Time limit: 80 minutes, and an extra 15 minutes for submission.

Maximum score: 184 points.

Instructions: For this test, you work in teams of up to eight to solve a multi-part, proof-oriented series of problems.

Problems that use the words "compute", "list", or "draw" only call for an answer; no explanation or proof is needed. Unless otherwise stated, all other questions require explanation or proof. Answers should be written on sheets of scratch paper, clearly labeled, with every problem *on its own sheet.* If you have multiple pages for a problem, number them and write the total number of pages for the problem (e.g. 1/2, 2/2).

Indicate your team ID number on each of paper that you submit. Only submit one set of solutions for the team. Do not turn in any scratch work.

In your solution for a given problem, you may cite the statements of earlier problems (but not later ones) without additional justification, even if you haven't solved them.

The problems are ordered by content, NOT DIFFICULTY. It is to your advantage to attempt problems from throughout the test.

While completing the round, you should not consult the internet or any materials outside of the content of this test (*including results not covered in this power round*). You may not use calculators.

It is critical that you select pages corresponding to each subpart of the problem carefully and correctly following submission of the test. Failure to do so may result in your test not being graded or answers being missed. Page selection does not count as a part of your test taking time – you can select pages immediately after time is over with no penalty.

For the submitting student only: please change your Gradescope ID to be your team ID. You may access this by going to Account → Edit Account → Student ID.

Good luck!

List of Problems

1 Introduction

In this Power Round, we will be discussing services through Queueing Theory, Market Design, and Matchings. To give a sense of the real world application of this mathematical concept, consider a call center. Customers join this call center line randomly, with some general average number per hour, and are served with some given frequency. However, it is also possible that customers who have been in line for too long simply drop off on their own. Given knowledge of how frequent each of the above events are, how long should you expect to wait until you are served if you join the line right now? We will look at a simplified model of this, and more, in the upcoming problems.

A note on color coding: problems are all in red boxes, definitions in blue boxes, and theorems in green boxes. Discussions (usually informal definitions, but important to read ones) are in orange boxes.

2 Probability Fundamentals

As alluded to in the introduction, we will start with looking at queues. As queueing theory is about random processes, we begin by familiarizing with some of the concepts.

2.1 Random Variables

Definition 2.1

A random variable X on a set S can be described as a function $p_X(x) : \mathbb{R} \to \mathbb{R}$ such that $\int_{-\infty}^{\infty} p_X(x)\,\mathrm{d}x = 1$ and $p_X(x) \geq 0$ for all x. The function $p_X(x)$ is called the "probability density function" of X. Here we use an integral as S may have weird structure (it might not be just integers, for example), and in this case the integral is only nonzero on points in S. This view as an integral is more useful when S is potentially infinite (specifically, when $S = \mathbb{R}$ or some large chunk of the reals).

When X is only defined on a subset of the (nonnegative) integers, we instead describe the probability density function as satisfying $\sum_{n=0}^{\infty} p_X(n) = 1$ and $p_X(n) \geq 0$ for all n.

Equipped with this definition, we can already begin to look at some natural random variables. Although the below may look like a slog of definitions, they will all be important to our later discussions (and will be a good reference).

Definition 2.2

Here are some common random variables.

- (Geometric Random Variable) We call a random variable $X \sim \text{Geom}(p)$ if X is the number of flips of a coin until we get heads, given that each flip comes up heads with probability p independent of all other throws (also known as **p-biased** coin).
- (Binomial Random Variable) We call a random variable $X \sim \text{Binomial}(n, p)$ if X is the number of heads when a p-biased coin is thrown n times, independently.
- (Poisson Random Variable) We call a random variable $X \sim \text{Poisson}(\lambda)$ if X can be modeled as the number of arrivals of a customer to a queue in (say, 1 hour) with fixed arrival rate λ. We say the arrival rate is fixed if it does not depend on past frequency of arrivals in any way.

With these definitions in mind, we state and prove a theorem about some properties of these random variables.

Theorem 2.1

Let x be a nonnegative integer (in the first case, also assume that $x \geq 1$).

1. If $X \sim \text{Geom}(p)$, then $p_X(x) = (1-p)^{x-1} \cdot p$.
2. If $X \sim \text{Binomial}(n, p)$, then $p_X(x) = \binom{n}{x} p^x (1-p)^{n-x}$.
3. If $X \sim \text{Poisson}(\lambda)$, then $p_X(x) = \frac{e^{-\lambda}\lambda^x}{x!}$.

Problem 2.1 (Probability Densities, 2 points)

[1 point each] Justify the first two cases (Geometric and Binomial) of the above theorem.

Random variables are often useful in regards to what results we see when observing them: in other words, the average values they may obtain after some transformations. To this end, we can define certain properties of random variables.

Definition 2.3

For a general random variable X, we define

- $\mathbf{E}[X] = \int_{-\infty}^{\infty} x \cdot p_X(x)\,\mathrm{d}x$ is the **expected value** of X.

If X is only defined on nonnegative integers, we often write this as $\mathbf{E}[X] = \sum_{n=0}^{\infty} n \cdot p_X(n)$ instead.

- $\mathbf{Var}(X) = \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] = \mathbf{E}\left[X^2\right] - \mathbf{E}[X]^2$ is the **variance** of X (otherwise known as the second central moment).
- $\mathbf{E}\left[X^k\right]$ is the **k-th moment** of X.
- $\widehat{X}(z) = \mathbf{E}\left[z^X\right]$ is the **z-transform** of X.
- $\widetilde{X}(s) = \mathbf{E}\left[e^{sX}\right]$ is the **moment generating function** of X.

Note that really, every concept is described in terms of the expectation of a variable X. In addition, note that the latter two definitions are the same up to transformation: substituting $s = \ln z$ gives $\widetilde{X}(\ln z) = \widehat{X}(z)$. So, we will use whichever one is more useful in the moment.
As is, though, the moment generating function $\widetilde{X}(s)$ may seem a bit mysterious: how did it get this name?

Problem 2.2 (Onion Peeling Theorem, 3 points)
Prove that the derivatives of the moment generating function are the moments of X. In particular, prove that $\widetilde{X}^{(k)}(0) = \mathbf{E}\left[X^k\right]$. The (k) superscript notation means taking the kth derivative, and then evaluating at $s = 0$.

We also note the following important theorem which we will not prove, and you can use freely.

Theorem 2.2
This theorem states important properties of general random variables.

1. (Linearity of Expectation) For *any* two random variables X, Y, we have $\mathbf{E}[X+Y] = \mathbf{E}[X] + \mathbf{E}[Y]$.
2. (Independent Splitting) If X and Y are independent (that is, for all x, y we have $p_{X,Y}(x,y) = p_X(x) \cdot p_Y(y)$), then for any functions f and g, $\mathbf{E}[f(X)g(Y)] = \mathbf{E}[f(X)]\,\mathbf{E}[g(Y)]$.
   - As a corollary, if X and Y are independent, then $\mathbf{Var}(X+Y) = \mathbf{Var}(X) + \mathbf{Var}(Y)$.
   - In addition, if X and Y are independent then $(\widetilde{X+Y})(s) = \widetilde{X}(s) \cdot \widetilde{Y}(s)$.

