CMPSCI 311: Introduction to Algorithms
Second Midterm Exam: Solutions

November 17, 2016.

Name: _____ ID: _____

Instructions:

- Answer the questions directly on the exam pages.
- Show all your work for each question. Providing more detail including comments and explanations can help with assignment of partial credit.
- If you need extra space, use the back of a page.
- No books, notes, calculators or other electronic devices are allowed. Any cheating will result in a grade of 0.
- If you have questions during the exam, raise your hand.
- **Hint:** To make sure you get as many points as possible, remember that it is often possible to answer later parts of a question without being able to answer earlier parts of a question.

Question	Value	Points Earned
1	10	
2	10	
3	10+2	
4	10	
5	10	
Total	50+2	

Question 1. (*10 points*) Indicate whether each of the following statements is TRUE or FALSE. No justification required.

1.1 (*2 points*): *In lectures we proved that* $P = NP$.

Answer: FALSE. Determining whether or not $P = NP$ is one of the biggest open problems in the entire field of computer science and mathematics.

1.2 (*2 points*): *In any max flow in any graph, the flow along every edge equals the capacity.*

Answer: FALSE. For example, consider a graph with nodes $\{s, a, b, c, t\}$ with edges

$$(s, a), (s, b), (a, c), (b, c), (c, t)$$

each of capacity one.

1.3 (*2 points*): *Given a graph, the size of the min vertex cover plus the size of the max independent set always equals the number of nodes.*

Answer: TRUE. This follows because if I is an independent set of nodes in a graph $G = (V, E)$ then $V \setminus I$ is a vertex cover.

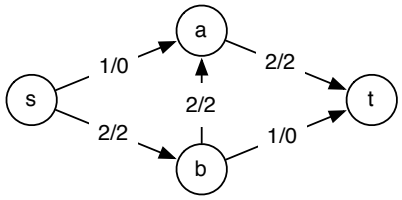
1.4 (*2 points*): *The formula* $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$ *has a satisfying assignment.*

Answer: TRUE. For example set x_1 to TRUE and x_2 to FALSE.

1.5 (*2 points*): *The nodes of a bipartite graph can always be colored with two colors such that endpoints of each edge get different colors.*

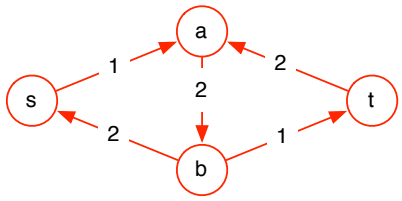
Answer: TRUE. A graph is bipartite iff it is 2-colorable.

Question 2. (*10 points*) In the first two parts of this question we consider the following directed, capacitated graph G where the first number on an edge indicates the capacity of that edge and the second number on an edge indicates the current flow on that edge.



2.1 (*6 points*): *Let* (A, B) *be an* s*-*t *cut where* $A = \{s, a\}$ *and* $B = \{b, t\}$.

1. What is the capacity of the cut? 4
2. What is the flow across the cut? 2
3. What is the flow out of s? 2
4. What is the capacity of the minimum cut? 3
5. Draw the residual graph corresponding to the above flow:



2.2 (*2 points*): *What is the value of the maximum flow from* s *to* t*?*

Answer: 3. This can be achieved by saturating every edge except from the edge (b, a) on which we place flow 1.

2.3 (*2 points*): *Consider the decision version of the max flow problem: given a capacitated graph and a value* k*, determine if there is a flow of size* k*. Is this problem in* NP*? Justify your answer.*

Answer: Yes this problem is in NP since, given a purported flow, it is possible to check in polynomial time whether it has size k and satisfies conservation of flow and the capacity constraints. Another argument is that since the problem P and $P \subseteq NP$ then the problem is also in NP.

