NAME: ____________________________

CMPSCI 611
Advanced Algorithms
Final Exam Fall 2020 Draft Solutions

A. McGregor

8pm, 12/2/20 to 8pm, 12/3/20

DIRECTIONS:

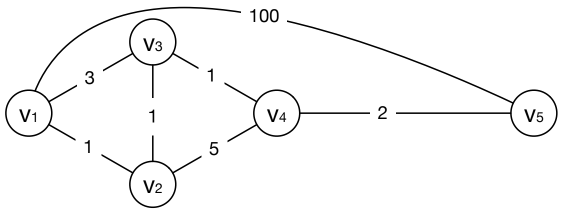
- **Honesty Policy:** No communicating with anyone (except the instructor and TAs via private posts on Piazza) about the exam during the 24 hours the exam is open. You are not allowed to use any resources except from the slides and scribed notes linked in Moodle.
- Answers to Questions 1 and 2 should be entered directly into Gradescope.
- You should type or handwrite the answers to Questions 3, 4, and 5 on separate pages and upload into Gradescope. It should be possible to answer each question using a single page. Most parts of each question can be answered in a few sentences. Remember that the best answers are those that are clear and concise (and of course correct).
- Once you are finished submit your solutions in Gradescope by 8pm Thursday. If you are using a camera to "scan" your answers, please take care to make the picture as legible as possible. If you handwrite, please write as clearly as possible.

1	/10
2	/10
3	/10
4	/10
5	/10
Total	/50

Question 1. For each of the following statements, enter in Gradescope whether they are TRUE or FALSE. No justification is required.

1. Given a list of n integers, it is possible to find the largest element in $O(n)$ time assuming that any two elements can be compared in $O(1)$ time.
   Answer: TRUE.
2. If every capacity in an instance of network flow is an odd integer then the size of the maximum s-t flow is always an odd integer.
   Answer: FALSE.
3. If $E = \{a, b, c\}$ and $\mathcal{I} = \{\{\}, \{a\}, \{c\}\}$ then $(E, \mathcal{I})$ is a subset system.
   Answer: TRUE.
4. If X is random variable then $E[X + X^2] = E[X] + E[X^2]$.
   Answer: TRUE.
5. Every tree with n nodes (assume n is even) has a matching of size $n/2$.
   Answer: FALSE.

Question 2. Enter in Gradescope the numerical answers in these questions. Consider the following graph with weights on the edges.



1. Let $d_{i,j}$ be the length of the shortest path between v_i and v_j What is the value of $d_{1,5}$ in the above graph?
   Answer: 5.
2. In the Floyd-Warshall algorithm, $d_{i,j}^{(k)}$ is the length of the shortest path between v_i and v_j which does not include any nodes in the set $\{v_{k+1}, v_{k+2}, \ldots, v_n\}$ as *intermediate* nodes. n is the number of nodes in the graph. What is the value of $d_{1,5}^{(2)}$ in the above graph.
   Answer: 100.
3. Suppose we compute the shortest paths from v_1 to all other nodes using Dijkstra's algorithm. This algorithm maintains an array D such that for each $i \in [n]$, $D[i]$ refers to the current distance from v_1 to v_i. For the above graph, the algorithm updates this array as follows:

   After Step 1: $D = (0, 1, 3, \infty, 100)$
   After Step 2: $D = (0, 1, 2, 6, 100)$
   After Step 3: $D = (0, 1, 2, ?, 100)$

   What is the missing value in the last line?
   Answer: 3

4. Seidel's Algorithm only works when all edge lengths are 1. However, you could get around this if all edge weights are strictly positive integers: replace every edge $e = (u, v)$ by a path that has end points u and v and $w_e - 1$ intermediate nodes. In the resulting graph, we set all edge weights to 1. If we transformed the above graph as suggested, how many nodes would be in the resulting graph?
   Answer: $5 + 99 + 1 + 0 + 0 + 0 + 4 + 2 = 111$.

5. What is the length of the shortest tour (a path that visits every node exactly once and returns to starting point) in the above graph?
   Answer: 105.

Question 3. In this question, you should assume that $P \neq NP$ and that any linear program can be solved in polynomial time.