Problem 2.3 (Moments of Random Variables, 7 points)
Compute the following, and show your work (not showing work for a subpart will result in a score of 0 for it).

1 pts. If $X \sim \text{Geom}(p)$, compute $\mathbf{E}[X]$ in terms of p.

3 pts. If $X \sim \text{Binomial}(n, p)$, compute $\mathbf{E}[X]$, $\mathbf{Var}(X)$, and $\widehat{X}(z)$ in terms of n and p.

3 pts. If $X \sim \text{Poisson}(\lambda)$, compute $\mathbf{E}[X]$, $\mathbf{Var}(X)$, and $\widehat{X}(z)$ in terms of λ.

2.2 Inter-arrival Times

With these three distributions at our disposal, let's dive a little deeper into the Poisson Distribution. It may seem a bit mystical why our distribution has this form, so let's give some reasoning for this fact.

Consider the random variables $X_n \sim \text{Binomial}(n, \frac{\lambda}{n})$, and $Y \sim \text{Poisson}(\lambda)$. We may always scale Y to be occurring in a time interval of length n where we expect λ arrivals total to occur, or we may instead scale X_n to take values $\{0, \frac{1}{n}, \dots, \frac{n-1}{n}, 1\}$, but still look at the number of heads.

Problem 2.4 (Limit of Binomial, 1 point)
Briefly give some "intuition" for how X_n and Y are connected, as $n \to \infty$.

Hint: Think about the arrivals of heads in X_n.
Note: this problem is one point: please do not get stuck up on it if the intuition doesn't come immediately.

To prove this formally, we introduce one notion of two random variables being equal in distribution.

Problem 2.5 (Probabilities in z-transform, 3 points)

Suppose that X is a random variable taking values in the natural numbers. Also suppose that instead of being given $p_X(k)$ for all k, we get $\widehat{X}(z)$. Find an expression for $p_X(k)$ as a function of $\widehat{X}(z)$ (and k itself).
Hint: Write out the z-transform explicitly as a sum, and start with $p_X(0)$.

The above problem implies that proving $\widehat{X}(z) = \widehat{Y}(z)$ implies that the two are equal in distribution.

This is very important to us, specifically because the z-transform is well understood for the Binomial and Poisson random variable. Using this, we may formalize the intuition on their "equality".

Problem 2.6 (Binomial converges to Poisson, 4 points)

As above, let $X_n \sim \text{Binomial}(n, \frac{\lambda}{n})$ and $Y \sim \text{Poisson}(\lambda)$.
Let $X = \lim_{n\to\infty} X_n$ and prove that X and Y are equal in distribution (that is, prove that $\lim_{n\to\infty} \widehat{X_n}(z) = \widehat{Y}(z)$).
Note: the formal statement we are proving is that the X_n converge in distribution to Y as $n \to \infty$.

With this intuition out of the way, we have some notion of a Poisson random variable being a limit of Binomials with increasingly less frequent coin flips. As the coin flips get less frequent, however, the un-normalized time between flips increases (recall that this is approximately Geometric, as we do have a stopping point after n steps).

We may then ask the natural extension of this to Poisson random variables: what is the distribution of **inter-arrival times**: that is, if people are arriving in a queue according to a Poisson random variable, then what is the distribution of the time in between arrivals?

To explore this question, we must look into what happens if we consider an "infinite" Poisson distribution, one not limited by how much time we capture arrivals in.

Definition 2.4

Define a **Poisson Process** with parameter λ as an infinite analogue of the Poisson random variable. Precisely, the number of arrivals up through time t is distributed according to Poisson(λt).

From this definition, it makes sense that we should have $\text{Poisson}(\lambda_1) + \text{Poisson}(\lambda_2) \sim \text{Poisson}(\lambda_1 + \lambda_2)$, and while this is true and not difficult to prove, you may use this whenever you need, without proof.
It turns out that the distribution of inter-arrival times will be highly important to us when looking at real queues.

Definition 2.5

Define the Exponential distribution as the continuous distribution with $X \sim \text{Exp}(\lambda)$ having **probability density function** $p_X(x) = \lambda e^{-\lambda x}$.

Notice that this looks somewhat like the Geometric (specifically, as $\lambda\left(e^{-\lambda}\right)^x \approx p \cdot (1-p)^x$ if $\lambda = p$ is very small), and as we will find out shortly, it is a continuous analogue. This has some reasoning behind it: since the Geometric is a rough simulation of the inter-arrival times of the Binomial, we'd expect a continuous analog which looks like it to be the distribution of inter-arrival times of the Poisson.
To prove the fact about inter-arrival times, however, note that we cannot use our distribution equality technique between the two unfortunately: the Exponential distribution can take on any nonnegative real value unlike the Geometric.

Problem 2.7 (Poisson interarrival is Exponential, 4 points)

Prove that the inter-arrival time X of a Poisson Process with parameter λ is distributed according to Exp(λ).

Hint: Compute the probability that there are no arrivals from time 0 *to time* t.

The fact that this is an inter-arrival time for the Poisson Process also implies that the Exponential satisfies a *memoryless* property similar to that of the Geometric: that is, if $X \sim \text{Exp}(\lambda)$ then $\mathbf{P}[X > t + s | X > s] = \mathbf{P}[X > t]$.

We will not emphasize conditional probability much in this round, but this means that the probability that $X > t$ is the same as the probability that $X > t + s$ given that $X > s$ (so, if we haven't seen an arrival in time s, we have no new knowledge on when the next arrival will occur). Again, this is not hard to prove, but you may use it without proof as well.

Problem 2.8 (Playing with Exponentials, 3 points)

Let $X \sim \text{Exp}(\lambda)$ and $Y \sim \text{Exp}(\mu)$. Show your work on the following parts.

1 pt. Compute $\mathbf{E}[X]$.

1 pt. Compute $\widetilde{X}(s)$.

1 pt. Compute $\mathbf{P}[X < Y]$.

Finally, with all of these definitions in place, we may begin to apply them to queueing systems.

3 Queueing Theory

We begin our queueing theory adventure by looking at the most goldfish of all queues: the $M/M/1/\infty$ queue. In this notation, each M refers to an Exponential distribution, the 1 corresponds to the number of queues, and the ∞ refers to the cap on the length of the line. In particular, we usually look at this queue as having independent arrivals each distributed as $\text{Exp}(\lambda)$ (λ is the *arrival rate*), and independent service times distributed as $\text{Exp}(\mu)$ (μ is the *service rate*).

Note that we do not specify how service occurs: in particular, someone who is currently being serviced could be put on hold to help another at any point. We will begin by fixing this order as FCFS, or "First Come First Serve". In this order, only after the current person is serviced is the next person begun to be serviced.

