Problem Set 9: NP and Polynomial Time Reductions
CS254, Fall 2019, Anna Rafferty

Solution: Please do not post these solutions or share them with anyone outside this class. Thanks!

You are welcome to talk to others about the problems on this problem set, but you should **read and attempt each problem prior do discussing it with others.** The write up must be your own work. That means that **you should write your answers by yourself, without looking at notes from sessions with collaborators, and you must be able to explain any answer you write down.** You should list anyone you collaborated with in your collaboration form.

Due dates: Two problems are due Monday, November 18; the remaining problems are due on Wednesday, November 20. As for all homeworks, if a problem is due on a specific day it must be turned in by 9PM on that day. See homework handout (a copy is posted on Moodle) for more specific homework policies. Failing to name your files properly may result in a loss of points and potentially receiving no credit for that problem; please name your files properly!

1. Closures and NP.
   (a) Show that NP is closed under union.

   Solution: We assume that we have two languages A, B that in NP, and we seek to show that $A \cup B \in NP$. Since A and B are in NP, there exist NTMs M_A and M_B that decide A and B in polynomial time. Assume their running times are in $O(n^{k_A})$ and $O(n^{k_B})$ respectively. Then we can construct an NTM M_U to decide $A \cup B$. M_U functions like M_A, but if a branch halts on entering the reject state, that branch then follows the steps for M_B (including creating new branches for the non-deterministic choices in M_B). We change M_B's accept state to the accept state for M_A, and make this the accept state for M_U. Thus, if $w \in A \cup B$, then either M_A will accept it or M_B will accept it, and thus M_U will also accept it. If $w \notin A \cup B$, the machine will still halt on all branches because each branch created by the steps in M_A will continue on to branching based on the steps in M_B, each of which also always halts.

   The longest branch in M_U when running on an input of size n will be no longer than the length of the longest branch from M_A running on an input of size n plus the length of the longest branch from M_B running on an input of size n. Since each of these branches are of size $O(n^{k_A})$ and $O(n^{k_B})$, M_U's longest branch must be no longer than $O(n^{k_A} + n^{k_B}) = O(\max(n^{k_A}, n^{k_B}))$. Since k_A and k_B are constants, this is polynomial in n (the length of the input).

   Thus, we have created an NTM that decides $A \cup B$ in polynomial time, so $A \cup B \in NP$. *Note: It is not valid to construct a machine that waits until all branches of M_A halt and then decides whether to run M_B.*

   (b) Show that NP is closed under concatenation.

   Solution: We assume that we have two languages A, B that in NP, and we seek to show that $A \circ B \in NP$. Since A and B are in NP, there exist NTMs M_A and M_B that decide A and B in polynomial time. Assume their running times are in $O(n^{k_A})$ and $O(n^{k_B})$ respectively. Then we can construct an NTM M_U to decide $A \circ B$. On an input w, M_U first non-deterministically chooses one of the $|w|+1$ break points in the string w; this can be accomplished by non-deterministically deciding that either the current location is the breakpoint or to continue moving right and choosing a break point in the remaining part of the string. Once the break point has been chosen, M_U functions like M_A on the part of w to the left of the break point. On rejecting branches of M_A, M_U halts in its reject state. On accepting branches of M_A, M_U acts like M_B on the part of w to the right of the break point. The accept state for M_U is the accept state from M_B, so M_U will accept if M_A accepted the first part of w and M_B accepted the second part of w. If $w \notin A \circ B$, then there will be no break point where the first part of w is in A and the second part in B, so all branches will either halt while following the M_A rules (in the case the branch rejects based on M_A) or will halt in a reject state after moving on to the M_B rules. Thus, each branch will always halt, and M_U will accept w iff $w \in A \circ B$.