1. Let $a(G)$ be the size of the minimum vertex cover of graph G. Does there exist a polynomial time algorithm to compute $a(G)$? Justify your answer.
   Answer: No. If it was, then for any graph G and integer k we would be able to determine in polynomial time whether there was a vertex cover of size at most k: we would just compute $a(G)$ and check whether $a(G) \leq k$. This contradicts the fact that vertex cover is an NP-hard problem and can't be solved in polynomial time if $P \neq NP$.

2. If T is an arbitrary tree with at least three nodes, show that there exists a vertex cover for T of minimum size that does not include any nodes of degree 1.
   Answer: Let U be a minimum vertex cover of T. Let L be the set of leaves of the tree. Removing $U \cap L$ from the vertex cover and adding the nodes adjacent to nodes in $U \cap L$ ensures we still have a vertex cover that is no larger than U and hence is also a minimum vertex cover that contains no leaf nodes.

3. Using the fact in Question 3.2, design a polynomial time algorithm that compute $a(T)$ for any tree T. No need to prove correctness or running time. Describing the algorithm in words (rather than pseudo-code) is fine.
   Answer: Let L be the leaves of T and let $P(L)$ be the neighbors of leaves. Add $P(L)$ to the vertex cover and remove $L \cup P(L)$ (and incident edges) from T. Recurse on the connected components of the remaining graph. If a component has one node, delete it. If a component has two nodes, add one of these nodes to the vertex cover and delete both from the graph.

Given a graph $G = (V, E)$ with nodes $\{v_1, \ldots, v_n\}$, we say the vector $(x_1, x_2, \ldots, x_n)$ is a *fractional vertex cover* of size $\sum_{i=1}^{n} x_i$ if

$$x_i + x_j \geq 1 \text{ for every edge } (v_i, v_j) \in E \qquad \text{and} \qquad x_i \geq 0 \text{ for every node } v_i \in V .$$

Let $b(G)$ be the size of the minimum fractional vertex cover. For example, if G is a triangle then $a(G) = 2$ and $b(G) = 1.5$.

4. Does there exist a polynomial time algorithm to compute $b(G)$ for an arbitrary graph G? Justify your answer.
   Answer: $b(G)$ can be written as the optimum value of a linear program. Hence, it can be computed in polynomial time.

5. Prove that $b(G) \leq a(G)$.

   Answer: For a minimum vertex cover U, consider the vector $(x_1, \ldots, x_n)$ where $x_i = 1$ if $v_i \in U$ and $x_i = 0$ otherwise. Then $(x_1, \ldots, x_n)$ is a fractional vertex cover of size $\sum x_i = |U| = a(G)$. The result follows because $b(G) \leq \sum x_i$.

Question 4. Let S be a set of n different integers. We say $x \in S$ is an *almost-median* if at least a third of the elements in S are strictly less than x and at least a third of the elements in S are strictly greater than x. For example, if $S = \{2, 5, 13, 1, 8, 3, 10, 6, 11\}$ then 5, 6 and 8 are almost-median. Throughout the question you may ignore rounding issues, e.g., you may assume that the sizes of all sets encountered are divisible by 3.

1. How many elements in a set of size n are almost-median?

   Answer: $n/3$.

2. Consider the algorithm FINDAM(S) for finding an almost-median of a set S:

   (a) Pick a random element x from S (each element is equally likely to be picked and note that when we pick an element we *do not* remove it from S)

   (b) Check if x is an almost-median by counting the elements in S strictly less than x.

   (c) If x is an almost-median, output x. If not, go back to Step (a).

   What is the expected running time of this algorithm? Justify your answer.

   Answer: The number of values we need to test is a geometric random variable with parameter 1/3. This has expectation 3. Each attempt requires $O(n)$ time so the total running time is $O(n)$.

3. What is the probability FINDAM takes at least twice as long as expected?

   Answer: $(2/3)^5$. The probability we need to test 6 or more values is equal to the probability that the first 5 value tests are not almost-medians.

4. Consider the following sorting algorithm AM-SORT(S):

   (a) If $|S| \leq 4$, return the list of the elements of S in sorted order.

   (b) $x \leftarrow$ FINDAM(S)

   (c) Compute $L = \{y \in S : y < x\}$ and $H = \{y \in S : y > x\}$

   (d) Return the list that concatenates AM-SORT(L) with x and then AM-SORT(H).