Question 3. (*12 points*)

Suppose you want to open some coffee shops along one side of a street. The possible locations are labeled $1, 2, \ldots, n$ in that order and you may open a coffee shop at any subset of these locations subject to the constraint that no two coffee shops are adjacent. For example, if $n = 5$ then the set of locations $\{1, 3, 5\}$ is allowed but $\{1, 2, 4\}$ is not. If you open a coffee shop at location i, you earn profit $p_i > 0$ and if you open coffee shops at a set S of locations, your total profit is $\sum_{i \in S} p_i$. You want to find the set of locations that maximizes your total profit.

3.1 (*2 points*): *Your friend suggests a greedy algorithm: consider locations in order of decreasing profit and add the next location unless you've already added an adjacent location. Give an example with* $n = 3$ *and* p_1, p_2, p_3 *all different where this approach doesn't maximize your total profit.*

$$p_1 = 2 \qquad p_2 = 4 \qquad p_3 = 3$$

Answer: If these were the profits then the greedy algorithm would only open a shop at location 2 whereas it would be more profitable to open shops at locations 1 and 3.

3.2 (*2 points*): *Another friend suggests that considering opening coffee shops at all even locations or opening coffee shops at all odd locations and picking whichever gives most profit will result in maximizing your total profit. Give an example with $n = 4$ where this is not true.*

$$p_1 = 10 \qquad p_2 = 2 \qquad p_3 = 1 \qquad p_4 = 11$$

Answer: If these were the profits then the odd locations would give profit $10 + 1$ and the even locations would give profit $2 + 11$. However it would be better to open locations 1 and 4 and get profit $10 + 11$.

3.3 (*2 points*): *Let $M[i]$ be the max total profit you can earn if you are only allowed to choose locations in the set $\{1, 2, \ldots, i\}$ subject to the constraint that no two are adjacent. If you have already computed $M[1], \ldots, M[i-1]$, how can you directly compute $M[i]$?*

$$M[i] = \max(M[i-1], M[i-2] + p_i)$$

3.4 (*4 points*): *Write pseudo-code for an efficient dynamic programing algorithm that finds the max total profit. What is the running time of your algorithm? You don't need to prove correctness.*

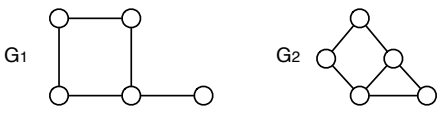
Answer: The running time is $O(n)$ and the pseudo-code is:

1. Set $M[0] = 0$ and $M[1] = p_1$
2. For $i = 2$ to n: $M[i] = \max(M[i-1], M[i-2] + p_i)$
3. Return $M[n]$

3.5 (*2 points*): **Extra Credit:** *How could you use your algorithm if the street was circular and this meant that location 1 and location n were neighboring?*

Answer: If 1 and n are neighboring, we can't pick both locations. So solve the problem restricted to locations $\{1, 2, \ldots, n-1\}$ and restricted to locations $\{2, 3, \ldots, n\}$. Return whichever solution yields the most profit.

Question 4. (*10 points*) In the first part of this question we consider the graphs G_1 and G_2:



4.1 (*4 points*):

What is the size of the maximum matching in G_1*?* 2
What is the size of the largest independent set in G_1*?* 3
What is the size of the maximum matching in G_2*?* 2
What is the size of the largest independent set in G_2*?* 2

4.2 (*2 points*): *For a graph* G *with edges* $e_1, \ldots, e_m$*, let* $L(G)$ *be the graph with* m *nodes* $v_1, \ldots, v_m$ *where there is an edge between a pair of nodes* v_i *and* v_j *whenever the edges* e_i *and* e_j *shared an endpoint. For example,* $L(G_1) = G_2$ *and each edge in* G_1 *corresponds to a node in* G_2*.*

- *Prove that if* G *has a matching of size* k *then* $L(G)$ *has an independent set of size* k*.*

  Answer: *Suppose* $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ *is a matching. Then the nodes* $v_{i_1}, v_{i_2}, \ldots, v_{i_k}$ *are an independent set because no pair of edges among* $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ *shared an endpoint.*

