# Midterm Exam

- The midterm contains five problems (and one just for fun). You have 100 minutes to earn 100 points.
- The midterm contains 18 pages, including this one and 3 pages of scratch paper.
- The midterm is closed book. You may bring one double-sided sheet of A4 paper to the midterm. (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 or lemma given in class without restating it—simply give the name of the algorithm and the running time. If you change the algorithm or lemma in any way, however, you must provide complete details.
- You may assume, for any problem, that you have access for free to a polynomial-time LP solver.
- Draw many pictures.
- Good luck!

| **Problem #** | **Name** | **Possible Points** | **Achieved Points** |
|---|---|---|---|
| 1 | You are the Computer | 10 | |
| 2 | Choosing Classes | 20 | |
| 3 | Drawing Pictures | 20 | |
| 4 | Approximate Matching | 25 | |
| 5 | DisneyLand-Singapore | 25 | |
| **Total:** | | 100 | |

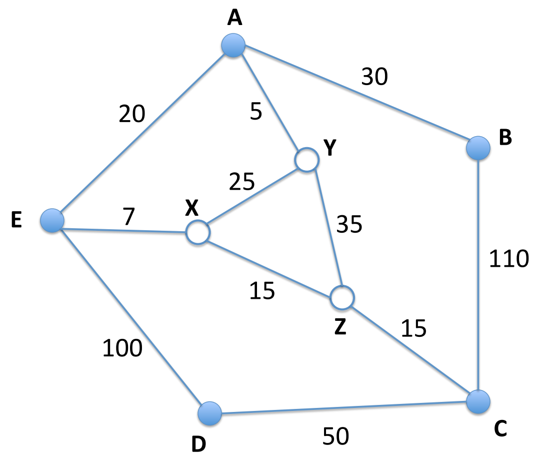
Name: ______________________________ Student Id:: ______________________________

**Problem 1.** You are the Computer [10 points]

Find a 2-approximate maximum weight Steiner tree in the following graph using the General Steiner Tree approximation algorithm from class. Show your work (i.e., give just enough detail that it is clear you understand how the algorithm works).

In the figure below, the solid nodes $(A, B, C, D, E)$ are the required nodes, while the empty nodes $(X, Y, Z)$ are the Steiner nodes. You may draw the final Steiner tree directly on the figure by (clearly) highlighting the edges you choose to include.

The total cost of your Steiner tree is: $\boxed{\phantom{000}}$



**Intermediate step of your solution:**

**Problem 2.** Choosing Classes [20 points]

Imagine that you are a student at a university. On the first day of your first year, you must choose the set of classes that you want to take before you graduate. The classes are given as a set $C = c_1, c_2, \ldots, c_n$.

Each class provides some *joy*, and your goal is to maximize the amount of joy from the classes you take. You are given a function $joy(c_i)$ that defines the joy of class $c_i$.

At the same time, you must satisfy the university requirements. Each class has a workload $work(c_i)$. In total, you may only take classes that have no more than 200 units of work.

Classes are divided into four categories: art (A), science (S), engineering (E), and space travel (T). You must take at least 70 units of work in space travel, which is your focus area. You are given the function $category(c_i, x) = 1$ if and only if class $c_i$ is in cateogory $x$. For example, $category(\text{“lunar mining”}, A) = 0$ and $category(\text{“lunar mining”}, T) = 1$.

A final requirement is that you cannot take too many more art classes than engineering classes. Specifically:

- The first 2 art classes are free. You can take them with no restrictions.
- After that, every 1 engineering class you take allows you to take 2 more art classes.

For example, you can take 2 art classes and no engineering classes; or you can take 12 art classes and 5 engineering classes; or you can 4 art classes and 1 engineering classes; or you can take 3 art classes and 1 engineering class. Notice that this restriction is based on *the number* of classes, not the work.

For example, consider the following available courses, where course $(x, y, z)$ is of type $x$, requires $y$ work, and gives joy $z$:

$$\begin{aligned} c_1 &= (T, 40, 10) & c_2 &= (T, 40, 10) \\ c_3 &= (A, 20, 20) & c_4 &= (A, 20, 20) \\ c_5 &= (A, 20, 20) & c_6 &= (A, 20, 20) \\ c_7 &= (E, 30, 10) & c_8 &= (S, 10, 5) \end{aligned}$$

In this case, a legal schedule might include all of these classes for a schedule with 200 work and 115 joy. It includes 80 units of work for space travel, and four art classes and one engineering class.

