Complexity Theory

1. Prove that P closed under the star operator ($*$). (Hint: Given a language $L \in P$ and input a, use dynamic programming to build up solutions for each substring of a to see if $a \in L^*$)

Assume that we have an arbitrary language $L \in P$, this means a DTM M_L exists that decides L in polynomial time ($M_L \in O(n^c)$ for some constant c).

Given a new string a, we want to construct a decider that can determine if $a \in L^*$. The naive algorithm would enumerate all the possible ways to divide the string a into any number of substrings. For each possible division, we loop over the substrings and run M_L on each. If M_L says *Yes* for each substring, then we accept. This decider works, but is too slow because there are $\Theta(2^n)$ possible ways to divide the string.

Using dynamic programming, we can get the runtime to a polynomial. Given a and $|a| = n$, create an array A of size $n + 1$. Each index i of A stores *True* iff the substring of a from index 0 to i is in L^*. We build up the values of this array as follows: For each index i, loop over every previous index $0 \geq j < i$. If we can find a j such that $A[j]$ is *True* and $a[j + 1 : i]$ (substring of a from j+1 to i) is in L (Run M_L to find out). Otherwise $A[i]$ is false. There are n entries in A to fill and each takes $\Theta(n * n^c)$ to fill out. The cell at $A[n]$ contains the final solution. Thus, if $L \in P$, then $L^* \in P$.

2. *PSPACE* is the complexity class containing all problems that can be solved using a *Deterministic Turing Machine* that uses at most a polynomial amount of additional space (on the tape). Prove that $P \subseteq$ *PSPACE*. (*Hint: Think about a generic problem in P, and the DTM that decides it. Try to use more than a polynomial space with this machine given the allotted time.*)

This one is pretty simple. Suppose we have a problem in P. It must have a TM decider that runs in polynomial time (polynomial steps of computation). A Turing Machine can only move its head one tape cell per step of computation, so in a polynomial amount of steps, it can only reach a polynomal number of tape cells to interact with. Thus, any TM that runs in polynomial time must use a polynomial amount of space as well. Thus, $P \subseteq$ *PSPACE*.

3. You are working for *Shoogle* (A data company that definitely does not exist) and you are given the following task from your boss: *Shoogle* is interested in investing in their employees by offering a large amount of professional development. They have collected a series of *workshops* that each cover a subset of *topics*. For example, workshop 1 might cover machine learning and databases, workshop 2 might cover machine learning and data mining, and workshop 3 might cover advanced data structures (note that each workshop can cover multiple topics and there can be overlap across workshops). In addition, *Shoogle* wants to make sure that over the course of the professional development workshops, at least one topic in each sub-area of interest is represented. For example, they might want to make sure there is at least one AI workshop (machine learning or reinforcement learning or neural networks, etc.), at least one systems workshop (databases or cloud computing), etc.

So...the problem is this: Given the availability of the professional development workshop, what topics each covers, and which areas need to be covered, can you devise a schedule over multiple weeks that ensures that at least one topic in every area is covered by at least one workshop?

Let's formalize this a bit: The input contains multiple items:

- T: A list of individual topics that can be covered (e.g., machine learning).
- $W = \{w_1, w_2, ..., w_m | w_i \subseteq T\}$: A list of workshops, each of which is a subset of the topics T that will be covered in that workshop. There are m workshops total.
- $S = \{s_1, s_2, ..., s_n | s_i \subseteq W\}$: The schedule availability for each week. Each s_i is the subset of workshops that can be scheduled in week i. There are n weeks.
- $A = \{a_1, a_2, ..., a_p | a_i \subseteq T\}$: The subject areas that need to be covered. Each area is a list of Topics in that area.

You want to output Yes if there is some schedule of workshops over the n weeks such that every area has at least one topic in T that is covered in at least one workshop.

This problem is *NP-Complete*. Prove it! For your reduction, use *3-SAT*.

Verification is simple, so I won't bother explaining it in detail. Simply look at what workshops were scheduled and what topics they cover and loop to make sure every area has at least one topic covered.

For the reduction, we use 3-SAT. Given a formula θ in 3-CNF form, we convert it into a scheduling problem as follows:

- Create one workshop for each variable in θ and its negation (e.g., $W = \{x_1, \bar{x_1}, x_2, \bar{x_2}, ...\}$)
- Create one topic covered per workshop. This effectively makes the list of topics and workshops the same, with a 1 to 1 correspondence.
- Create one area per clause in θ. For each clause, the three literals in that clause map to the three topics / workshops within that area.
- Create the schedule so that each variable and its negation workshop are available each week. For example, x_1 annd $\bar{x_1}$ are available during week 1, x_2 and $\bar{x_2}$ during week 2, etc.

This is a valid reduction because if the schedule can be done such that all areas are covered, then the formula is satisfiable. The workshops that get scheduled correspond to the variables that should be assigned to True, and since each area is associate with a clause, some variable from each clause in θ was selected.

4. You are working for *Amaphon* (an online realtor that definitely does not exist) and you are given the following task from your boss: Your users have a profile and may or may not have an in-store balance (i.e., a sum of money that can be directly spent on items on *Amaphon*). For example, maybe one user has $\$31.47$ in their account right now. Your team wants to

build a new feature in which customers can select a type of product (e.g., Holiday gifts or clothes or electronics) and the site will automatically suggest a set of items they can purchase for which the total price is exactly their current balance. This means the customer gets a bunch of cool products AND they get to clear out their balance to exactly 0 all at once.

To clarify this problem a bit, consider the following: You are given as input the users balance B, and a list of n products in their chosen category along with the prices of those products. Only the prices really matter, so let's call this list $P = \{p_1, p_2, ...p_n\}$ Your task is to find any subset of these prices that totals exactly B.

You suspect this problem might be *NP-Complete*. Prove it! For your reduction, use *3-SAT*.

This is a thinly veiled subset sum. Verification is easy, simply add the proposed items together and see if they hit the target.

For a reduction from 3-Sat, assume we have a formula θ in 3-CNF form. We convert the problem into a set of item prices as follows:

- Let n be the number of variables in θ and let c be the number of clauses in θ. Each item in the list P will have be an $n + c$ digit number.
- Each $n + c$ digits have a meaning. This first n digits represent if this is a variable or its negation AND what variable it is. For example, if the item represents x_1 or $\bar{x_1}$ then the first digit will be set to 1, otherwise 0.
- The last c digits of the number represent which clauses that variable is present in. For example, if x_1 is in clauses 1, 4, and 9 then the first, fourth, and ninth digits in this section of the number will be set to 1, otherwise 0.
- In addition, for each clause digit, there are two extra items in the list, each of which contains all 0s except for a 1 in one clause digit location. This is used for adding in up to 2 extra values in each clause column (if only 1 or 2 of the variables is selected to be true).
- The target value is n 1s followed by c 3s.

This works because the first n digits represent the variables and their negations, but the target is 1 in each of these digit columns. So we are forced to select only 1 of these items (a variable OR its negation for each respective digit). In addition, the clause columns need to add up to exactly three, but the extra dummy items allow you to select 2 of these 3 for free if you want. Thus, you MUST select at least one item from each clause to hit the target of 3 for each of these digits.