*Note*: Your TA probably will not cover all the problems. This is totally fine. The discussion worksheets are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

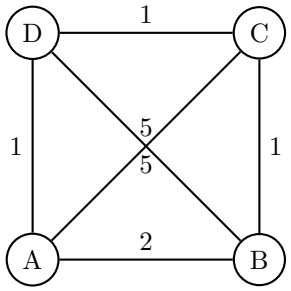
# 1 Approximating the Traveling Salesperson Problem

Recall in lecture, we learned the following approximation algorithm for TSP:

1. Given the weighted complete graph $G = (V, E, w)$, compute its MST.
2. Run DFS on the MST from the previous step, and record the vertices visited in pre-order.
3. The output tour is the list of vertices computed in the previous step, with the first vertex appended to the end.

When $G$ satisfies the triangle inequality, this algorithm achieves an approximation factor of 2. But what happens when the triangle inequality does not hold?

Suppose we run this approximation algorithm on the following graph:




The algorithm will return different tours based on the choices it makes during its depth-first traversal.

1. Which DFS traversal leads to the best possible output tour?

2. Which DFS traversal leads to the worst possible output tour?

3. What is the approximation ratio given by the algorithm in the worst case for the above instance? Why is it worse than 2?

## 2 Boba Shops

A rectangular city is divided into a grid of $m \times n$ blocks. You would like to set up boba shops so that for every block in the city, either there is a boba shop within the block or there is one in a neighboring block (assume there are up to 4 neighboring blocks for every block). It costs $r_{ij}$ to rent space for a boba shop in block $ij$.

Write an integer linear program to determine on which blocks to set up the boba shops, so as to minimize the total rental costs.

(a) What are your variables, and what do they mean?

(b) What is the objective function? Briefly justify.

(c) What are the constraints? Briefly justify.

(d) Solving the non-integer version of the linear program yields a real-valued solution. How would you round the LP solution to obtain an integer solution to the problem? Describe the algorithm in at most two sentences.

(e) What is the approximation ratio of your algorithm in part (d)? Briefly justify.

# 3 Randomization for Approximation

Oftentimes, extremely simple randomized algorithms can achieve reasonably good approximation factors.

(a) Consider Max 3-SAT: given an instance with $m$ clauses each containing exactly 3 distinct literals, find the assignment that satisfies as many of them as possible. Come up with a simple randomized algorithm that will achieve an approximation factor of $\frac{7}{8}$ in expectation. That is, if the optimal solution satisfies $c$ clauses, your algorithm should produce an assignment that satisfies at least $\frac{7c}{8}$ clauses in expectation.

*Hint: use linearity of expectation!*

(b) Given a Max 3-SAT instance $I$, let $\mathsf{OPT}_I$ denote the maximum fraction of clauses in $I$ satisfied by any variable assignment. What is the smallest value of $\mathsf{OPT}_I$ over all instances $I$? In other words, what is $\min_I \mathsf{OPT}_I$?

*Hint: use part (a), and note that a random variable must sometimes be at least its mean.*

# 4 Randomized algorithms basics

**Philosophy of analyzing randomized algorithms.** The first step is to always identify a *bad event* – you want to identify when your randomness makes your algorithm fail. We will review some techniques from class using the following problem as a way to learn how to use them in the context of randomized algorithms.

Let $G$ be a bipartite graph with $n$ left vertices, and $n$ right vertices on $n^2 - n + 1$ edges.

- Prove that $G$ always has a perfect matching.
- Give a polynomial in $n$ time algorithm to find this perfect matching.

We will now use some common techniques to analyze the following algorithm `BlindMatching`:

- Let $\boldsymbol{\pi}$ and $\boldsymbol{\sigma}$ be independent and uniformly random permutations of $[n]$.
- If $\{\boldsymbol{\pi}(1), \boldsymbol{\sigma}(1)\}, \{\boldsymbol{\pi}(2), \boldsymbol{\sigma}(2)\}, \ldots, \{\boldsymbol{\pi}(n), \boldsymbol{\sigma}(n)\}$ is a valid matching output it.
- Else output `failed`.