   What is the expected running time of AM-SORT? Justify your answer.

   Answer: The depth of the recursion is at most $\log_{3/2} n$ because $|L| \leq 2|S|/3$ and $|H| \leq 2|S|/3$. Suppose at level i of the recursion, the sets being considered are $S_1, S_2, \ldots$. The expected time to run FINDAM and define the subprogram for all these sets is $O(|S_1| + |S_2| + \ldots) = O(n)$. Hence the total expected running time is $O(n \log n)$.

Question 5. Given graph $G = (V, E)$ and any subset of nodes U, let $\delta(U)$ be the number of edges with exactly one endpoint in U. The *k-constrained max-cut problem* is to find the subset $S \subseteq V$ of *at most* k nodes that maximizes $\delta(S)$. Note the optimum may contain $< k$ nodes.

1. If $k = 10$, design a polynomial time algorithm that solves this problem exactly. Describing the algorithm in words (rather than pseudo-code) is fine. **Hint:** For any $k \leq n/2$, $\binom{n}{0} + \binom{n}{1} \ldots + \binom{n}{k} = O(n^k)$.

   Answer: For each of the $\binom{n}{0} + \ldots + \binom{n}{10} = O(n^{10})$ subsets S of size at most 10, compute $\delta(S)$ and return the subset the gives the max value. Computing $\delta(S)$ takes $O(m)$ time so the total time is $O(n^{10}m)$. Correctness follows since all relevant subsets were considered.

In what follows, we analyze the following approximation algorithm: 1) Let S be the set of k nodes with the highest degree. 2) For every subset $S' \subseteq S$, calculate $\delta(S')$ and return the subset $S^* \subseteq S$ that gives the maximum value.

2. Show that the running time of this algorithm is polynomial in n if $k \leq \log n$.

   Answer: It takes $O(m + n \log n)$ to find S. There are $2^k \leq n$ subsets S' of S to consider and computing $\delta(S')$ for each takes $O(m)$ time. Hence, the total running time is $O(m + n \log n + nm)$ and using the fact $m = O(n^2)$, this is $O(n^3)$.

3. Prove that $\delta(S^*) \geq (d_1 + \ldots + d_k)/4$ where $d_1 \geq d_2 \geq \ldots \geq d_k$ are the k highest degrees. **Hint:** Recall that the max cut of any graph includes at least half the edges. Furthermore, if there are m_1 edges with one endpoint in S and m_2 edges with two endpoints in S then $m_1 + 2m_2 = d_1 + \ldots + d_k$.

   Answer: By the hint, it is possible to partition S into S_1 and S_2 such that there are $\geq m_2/2$ edges split between S_1 and S_2. Since $\delta(V \setminus S) = m_1$, there are at least $m_1/2$ edges between $V \setminus S$ and one of S_1 or S_2. Hence,

$$\delta(S^*) \geq \max(\delta(S_1), \delta(S_2)) \geq \frac{m_1}{2} + \frac{m_2}{2} \geq \frac{m_1}{4} + \frac{m_2}{2} = \frac{d_1 + \ldots + d_k}{4} \ .$$

   .

4. Prove that $\delta(T^*) \leq \delta(S^*) + kd_k$ where T^* is the optimum solution for the k-constrained max-cut problem.

   Answer: Every edge contributing to $\delta(T^*)$ either has exactly one endpoint in $T^* \cap S$ or is incident to $T^* \setminus S$. There are $\delta(T^* \cap S) \leq \delta(S^*)$ edges of the first type and at most $|T^* \setminus S^*| d_k \leq kd_k$ edges of second type.

5. Using the inequalities in part 3 and part 4 (whether or not you proved them), prove that $\delta(T^*) \leq 5\delta(S^*)$, i.e., that the above algorithm is a 5 approximation.

   Answer: $\delta(T^*) \leq \delta(S^*) + kd_k \leq \delta(S^*) + (d_1 + \ldots + d_k) \leq \delta(S^*) + 4\delta(S^*) = 5\delta(S^*)$.[1]

[1] It's possible to show a 4-approximation and it's actually easier. Specifically, $\delta(T^*) \leq d_1 + \ldots + d_k \leq 4\delta(S^*)$.