CSCI 2300 — Introduction to Algorithms
Exam 2 Prep and Sample Questions (document version 1.1)

- Exam 2 is scheduled for 5:00-9:59PM EDT on Thursday 4/23
- Exam 2 is open book(s), open notes; given that you are working remotely, you may use any and all of the posted course materials, including all previous questions and answers posted in the Discussion Forum.
- Make-up exams are given only with an official excused absence (`http://bit.ly/rpiabsence`)
- Exam 2 covers everything from after Exam 1 through Thursday 4/16 lecture, which therefore includes DPV sections 2.1-2.4, 5.1-5.2, 5.4, 6.1-6.4, 6.6-6.7, and 7.1-7.2 (i.e., Homeworks 3, 4, 5, and 6, as well as Labs 4, 5, and 6)
- To prepare for the exam, focus on the warm-up and homework/lab problems, as well as the problems discussed and/or proposed in the corresponding lectures slides
- Key topics are greedy algorithms, divide-and-conquer, dynamic programming, linear programming, and network flow
- All work must be your own; do not even think of copying or communicating with others
- Be as concise as you can in your answers; long answers are difficult to grade

- **(v1.1)** Our TAs and mentors have recorded mini-lectures in which they review solutions to these sample problems. Please see the URLs included for each of the problems below.

Sample problems

Work on the problems below as practice problems as you prepare for the exam. **Also work on the "Warm-up problems" and graded problems from the homeworks and labs noted above.**

Feel free to post your solutions in the Discussion Forum (except for graded Homework 6 problems); and reply to posts if you agree or disagree with the proposed approaches/solutions.

Some selected solutions will be posted on Wednesday 4/22.

1. Show that if an undirected graph with n vertices has k connected components then it has at least $n - k$ edges.

   SOLUTION: (v1.1) URL: `https://tinyurl.com/ycp8twyz`

   Intuitively this makes sense, but to prove this, we need to be more specific. Let's try a direct proof.

   For graph $G = (V, E)$, let e_i denote the number of edges and let v_i denote the number of vertices in the ith connected component. We can describe $|E|$ as

   $$|E| = \sum_{i=1}^{k} e_i$$

   Next, since a connected graph with t vertices must have at least $t - 1$ edges, we then have

   $$\sum_{i=1}^{k} e_i \geq \sum_{i=1}^{k} (n_i - 1)$$

   Observe that the rightmost term above is equal to $n - k$.

2. Using a Huffman encoding of n symbols with frequencies $f_1, f_2, \ldots, f_n$, what is the longest a codeword could possibly be?

   SOLUTION: (v1.1) URL: `https://tinyurl.com/ybojlfq9`

   The longest codeword given n symbols is a codeword of length $n - 1$.

   An encoding of n symbols with $n-2$ of them having decreasing probabilities $1/2, 1/4, \ldots, 1/2^{n-2}$ and two of them having probability $1/2^{n-1}$ will achieve this encoding scheme. (You should also try to show a few specific example frequency sets that produce the case described.)

3. Give a linear-time greedy algorithm that takes as input a tree and determines whether it has what is called a *perfect matching*, i.e., a set of edges that touches each vertex exactly once.

   SOLUTION: (v1.1) URL: `https://tinyurl.com/y8tmzwe2`

   First observe that for each leaf of the tree, the edge incident on the leaf must be in the matching. Also, if we remove all of these leaves, the leaves of the resulting tree have already been covered, so the edges incident on them must not be in the matching.

   We can then construct the perfect matching bottom up (i.e., from the leaves), including and excluding edges at alternating steps. If at any step we need to include two edges incident on the same vertex, then we can conclude that no perfect matching exists.

   The above algorithm is a linear time algorithm since we can keep track of the degree of each vertex and identify the leaves (i.e., vertices with degree 1) in a list. At each step, we update the degree of the endpoints of edges that we delete.

4. For a set of n variables $x_1, x_2, \ldots, x_n$, we are given q *equality* constraints of the form $x_i = x_j$ and p *disequality* constraints of the form $x_i \neq x_j$. Let $m = q + p$.

   Is it possible to satisfy all given constraints? More specifically, give an efficient greedy algorithm that models the problem as a graph and takes as input m constraints over n variables, providing as output whether the constraints can be satisfied or not. What is the runtime of your algorithm?

