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 Breaking Encryption

After years of research, Horizon Wireless released an encryption algorithm E that encrypts an n-bit message x in time $O(n^2)$, i.e., it takes x and outputs a message $E(x)$ corresponding to it (with length $O(n^2)$ or shorter). Show that if $\mathsf{P} = \mathsf{NP}$, then this encryption algorithm can be broken in polynomial time. More precisely, argue that if $\mathsf{P} = \mathsf{NP}$, then the following decryption problem can be solved in polynomial time.

DECRYPT :

Input: An encrypted message m_e (encrypted using the algorithm E on an unkown input)

Output: Decryption x of the message m_e, i.e an n bit string x such that encrypting x produces m_e.

2 Public Funds

You are looking to build a new fence for your mansion, to keep out pesky people protesting profligate purchases. You have m bank accounts at your disposal to use to pay for your fence; each account i has a balance of b_i. You must choose one of n options for your fence; each fence j costs c_j dollars. You would like to withdraw from at most k of the bank accounts to build the fence, and due to peculiar UC accounting rules, if you use a particular bank account, you must use the whole balance (all b_m dollars).

Determine whether it is possible to exactly pay for some fence j; that is, whether there is a j between 1 and n such that you can withdraw exactly c_j dollars given the bank account balances $b_1, \ldots, b_m$, the fence costs $c_1, \ldots, c_n$, and k.

Your task is to prove that Public Funds is **NP**-complete.

(a) Prove that Public Funds is in **NP**.

(b) Prove that Public Funds is **NP**-hard by providing a reduction from Subset Sum.

Note: recall that, to rigorously prove the correctness of a reduction from A to B, you must show two things:

1. *If an instance of A has a solution, then the transformed instance of B has a solution.*
2. *If an instance of B in the format of the transformation has a solution, then the corresponding instance of A has a solution.*

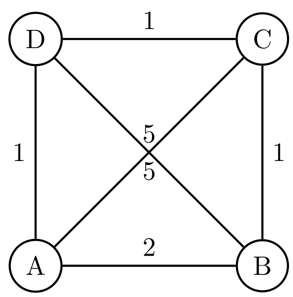
3 Approximating the Traveling Salesperson Problem

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

1. Given the complete graph $G = (V, E)$, compute its MST.
2. Run DFS on the MST computed in the previous step, and record down 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?

4 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.

5 Runtime of NP

True or False (with brief justification): Suppose we can show for some fixed k, an NP-complete problem P has a time $O(n^k)$ algorithm. Then every problem in NP has a $O(n^k)$ time algorithm.

6 A Reduction Warm-up

Consider the two problems below:

> **Undirected Rudrata path problem:** we are given an undirected graph G as input, and want to determine if there exists a path in G that uses every vertex exactly once.
>
> **Longest Path in a DAG:** we are given a DAG and a variable k as input, and want to determine if there exists a path in the DAG that is of length k or more.

Now, consider the reduction below; is it correct? If so, provide a concise proof. If not, justify your answer and provide a counter-example.

> Undirected Rudrata Path can be reduced to Longest Path in a DAG. Given the undirected graph G, we will use DFS to find a traversal of G and assign directions to all the edges in G based on this traversal. In other words, the edges will point in the same direction they were traversed and back edges will be omitted, giving us a DAG. If the longest path in this DAG has $|V| - 1$ edges, then there must be a Rudrata path in G, as any simple path with $|V| - 1$ edges must visit every vertex.

7 Exact 4-SAT

The Exact 4-SAT problem is defined as follows.

Input: n boolean variables $\{x_1, \ldots, x_n\}$ and clauses $\{C_1, \ldots, C_m\}$ with each containing exactly {four distinct} literals. For example, the following is an instance of Exact 4-SAT,

$$(x_1 \vee x_2 \vee \overline{x_4} \vee \overline{x_5}) \wedge (x_3 \vee \overline{x_4} \vee \overline{x_1} \vee x_2) \wedge (x_1 \vee x_3 \vee x_4 \vee x_5)$$

Note that all the 4 literals within a clause have to be distinct.

Goal: Find an assignment to the variables $x_1, \ldots, x_n$ that satisfies all the clauses.

(a) Give a polynomial time reduction from 3-SAT to Exact 4-SAT.

(b) Give a polynomial time reduction from Exact 4-SAT to 3-SAT.

8 Graph Coloring Problem

An undirected graph $G = (V, E)$ is k-colorable if we can assign every vertex a color from the set $1, \ldots, k$, such that no two adjacent vertices have the same color. In the k-coloring problem, we are given a graph G and want to output "Yes" if it is k-colorable and "No" otherwise.

(a) Show how to reduce the 2-coloring problem to the 3-coloring problem. That is, describe an algorithm that takes a graph G and outputs a graph G', such that G' is 3-colorable if and only if G is 2-colorable. To prove the correctness of your algorithm, describe how to construct a 3-coloring of G' from a 2-coloring of G and vice-versa. (No runtime analysis needed).

(b) The 2-coloring problem has a $O(|V| + |E|)$-time algorithm. Does the above reduction imply an efficient algorithm for the 3-coloring problem? If yes, what is the runtime of the resulting algorithm? If no, justify your answer.