Final Exam

- Do not open the exam until you are directed to do so.
- Read **all** the instructions first.
- The exam contains four problems. You have 120 minutes to earn 100 points.
- The exam contains 18 pages, including this one and 4 pages of scratch paper.
- The exam is closed book. You may bring two double-sided sheet of A4 paper to the exam. (You may not bring any magnification equipment!) You may not use a calculator, your mobile phone, or any other electronic device.
- Write your solutions in the space provided. If you need more space, please use the scratch paper at the end of the midterm. Do not put part of the answer to one problem on a page for another problem.
- Read through the problems before starting. Do not spend too much time on any one problem.
- Show your work. Partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- You may use any algorithm given in class without restating it—simply give the name of the algorithm and the running time. If you change the algorithm in any way, however, you must provide complete details.
- Good luck!

Problem #	**Name**	**Possible Points**	**Achieved Points**
1	Drawing Pictures	20	
2	Short Answer	10	
3	Edge-Dominating Set	30	
4	Spacely Sprockets	20	
5	Startups 101	20	
Total:		100	

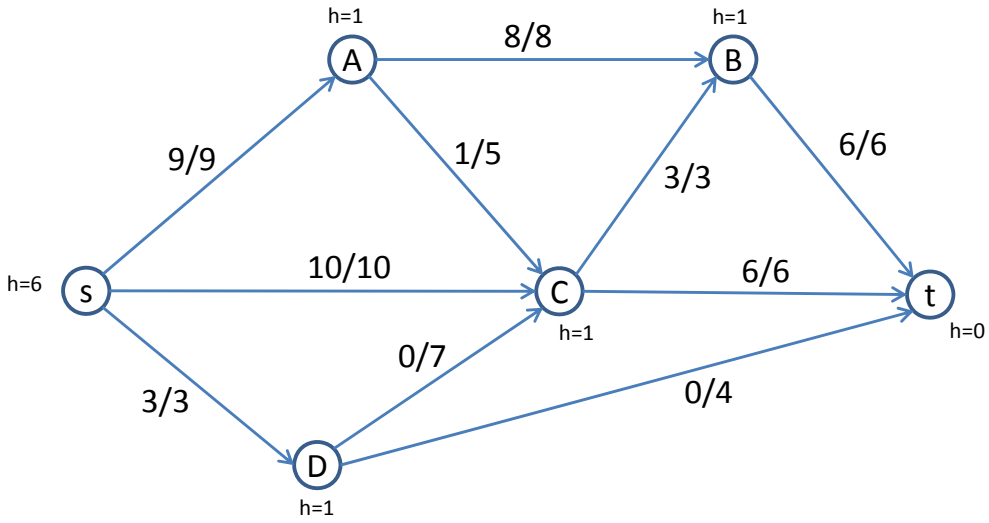
Matric. Num.: _____

Problem 1. (*Drawing pictures*) [20 points]

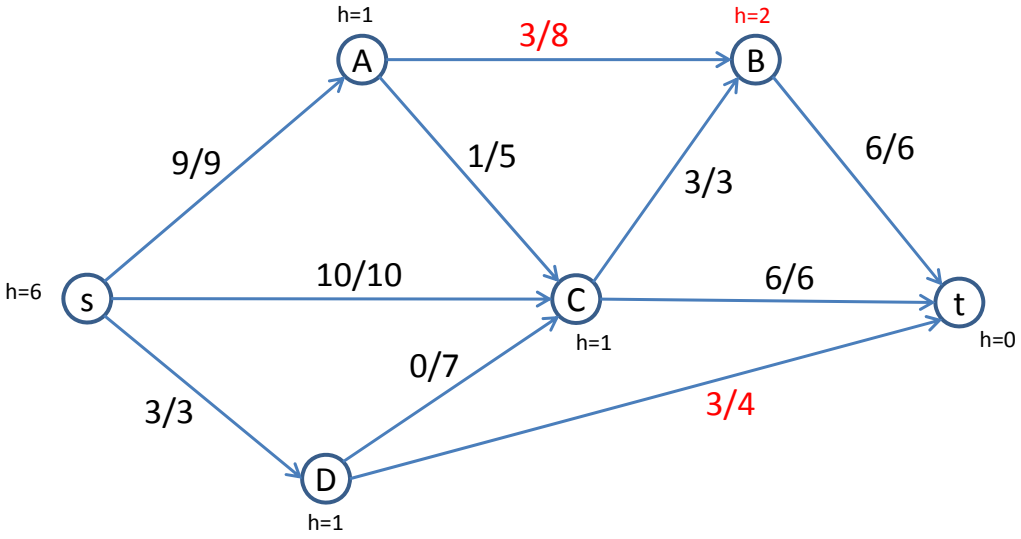
(a) [4 points] Draw a graph that is *not* a planar graph. Give a proof that your graph is not planar.