   The longest branch in M_U when running on an input of size n will be no longer than the length of the longest branch from M_A running on an input of size $n' < n$ plus the length of the longest branch from M_B running on an input of size $n'' < n$. Since M_A and M_B run in $O(n^{k_A})$ and $O(n^{k_B})$ time, decreasing the length of the inputs (e.g., n' and n''), can only decrease the length of the branches as long as the inputs are greater than some fixed length (i.e., N_A and N_B in the big-O definitions). Thus, as above, we can bound the running time of M_U as $O(n^{k_A} + n^{k_A})$, which is polynomial in the length of the input.

Thus, we have created an NTM that decides $A \circ B$ in polynomial time, so $A \circ B \in NP$.

(c) It is not known whether NP is closed under complementation (although it is widely believed that it is not). Here are two specious proofs that it is - explain what's wrong with each "proof." (What's wrong may be a false statement/conclusion or a statement/conclusion that we do not know to be true and that lacks sufficient evidence to support it.)

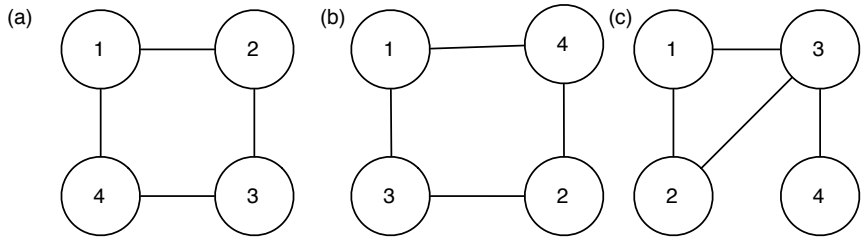
i. Let $A \in NP$. Then there exists an NTM N and natural number k such that $L(N) = A$ and the running time of N is $O(n^k)$. Define a deterministic TM M that on input w, runs N on w and outputs the opposite of what N outputs. Then $L(M) = \overline{L(N)} = \overline{A}$ and the running time of M is $O(n^k)$. So $\overline{A} \in NP$.

Solution: N runs in time $O(n^k)$ on a nondeterministic Turing machine. This does not guarantee that a deterministic TM M can simulate N in time $O(n^k)$. Indeed, the only bound we know is that there exists a deterministic Turing machine that takes time $2^{O(n^k)}$ - this comes from how we're used to simulating NTMs on deterministic TMs. Thus, we don't know that deterministic TM M will run in time $O(n^k)$, and this thus doesn't show us that an NTM exists that decides $\overline{A}$ in time $O(n^k)$ - we don't have a way to map back from deterministic machines to nondeterministic machines that will necessarily cut the time required.

ii. Let $A \in NP$. Then there exists an NTM N and natural number k such that $L(N) = A$ and the running time of N is $O(n^k)$. Define an NTM M that is exactly like N except what was q_{acc} is now q_{rej} and what was q_{rej} is now q_{acc}. Then $L(M) = \overline{L(N)} = \overline{A}$ and the running time of M is $O(n^k)$. So $\overline{A} \in NP$.

Solution: This machine is not necessarily returning the complement of A. Specifically, if on each branch, it outputs the opposite of what N would output, then if will accept if any branch was a reject branch, rather than if all of them were reject branches.

2. Two graphs G and H over a set of vertices $\{1, \ldots, n\}$ are isomorphic to one another if the nodes of G can be relabeled so that it is identical to H. The first two graphs shown below are isomorphic to each other while the third is isomorphic to neither:



Define $ISO = \{\langle G, H\rangle \mid G \text{ and } H \text{ are graphs isomorphic to one another}\}$. Show that $ISO \in NP$. (Interesting note: This problem is not known to definitely be in P, but nor is it known to be NP-complete.)

