* instability with respect to the weak preferences would imply a *instability* with respect to the fully-ordered preferences, so a strong stability cannot exist in M.

   (b) There doesn't always exist a perfect matching with no weak instability. Consider the case where $n = 2$ and m_1 is indifferent between w_1 and w_2 while both women prefer m_1 to m_2. No matter which woman matches with m_1, there will always be a weak instability since the woman not matched with m_1 is going to prefer m_1 while m_1 is indifferent between both women.

3. **(15 points) Big-O. K&T Ch 2 Ex 4.** Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows $f(n)$, then it should be the case that $f(n)$ is $O(g(n))$.

$$
\begin{aligned}
g_1(n) &= 2^{\sqrt{\log n}} \\
g_2(n) &= 2^n \\
g_3(n) &= n(\log n)^3 \\
g_4(n) &= n^{4/3} \\
g_6(n) &= n^{\log n} \\
g_7(n) &= 2^{n^2}
\end{aligned}
$$

   Solution: We order the functions as follows:

   - g_1 comes before g_3. If we take logarithms, we are comparing $\sqrt{\log n}$ to $\log n + 3\log(\log n) \geq \log n$

- g_3 comes before g_4: Dividing both by n, we are comparing $(\log n)^3$ with $n^{1/3}$, or (taking cube root), $\log n$ with $n^{1/9}$. Now we use the fact that logarithms grow slower than polynomials.
- g_4 comes before g_6: Taking logarithms, we are comparing $4/3$ with $(\log n)^3$. Now we use the fact that constants grow slower than logarithms.
- g_6 comes before g_2: Taking logarithms, we are comparing $(\log n)^2$ with n. Now we use the fact that logarithms grow slower than polynomials.
- g_2 comes before g_7: Taking logarithms, we are comparing n to n^2, and n^2 is the polynomial of larger degree.

$$g_1, g_3, g_4, g_6, g_2, g_7$$

4. **(20 points) Asymptotics. K&T Ch 2, Ex 6.** Given an array A consistent of n integers, you'd like to output a two-dimensional $n \times n$ array B in which $B[i, j] = A[i] + A[i+1] + \ldots + A[j]$ for each $i < j$. For $i \geq j$ the value of $B[i, j]$ can be left as is.

   For $i = 1, 2, \ldots, n$
       For $j = i + 1, \ldots, n$
           Add up $A[i] + A[i+1] + \ldots + A[j]$.
           Store in $B[i, j]$.

   (a) What is the running time of this algorithm as a function of n? Specify a function f such that the running time of the algorithm is $\Theta(f(n))$.

   (b) Design and analyze a faster algorithm for this problem. You should give an algorithm with running $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.

   Solution:

   (a) The given algorithm has run time complexity $= O(n^3)$. The outer for loop executes exact n times and the inner for loop executes at most n times. So, the line of code that performs addition of array is executed n^2 times. Also, the addition of array A takes $O(j - i - 1)$ time which is again at most $O(n)$. Storing the result in array B takes constant time. So, the run time complexity is at most $O(n^2 * n) = O(n^3)$.

   To prove that run time of this algorithm is $\Theta(n^3)$, then we must prove that it is lower bound by cn^3 for some c.

   Consider the case when $i \leq n/4$ and $j \geq 3n/4$. Then the minimum number of addition operations needs to done are $j - i - 1 \geq n/2$. This situation will arise for all pairs when $i \leq n/4$ and when $j \geq 3n/4$ so there will be $(n/4)^2$ pairs and there are at least $n/2$ addition operations for all such pairs. So, this algorithm must perform at least $(n/2) * (n/4)^2 = n^3/32$ operations. Hence it is $\Omega(n^3)$.