3.1 Little's Law

For a queueing system Q, it is helpful to be able to ask questions about its *steady state*: that is, after a long time of running the system, what do various statistics of it look like?

We won't rigorously define what a steady state is (as it would require an introduction to Markov Chains), but one interpretation is that if the system is currently in the steady state, then at any point after this it remains in steady state. In this way, we can view the steady state as the limit state of running the process for a long time (assuming there is a unique steady state, which you may assume throughout).

Definition 3.1

We define several random variables (and their conventional symbols) for a queueing system Q.

- We define N as the number of people waiting to be serviced (including the person currently being serviced) in the steady state distribution.
- We define T as the time you must wait until exiting the system after arriving, in the steady state.

Problem 3.1 (Memoryless queue, 7 points)

Suppose first we are in an $M/M/1/\infty$ queue with arrival rate λ and service rate μ, with $\lambda < \mu$ (this is required so that the system does not go off to infinity in the steady state). Furthermore, suppose that people are serviced in the order they come in. (You may *not* use Little's Law, stated below, in this problem)

2 pts. Express $\mathbf{E}[N]$ and $\mathbf{E}[T]$ in terms of each other and μ.

5 pts. Compute $\mathbf{E}[N]$ and $\mathbf{E}[T]$. It may be helpful to express things in terms of $\rho = \frac{\lambda}{\mu}$ in your proof (known as the *traffic density*).

Of course, the above results are very specific to the $M/M/1/\infty$ queue: we can't expect the same kind of analysis to extend to any other queue. However, the first part of the above problem is a bit intriguing: is there still a general relationship between $\mathbf{E}[N]$ and $\mathbf{E}[T]$ that we may compute? It turns out that there is a surprisingly simple relationship here.

Theorem 3.1 (Little's Law)

Let Q be a **general** queueing system with a steady state and having average arrival rate λ (this need not be memoryless in any way). Then $\mathbf{E}[N] = \lambda \mathbf{E}[T]$.

We will dedicate the rest of this section to proving Little's Law. To do so, we will actually define $\overline{N}$ and $\overline{T}$ as *time-averages* of N and T: specifically, they are the average number of jobs in the system and the average time from arrival starting from the system having nobody in it. To look at an example, suppose that you are a cashier at a supermarket. People join your line, and each person has some time T_i that it takes from them arriving to them paying for their groceries. The average time spent in the system, if there are n total people that join your line, is $\frac{\sum T_i}{n}$. The *time-average* $\overline{T}$ is this expression if we were to extend time to infinity: that is, people keep arriving and we take this fraction, but as time goes to infinity. Defining $\overline{N}$ proceeds similarly.

It turns out that for a system with a steady state, we have the following theorem, which you can assume freely.

Theorem 3.2 (Time average is Expectation)

In a system with a steady state, $\overline{N} = \mathbf{E}[N]$ and $\overline{T} = \mathbf{E}[T]$.

For the proof of Little's Law, we will also need a result from calculus which you may use without proof.

Theorem 3.3 (Squeeze Theorem)

Suppose that for every x, $f(x) \le g(x) \le h(x)$. Then, if $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} h(x) = L$, it follows that $\lim_{x\to\infty} g(x) = L$.

Problem 3.2 (Little's Law, 10 points)

We split this proof up into several parts and definitions.

0 pts. (No need to submit anything) Define $A(t)$ as the number of arrivals by time t, and $C(t)$ the number of people served by time t. Convince yourself that $\lambda = \lim_{t\to\infty} \frac{A(t)}{t} = \lim_{t\to\infty} \frac{C(t)}{t}$ when there is a steady state.

2 pts. Let $N(t')$ be the number of jobs in the system at time t', and let T_i be the time it takes for job/person i to be serviced (from when they arrive to when they depart). Express $\overline{N}$ and $\overline{T}$ in terms of the quantities we have defined up to now.

8 pts. Prove the *time-average* version of Little's Law: $\overline{N} = \lambda \overline{T}$.

With Little's Law in mind, suddenly the $M/M/1/\infty$ solution gets a lot easier.

Problem 3.3 (Memoryless queue with Little's Law, 1 point)

Rederive $\mathbf{E}[N]$ and $\mathbf{E}[T]$ for the $M/M/1/\infty$ queue using Little's Law and the first part of Problem 3.1.

3.2 PASTA...yum

When proving Little's Law, we transitioned from expectations to time averages, and claimed without proof that these two were equal: that is, a random observer sees time averages. Is this fact true for arrivals? That is, if a person arrives to the queue, will they necessarily see the time-average or steady state in action?
The answer turns out to be no in general, but for one very special and important case it is indeed correct.

Theorem 3.4 (Poisson Arrivals See Time Averages)

Let Q be a queueing system. Suppose that the arrivals follow a Poisson Process with rate λ (Exponential arrivals). Let a_n be the probability that an arrival into the system sees n people in front of them. Then, $a_n = \pi_n$, where π_n is the time average/steady state probability of having n people in the system.
In particular, (using terminology from the proof of Little's Law) it is true that

$$\lim_{t\to\infty} \mathbf{P}\left[N(t) = n\right] = \lim_{t\to\infty} \mathbf{P}\left[N(t) = n \mid \text{arrival happens right after time } t\right]$$

Problem 3.4 (PASTA, 3 points)

Prove PASTA (the above theorem).

Finally, we end queueing theory by looking at a more real world type of queue.

3.3 Bounded Queues and Insensitivity

In particular, we turn our attention to a slightly restricted and more realistic version of queues: queues of bounded size. Imagine that it is Covid times and so each store only has capacity for one person to go inside. However, there is a no loitering policy on the sidewalk, so if all of the stores have one person in them, then everyone else who arrives must go home. If there are k stores and both arrivals and service times are memoryless (Exponential), what is the time-average probability that someone who arrives is turned away?

Definition 3.2

This probability is known as the *blocking probability* P^{block} of the $M/M/k/0$ queue (note that the 0 represents there being no waiting room). That is, P^{block} is the probability that all storefronts are full when a new person arrives.

Problem 3.5 (Blocking of simple queue, 6 points)

Compute (with proof) P^{block} for the $M/M/k/0$ queue (with arrival rate λ and service rate μ).
Note: We suggest using $\rho = \frac{\lambda}{\mu}$ as before.

As always, having memoryless properties makes proofs easier: but can we generalize this in any way? As is, we have PASTA which works for general service times. Is it possible that P^{block} is also the same for general service times? This surprising fact is indeed true, and is also known the insensitivity theorem.

Theorem 3.5 (Insensitivity of Blocking Probabilities)

Consider any $M/G/k/0$ queue with expected service time $\frac{1}{\mu}$ (and arrival rate λ). Then, P^{block} is invariant of the distribution G, provided that service times are independent.

While this proof is not easy, it is essentially doable with all that we have discovered so far. For sake of exploring more topics, we won't prove it now.

