

**Problem Set 1**

CS265/CME309, Winter 2026

Due: Friday 1/16, 11:59pm on Gradescope

Please follow the homework policies on the course website.

1. **(4 pt.) [Coin Flipping]** Suppose you are flipping a fair coin repeatedly. (Here, a *fair coin* means that it comes up heads with probability  $1/2$  and tails with probability  $1/2$ ; each flip is independent).

- (a) **(4 pt.)** What is the expected number of flips until you get two heads in a row (counting both heads)? Justify your answer.

[**HINT:** *If you find yourself doing a tedious computation, try to think of a simpler way. Perhaps look to the mini-lecture on linearity of expectation for some inspiration...* ]

- (b) **(0 pt.) [Optional; this part won't be graded]** What is the expected number of flips until you get  $k$  heads in a row (counting all  $k$  heads)? Justify your answer.

[**HINT:** *You may want to do it for  $k = 3$  first to get some intuition.* ]

[**HINT:** *Depending on how you approach it, the following facts might be helpful:  $\sum_{j=1}^t \frac{j}{2^j} = \frac{1}{2^t}(2^{t+1} - t - 2)$ , and  $\sum_{j=1}^t 2^{-j} = 1 - 2^{-t}$ .* ]

**Note:** *There are many ways to do this problem, but there is a not-too-long way to do it by doing something similar to something we did in the mini-lecture on linearity of expectation.*

2. **(10 pt.) [More coin flipping]** Suppose you are given a fair coin and want to use it to “simulate” a coin that lands heads with probability exactly  $1/3$ . Specifically, you will design an algorithm whose only access to randomness is by flipping the fair coin (repeatedly, if desired), and your algorithm should return “heads” with probability exactly  $1/3$  and “tails” with probability exactly  $2/3$ .

- (a) **(4 pt.)** Prove that it is *impossible* to do this if the algorithm is only allowed to flip the fair coin at most 1,000,000,000 times.

[**HINT:** *Read the next two parts of the problem first...* ]

- (b) **(4 pt.)** Design an algorithm for the above task that flips the fair coin a finite number of times *in expectation*.

- (c) **(2 pt.)** Show that for *any* value  $v$  in the interval  $[0, 1]$ , there is an algorithm that flips a fair coin at most 2 times in expectation, and outputs “heads” with probability  $v$  and “tails” with probability  $1 - v$ .

**Note:** if you do this part correctly, you can write “follows from (c)” in part (b) get full credit for both parts. But doing part (b) first might be helpful...

[**HINT:** *Think about representing the desired probability in its binary representation.* ]

3. (8 pt.) [Counting small cuts]

Recall that a cut of an undirected graph  $G = (V, E)$  is a partition of the vertices  $V$  into nonempty disjoint sets  $A$  and  $B$ . A *min cut* of  $G$  is a cut that minimizes the number of edges that cross the cut (that is, edges that have one endpoint in  $A$  and one endpoint in  $B$ ).

In the following problems, assume  $G$  is a connected graph on  $n$  vertices (i.e., there is no cut with 0 edges that cross it).

- (a) (2 pt.) A graph may have many possible min cuts. Prove that  $G$  has *at most*  $n(n-1)/2$  min cuts.

[HINT: *What did we prove about Karger's algorithm?* ]

- (b) (2 pt.) Show that part (a) is tight; for every  $n \geq 2$ , give a connected graph on  $n$  vertices with exactly  $n(n-1)/2$  min cuts.

- (c) (4 pt.) Let  $\alpha$  be a positive integer. Suppose that any min cut of  $G$  has  $k$  edges that cross the cut. An  $\alpha$ -small cut of  $G$  is a cut that has at most  $\alpha k$  edges that cross the cut. Prove that the number of such cuts is at most  $O(n^{2\alpha})$ .

[Note: If you find it easier, you'll still get full credit if you prove a bound of  $O((2n)^{2\alpha})$ .]

[HINT: *Consider stopping Karger's algorithm early and outputting a random cut in the contracted graph. What's the probability that this returns a fixed  $\alpha$ -small cut of  $G$ ?* ]

- (d) (0 pt.) [Optional: this part won't be graded] Let  $f(n, \alpha)$  be the maximum number of  $\alpha$ -small cuts that an  $n$  vertex graph can have. What are the tightest upper and lower bounds you can find for  $f(n, \alpha)$ ?

4. (4 pt.) [Logistics] We have a semi-out-of-class midterm in this class: The midterm is **Wednesday February 11, 9:00am-11:50am**. Note that class is 10:30am-11:50am, so the midterm includes an extra 90 minutes before class.

