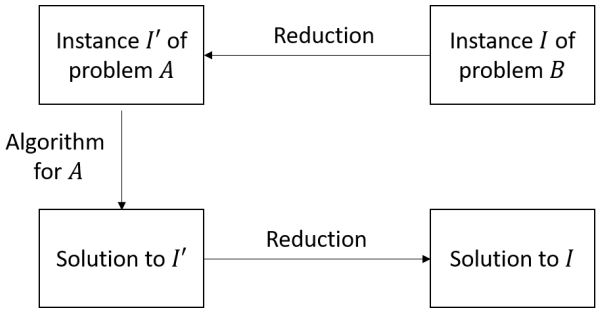
Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings,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

Reduction: Suppose we have an algorithm to solve problem A, how can we use it to solve problem B?

This has been and will continue to be a recurring theme of the class. Examples so far include

- Use LP to solve max flow.
- Use max-flow to solve min s-t cut.
- Use minimum spanning tree to solve maximum spanning tree.

In each case, we would transform the instance I of problem B we want to solve into an instance I' of problem A that we can solve, and also describe how to take a solution for I' and transform it into a solution for I:



Importantly, the transformation should be efficient, i.e. takes polynomial time. If we can do this, we say that we have reduced problem B to problem A.

Conceptually, a efficient reduction means that if we can solve problem A efficiently, we can also solve problem B efficiently. On the other hand, if we think that B cannot be solved efficiently, we also think that A cannot be solved efficiently. Put simply, we think that A is "at least as hard" as B to solve.

To show that the reduction works, you need to prove **both**:

(1) If there is a solution for instance I' of problem A, there must be a solution to the instance I of problem B

(2) If there is a solution to instance I of B, there must be a solution to instance I' of problem A.

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.

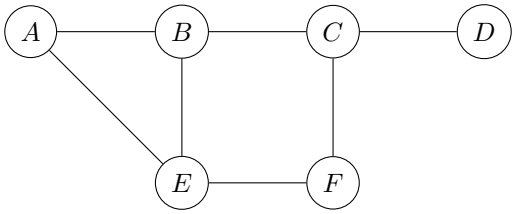
Note: recall that, to rigorously prove the correctness of a reduction from A to B, you must show two things:

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 Vertex Cover to Set Cover

To help jog your memory, here are some definitions:

> **Vertex Cover:** given an undirected unweighted graph $G = (V, E)$, a vertex cover C_V of G is a subset of vertices such that for every edge $e = (u, v) \in E$, at least one of u or v must be in the vertex cover C_V.
>
> **Set Cover:** given a universe of elements U and a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, a set cover is any (sub)collection C_S whose union equals U.
>
> In the *minimum vertex cover problem*, we are given an undirected unweighted graph $G = (V, E)$, and are asked to find the smallest vertex cover. For example, in the following graph, $\{A, E, C, D\}$ is a vertex cover, but not a minimum vertex cover. The minimum vertex covers are $\{B, E, C\}$ and $\{A, E, C\}$.



> Then, recall in the *minimum set cover problem*, we are given a set U and a collection $\mathcal{S} = \{S_1, \ldots, S_m\}$ of subsets of U, and are asked to find the smallest set cover. For example, given $U := \{a, b, c, d\}$, $S_1 := \{a, b, c\}$, $S_2 := \{b, c\}$, and $S_3 := \{c, d\}$, a solution to the problem is $C_S = \{S_1, S_3\}$.

Give an efficient reduction from the minimum vertex cover problem to the minimum set cover problem. Briefly justify the correctness of your reduction (i.e. 1-2 sentences).

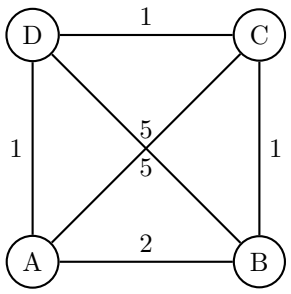
4 Approximating the Traveling Salesperson Problem

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 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 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?

5 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.