Solution: A clique is not planar. This can be proved readily in several ways, e.g., Euler's formula says that in a planar graph $v - e + f = 2$. Alternatively, we know that for a planar graph, $m = \Theta(n)$.

(b) [9 points] Execute the Push-Relabel algorithm for three steps on the following graph. (Note that one step consists of either a *push* or a *relabel*). Fill in the table below identifying each step, indicating whether each operation is a relabel, a saturating push, or a non-saturating push. Clearly draw the final state of the graph on the picture, indicating the result of each operation.

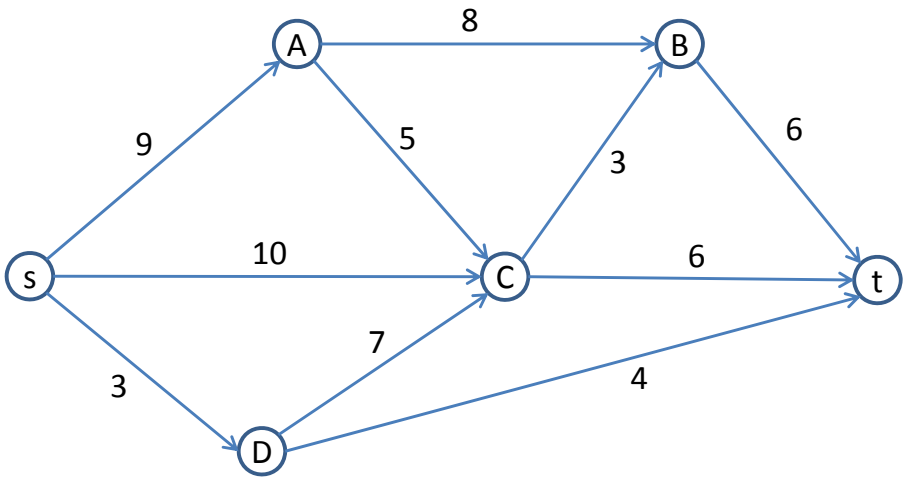


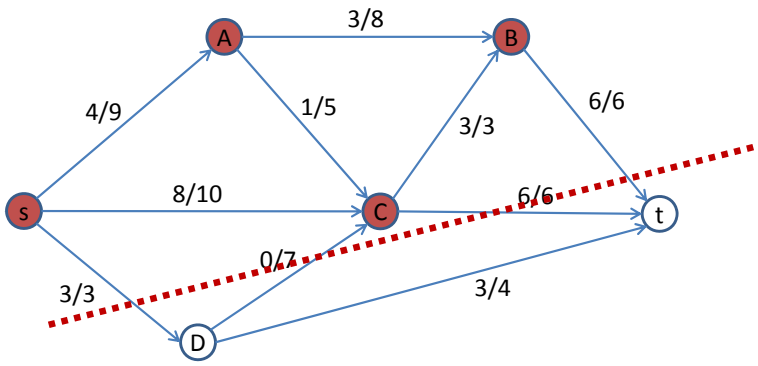
Step	Node/Edge	Type (saturating push/non-saturating push/relabel)
1		
2		
3		



Solution:

(c) [7 points] Identify the minimum cost st-cut in the following graph. Prove that the cut you have identified is the minimum cost cut.





Solution:
The cut is minimal because we have exhibited a flow that has the same capacity as the cut. By MaxFlow/MinCut, then, it is impossible to find a smaller cut.

Problem 2. (*Short Answer*) [10 points]

(a) [5 points] Let G be an arbitrary flow network, and let f be a maximum flow for G.

True or false: there always exists an edge e such that increasing the capacity on e increases the maximum feasible flow in the network. Explain your answer.