- *Prove that if* $L(G)$ *has an independent set of size* k *then* G *has an matching of size* k*.*

  Answer: *Suppose* $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ *is an independent set. Then no pair of edges in the set* $e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ *share an endpoint (and hence these edges are a matching) because if they did then the corresponding nodes in* I *would have had an edge between them and this would have violated the assumption that* I *was an independent set.*

4.3 (*2 points*): *Given that independent set is NP-complete and maximum matchings can be found in poly-time, is it likely that there's a poly-time reduction from independent set to matching? Explain your answer.*

Answer: It is unlikely. If there was such a reduction then this would imply that independent set could be solved in polynomial time (it is can be reduced in polynomial time to a problem that can be solved in polynomial time). Since independent set is NP complete this would imply that all NP problems can be solved in polynomial time and hence $NP = P$ (which is unlikely.)

4.4 (*2 points*): *Suppose* G *is a bipartite graph with* 100 *edges where every node has degree two. What is the size of the maximum matching in* G*? Justify your answer.*

Answer: 50. Any graph in which every node has degree 2 is a set of node-disjoint cycles. Since G is bipartite then all these cycles have even length and hence exactly half of the edges can be included in a matching.

Question 5. (*10 points*) Here we consider finding the length of the shortest path between all pairs of nodes in an undirected, weighted graph G. For simplicity, assume that the n nodes are labeled $1, 2, \ldots, n$, that the weight w_{ij} of any edge $e = (i, j)$ is positive and that there is an edge between every pair of nodes. In this question, the goal is to solve this via dynamic programming. Note that the algorithm you will develop is *not* the Bellman-Ford algorithm.

5.1 (*2 points*): *Let $D[i, j, k]$ be the length of the shortest path from node i to node j in which the intermediate nodes on the path are all in the set $\{1, 2, \ldots, k\}$. Note that $D[i, i, k] = 0$ for all i and k and that $D[i, j, 0] = w_{ij}$ for all $i \neq j$. A formula for $D[i, j, k]$ with two missing arguments is:*

$$D[i, j, k] = \min\left(D[i, j, k-1], D[i, \mathbf{k}, k-1] + D[k, j, \mathbf{k-1}]\right)$$

when $k \geq 0$. Fill in the two missing arguments.

Answer: The intuition is that the shortest path from i to j that may only use nodes $\{1, 2, \ldots, k\}$ as internal nodes, either goes via node k (in which case it has length $D[i, k, k-1] + D[k, j, k-1]$) or it doesn't go via node k (in which it has length $D[i, j, k-1]$).

5.2 (*4 points*): *Write pseudo-code for an efficient dynamic programming (using the above formula) to compute the shortest path between every pair of nodes.* **Hint:** *You need to specify how each $D[i, j, k]$ is computed, the order in which they are computed, and the final output.*

Answer:

1. For all $1 \leq i, k \leq n$: set $D[i, i, k] = 0$.
2. For all $1 \leq i, j \leq n, i \neq j$: set $D[i, j, 0] = w_{ij}$.
3. For $k = 1$ to n:
   (a) For all $1 \leq i, j \leq n$, $i \neq j$: set $D[i, j, k] = \min\left(D[i, j, k-1], D[i, k, k-1] + D[k, j, k-1]\right)$
4. For all $1 \leq i < j \leq n$: Output $D[i, j, n]$ as the distance between i and j

5.3 (*2 points*): *What is the running time of your algorithm? Justify your answer.*

Answer: $O(n^3)$ since all lines get run $O(n^2)$ times except 3a which gets run $O(n^3)$ times.

5.4 (*2 points*): *What is the number of edge-disjoint paths between node 1 and node 2 in G? Your answer should be a function of n. Recall that G had an undirected edge between every pair of nodes.*

Answer: There are $n - 1$ edge-disjoint paths: e.g., $1 - 2, 1 - 3 - 2, 1 - 4 - 2, \ldots 1 - n - 2$.