**Problem 2.a.**

Give an integer linear programming that finds the best set of classes to take for a given set $c_1, c_2, \ldots, c_n$ of classes where class $c_i$ has $joy(c_i)$, $work(c_i)$, and $category(c_i, \cdot)$. (Your ILP does *not* need to be in normal form.) For each variable and constraint, explain (briefly) its intent.

**Problem 2.b.** Imagine that you solve this integer linear program as a linear program. In this case, you may get a fractional solution where you may take a fraction of a class (e.g., you can take 1/5 of a class). Notice that in the fractional case, the constraint on art and engineering classes is that, after you have completed your 2 free art classes, you can take at most twice as many (fractional) art classes as (fractional) engineering classes.

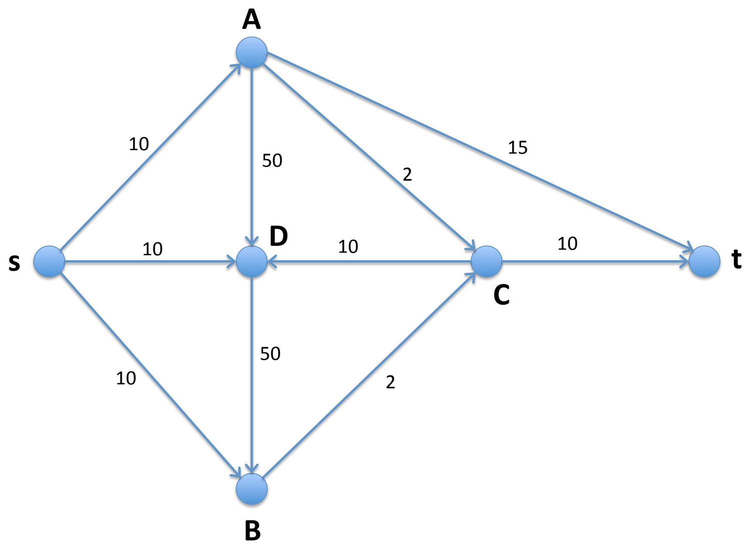
Give an example where the best integer solution is at most 3/4 of the best factional solution (or as close to 3/4 as you can come).

(Hint: imagine a situation where after your required classes in space travel, you are very close to the maximum work limit.)

**Problem 3.** Drawing Pictures [20 points]

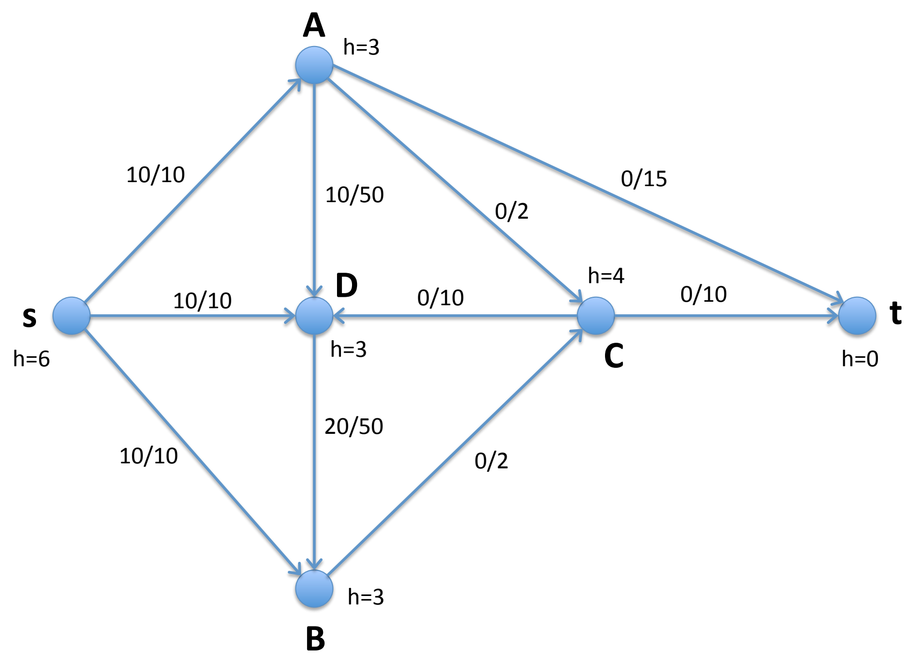
**Problem 3.a.** Identify a minimum capacity cut in the following graph. (Draw the cut *clearly* on the graph.) Prove, using the Max-Flow / Min-Cut Theorem, that it is really the minimum capacity cut.