Solution: False. Consider a flow network consisting of a path all with the same capacity. There is no one edge that can be changed to increase the feasible flow, since all the edges act as bottlenecks.

(b) [5 points] Let G be an arbitrary weighted, bipartite graph, and let M be a maximum weight matching.

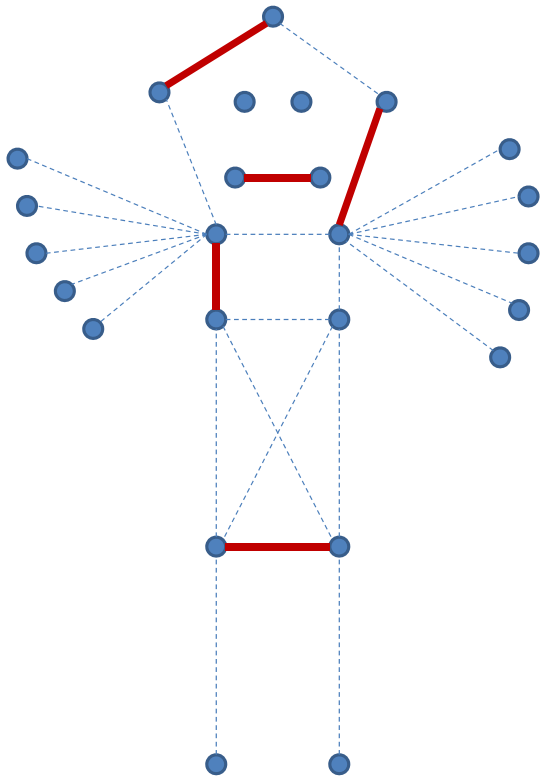
True or false: if every edge in G has its value increased by a constant c, then is M still a maximum weight matching? Explain your answer.

Solution: False. The simplest example involves all negative edges, which yield an empty matching M. Another example is where the augmenting path consists of two unmatched edges of weight 1 and a matched edge of weight 3. By increasing the edge weights by 2, we find the alternate matching to be better.

Problem 3. (*Edge-Dominating Set*) [30 points]

Consider the problem of finding an **Edge-Dominating Set**: Given a connected, unweighted graph $G = (V, E)$, a set of edges $D \subseteq E$ is an *edge-dominating set* if, for every edge $e \in E$, either: (i) $e \in D$, or (ii) some $e' \in D$ where e' and e are both adjacent to a shared vertex v. The goal is to find the minimum sized edge-dominating set.

Here is an example of an edge-dominating set. The thick edges are in the dominating set D, while the dashed edges are dominated:



(a) [6 points] Consider the following attempt to solve the problem of a minimum edge-dominating set by linear programming. For every edge e, let $edges(e)$ be the set of edges adjacent to e. For each edge e, we instantiate a variable x_e which indicates whether or not edge e is in the edge-dominating set.

$$\min \sum_{e \in E} x_e \tag{1}$$

$$\sum_{j \in edges(e) \cup e} x_j \geq 1 \quad \forall e \in E \tag{2}$$

$$x_e(1 - x_e) = 0 \quad \forall e \in E \tag{3}$$

$$x_e \leq 1 \quad \forall e \in E \tag{4}$$

$$x_e \geq 0 \quad \forall e \in E \tag{5}$$

Line (2) is intended to ensure that every edge is dominated by some edge in the dominating set. Line (3) is intended to ensure that each edge is either included or excluded from the dominating set. Lines (4) and (5) ensure that each x_e is between 0 and 1.

Does this strategy work? Either prove that you can use an LP solver to find the minimum edge-dominating set, or explain why you cannot use this strategy.

Solution: This strategy does not work because this is not a linear program.

(b) [12 points] Consider the following heuristic for constructing an edge-dominating set for a connected graph $G = (V, E)$:

a. Choose an arbitrary, fixed root node $r \in V$.

b. Let T be a spanning tree of G rooted at r constructed via depth-first search.

c. If T has depth 1, then add any arbitrary edge to D and return D.

d. Otherwise, let D be the set of edges in T that are *not* adjacent to a leaf in T.

e. Return D as the edge-dominating set.