4 Kidney Exchange

We will use some of what we have learned about probability and queues to conduct some analysis of the USA Kidney Exchange market.

Let's set up the problem. We may think of the kidney exchange market at any given point in time as a set of *patient-donor* pairs. There are different types of kidneys, and the chance that any particular pair of people have "compatible" kidneys is around 21% (we will later abstract away this exact number).

Let's say there are already some patient-donor pairs in a kidney market, and imagine a pair P, D enters the market. Then, we can make a link between this pair and every other pair P', D' where P, D' are compatible and P', D are compatible.

This is a simplified view of the kidney exchange market where the only possible donations happen as a result of direct swaps.

As a result of patient-donor pairs entering the market, we can construct a graph where the nodes are these pairs and the edges form exactly the possible swaps. When we *match* two pairs, they both exit the market: the edge between them and all of their other edges disappear.

In addition, we have a condition known as *criticality*: that is, if a patient becomes *critical* (the node vanishes from the system) then this is the last possible time to match them. After this point in time, the patient-donor pair will exit the market. Note that in the scenario that a particular patient becomes critical, the best move will be to match this person given the availability of a donor – because the best case given non-matching is that we will match the unused donor with at most one other patient in need.

With this, we can define more formally a kidney exchange market.

Definition 4.1

We call a (m, p) kidney exchange market as one with the following properties.

- The rate of arrival of patient-donor pairs is a Poisson Process with rate m.
- When a patient-donor pair arrives, the probability of it being linked to any particular patient-donor pair already in the market is p, independent of other pairs (in the real world, $p \approx 4\%$).
- Patient-donor pairs become critical as a Poisson Process with rate 1.

Usually, we write $d = pm$ as the "average number of links".

We generally make the assumption that if a system has equal arrival and departure rates, then it is in steady state (and vice versa). **You may assume that this is the definition of a steady state throughout.**

Problem 4.1 (Expected market size, 2 point)

Suppose that we have an (m, p) kidney exchange market, and we never match any pairs. If the system is in steady state, show that the expected number of patient-donor pairs in the market that a new pair sees when it arrives is m.

Of course, a kidney exchange market is not useful unless people actually get matched. For the purposes of this round, we will define a matching algorithm in terms of events: that is, pairs can only be matched when someone has just arrived or someone has become critical.

Definition 4.2

A matching algorithm is a function $f : G \times \{A, C\} \times V \to M$ where

- G is the state of the system (the set of pairs and their links), and also who is critical.
- $\{A, C\}$ denotes whether an arrival or criticality event just occurred.
- V denotes the pair that just had an arrival or criticality event.
- M is a (possibly empty) set of disjoint linked pairs who should exit the market now.

With this definition in mind, we can reason about how good a candidate function f is.

Definition 4.3

Let $\overline{C}$ be the time average number of pairs that become critical but are not matched by f. Then, define the loss $L(f)$ of an algorithm as $L(f) = \frac{\overline{C}}{m}$.

The reason for normalization by m is because this is the expected number of people we'll see in the system at any point in time, so it makes sense to consider L as a fraction of number of unmatched critical pairs over the total number of pairs.

We begin by defining our first function f, known as the greedy algorithm.

Definition 4.4

The greedy algorithm is the function $f : G \times \{A, C\} \times V \to M$ which does the following.

- If we are in an arrival event and (P, D) is the arrival, if (P, D) has any link to a (P', D') already in the market we match them and return $M = \{((P, D), (P', D'))\}$. Note that (P', D') is an *arbitrary* linked pair.
- If (P, D) has no links, output the empty matching.
- If we are in a criticality event, do nothing (we analyze why in Problem 5.2).

Theorem 4.1

Let L be the loss of the Greedy Algorithm. Then, as $m \to \infty$ and d stays fixed, $L \geq \frac{1}{2d+1}$.

Problem 4.2 (Analysis of greedy in kidney market, 9 points)

We are in an (m, p) kidney market.

1 pt. Under the greedy algorithm, how many links are in the pair graph at any point in time? Using this, justify why we do nothing at a criticality event.

1 pt. Define N to be the random variable of the number of pairs in the market at steady state (that is, suppose the system is already in steady state. We let the kidney exchange market run for some amount of time, and then look at the system). Compute L, the loss of the greedy algorithm.

1 pt. Suppose that the current market size is *exactly* z. Compute the probability that a new arriving pair finds a match.

5 pts. Show that if z is the market size which achieves the steady state conditions mentioned under Definition 4.1, then $z \geq \frac{m}{2d+1}$.

1 pt. Combine the parts 2 and 4 to prove the lower bound for L and hence Theorem 4.1.

One of the major issues with the greedy algorithm is that we never wait to see what other connections (P, D) could obtain. So, a natural question is: what is the value of waiting? That is, if we wait for someone to become critical before matching them, can we do a lot better?

It turns out the answer is *yes*: we can do exponentially better. To this end, we define the patient algorithm.

Definition 4.5

The patient algorithm is the function $f : G \times \{A, C\} \times V \to M$ which does the following.

- If we are in an arrival event, do nothing.
- If (P, D) had a criticality event and has some link (P', D'), match them and return $M = \{((P, D), (P', D'))\}$.

- If (P, D) has no links, do nothing.

Theorem 4.2
The loss L of the Patient algorithm, as $m \to \infty$ and d stays fixed, satisfies $L \leq \frac{e^{-d/2}}{2}$.

To prove this, we will focus on the *distribution* of the number of patient donor pairs in the market. Toward this, we need a helpful lemma on concentrated random variables.

Definition 4.6
For $c_1, c_2, m > 0$, define a discrete random variable X as (c_1, c_2, m) well-concentrated if

$$\mathbf{P}\left[|X - \mathbf{E}[X]| > k\sqrt{m}\right] \leq c_1\sqrt{m}e^{-c_2k^2}$$

.

You may use the fact that $1 + x \leq e^x$ for any x without proof.

Problem 4.3 (Bounds with well-concentrated, 18 points)
Suppose that X is (c_1, c_2, m) well-concentrated and always satisfies $X \geq 0$. In addition, suppose that X takes on only nonnegative integer values. In this problem, we will give an upper bound on $\mathbf{E}\left[X(1 - \frac{d}{m})^X\right]$ for fixed d and $m \to \infty$.

4 pts. Prove that for any nonnegative random variable Y, $\mathbf{E}[Y] \leq k + \int_k^\infty \mathbf{P}[Y > y]\, dy$.

6 pts. Suppose that $f(x)$ taking in real numbers is an increasing function and $g(x)$ a decreasing function. Suppose that X is in addition a random variable taking on each value in $\{1, 2, \ldots, n\}$ with probability $\frac{1}{n}$. Prove that

$$\mathbf{E}[f(X)g(X)] \leq \mathbf{E}[f(X)]\,\mathbf{E}[g(X)]\,.$$

The same fact is true for any nonnegative random variable X, which you can use without proof after this part.