Solution: To show $ISO \in NP$, we can show that there exists a polynomial time verifier for ISO. Let's label the vertices in G as $g_1, \ldots, g_n$ and the vertices in H as $h_1, \ldots, h_n$ (if the two don't have the same number of nodes, we can clearly verify in polynomial time that this is true and reject). The certificate for our verifier will be a list in which the item at the ith index corresponds to the node that g_i maps to in the isomorphism. I.e., if we had the list [3,1,2,4], vertex g_1 would correspond to h_3, vertex g_2 to h_1, vertex h_3 to h_2, and vertex g_4, to h_4. We then check that this correspondence is an isomorphism. Specifically, we first make sure it's the right length and doesn't contain any duplicates - this can be done in $O(|V|^2)$ time. We can check that for each pair of nodes in G, an edge exists between them if and only if an edge exists in the mapped version of H (i.e., with the vertices as specified in the certificate). This will take polynomial time, as we have a polynomial number of such pairs of nodes. Thus, we can check that the certificate is valid in polynomial time, and a valid certificate will exist if and only if the two graphs are actually isomorphic.

3. Recall from your reading and the worksheet that

$$\text{Clique} = \{\langle G, k\rangle \mid G \text{ is an undirected graph} = \{V, E\} \text{ and } k \geq 1, \exists S \subseteq V \text{ s.t. } |S| \geq k \text{ and } \forall u, v \in S, (u, v) \in E\}$$

We'll also define the following language for any $k \geq 1$:

$$\text{Clique}_k = \{\langle G\rangle \mid G \text{ is an undirected graph} = \{V, E\}, \exists S \subseteq V \text{ s.t. } |S| \geq k \text{ and } \forall u, v \in S, (u, v) \in E\}$$

We'll cover in class sometime soon that Clique is NP-complete - it's as hard as any problem in NP, and thus if Clique $\in P$, then $P = NP$. As we've mentioned, most people believe that $P \neq NP$.

(a) Show that $\text{Clique}_k \in P$ for all k. (It may help for solving the second part to get relatively specific about the run time.)

Solution: Take an input $\langle G \rangle$ for some k. The Turing machine that runs in P works as follows. Let n be the number of nodes in G. Then the machine enumerates all subsets of size k, and tests to see if any are a clique. There are $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ such subsets. We can show this is in $O(n^k)$:

$$\frac{n!}{(n-k)!k!} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1}{(n-k) \cdot (n-k-1)) \cdot \ldots 1 \cdot k \cdot (k-1) \cdot \ldots \cdot 1} \tag{1}$$

$$= \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-(k-1))}{k \cdot (k-1) \cdot \ldots \cdot 1} \tag{2}$$

$$= \prod_{i=1}^{k} \frac{n+1-i}{i} \tag{3}$$

$$\in O(n^k) \tag{4}$$

Thus, we can enumerate all of these subsets in time $O(n^k)$, and checking if a particular subset is a clique, requires simply checking all edges between pairs of nodes in the proposed clique. There are k^2 such nodes, and even counting time to scan to the edges to check them, this can be done in time no worse than $O(n^2)$. Thus, our machine runs in $O(n^{k+2})$, which is polynomial time, so Clique_k is in P.

(b) Explain how it is possible that $\text{Clique}_k \in P$ for any k, and yet Clique is NP-complete.

Solution: The problem arises from the fact that we need some constant factor in the exponent that doesn't rely on the size of the input. Specifically, recall that $P = \bigcup_c TIME(n^c)$. Thus, for Clique to be in P, we would need to be able to state a particular c such that Clique $\in TIME(n^c)$. However, for any fixed c, we can consider the time to decide for an input with $k = c + 1$, and this will not be decidable on the same machine in time $O(n^c)$. (The underlying issue here is that the magnitude of the input k increases faster than the length of it: its length is the log of its magnitude; thus, the overall length of the input can increase much more slowly than the magnitude of k.)