   As an example, the following set of constraints cannot be satisfied:

   $$x_1 = x_2, x_2 = x_3, x_3 = x_4, x_1 \neq x_4$$

   SOLUTION: (v1.1) URL: `https://tinyurl.com/ybqsqkzs`

   We can model this problem using two graphs, $G_{eq} = (V, E_{eq})$ and $G_{neq} = (V, E_{neq})$ where $V = \{x_1, x_2, \ldots, x_n\}$ from the problem definition. In other words, we model each variable as a vertex. Further, we have edge $(i, j) \in E_{eq}$ if $x_i = x_j$ and similarly edge $(i, j) \in E_{neq}$ if $x_i \neq x_j$.

   From this graph, since equality is an equivalence relation, all variables in each connected component of G_{eq} must have the same value. Therefore, the constraints can be satisfied if and only if there is no edge $(i, j) \in E_{neq}$ such that x_i and x_j are in the same connected component in G_{eq}.

   Our algorithm then consists of decomposing both graphs into their connected components and checking for edges from E_{neq} in E_{eq}, all of which can be done in $O(m + n)$ time, which is linear.

5. Given two sorted lists of size m and size n, describe a greedy algorithm that computes the kth smallest element in the union of the two lists. Use a divide-and-conquer approach and make sure your algorithm runs in $O(\log m + \log n)$ time.

   SOLUTION: (v1.1) URL: `https://tinyurl.com/ybcp9add`

6. DPV Problem 2.18

   HINT: As a hint, see the section on pages 52-53 of DPV (**An $n \log n$ lower bound for sorting**) for inspiration, i.e., set the problem up as a binary tree in which a path from the root to a leaf represents a "run" of the algorithm. At each vertex of this tree, a comparison takes place and, according to its result, a new comparison follows until we reach a leaf, which is an output of the algorithm.

   SOLUTION: (v1.1) URL: `https://tinyurl.com/ya4luwuu`

7. DPV Problem 6.7

   SOLUTION: (v1.1) URL: `https://tinyurl.com/y7f3n4gp`

   Subproblems: Define variables $L(i, j)$ for all $1 \leq i \leq j \leq n$ such that each $L(i, j)$ represents the length of the longest palindromic subsequence of given string $x[i, \ldots, j]$.

   Algorithm and Recursion: From the above, the recursion will be

$$L(i,j) = \begin{cases} L(i+1,j) & \\ L(i,j-1) & \\ L(i+1,j-1)+1 & \text{if } x_i = y_j \\ L(i+1,j-1) & \text{if } x_i \neq y_j \end{cases}$$

   The initialization is, for all $1 \leq i \leq n$, we set $L(i, i) = 0$. Further, for all $1 \leq i \leq n-1$, we initialize

$$L(i,i+1) = \begin{cases} 1 & \text{if } x_i = x_i + 1 \\ 0 & \text{if } x_i \neq x_i + 1 \end{cases}$$

   Correctness and Runtime: Consider the longest palindromic subsequence s of $x[i, \ldots, j]$ and focus on the elements x_i and x_j. There are three possible cases here (from top to bottom):

   - If both x_i and x_j are in s then they must be equal and

$$L(i,j) = \begin{cases} L(i+1,j-1)+1 & \text{if } x_i = x_j \\ L(i+1,j-1) & \text{if } x_i \neq x_j \end{cases}$$

   - If x_i is not a part of s then $L(i, j) = L(i + 1, j)$.
   - If x_j is not a part of s then $L(i, j) = L(i, j - 1)$.

   Given the above, the recursion handles all possible cases correctly. And the runtime of this algorithm is $O(n^2)$ since there are $O(n^2)$ subproblems, each of which takes $O(1)$ to evaluate.

8. DPV Problem 6.9

   SOLUTION: (v1.1) URL: `https://tinyurl.com/y7rn8swx`

9. DPV Problem 6.11

   SOLUTION: (v1.1) URL: `https://tinyurl.com/y7qzxudv`

10. DPV Problem 7.5

    HINT: For part (a), use the same approach as that used in the Homework 6 warm-up problems, i.e., DPV Problems 7.1, 7.2, and 7.3.

    SOLUTION: (v1.1) URL: `https://tinyurl.com/yblbxy6b`

11. DPV Problem 7.6

    SOLUTION: (v1.1) URL: `https://tinyurl.com/y8vx4x8a`

12. DPV Problem 7.7

    SOLUTION: (v1.1) URL: `https://tinyurl.com/y8vx4x8a`

    (a) **SOLUTION:** This LP is never infeasible since the origin $(0, 0)$ will satisfy $ax + by \leq 1$ for any choice of a and b.

    (b) **HINT:** Consider what happens if we have $a \leq 0$ or $b \leq 0$, then consider what happens if we have $a > 0$ and $b > 0$.