Handout 10b (Complexity Review Solutions)

Kelly Jones

December 2022

1 Solutions

True or False

1. All polytime verifiers are deciders.

True. A property of verifiers is that they always halt because the certificate is either a valid solution and the verifier accepts or it is not and the verifier rejects.

2. If $A \leq_p B$, then it is possible for A to be in P and for B not to be.

True. If $A \leq_p B$ then A is as hard to solve as B or easier. Therefore, it is possible for A to be in P while B is not. Consider the example where A is some language that can be decided in polytime and B is the halting problem $Halt_{TM}$. Now imagine that we do a polytime reduction of $Halt_{TM}$ from A. $A(x) = Halt_{TM}(F(x))$ where $F(x)$ runs $A(x)$ and if $A(x)$ accepts, outputs a pair of a TM and string M, w such that M, w halts on w and if $A(x)$ rejects, outputs some pair of TM and string M, w such that M does not halt on w.
This polytime reduction function meets the requirements of a polytime reduction because a TM that does or doesn't halt can be constructed in polytime and it is clear that every input x will have an output $F(x)$ because $A(x)$ is in polytime and either rejects or accepts.
Therefore, A is in polytime but we know that the halting problem is certainly not so the statement above is true because of examples like this.

3. The language SAT is NP complete.

True. In class, we saw that SAT was the first problem to be proven to be NP complete. It is in NP in that it can be verified in polytime by checking a sequence of boolean values to see if they satisfy the equation when plugged in. It is in NP Hard because all other problems in NP reduce to it as shown by Cook and Levin.

4. All NP Complete problems are polytime reducible to each other.

True. We often prove a problem is NP Complete by reducing it to a known NP Complete problem because a problem is NP Complete if and only if it reduces to NP Complete problems.

5. A nondeterministic TM always runs in constant time.

False. A nondeterministic TM runs in different amounts of time depending on the algorithm. For example, even if a nondeterministic TM picks a correct se-

quence of assignments for the variables in a boolean expression, it will still take linear time to check if the expression evaluates to true when those values are plugged in.
6. Algorithms with runtimes of $O(n \log n)$,$O(n^{3.2})$, and $O(n!)$ all count as having polynomial time complexity.
False. $n!$ is considered to be exponential because it grows even faster than $O(2^n)$. This is true for the other expressions listed though because they run in $O(n^k)$ or less for some constant k.
7. In this class, when we state an algorithm's time complexity, we are referring to the average time it takes to run with respect to input length.

False. Time complexity is based on the worst case run time, not the average.

Proofs
1. For an undirected graph G with a set of nodes V and edges E, a set $S \subset V$ is an independent set if and only if $\nexists e \in E | e$ connects nodes u and v and $u, v \in S$. L is the language such that $\forall G, k \in L | \exists S \subset V$ where S is independent and $|S| \geq k$
Prove that $L \in NP$

Given a graph G with a set of vertices V and a set of edges E and an integer k, accept if $\exists S \subset V$ such that $|S| \geq k$ and no two vertices in S share an edge.

For proofs like this, we want to show that the problem can be solved in polynomial time by a nondeterministic TM or verified in polynomial time by a deterministic TM. Here are both forms of the proof.

Let M be a nondeterministic TM which takes in G, k

- Non Deterministically Pick a $S \subset V$ where $|S| = k$
- For every pair of nodes in S u, v check through the
- set of edges E to ensure that no edge connects u and v
- If S exists such that no edge connects any two nodes within it, accept
- Otherwise, reject

$G, k \in L \iff \exists$ NTM M where $M(G, k)$ accepts
If $G, k \in L$ then this means there is a set of size at least k within the vertices of G such that the set is independent. If that set exists, M will pick it because the NTM picks the choice that works if it exists.
within the vertices of G If $G, k \notin L$ then there is no set of size k within the vertices of G such that the set is independent and M will clearly reject any

solutions it picks when it iterates through the checking loop.
This will run in polynomial time because it only requires $k\dot{E}$ operations to check that each edge does not connect each pair which is linear with respect to the size of the input graph. Therefore, $L \in NP$ because it can be decided by a nondeterministic TM in polytime.

Alternative Proof

Let V be a TM that verifies the independent set problem by taking in G, k, S where Svis the potential independent set that could solve the problem.

- Check that $|S| \geq k$ by counting
- For each pair of vertices $u, v \in S$, iterate through E to
- Check that no edge connects them
- If an edge is found connecting a pair u, v, reject
- Otherwise, accept

$G, k \in L \iff \exists$ a verifier V where $\exists c\ V(G, k, c)$ where c is a certificate (possible solution) for G, k
If $G, k \in L$ then by our definition of L there must exist some independent set within the vertices of G of size at least k. The encoding of this set would be a c that would cause $V(G, k, c)$ to accept because no two vertices in the set would connect and the verifier would run through all the checks without rejecting and accept.
If $G, k \notin L$ then there is no set of size k within the vertices of G that would be independent. This means that $\forall c\ V(G, k, c)$ rejects because it will find a connected edge within the set of vertices encoded by c.
This runs in polynomial time with respect to the length of the input graph G because it has to run through all edges in E a constant number of times. Therefore, $L \in NP$ because it can be verified in polytime.