The capacity of your cut is: $\boxed{\phantom{000}}$



**Problem 3.b.** Imagine you are running the Push-Relabel algorithm on the following graph. Notice that for each node, the current height $h$ is indicated. For each edge, the current flow is indicated.

Execute two steps of the Push-Relabel algorithm. For each step, indicate what type of step (i.e., push or relabel) and how the state changes as a result.



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

**Problem 4.** (Approximate Matchings) [25 points]

Joe Kaynaim decided to try to implement Christofides Algorithm. It went well until he came to the part on Maximum Weight Matchings. (Oops, he has already made a big mistake in understanding Christofides Algorithm.) Alas, the lecture notes from CS4234 did not quite provide enough information on how to efficiently implement this step.

Joe had an idea: while implementing an optimal maximum weight matching algorithm is a bit tricky, it is really easy to implement an *approximate* maximum weight matching algorithm. Since Christofides Algorithm is an approximation algorithm anyways, maybe it is ok?

Joe focuses on the following problem: given an undirected graph $G = (V, E)$ with edge weights $w$, find a matching $M \subseteq E$ with maximum weight, i.e., $\sum_{e \in M} w(e)$ is maximized. (Note that the input $G$ is an arbitrary graph, and the output does not have to be a *perfect* matching, just a matching with maximum weight.)

Joe implemented the following greedy algorithm for finding the maximum weight matching for a graph $G = (V, E)$ with weights $w$:

**Approximate Maximum Weight Matching**

```
1. Sort all the edges by weight from largest to smallest: $e_1, e_2, \ldots, e_m$.
2. Let $e_j = (u, v)$ be the largest weight edge.
3. Add $e_j$ to the matching $M$.
4. Delete all edges adjacent to $u$; delete all edges adjacent to $v$.
5. If there are any edges left, go to Step 2.
```

**Problem 4.a.** Prove that this algorithm gives a 2-approximate maximum weight matching. (For partial credit, show instead that it gives a 2-approximate maximum sized matching.) **Remember, Joe is solving the Maximum Weight Matching problem on an arbitrary graph** $G = (V, E)$, so the output may not be a perfect matching.

**Problem 4.b.** Give an example that shows that the Approximate Maximum Weight Matching algorithm is, asymptotically, no better than a 2-approximation.

**Problem 4.c.** Joe suddenly realizes that for Christofides Algorithm, he needs a *minimum* weight *perfect* matching, not a maximum weight matching. After a little more thought, he finds an Approximate Minimum Weight Perfect Matching algorithm that is a 3-approximation of optimal. (That is, it turns out that the work Joe did in parts (a) and (b) was not helpful for implementing Christofides Algorithm.)

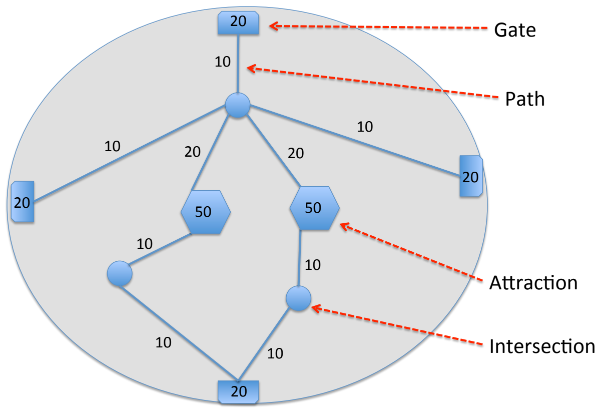
Assume Joe implements Christofides Algorithm for approximate (metric) TSP using a 3-Approximate Minimum Weight Perfect Matching algorithm $A$ (which is guaranteed to find a 3-approximation of the minimum weight perfect matching).

