Name: Ashna Gupta

CS 4510 Automata and Complexity 11/7/2024

Homework 6: Time Complexity

Due:11/19/2024

This assignment is due on **11:59 PM, November 19, 2024**. You may turn it in up to 48 hours late, but assignments turned in by the deadline receive 3% extra credit.

You should submit a typeset or *neatly* written pdf on Gradescope. The grading TA should not have to struggle to read what you've written; if your handwriting is hard to decipher, you will be asked to typeset your future assignments.

You may collaborate with other students, but any written work should be your own.

1. **co-NP** (4 points)

   Show that if SAT is in co-NP, then NP = co-NP. (Recall that to prove two sets are equal, show that each set is a subset of the other.)

   Answer: To show that NP = co-NP, we need to show that NP is a subset of co-NP and that co-NP is a subset of NP. Let's start with looking at SAT. We know SAT to be NP-complete. So, we know that for anything in NP, there is a polynomial reduction for it to SAT. If SAT is in co-NP, then it follows that problems in NP can also be reduced to co-NP. Thus, NP is a subset of co-NP.
   Now that we know that NP is a subset of co-NP, let's show that co-NP is a subset of NP. If SAT is NP-complete, then we know that SAT is in NP. If SAT is in co-NP, then the complement of SAT (co-SAT) is in NP. We can say this because there is a polynomial verifier to show SAT is in co-NP, and we can use that same verifier to show the opposite (anything "yes" in co-NP would be a "no" in co-SAT). So, anything not in co-NP (its complement or NP) can be reduced to SAT, then NP can be reduced to SAT, so co-NP can be reduced to NP. Thus, co-NP is a subset of NP.
   Since we can show that NP is subset of co-NP and co-NP is a subset of NP, we have shown that NP = co-NP if SAT is in co-NP.

2. **Polynomial Reductions** (4 points)

   Show that if P = NP, then every language in P is NP-complete except for $\emptyset$ and Σ^*.

   Answer: We basically want to show that all languages besides $\emptyset$ and Σ^* in P are NP-complete. To be NP-complete, a language must belong to NP and all other languages in NP can be reduced to it. Let's take some language L in P for example. Since P = NP, L is also in NP. Now, let's look into if all other languages in NP can be reduced to L. Since P = NP, all languages in P can be decided in polynomial time. Taking it further, we can reduce every language in NP to L. This is where we have the issue if L = $\emptyset$ or Σ^*. If L = $\emptyset$, then we can automatically say that a string doesn't belong to it because nothing belongs to it. Likewise, if L = Σ^*, we can automatically say that a string does belong to it because it includes everything. So, if L = $\emptyset$ and Σ^*, the polynomial reductions become trivial and not in NP-complete based on its two requirements.

3. **NP-Complete** (8 points) Show that the following problems are NP-complete:

   - $4_2 - SAT$: given a SAT problem φ with four literals per clause, is there an assignment of the variables of φ such that each clause contains at least two true literals?

     Answer: To show that $4_2 - SAT$ is NP-complete, let's use a problem we know to be NP-complete and reduce it to $4_2 - SAT$. But first, we can establish that $4_2 - SAT$ is in NP because we can check if each 4 literal clause has at least 2 true literals in polynomial time by going through each clause. Now, let's reduce 3SAT to $4_2 - SAT$. In 3SAT, each clause has 3 literals and at least one of those 3 literals must be true in order for the formula to be evaluated as true. To reduce 3SAT to $4_2 - SAT$, we can change each of the 3 literal clauses in 3SAT to have 4 literals by adding a 4th literal to each clause. This new literal can always be set to true. Since in 3SAT at least one of the 3 literals must be true, then by adding this new always true literal to each clause in $4_2 - SAT$, we are ensuring that the new 4 literal clause will have at least 2 true literals. This addition of the 4th true literal to each clause can be done in polynomial time as it requires just going through each clause. Thus, $4_2 - SAT$ is NP-complete because it is in NP and can be reduced from 3SAT.

   - $SUBGRAPH - WEIGHT$: Given a graph G whose vertices and edges have associated integer weights, and an integer k, does G contain a subgraph of weight k? (Define the weight of a graph to be the sum of the weights of the vertices and edges of that graph.)

     Answer: To show that $SUBGRAPH - WEIGHT$ is NP-complete, we need to use a problem we already know to be NP-complete and reduce it to $SUBGRAPH - WEIGHT$. Also, we need to show that $SUBGRAPH - WEIGHT$ is in NP. $SUBGRAPH - WEIGHT$ is in NP because we can check in polynomial time by summing up the chosen vertices and edges in the subgraph to k by going through the vertices and edges. Now, let's reduce Subset-Sum (a known NP-complete problem) to $SUBGRAPH - WEIGHT$. Subset-Sum takes a set and finds a subset which sums to a specific number. We can use this with the graph G to see if there is a subset that sums up to k. Then we can use that input to make a new graph where the vertices in this new graph are a weight of a graph for this input. We then can use this new graph in $SUBGRAPH - WEIGHT$ and it can check if there are weights that add up to k. If there is a subgraph in this new graph whose weight adds up to k, then we know that there is a subset from the original graph who also adds up to k. Since we can verify $SUBGRAPH - WEIGHT$ in polynomial time (explained earlier) and we can reduce Subset-Sum (NP-complete) to $SUBGRAPH - WEIGHT$, then we can say that $SUBGRAPH - WEIGHT$ is NP-complete.

4. **Bonus** (1 points) In your opinion, what are the three most important concepts or arguments or ideas presented in this course? Give a 1-2 sentence explanation for why you chose each item.
   Answer:

1) Finite Automatas - The idea of having machines that can recognize regular languages is important because they are the foundation of understanding patterns. They kind of ”generalize” how to achieve a certain end state and that can be used in so many scenarios.
2) Turing Machines - TM’s expand on determining what can and what can’t be computed by a machine and that is critical when it comes to thinking about just how much machines can or can’t do. Especially today with modern day computers, it is foundational in understanding why complexity cannot always be achieved by machines.
3) Complexity - The difficulty of a problem is important in understanding what resources are needed to solve that problem. I feel like complexity gives a better understanding of what a problem entails and makes them less ”theoretical” because we are using time to define it.