- (a) (1 pt.) Is the midterm on your calendar? (The correct answer is "yes.")  
(b) (3 pt.) Can you make the midterm?

For non-CGOE students without OAE exam accommodations:

- If you can make the exam, write "Yes, and I understand that it is my responsibility to alert the course staff ASAP if that changes." Note that we may not be able to make last-minute accommodations for forecast-able conflicts (e.g., if you email us the day before the exam about a conflicting class that you have been enrolled in for a month, or about travel that you have had booked for weeks).
- If you cannot make the exam, please write "No, and I have emailed the course staff list describing my conflict," and immediately email the course staff list with your conflict if you haven't already. We'll work with you to figure out an alternative plan.

*If you have OAE accommodations on exams*, please email the course staff list your OAE letter if you haven't already, and we will work with you to figure out the details. In that case, write "I have OAE accommodations and I have emailed the course staff list."

*CGOE Students*: You have a separate process for taking exams; please write "I am a CGOE student and I will work with the CGOE office to schedule exams."

**Note**: Please also alert us ASAP about any conflicts with the final exam: Monday March 16, 3:30pm-6:30pm.

5. (0 pt.) [Optional: this entire problem won't be graded.] [Perfect Matchings.]

Let  $G = (V, E)$  be a *bipartite graph* with  $n$  vertices on each side. A *perfect matching* in  $G$  is a list of edges  $M \subset E$  so that every vertex in  $V$  is incident to exactly one edge.

For example, here is a bipartite graph  $G$  (on the left), and a perfect matching in  $G$  (shown in **bold** on the right):



Your goal is to determine if the graph  $G$  has a perfect matching.

There are efficient deterministic algorithms for this problem, but in this problem you'll work out a simple randomized one.<sup>1</sup>

- (a) Recall that the *determinant* of an  $n \times n$  matrix  $A$  is given by

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^n A_{i,\sigma(i)},$$

where the sum is over all permutations  $\sigma : \{1, \dots, n\} \rightarrow \{1, \dots, n\}$ , and where  $\text{sgn}(\sigma)$  denotes the signature<sup>2</sup> of the permutation  $\sigma$ . (For example, if  $n = 3$ , then the function  $\sigma : \{1, 2, 3\} \rightarrow \{1, 2, 3\}$  that maps  $1 \mapsto 2$ ,  $2 \mapsto 1$ ,  $3 \mapsto 3$  is a permutation in  $S_3$ . The signature of  $\sigma$  happens to be  $-1$ , although as noted in the footnote, if you haven't seen this definition before, don't worry about it).

Let  $A$  be the  $n \times n$  matrix so that

$$A_{ij} = \begin{cases} x_{ij} & (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

where the  $x_{ij}$  are variables, and  $(i, j) \in E$  if and only if the  $i$ -th vertex on the left and the  $j$ -th vertex on the right are connected by an edge in  $G$ . Notice that  $\det(A)$  is a multivariate polynomial in the variables  $x_{ij}$ .

Explain why  $\det(A)$  is not identically zero if and only if  $G$  has a perfect matching.

- (b) Use the part above to develop a randomized algorithm for deciding whether or not there is a perfect matching. Your algorithm should run in  $O(n^3)$  operations. If  $G$  has no

<sup>1</sup>This randomized algorithm has the advantage that (a) it generalizes to all graphs (not necessarily bipartite), and (b) it can be parallelized easily. Moreover, it's possible to generalize it to actually recover the perfect matching (and not just decide if there is one or not).

<sup>2</sup>The *signature* of a permutation is defined as  $-1$  if the permutation can be written as an odd number of transpositions, and  $+1$  otherwise. **The exact definition isn't important** to this problem, all you need to know is that it's either  $\pm 1$  in a way that depends on  $\sigma$ .

perfect matching, your algorithm should return “There is no perfect matching” with probability 1. If  $G$  has a perfect matching, your algorithm should return “There is a perfect matching” with probability at least 0.9.

You should clearly state your algorithm and explain why it has the desired properties.

**[HINT:** *You may use the fact that one can compute the determinant of a matrix  $A \in \mathbb{R}^{n \times n}$  in  $O(n^3)$  operations.* ]

- (c) Extend your algorithm to actually *return* a perfect matching. And/or, extend your algorithm to non-bipartite graphs. As a hint, consider the matrix

$$A = \begin{cases} x_{ij} & \{i, j\} \in E \text{ and } i < j \\ -x_{ji} & \{i, j\} \in E \text{ and } i \geq j \\ 0 & \text{else} \end{cases}$$