**Union Bound.** Suppose $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are (not necessarily independent) Bernoulli random variables (i.e. random variables valued $\{0, 1\}$). Then, we have the following identity:

$$\Pr[\boldsymbol{X}_1 + \cdots + \boldsymbol{X}_n \geq 1] \leq \Pr[\boldsymbol{X}_1 = 1] + \Pr[\boldsymbol{X}_2 = 1] + \cdots + \Pr[\boldsymbol{X}_n = 1].$$

Now, using union bound, we analyze our algorithm for `BlindMatching`. Note that an output

$$M = (\{\boldsymbol{\pi}(1), \boldsymbol{\sigma}(1)\}, \ldots, \{\boldsymbol{\pi}(n), \boldsymbol{\sigma}(n)\})$$

is a valid perfect matching exactly when all edges of the form $\{\boldsymbol{\pi}(i), \boldsymbol{\sigma}(i)\}$ are present in $G$. A "bad event" happens if any of those pairs are not edges in $G$.

Let $\boldsymbol{X}_i$ be the indicator of the event that $\{\boldsymbol{\pi}(i), \boldsymbol{\sigma}(i)\}$ is *not* present in our graph.

1. What is the probability that $\boldsymbol{X}_i = 1$?

2. Use the union bound to upper bound the probability that $M$ is *not* a valid perfect matching.

3. Conclude that $G$ has a valid perfect matching.

The upper bound obtained on the probability of our bad event, i.e. of $M$ not being a valid perfect matching, is fairly high. In light of this, we introduce the technique of *amplification*.

**Amplification.** The philosophy of amplification is that if we have a randomized algorithm that fails with probability $p$, we can repeat the algorithm many times and aggregate the output of all the runs to produce a new output such that the failure probability of the randomized algorithm is significantly smaller. Now consider the following algorithm `SpamBlindMatching`:

- Run `BlindMatching` independently $T$ times.
- If at least one of the runs outputted a valid perfect matching, return the output of such a run.
- Else output `failed`.

To see the effectiveness of amplification, let us answer the following questions:

1. What is tight upper bound on the failure probability of `SpamBlindMatching`?

2. How large should we set $T$ if we want a failure probability of $\delta$?

Notice that the failure probability of `SpamBlindMatching` is not only lower than that of `BlindMatching`, but it can also be adjusted for based on the number of "amplifications" we make!

---

Now we switch gears and turn our attention to concentration phenomena and its usefulness in analyzing randomized algorithms.

**Markov's inequality.** Let $\boldsymbol{X}$ be a *nonnegative valued* random variable, then for every $t \geq 0$:

$$\Pr[\boldsymbol{X} \geq t] \leq \frac{\mathbf{E}[\boldsymbol{X}]}{t}.$$

1. Markov's inequality is *false* for random variables that can take on negative values! Give an example.

2. Give a tight example for Markov's inequality. In particular, given $\mu$ and $t$, construct a random variable $\boldsymbol{X}$ such that $\mu = \mathbf{E}[\boldsymbol{X}]$ and $\Pr[\boldsymbol{X} \geq t] = \frac{\mu}{t}$.

**Chebyshev's inequality.** Let $\boldsymbol{X}$ be any random variable with well-defined variance[1], then

$$\Pr\left[|\boldsymbol{X} - \mathbf{E}[\boldsymbol{X}]| > t\sqrt{\mathbf{Var}[\boldsymbol{X}]}\right] \leq \frac{1}{t^2}.$$

To see the above inequality in action, consider the following problem:

[1] In this course, all random variables will have well-defined variance (i.e. $\mathbf{Var}[\cdot] < \infty$).

Let $B$ be a bag with $n$ balls, $k$ of which are red and $n-k$ of which are blue. We do not have knowledge of $k$ and wish to estimate $k$ from $\ell$ independent samples (with replacement) drawn from $B$.

Let $\boldsymbol{X}$ be the number of red balls sampled.

1. What is $\mathbf{E}[\boldsymbol{X}]$?

2. What is $\mathbf{Var}[\boldsymbol{X}]$?

3. Choose a value for $\ell$ and give an algorithm that takes in $n$ and $\boldsymbol{X}$ and outputs a number $\widetilde{k}$ such that $\widetilde{k} \in [k - \varepsilon\sqrt{k}, k + \varepsilon\sqrt{k}]$ with probability at least $1 - \delta$.

# 5 4-cycles

We use $G(n,p)$ to denote the distribution of graphs obtained by taking $n$ vertices and for each pair of vertices $i,j$ placing edge $\{i,j\}$ independently with probability $p$.

(a) Compute the expected number of edges in $G(n,p)$?

(b) Compute the expected number of 4-cycles in $G(n,p)$?

(c) Describe, with proof of correctness, a polynomial time randomized algorithm that takes in $n$ as input and in poly$(n)$-time outputs a graph $G$ such that $G$ has no 4-cycles and the expected number of edges in $G$ is $\Omega(n^{4/3})$.