Prove that this heuristic returns a **correct** edge-dominating set (i.e., one in which every edge is dominated by some edge in the set).

Solution: Consider some edge $e = (u, v) \in E$. Assume that $e \notin D$. There are two main cases:

- Edge e is adjacent to a non-leaf node v. If $v = r$ and the tree has depth 1, then the tree is actually a star centered at the root; in this case, there is exactly one edge in D and it neighbors e. If $v = r$ and the tree has depth > 1, then v has at least one child w that is not a leaf and hence the edge (v, w) is in D and adjacent to e. Finally, if $v \neq r$, then v has a parent node w and the edge (v, w) is in D and adjacent to e. In each case, e is dominated by an edge in D.
- Both the nodes adjacent to edge e are leaves. Recall, however, that T is a DFS-tree, i.e., two leaves cannot be connected by an edge—otherwise the DFS would have continued its search when it encountered the first of the two leaves. (Notably, in a DFS of an unweighted graph, all non-tree edges are back edges pointing to an ancestor.) Hence there cannot be an edge e connecting two leaves of the tree T.

(c) [12 points] Prove that the heuristic above returns a 6-approximation of the minimum-sized edge-dominating set. (A better approximation ratio may be possible, but is not necessary here.)

Solution: The proof will proceed in two stages. First, we will argue that the optimal edge-dominating set for a sub-graph G' is no larger than twice the optimal edge-dominating set for the entire graph. Second, we will argue that if the above heuristic finds an edge-dominating set E', then we can construct such a sub-graph G' which has an optimal edge-dominating set of size at least $|E'|/3$.

First, let OPT_G be the optimal edge-dominating set for the entire graph. We show that you when you delete an edge from the graph G, you can find another edge-dominating set containing at most 2 other edges. Notably, imagine that you remove edge $e = (u, v)$ that is in OPT_G. If there are no other edges incident to u or v, then stop. Otherwise, if there are any edges incident to u, then add an arbitrary edge incident to u to the edge-dominating set. Similarly, if there are any edges incident to v, then add an arbitrary edge incident to v to the edge-dominating set. This ensures that any edge depending on e to be covered is still covered by one of the newly added edges. Moreover, we have replaced one edge in OPT_G with two edges in the new edge-dominating set. For any subgraph G', we can continue inductively until you have removed all the edges not in G'. This shows that for any $G' \subseteq G$, we have $OPT_{G'} \leq 2OPT_G$.

Second, we identify a graph G'. Specifically, for each non-leaf node in the tree T, choose exactly one child from T (arbitrarily) and add that edge to G'. Omit all the other edges out of G. Notice that G' now consists only of paths and disconnected nodes. Let t be the number of edges in the edge-dominating set constructed by the heuristic. Notice that there are then $t+1$ non-leaf nodes in T: for every non-leaf node except the root, its parent edge is in D. Since for each non-leaf node in T, we add one edge to G', we conclude that G' contains exactly $t + 1$ edges.

Finally, on any given path of length x, the optimal minimum edge-dominating set much contain at least $x/3$ edges—since each edge can cover at most three edges: two other edges plus itself. Putting these facts together, we conclude that $OPT_{G'} \geq (t+1)/3 \geq |D|/3$. Since $OPT_{G'} \leq 2OPT_G$, we conclude that $|D| \leq 6OPT_G$.

You may or may not find it simpler to convert the problem from *edge dominating set* to *vertex dominating set* by building a new graph where each edge $e \in E$ is a vertex in the new graph, and two adjacent edges in G are connected by an edge in the new graph. In this case, the problem is more similar to vertex cover.

Problem 4. (*Spacely Sprockets*) [20 points]

(a) [10 points] Spacely Sprockets produces four types of sprockets: A, B, C, D. Making sprockets requires energy, and your job is to minimize the energy used. Here are the rules:

- Sprocket A costs 2 kilowatts to make, sprocket B costs 1 kilowatt, sprocket C costs 1 kilowatt, and sprocket D costs 4 kilowatts.
- Spacely Sprockets has two factories: X and Y. Factory X makes only sprockets of type A and D, and Factory Y makes only sprockets of type B and C. Spacely has a contract with the unions promising to make a certain number of sprockets at each factory every day. At Factory X, he has agreed to make at least 8 sprockets per day (in total, of either type). At Factory Y, he has agreed to make *exactly* four sprockets a day (in total, of either type).
- The number of sprockets of type C should *not* exceed 4 per day. (Don't ask why.)
- Lastly, sprockets of type A take up one cubic meter of space, and sprockets of type D take up four cubic meters of space. Every day, your client Fred requires that you make enough sprockets of type A and type D to fill up a 12 cubic meters truck.
- For the purpose of this problem, sprockets can be subdivided into smaller pieces. Hence if you need to produce 0.3357 sprockets of type C, that is doable.