8 pts. Let $\varepsilon > 0$ and $d > 1$. Prove that

$$\mathbf{E}\left[X\left(1 - \frac{d}{m}\right)^X\right] \leq \mathbf{E}[X]\, e^{-\frac{d\mathbf{E}[X]}{m}}(1 + \varepsilon).$$

for all large enough m.

To conclude, we have proven that $\mathbf{E}\left[X\left(1 - \frac{d}{m}\right)^X\right] \leq \mathbf{E}[X]\, e^{-\frac{d}{m}\mathbf{E}[X]}$ when $m \to \infty$, for fixed d.

We now see how to use this result to analyze the patient algorithm.

Problem 4.4 (Analysis of patient in kidney market, 8 points)
We are in an (m, p) kidney exchange market.

1 pt. Suppose that there are currently z pairs in the market. If a pair becomes critical, what is the probability that they have a match?

2 pts. As before, let N be the random variable of the number of pairs in the market at steady state. Compute L, the loss of the patient algorithm, as a function of N.

3 pts. Show that $\mathbf{E}[N] \geq \frac{m}{2}$. (Hint: use the alternative description of a steady state).

2 pts. Suppose that N is (c_1, c_2, m) well-concentrated (this is true, but we will not prove it here). Prove Theorem 4.2.

What are the takeaways from the kidney exchange market? Probably the most important takeaway is the importance of waiting: if d is large then waiting leads to a much lower number of people not being matched. So, even though in this queuing system we have the Little's Law relation $\overline{N} = m\overline{T}$, when we split our output into types (here these are matched and unmatched), it ends up being worth it sacrificing T of one to decrease the probability of the other.

In addition, this problem gives us a foray into matching theory. This problem is an example of a *dynamic online matching*. That is, the set of nodes and edges is constantly changing. Such change makes it difficult to construct an optimal algorithm (there do exist algorithms better than the patient algorithm, at least heuristically). In the next section, we will look at simpler scenarios of matchings.

5 Fixed Matchings

To look at matchings properly, let us begin with some graph theory.

5.1 Graph Theory and Matchings definitions

Definition 5.1

- A ***graph*** is denoted as $G = (V, E)$ where V is some collection of vertices and E is a collection of pairs of distinct vertices. We will often work with undirected graphs: that is, E is a collection of unordered pairs of vertices. Commonly, we write $n = |V|$ and $m = |E|$.
- A bipartite graph G is commonly written as $G = (X, Y, E)$ where $X \cup Y = V$, they are disjoint, and all edges have one endpoint in X and one in Y.
- The degree of a vertex $\deg(v)$ is the number of edges $e \in E$ with $v \in e$.
- The neighborhood of a vertex v (written $N(v)$) is the set of all u such that $\{u, v\} \in E$.
- A list of vertices $v_1, v_2, \ldots, v_k$ is called a walk if $\{v_i, v_{i+1}\} \in E$ for all i. A walk is called a path if additionally all of the v_i are distinct. A walk is called a cycle if $v_1 = v_k$ and all other nodes are distinct from them.

In the kidney exchange market, V is the set of patient-donor pairs and E is the set of links.

For practice, here is one of the most useful results in graph theory.

Problem 5.1 (Handshake Lemma, 2 points)
Let $G = (V, E)$ be an undirected graph. Prove that $\sum_{v\in V} \deg(v) = 2|E|$. This is called the Handshake Lemma.

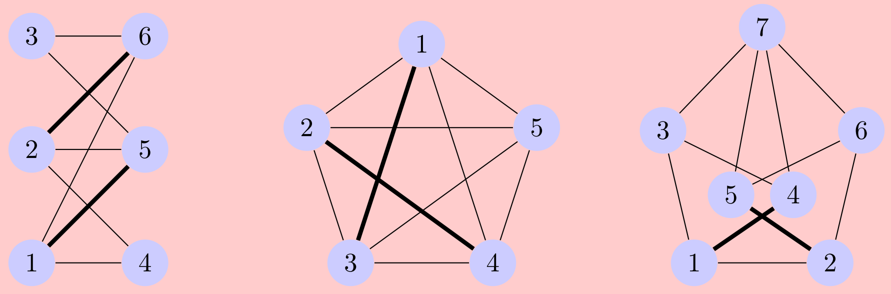
Now we define some important definitions of matchings.

Definition 5.2
Let $G = (V, E)$ be an undirected graph. For a subset $S \subseteq E$, let $V(S)$ be the set of endpoints of these edges.

- A subset $M \subseteq E$ is called a matching if $|V(M)| = 2|M|$: that is, all the edges are "disjoint" and have no overlapping endpoints.
- A matching M is called **maximal** if there are no edges $e \in E \setminus M$ with $M \cup \{e\}$ still being a matching.
- A matching M is called **maximum** if $|M|$ is largest possible among all matchings on G. A graph can have multiple maximum matchings. If $|V(M)| = |V|$ as well, we call M a **perfect matching**.

Problem 5.2 (Practice with matchings, 3 points)

For each of these 3 graphs, say whether the highlighted edges form a maximal matching and whether they form a maximum matching. If they do not form a maximum matching, find a maximum matching. No justification required.



5.2 Algorithms for Static Matchings

Matchings are nice in graphs precisely because of their application to real world problems: much like kidney exchange, an (embarassingly) large number of economics-related problems can be written through matchings. Some other examples are medical residency matching and school system matching. So, a natural question to ask is: can we find matchings efficiently?

We will measure efficiencies of algorithms in terms of m (the number of edges) and n (the number of vertices). In particular, we will say that any time we "ask" for a vertex or edge, this takes time 1. So, an algorithm looking at every vertex in a graph takes time n and an algorithm looking every edge in a graph takes time m.
So, we can look at algorithms as having runtime as a function of m and n. The algorithms we will be working with here will have *polynomial* running time in both: that is, there is a multivariate polynomial $p(m, n)$ giving an upper bound on the runtime of our algorithm for large enough m, n.

There are some basic operations that we will take as constant for the remainder of this round, for simplicity. We list them here.

Theorem 5.1

The following take time 1.

- Given a list of vertices, check if a vertex is present in the list.
- Given a list of edges, check if an edge is present in the list.
- Given a list of edges, check if a vertex is an endpoint of some edge in the list.
- Add a vertex (resp. edge) to a list of vertices (resp. edges)

Problem 5.3 (Finding a maximal matching, 3 points)

Let $G = (V, E)$ be a graph. Propose an algorithm which computes a maximal matching in time at most $c \cdot m$, and prove this running time bound. In addition, find the c of your algorithm.
Hint: think about the definition of a maximal matching.

Unfortunately, finding a maximum matching is a little bit harder, and we need a few more definitions.

Definition 5.3

Let G be a graph and M be a matching.

- An alternating path with respect to M is a path $v_1, v_2, \ldots, v_k$ such that **every other** consecutive pair $\{v_i, v_{i+1}\} \in M$ (note: we make no suppositions of whether the first pair is in M).