   (b) Consider the below algorithm:

   $cur\_sum = 0$
   $for \quad i = 1 \quad to \quad n$
       $cur\_sum = A[i]$
       $for \quad j = 1 + 1 \quad to \quad n$
           $cur\_sum += A[j]$
           $B[i, j] = cur\_sum$

   The outer loop will execute for n times and inner loop will run for at most n-1 times. Addition and storing will take constant time here. So total operations would be:
   $(n - 1) + (n - 2) + .. + 2 + 1 = O(n^2)$

5. **(10 points) DFS and BFS. K&T Ch 3, Ex 5.** Suppose we have a connected graph $G = (V, E)$ and a vertex $u \in V$. If we run DFS from u, we obtain a tree T. Suppose that if we run BFS from u we obtain exactly the same tree T. Prove that $G = T$.

   Solution: Let T be the tree returned by both BFS and DFS. We will show that G is equal to T.

   Suppose for the sake of contradiction that G has an edge (x, y) that does not belong to T. By (3.7), either x is an ancestor of y or vice versa. Assume without loss of generality that x is an ancestor of y. By (3.4), the layers of x and y differ by at most one. Since x is an ancestor of y, this implies that x must be the immediate parent of y in T, or, in other words (x, y) belongs to T. This is a contradiction.

6. **(20 points) Butterfly ID. K&T Ch 3 Ex 4.** Some of your friends have become amateur lepidopterists (they study butterflies). Often when they return from a trip with specimens of butterflies, it is very difficult for them to tell how many distinct species they've caught—thanks to the fact that many species look very similar to one another.

   One day they return with n butterflies, and they believe that each belongs to one of two different species, which well call A and B for purposes of this discussion. They'd like to divide the n specimens into two groups—those that belong to A and those that belong to B—but it's very hard for them to directly label any one specimen. So they decide to adopt the following approach.

   For each pair of specimens i and j, they study them carefully side by side. If theyre confident enough in their judgment, then they label the pair (i, j) either "same" (meaning they believe them both to come from the same species) or "different" (meaning they believe them to come from different species). They also have the option of rendering no judgment on a given pair, in which case we'll call the pair ambiguous.

   So now they have the collection of n specimens, as well as a collection of m judgments (either "same" or "different") for the pairs that were not declared to be ambiguous. They'd like to know if this data is consistent with the idea that each butterfly is from one of species A or B. So more concretely, we'll declare the m judgments to be consistent if it is possible to label each specimen either A or B in such a way that for each pair (i, j) labeled "same," it is the case that i and j have the same label; and for each pair (i, j) labeled "different," it is the case that i and j have different labels. They're in the middle of tediously working out whether their judgments are consistent, when one of them realizes that you probably have an algorithm that would answer this question right away. Give an algorithm with running time $O(m + n)$ that determines whether the m judgments are consistent.

   Solution sketch: The idea is to modify BFS. Assume the graph is connected. If not, the same algorithm can be run on each connected component. Pick a starting node s and an arbitrary label, say A, for the starting node. When running BFS, when a new node v is explored via edge (u, v) (i.e., when node v is put in layer L_{i+1} by node $u \in L_i$), use the label of u and the judgment on edge (u, v) to label v. This process will uniquely label all nodes based on tree edges and the arbitrary choice of label A for s. Then, check all non-tree edges (u, v) to see if the endpoints (u, v) are consistent—either same or different—with the judgment on the edge. If so, report "consistent". If not, report "inconsistent."

   To argue correctness, we argue two cases. First, if the algorithm reports "consistent", then the judgments are consistent because the algorithm has labeled all nodes using the labels A or B and: (1) the labeling process guarantees that the labels are consistent with tree ediges, and (2) the algorithm explicitly checks that the labels are consistent with non-tree edges. Second, if the algorithm reports "inconsistent", the the judgments are inconsistent. This is true because the nodes were labeled in the unique way possible based on only the tree edges (up to switching the labels A and B)—and the resulting labeling is inconsistent on some non-tree edge. There is no other way to change the labels to make all edges consistent.

   The algorithm takes $O(m + n)$ time because it is a modification of BFS, with only constant-time operations added within the execution of BFS, and then $O(m)$ time to check the non-tree edges after BFS completes.

7. **(0 points).** How long did it take you to complete this assignment?