What is the approximation ratio for the resulting implementation of Christofides Algorithm? Explain your answer.

**Problem 5.** Disneyland-Singapore [25 points]

You are on the team designing the layout for the new Disneyland-Singapore! Here are the constraints:

- Disneyland has a collection of gates $G = g_1, g_2, \ldots, g_k$. Each gate only allows a fixed number of people to enter every day. For all gates $g_i \in G$, we define $\mathit{number}(g_i)$ to be the number of people who can pass through gate $g_i$ in a day.
- Disneyland has a collection of attractions $A = a_1, a_2, \ldots, a_\ell$. There is a large crowd of $P$ people that each want to enter at some gate and visit exactly one of the attractions. For all attractions $a_i \in A$, define $\mathit{number}(a_i)$ to be the number of people who want to visit attraction $a_i$ in a given day.
- There is a networks of walking paths connecting the gates to the attractions. The network consists of a set $V$ of $n$ intersections (i.e., nodes) and a set $E$ of $m$ paths (i.e., edges). Each path $e$ has a capacity $c(e)$ that determines the maximum number of people that can use the path in a given day. Note that the paths are undirected: up to $c(e)$ people can use the path *in either direction*—however no more than $c(e)$ people can use it in total.



In this example of a possible Disneyland layout, there are four gates, each of which allows 20 people through. There are two attractions, and there are 50 people that want to visit each attraction. The set of paths contain nine paths and three intersections (including a main intersection with a very pretty fountain).

Notice that in this example, it is impossible for all 100 people to visit their preferred attraction. Notably, only 30 people can visit each attraction.

**Problem 5.a.** We want to allow everyone to visit their desired attraction. However, the gates are limited (because each gate attendant can only check a fixed number of tickets per unit time). Given a specific design, your job is to decide whether the design meets this requirement.

That is, given gates $G$, attractions $A$, intersections $V$, paths $E$ and cost functions *number* and $c$, determine whether everyone who wants to visit an attraction can: for all attractions $a_i \in A$, is it possible for $number(a_i)$ people to reach attaction $a_i$. Your algorithm should output either **YES** (i.e., everyone can get to their attraction) or **NO** (i.e., some people cannot get to their attraction).

Give an algorithm, and explain why it works.

**Problem 5.b.** What is the running time of your algorithm from part (a)? (The running time should be given as a function of (some subset of) the input parameters, i.e., $G$, $A$, $V$, $E$, *number*, and $c$. Briefly explain your answer.

**Problem 5.c.** Assume you discover that the park design is sufficient, i.e., enough people can get to every attraction. Explain how to find a route for each individual from some gate to their attraction of choice. (Note that a person is willing to enter at any gate, but must arrive at their desired attraction.) That is, give an algorithm for finding the individual routes from gates to attractions (that satisfy the contraints, of course).

**Problem 6.** (Just for fun!) [0 points]

While you take this midterm, I am currently on my way to a conference, spending much time in large international airports. That reminds me of a little puzzle, courtesy of Terence Tao[1].

Suppose you are trying to get from one end of a terminal to the other end. (For simplicity, assume the terminal is a one-dimensional line segment.) Some portions of the terminal have moving walkways going in the direction you want. Your walking speed is a constant $v$, but while on a walkway, it is boosted by the speed $u$ of the walkway for a net speed of $v + u$. Your objective is to get from one end to the other in the shortest time possible.

Suppose you need to pause for some period of time, say to tie your shoe. Is it more efficient to do so while on a walkway, or off the walkway? Assume the period of time required is the same in both cases.

Or, alternatively, suppose you have a limited amount of energy available to run and increase your speed to a higher quantity $v'$ (or $v' + u$, if you are on a walkway). Is it more efficient to run while on a walkway, or off the walkway? Assume that the energy expenditure is the same in both cases.

Can you give a simple and intuitive explanation for your answer? (Presumably, you can give a detailed and mathematical explanation as well.)

[1] I am quoting `https://terrytao.wordpress.com/2008/12/09/an-airport-inspired-puzzle/`

## Scratch Paper

## Scratch Paper

## Scratch Paper