- An augmenting path with respect to M is an alternating path $v_1, v_2, \ldots, v_k$ such that both v_1, v_k are not in $V(M)$.

With this definition in mind, we can present a more useful way to look at maximum matchings.

Problem 5.4 (Berge's Lemma (Max Matching), 5 points)

Let G be a graph. Prove that M is a maximum matching if and only if there is no augmenting path with respect to M.

It turns out that on bipartite graphs, this "immediately" gives an algorithm for computing maximum matchings. Although this observation also gives an algorithm for general graphs, this is outside the scope of this power round.

Problem 5.5 (Efficiently finding max matching, 15 points)

Let $G = (X, Y, E)$ be a bipartite graph.

8 pts. Suppose that M is a matching in G. Propose an algorithm which either finds an augmenting path with respect to M or returns that none exist, and prove that your algorithm satisfies this guarantee. Your algorithm must run in time at most $c(n + m)$ for some suitable constant c.

Hint: What does an augmenting path in a bipartite graph look like?

4 pts. Prove the runtime of your algorithm satisfies the above.

3 pts. Propose an algorithm for computing a maximum matching in G, and give an upper bound on its runtime (don't worry about optimizing constants).

You may have noticed that finding a maximum matching is far more work than finding a maximal matching (at least, considering the number of points each problem is worth). Is there a good reason for this? More specifically, is there a large gap between the sizes of maximal and maximum matchings ala the greedy and patient algorithm for kidney exchange?

Problem 5.6 (Maximal matchings aren't so bad, 3 points)

Let M be a maximal matching in a graph G, and M' a maximum matching. Prove that $|M| \geq \frac{1}{2}|M'|$.

6 Dynamic Matchings

The static matching setting, while applicable in the real world, is sometimes not the one we really care about: at times, we may have people or things arriving dynamically and we want to match them when they arrive.

In particular, let $G = (\{\}, Y, \{\})$ be a bipartite graph (that is, there are no vertices in X yet and also no edges as a result). Vertices of X begin arriving, along with their edges to Y. When a vertex $x \in X$ and its edges arrive, an algorithm can match it to some $y \in Y$ or not. If not, then x can never be matched again in the future.

To formalize, we have the following definition.

Definition 6.1

An online matching algorithm is a function $f : G \times X \times \mathcal{P}(Y) \to Y \cup \{\perp\}$ where

- G is the currently uncovered graph: it contains the current information about which pairs (x, y) are matched.
- X denotes the new vertex that has just arrived.
- $\mathcal{P}(Y)$ denotes the powerset of Y: this encodes the neighbors of x.
- The return value says that either we match (x, y) or we do not match x.

We define $\mathsf{ALG}_f(G, L)$ as the number of matched edges when we run f on a graph G with arrival order L. We drop the subscript f when it is clear from context.

We can then compute the quality of an algorithm by comparing it to the maximum matching, which would be the optimal “omniscient” (knows everything that will happen) algorithm.

Definition 6.2

We define what it means for an algorithm to perform well in the online setting.

- The *performance* of a matching algorithm f on an input graph G and arrival order L is defined as $\frac{\mathsf{ALG}(G,L)}{|M|}$ where M is a maximum matching of G.
- The competitive ratio c of an algorithm is defined as the minimum performance over any possible input. That is, $c = \min_{G,L} \frac{\mathsf{ALG}(G,L)}{|M(G)|}$.

How good of a competitive ratio can we expect to get if we can only consider deterministic algorithms?

Problem 6.1 (Deterministic Online Matchings, 6 points)

We will show in this problem that determinism can only go so far.

4 pts. Propose an algorithm which achieves a competitive ratio of $\frac{1}{2}$.

2 pts. Prove that this is tight: there can be no algorithm which achieves better than $c = \frac{1}{2}$.

Hint: You should only need 2 *vertices in* X*.*

What if we allow randomness in a solution? In particular, suppose that we are allowed to make each matching decision with some probability: can we improve our performance? It turns out the answer is yes: in fact, we can achieve an *expected* competitive ratio of $1 - \frac{1}{e}$.

Definition 6.3

A randomized algorithm ALG for online matching is a decision process where whenever $x \in X$ arrives, ALG chooses each match $x \to y$ (or x unmatched) with some probability depending on what the graph currently looks like and the information it knows about x. Crucially, it cannot assume anything about future arriving x. In the matching function f framework, we have that $f : G \times X \times \mathcal{P}(Y) \to Y \cup \{\perp\}$ actually is a random variable, which selects $y \in Y$ or no match, each with some probability.

Now, under this definition of randomized algorithms, we can define a similar notion of competitive ratio.

Definition 6.4

Let $G = (X, Y, E)$ be a bipartite graph and L an ordering on adding the vertices of X to $(\{\}, Y, \{\})$. As before, let $\mathsf{ALG}(G, L)$ be a random variable denoting the size of the matching returned by the algorithm ALG on this input.
Then, the expected competitive ratio is defined as $c = \min_{G,L} \frac{\mathbf{E}[\mathsf{ALG}(G,L)]}{|M(G)|}$.

To bound this in expectation, we will need a notion of *fractional* matchings (the regular matchings we had before are sometimes called *integral* matchings) which we will show to be equivalent to randomized matchings.

Definition 6.5

A *fractional matching* on a graph $G = (X, Y, E)$ is a collection of variables $m_{x,y}$ for $(x, y) \in E$. satisfying $\sum_{y \in N(x)} m_{xy} \leq 1$ when $x \in X$ and $\sum_{x \in N(y)} m_{xy} \leq 1$ when $y \in Y$.

Intuitively, this is saying that instead of matching x to a specific vertex $y \in Y$, we split this matching into possibilities for each neighbor of x. We will make this intuition precise in the following problem.

Problem 6.2 (Randomized = Fractional, 6 points)

Let ALG be a randomized algorithm for the online matching problem. Then, prove that there exists a deterministic $\mathsf{FracALG}$ which (given the same graph G and order L) satisfies

$$\mathbf{E}\left[\mathsf{ALG}(G, L)\right] = \sum_{(x,y) \in E} m_{xy}$$

at the end of the process.
Briefly describe why we can also go from any deterministic fractional algorithm $\mathsf{FracALG}$ to a randomized ALG.

Hence, if we prove that the competitive ratio of fractional algorithms is at least some c, this implies that the competitive ratio of randomized integral algorithms is also at least c. To this end, we prove the following theorem in the next set of problems.

Theorem 6.1

The competitive ratio of any randomized integral algorithm is at most $1 - \frac{1}{e}$, and this bound is tight.

To prove this theorem, we introduce a fundamental fractional algorithm known as the *water level* algorithm. Toward this algorithm, we will need a bit more notation about fractional matchings.

Definition 6.6

For any fractional matching $M = \{m_{xy}\}$ on the graph $G = (X, Y, E)$, define $W_x = \sum_{y \in N(x)} m_{xy}$ and $W_y = \sum_{x \in N(y)} m_{xy}$. We call these the "load" of a vertex (how important it is, in some sense).