4. An oracle is a black box machine that we can ask questions of and receive answers (without knowing how it works). Imagine we have an oracle for the NP-complete problem SubsetSum: SubsetSum is the decision problem of whether given a set of n binary integers $S = \{a_1, \ldots, a_n\}$ and a target binary integer t, does some $R \subseteq S$ sum up to t. We can ask the oracle whether an input $\langle S, t \rangle$ is a member of SubsetSum or not. Show how we can use this oracle to actually find the set that sums up to t. That is, create a machine that given input $\langle S, t \rangle \in$ SubsetSum, outputs a set R such that $\sum_{a_i \in R} a_i = t$. You may use the oracle multiple times, and if you assume the oracle runs in polynomial time, your machine to output the set R should run in polynomial time as well.

Solution: On input $S = \{a_1, \ldots, a_n\}$ and target integer t, we work as follows:

1. First, ask the oracle if there exists a subset of S that sums to t. If it says no, return that no set exists (might be encoded as emptyset/false/etc).
2. We'll construct the set S' of integers that sum to t. Initialize $S' = \varnothing$. Then, for $i = 1, 2, 3, \ldots, n$:
   - Ask the oracle if $\langle S - a_i, t - a_i \rangle \in$SubsetSum.
   - If it says yes, set $t \leftarrow t - a_i$ and add a_i to S'.
   - Regardless of the oracles answer, remove a_i from S.
3. Return S'.

This algorithm works because we check for each integer whether the remaining items in S could combine with that one to form the current sum (i.e., if some subset of S sums to $t - a_i$, then that subset plus a_i sums to t). We place each of these integers that could be part of the set into S', decreasing t as we add more items to S', so that at the end, since we know we must have a subset that works, $t = 0$ and the sum of all the items in S' is the original t.

5. As introduced in the previous problem, SubsetSum = $\{\langle S, t\rangle \mid \exists R \subseteq S \text{ s.t. } \sum_{a\in R} a = t\}$ where $S = \{a_1, \ldots, a_n\}$, a set of binary integers, and t is also an integer in binary. The partition problem is determining given a set of binary integers, does there exist a subset of the integers R such that the sum of the integers in the subset is the same as the sum of the integers that are not in the subset? That is, Partition = $\{\langle S\rangle \mid \exists R \subseteq S \text{ s.t. } \sum_{a\in R} a = \sum_{b\in S-R} b\}$ where $S = \{a_1, \ldots, a_n\}$. Show that SubsetSum $\leq_p$ Partition[1].

Solution:

We show how to convert inputs $\langle S, t\rangle$ to SubsetSum into inputs for Partition. Let $A = \sum_{i=1}^{n} a_i$ be the sum of all the integers in S. Then we construct $S' = S \cup \{2t - A\}$. We claim that $\langle S, t\rangle \in$ SubsetSum $\iff S' \in$ Partition.

First, we show that if $\langle S, t\rangle \in$ SubsetSum, then $S' \in$ Partition. Consider the sum of all numbers in S': $2t - A + \sum_{i=1}^{n} a_i = 2t$. Let T be the subset of S that sums to t. Then:

$$\sum_{a\in T} a = t$$
$$\sum_{b\in(S'-T)} = \sum_{b\in S'} b - \sum_{c\in T} c = 2t - t = t$$

Thus, T and $S' - T$ are subsets of S' that partition S' into two equal sums.

Then we show the other direction: If $S' \in$ Partition, then $\langle S, t\rangle \in$ SubsetSum. Let $T' \subseteq S'$ be some subset that sums to t (we know the partition must sum to t since the total is $2t$); then $S' - T'$ also sums to t. Then $2t - A \in T'$ or $2t - A \in S' - T'$. Without loss of generality, assume $2t - A \in S' - T'$. Then T' is a subset of S, and sums to t: it satisfies the conditions for subset sum.

[1]For these problems, it is okay if the original S is a multiset - a set with repeating elements like $S = \{a_1, a_2, a_3\} = \{2, 3, 2\}$. You can use each a_i once. (This note is to indicate that you don't have to worry if your reduction has an edge case where some set could have multiple copies of the same number. Ignore it if it doesn't make sense.)