Give a linear program that will solve this problem, minimizing the number of kilowatts needed. Write your linear program in standard form.

Solution:

$$\begin{aligned} \min 2A + B + C + 4D & \\ A + D &\geq 8 \\ B + C &\geq 4 \\ -B - C &\geq -4 \\ A + 4D &\geq 12 \\ -C &\geq -4 \\ A, B, C, D &\geq 0 \end{aligned}$$

(b) [10 points] Give the dual of the linear program from the previous part. (If you did not succeed in coming up with a linear program for part (a), then for partial credit make up any linear program with between three and five variables and between three and five constraints and give the dual of it.)

Solution:

$$\begin{aligned} \max 8z_1 + 4z_2 - 4z_3 + 12z_4 - 4z_5 & \\ z_1 + z_4 &\leq 2 \\ z_2 - z_3 &\leq 1 \\ z_2 - z_3 - z_5 &\leq 1 \\ z_1 + 4z_4 &\leq 4 \\ z_1, z_2, z_3, z_4, z_4 &\geq 0 \end{aligned}$$

Problem 5. (*Startups 101*) [20 points]

NUS has decided to offer a new module on how to start a startup up. On the first day of class, we divide the students into three groups: (i) the Computer Scientists, (ii) the Engineers, and (iii) the Business People.

Next, we need to form teams. Each startup team must have exactly one computer scientist, one engineer, and one business person. (And each person can be on at most one team.) To accomplish this, each computer scientist p lists the engineers $E(p)$ and the business people $B(p)$ that he/she is willing to work with. Each engineer and business person p lists the computer scientists $C(p)$ that he/she is willing to work with. (We assume that every engineer is willing to work with every business person, and vice versa.)

In this problem, you will give an algorithm for creating the maximum number of teams possible. (Everyone who is not assigned a team has to drop the class!) Give the most efficient algorithm you can.

(a) [8 points] Describe your algorithm, briefly. It is recommended that you draw a picture.

Solution: We can solve this as a matching problem. Add three sets of nodes: E for engineers, C for computer scientists, and B for business people. For each node in C, add two nodes to the graph c_1 and c_2. For every $i \in C$, connect i_1 and i_2 with an edge. Add an edge from a node $i \in E$ to a node $j \in C_1$ if $i \in E(j)$ and $j \in C(i)$, i.e., if i and j are willing to work together. Similarly, add an edge from a node $i \in C_2$ to a node $j \in B$ if $i \in C(j)$ and $j \in B(i)$.

Now, add a source node s and connect it to every engineer in E. Add a target node t and connect every business person to t.

Give every edge in the graph capacity 1. Execute a maximum-flow algorithm. Each $s - t$ path represents a team.

(b) [4 points] What is the (asymptotic) running time of your algorithm?

Solution: If there are n people total, then there are at most $2n^2$ edges in the graph. Using Push-Relabel, which runs in time $O(n^2m)$, the total running time is $O(n^4)$. Using Ford-Fulkerson, the maximum flow is n, and so the total running time is $O(n^3)$. Using matching algorithms (not covered in this module) directly, you can get even better running times (e.g., $O(n^{2.5})$).

(c) [8 points] Prove that your algorithm is correct.

Solution: Notice that each such team found by the algorithm is feasible: it contains exactly three people, one of each type. Moreover, each person is only on one team since each person has an incoming or outgoing capacity of 1. (This was the reason for dividing up the computer scientists into two nodes.) Notice that this depends on flow integrality: each flow in the network is either 0 or 1.
Finally, it is optimal since any team assignment is a feasible flow, where the number of teams is equal to the capacity of the flow.

Scratch Paper

Scratch Paper

Scratch Paper

Scratch Paper