The intuitive description behind the water level algorithm (before presenting it) is as follows: suppose there is a set of containers of water, each corresponding to a vertex $y \in Y$. When a new vertex $x \in X$ arrives, it brings with it 1 gallon of water. Then, it can distribute this gallon amongst the edges (x, y) by increasing m_{xy}. However, it does this addition to try to "even out" the sets W_y as much as possible. In particular, if x neighbors every vertex $y \in Y$, and the water weights W_y are all equal, then x will set m_{xy} to be the same for all y.

Here is the actual algorithm.

```
Water Level Algorithm (Step)

**Input:** A graph $G = (X', Y, E)$ with fractional matching $M = \{m_{xy}\}$, and a vertex $x \in X \setminus X'$ with its neighborhood $N(x) \subseteq Y$.

**Output:** A fractional matching $M' = \{m_{xy}\}$ on $G = (X' \cup \{x\}, Y, E \cup N(x))$.

  • Set the "budget" $w = 1$.
  • Order the neighbors $y_i \in N(x)$: $W_{y_1} < W_{y_2} < W_{y_3} < \ldots < W_{y_k}$. Assume that there are no ties (see the discussion after this algorithm for how to deal with ties).
  • **While** $w > 0$ and $W_{y_1} < 1$:
      – Set $m_{x,y_1} \leftarrow m_{x,y_1} + (W_{y_2} - W_{y_1})$. If $W_{y_2} - W_{y_1} > w$, then set $m_{x,y_1} \leftarrow m_{x,y_1} + w$.
      – Set $w \leftarrow w - (W_{y_2} - W_{y_1})$.
      – Reorder the neighbors $N(x)$ according to their new loads $W$.
  • Output $\{m_{xy}\}$
```

Note that we swept under the rug issues of equal loads, even though this is exactly what happens after one iteration of the while loop.

To combat this case, we will increase *both* W_{y_1} and W_{y_2}: in particular, we will try to set both

$$\begin{aligned} m_{x,y_1} &\leftarrow m_{x,y_1} + (W_{y_3} - W_{y_2}) \\ m_{x,y_2} &\leftarrow m_{x,y_2} + (W_{y_3} - W_{y_2}). \end{aligned}$$

If $2(W_{y_3} - W_{y_2}) > w$ (our budget), then we we set $m_{x,y_1} \leftarrow m_{x,y_1} + \frac{w}{2}$ and similarly for $m_{x,y_2} \leftarrow m_{x,y_2} + \frac{w}{2}$.

What this does is effectively say that if $W_{y_1} = W_{y_2}$ when the budget $w > 0$, then we will keep this invariant true forever. If $W_{y_2} = W_{y_3}$ as well, we extend the same discussion as above to instead add $\frac{w}{3}$, and so on. In this way, the water level algorithm always leaves the sets $\{W_y : y \in N(x)\}$ with at least as much equality as they had before starting (in fact, strictly more equality).

6.1 Analysis of Water Level Algorithm

To analyze the water level algorithm, we use a technique known as *money analysis*. We let a_x be the money of $x \in X$ at any point, and b_y be the money of $y \in Y$. Note that money can be fractional or irrational: all that matters is that it is nonnegative. When $x \in X$ arrives, it comes with \$1 which it is allowed to allocate to a_x, b_y for $y \in N(x)$. So, whenever we increase m_{xy} in the Water Level Algorithm step, we will change a_x, b_y in such a way that $\sum_{(x,y)\in E} m_{xy} = \sum_x a_x + \sum_y b_y$. That is, the total amount of allocated money is equal to the total load allocation (just not necessarily in the same proportions).

The natural question to ask now is: "why is this useful?".

Problem 6.3 (Money is competitive, 3 points)

Let $M = \{m_{xy}\}$ be the Water Level fractional matching of a graph $G = (X, Y, E)$, and let a_x, b_y be the moneys associated to $x \in X$ and $y \in Y$ respectively.
Prove that if for all $(x, y) \in E$ we have $a_x + b_y \geq c$, then $\mathsf{FracALG}$ is c-competitive.
Hint: Let $M^* = \{m^*_{xy}\}$ be the optimal matching, and find a way to compare them.

One analogy is to the deterministic integral case: there, we can guarantee that at least one endpoint of every edge in a matching is contained in a maximal matching. Hence, splitting money 50/50 for matches gives the desired $\frac{1}{2}$ approximation (we are intentionally leaving out details: this is not the intended approach for deterministic matchings).

Although it may seem a bit magical at first glance, we will let $g : [0,1] \to [0,1]$ be a differentiable increasing function and set $b_y = \int_0^{W(y)} g(s)\,\mathrm{d}s$. The intuition here is saying that small changes to m_{xy} correspond to changes $g(W_y)$ in the money of y (correspondingly, small changes to m_{xy} yield changes $1 - g(W_y)$ for the money of x).

g being increasing essentially tells us that as y fills up, we should allocate more money to it: if later an x' with $y \in N(x')$ arrives, y should serve as "protection" so that $a_x + b_y \geq c$ is satisfied. Then, integrating over small changes to m_{xy} gives the integral representation written here.

The final step here is choosing a function g which maximizes $a_x + b_y$.

Problem 6.4 (Analysis of Water Level, 9 points)

Fix an edge $(x, y) \in E$, and let W^f_y be the final load of y. Furthermore, let G be an antiderivative of g.

2 pts. Prove that $a_x + b_y \geq G(1) - G(0)$.

4 pts. Prove that $a_x + b_y \geq G(W^f_y) - G(0) + 1 - g(W^f_y)$.

3 pts. By some trickery, we can reduce our search to determining a function $g(z)$ satisfying $G(1) - G(0) = c$ and $G(z) - G(0) + 1 - g(z) = c$ for all $z \in [0,1]$. Find such a g, and on the way compute c.
Hint: When faced with an antiderivative...

We have now shown that there is an algorithm achieving the competitive ratio bound $1 - \frac{1}{e}$. To finish the proof of the theorem, we show that there cannot exist a better bound.

Problem 6.5 (Optimality of Water Level, 14 points)

Consider the graph $G = (X, Y, E)$ where $|X| = |Y| = n$ (so we can label of each of these by $\{1, 2, \ldots, n\}$) and the edges are $E = \{(x, y) : y \geq x\}$.

1 pt. Show that the size of the maximum matching in G is n.

4 pts. Consider the arrival order $L = [1, 2, \ldots, n]$. Prove that as $n \to \infty$, the competitive ratio of Water Level on this input is $1 - \frac{1}{e}$.

Note: You may use without proof that $H_n \to \ln n + d$ as $n \to \infty$ for some constant d.

6 pts. Suppose ALG is a **randomized integral** algorithm for online bipartite matching.