2. Let I be a set of positive integers, let k be a positive integer, and let L be the language such that $\forall I, k \in L\ \exists S \subset I$ such that $\Sigma S = k$.
Prove that $L \in NP$

Given set of integers I and integer k, $I, k \in L | \exists S \subset I$ where $\Sigma S = k$

Let M be a nondeterministic TM that takes in a set of integers $I$and an integer k

- Nondeterministicly pick $S \subset I$
- Sum over all values in S

- If the sum is equal to k accept
- Otherwise, reject

$I, k \in L \iff M(I, k)$ accepts
If $I, k \in L$ this means there is some subset of the integers in I which sums to k. TM M is nondeterministic so it can pick that subset correctly and then will sum over it and pass the check that the sum is equal to k and accept.
If $I, k \in L$ then there is no possible subset of I with members that sum to k. This means that, whatever solution the machine picks, it will never find that the sum equals k and it will reject.
This runs in linear time with respect to $|I|$ because the maximum size of S is $|I|$ and the TM just needs to loop through S to get the sum. Therefore, the integer sum problem is in NP because it can be decided by a nondeterministic TM in polynomial time.

Alternative Proof

Let V be a deterministic TM which takes in I, k, S where S is a subset of I

- For each integer in S check that the integer is in I if not reject
- Add the integer to the sum
- If the final sum is equal to k accept
- Otherwise, reject

$I, k \in L \iff \exists c$ such that $V(I, k, c)$ accepts where c is a certificate (encoding of a solution).
If $I, k \in L$ then there is a subset of I that sums to k by our definition of L so then there is a c which is an encoding of that subset where V wil sum over those values and find that the sum equals k at which point it accepts.
If $I, k \notin L$ then by definition there is no subset of I that sums to k and therefore, there is no possible c where the verifier finds that the sum equals k and when sum does not equal k it will reject.
This runs in polynomial time with respect to $|I|$ because in the worst case it will need to do $|I|^2$ checks to see that integers are in I. Therefore, the integer sum problem is in NP because it can be verified in polynomial time by a deterministic TM.

Note: For proofs like these, it is possible but not necessary to describe TMs in greater detail/at a lower level, as long as it is clear that operations described could be performed by a TM (reading and writing values, checking conditions, iterating over loops are all things that can easily be done by a TM).

3. $3SAT$ is a subset of the SAT language and is in NP Hard. It consists of a boolean algebra expression of the form $(x_1 \vee x_2 \vee x_3) \wedge (x_{n-1} \vee x_{n-1} \vee x_n)$ in other words, it has n boolean variables which appear in k clauses where each clause is the disjunction of three of the variables or their negations and, for the expression to be true, all clauses must be true. Note that the same variables can appear multiple times in the same clause or in different clauses
$4SAT$ is a language which is similar to $3SAT$ except that clauses consist of the disjunction of 4 variables instead. Expressions in $4SAT$ take the form $(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (x_{n-3} \vee x_{n-2} \vee x_{n-1} \vee x_n)$ where the same variable may appear multiple times in the same clause or different clauses.
Prove that $4SAT$ is NP Complete. (Hint: I will reduce $3SAT$ to $4SAT$ in this proof)

$4SAT$ is NP Complete if and only if $4SAT \in NP \wedge 4SAT \in NP$ Hard
$4SAT \in NP$
Let M be a nondeterministic TM that takes as input an encoding of a boolean algebra expression E

- Check through E to see that all clauses are disjunctions of 4 variables and the expression is the conjunction of all clauses. If not, reject
- Nondeterministicly pick an assignment for variables $x_1 x_n$ (a binary string of length n
- Iterate through all clauses and plug in variable assignments. If any clause evaluates to false, reject. Otherwise accept

M decides $4SAT$ in polytime because it always halts because expressions can only evaluate to true or false and plugging in and checking variable assignments will only take linear time with respect to the length of E. Therefore, $3SAT \in NP$

$4SAT \in NP$ Hard
$3SAT \leq_p 4SAT$ and $4SAT \in NP$ Hard Let M be a deterministic TM that takes in an encoding of a boolean expression E

- Check all clauses in E to see that they are disjunctions of 3 variables and that E is the conjunction of all clauses. If not, reject
- Let F be a function which takes in a boolean expression encoding and returns a different boolean expression encoding as follows
- $F(E) \rightarrow E'$ where for every pair of two clauses in E $(x_i \vee x_j \vee x_j) \wedge (x_l \vee x_m or x_p)$ E' simply adds another variable to each clause like so $(x_i \vee x_j \vee x_j \vee v) \wedge (x_l \vee x_m or x_p \vee \not{v})$

- Let T be a TM which decides $4SAT$
- Run $F(E)$ and put the result into T. If T accepts, accept. If T rejects, reject.

M is able to decide $3SAT$ using $4SAT$ and function F which only takes poly-time to run because it only needs to iterate through the clauses of E once to add new variables. F is also one-to-one because it results in a unique new expression for every unique input because it only adds extra variables without removing anything. Therefore, $3SAT \leq_p 4SAT$ and $3SAT \in NP$ Hard was given. Therefore, $4SAT$ is NP Complete.

Note: The general form of proofs of NP Hardness is to find a known NP Hard rpblem and then reduce it to the problem we are proving is NP Hard. This way, we know that the problem we want to prove is NP Hard is at least as hard as the known NP Hard problem and that therefore, the new problem must also be in NP Hard.