We choose a random ordering L of $\{1, 2, \ldots, n\}$. Prove that

$$c = \frac{1}{n}\mathbf{E}\left[\mathsf{ALG}(G, L)\right] \leq \frac{1}{n}\sum_{j=1}^{n} \min\left(1, \sum_{i=1}^{j} \frac{1}{n-i+1}\right),$$

that is, the expected size of the matching formed (with expectation over the ordering L and the randomness of ALG) is bounded above.

Hint: Prove this for **deterministic** *integral algorithms first.*

3 pts. Suppose that the right hand side of the above expression approaches $1 - \frac{1}{e}$ as $n \to \infty$ (it does). Show that for any $\varepsilon > 0$, there is some n and ordering L_n so that $\mathbf{E}\left[\mathsf{ALG}(G, L_n)\right] \leq 1 - \frac{1}{e} + \varepsilon$ as $n \to \infty$.

The previous problems prove that Water Level always has $c \geq 1 - \frac{1}{e}$ for any input, and that there is an input achieving $1 - \frac{1}{e}$ in the limit. Therefore, $c = 1 - \frac{1}{e}$ for the Water Level algorithm. Furthermore, for any randomized integral algorithm (and hence any deterministic fractional algorithm) there is some input achieving $c \leq 1 - \frac{1}{e}$ in the limit which essentially shows that the Water Level algorithm is "optimal" in some sense: on a worst case input, it achieves a better competitive ratio than any other algorithm.

7 Stable Matchings

In the final section of this power round, we will see an application of matchings with explicit real world applicability.

Let H be a set of high school students and S a set of colleges. In this unrealistic world, we have $|H| = |S| = n$: that is, there are the same number of high schoolers and colleges. We wish to construct a matching in the graph $G(H, S, \{(h, s) : h \in H, s \in S\})$: that is, the graph has every possible edge.

Finding a matching here is easy: we can just greedily match students to schools. The problem is made more difficult with the addition of *preference lists*.

Definition 7.1

A stable matching problem is a graph $G = (H, S)$ with each $h \in H$ and $s \in S$ having a preference list: for each $h \in H$ this is an ordering $L_h = [s_1, s_2, \ldots, s_n]$ and for $s \in S$ this is $L_s = [h_1, h_2, \ldots, h_n]$. Note that we drop the set of edges, since we will always assume that every edge exists.

Preference lists (from the student view) encode where a student would like to go. For example, a student h may have the preference list [Stanford, CMU, . . .] indicating that their top choice is Stanford, the next CMU, and so on. So, we would say that Stanford $\succ$ CMU in L_h (read this as "Stanford is strictly higher preference than CMU on the preference list of h"). Importantly, each preference list ranks *every* school (respectively, every student).

We may now look at the property of stability in matchings.

Definition 7.2

Let M be a matching of $G = (H, S)$. We say a pair (h_1, s_2) is an **unstable pair** if $s_2 \succ s_1$ in L_{h_1} and $h_1 \succ h_2$ in L_{h_2}, where $(h_1, s_1), (h_2, s_2) \in M$. That is, (h_1, s_2) prefer each other to their current matches.
A matching M is called **stable** if it has no unstable pairs.

The question now arises: does every stable matching problem have a stable matching? The answer, surprisingly, is yes.

Theorem 7.1

Let $(G = (H, S), L = \{L_h : h \in H\} \cup \{L_s : s \in S\})$ be a stable matching problem. Then, there exists a stable matching M.

We will prove this theorem constructively: that is, we will give an algorithm which finds a stable matching of G. The algorithm itself is called the "proposal algorithm" (the stable matching problem was originally written with marriage as an example).

Proposal Algorithm

Input: A stable matching problem $(G = (H, S), L = \{L_h : h \in H\} \cup \{L_s : s \in S\})$.

Output: A stable matching M.

```
• For each $h \in H$, keep track of the highest preference person in $L_h$ that they have not yet proposed to.
• Let $M$ be the empty matching.
• While there exists an unmatched $h \in H$:
    – $h$ proposes to $s$, the top unproposed-to person in $L_h$.
    – If $s$ is unmatched, add $(h, s)$ to $M$.
    – If $(h', s)$ are matched and $h \succ h'$ in $L_s$, remove $(h', s)$ from $M$ and add $(h, s)$.
    – Else, continue the while loop.
• Return $M$.
```

We will now analyze this "simple" algorithm.

Problem 7.1 (Properties of proposal algorithm, 4 points)

We show some base properties of the algorithm: that it terminates and returns a valid matching.

2 pts. Suppose that $|H| = |S| = n$. Prove that the proposal algorithm terminates and give an upper bound on the number of iterations of the loop.

2 pts. Prove that the returned matching M is full: that is, $|M| = n$.

Next, we prove that this algorithm indeed returns a stable matching.

Problem 7.2 (Stability of proposal algorithm, 4 points)

Prove that the proposal algorithm finds a stable matching.

Hence, we have proven the stable matching theorem. However, there is more to explore here as well.

7.1 Applications of Stable Matchings

The structure of proposing might suggest that there is more hidden structure that we are missing in the proposal algorithm: in fact, we can say more about this matching.

Definition 7.3

For a student h, let $\text{best}(h)$ be the highest $s \in L_h$ that h could be matched to in any stable matching. Similarly, let $\text{worst}(s)$ be the lowest $h \in L_s$ that s could be matched to in any stable matching.
We say a matching M is **student-optimal** if $M = \{(h, \text{best}(h)) : h \in H\}$ and we call it **school-pessimal** if $M = \{(\text{worst}(s), s) : s \in S\}$.

It may make sense that the proposal algorithm is worse for schools than for students: if s is unmatched, then it has to accept any matching request is receives. The surprising fact is that it is both school-pessimal and student-optimal in addition to this.

Problem 7.3 (Optimality of proposal algorithm, 8 points)

Prove that the matching M returned by the proposal algorithm is both student-optimal and school-pessimal.

A natural extension of best and worst is the idea of *soulmates*.

Definition 7.4

A pair (h, s) are called soulmates if they are matched in every stable matching M.

Historically, the name "soulmates" comes from the original marriage version of stable matching.

Problem 7.4 (Soulmates, 3 points)

Give an algorithm which, given a stable matching problem (G, L), determines if there is a pair of soulmates (and if so, returns such a pair).

We end the power round by extending the proposal algorithm slightly to be more realistic in terms of students and schools.

Problem 7.5 (Realistic schools, 5 points)

Suppose that we have a set of $m \cdot n$ students H for integer $m \geq 1$, and a set of n schools S. Each school has m spots in it to accept students into. Each student has a preference list of schools L_h and each school a preference list of students L_s, as before.
An unstable pair in this setting is an unmatched pair (h, s) where $h \succ h'$ in L_s for some matched (h', s) and $s \succ s'$ in L_h (where (h, s') are matched). Prove that there exists a stable matching in this setting: that is, there are still no unstable pairs.
Hint: How can you simulate $m \cdot n$ schools?