Handout 10a (Complexity Review)

Kelly Jones

December 2022

1 Key Definitions and Ideas

Time Complexity
For a given TM M $t(n)$ is the maximum number of steps taken by M on an input of length n (worst case number of steps)
$TIME(t(n))$ is the set of all languages L such that $\forall L \in TIME(t(n)) \exists M$ which is a TM that decides L in $O(t(n))$ In this class, we use Big O which removes constants from $t(n)$)
$a * n^b = O(n^k)$ (all constant coefficients go to 1 and all constant exponents go to k
$a_1 n^{x1} + a_2 n^{x2} + ...a_j n^{xj} = O(n^k)$ (all polynomial terms are represented as one polynomial term)
$log_a n = O(log_2 n)$ (all logs can be assumed to be equivalent to log base 2)
If $t(n) = O(t'(n))$ then $TIME(t(n)) \subseteq TIME(t'(n))$ because Big O has removed constants and small polynomial terms.
When we discuss complexity in this class, we ignore time complexity differences if the Big O for two functions is the same
Polytime is when a function has $O(n^k)$ for some constant k.

Church-Turing Thesis
For any realistic model of computation, there exists a TM which can perform that computation.

Strong Church-Turing Thesis
Any realistic model of computation is polytime reducible to a computation that can be performed on a TM.
Much more contraversial than Church-Turing Thesis

P
The class of all languages that can be decided in polynomial time with respect to input length by a deterministic TM (or your favorite programming language)

NP
The class of all languages that can be decided in polynomial time by a nonde-

terministic TM or (equivalently) for which there exists a verifier (consisting of a deterministic TM) that runs in polynomial time.

$P \subseteq NP$
We don't know if $P = NP$ or if P is a strict subset of NP
It is widely believed that $P \neq NP$, otherwise, any problem with a solution that can be verified in polynomial time could be solved in polynomial time

Verifier
Verifier: a TM V that for some language L, takes as input X, c where $X \in \Sigma*$ of L and c is a certificate.
$X \in L \iff \exists c$ such that V accepts
V always halts
If $V(X, c)$ accepts for some c then $X \in L$
If $V(X, c)$ rejects then either X is not in L or c is awrong certificate for X

Intuition: A language L is all the encodings X of a problem that have a solution c. A verifier can tell you if the solution is correct if you put in the problem X and the solution c.

Polytime Reducible
Language A is polytime reducible to language B ($A \leq_p B$) if and only if there exists a function F which is one-to-one such that $\forall x \in A$, $x \in A \iff F(x) \in B$ and the transformation from x to $F(x)$ runs in polynomial time with respect to the length of x.

Intuition: To reduce one problem to another in polytime, you need a function that can transform the input to the first problem into an input for the second problem.
A polytime reduction proof can be used to show that two languages are both in NP Hard if we know one language is in NP Hard and can reduce the new language from it. This will show that the new language is as hard as the known language and therefore the new language is in NP Hard. In such a proof, it is also necessary to show that $F(x)$ runs in polytime and is one-to-one.

NP Hard and NP Complete
We define a problem to be NP Hard if all problems in NP are polytime reducible to it. This is the case if and only if the problem is reducible from another NP Hard problem. NP Complete problems are the class of problems which are in both NP and NP Hard. Therefore, to prove a problem is NP Complete you need to show that it is in NP by providing a nondeterministic TM or a verifier for it that runs in polytime and show that it is polytime reducible from another NP Hard problem. For example, SAT is the problem where, given a boolean algebra expression with many variables, you need to determine if there is some set of true or false values for the variables such that the expression evaluates to true. All other NP Complete problems reduce from this problem and from

each other.

2 Practice Problems

True or False
1. All polytime verifiers are deciders.
2. If $A \leq_p B$, then it is possible for A to be in P and for B not to be.
3. The language SAT is NP complete.
4. All NP Complete problems are polytime reducible to each other.
5. A nondeterministic TM always runs in constant time.
6. Algorithms with runtimes of $O(n \log n)$,$O(n^{3.2})$, and $O(n!)$ all count as having polynomial time complexity.
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.

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 $L \in NP$

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 $L \in NP$

Note: you will not need to be able to produce a proof like the one below on the exam. However, it is important that you understand them as you may need to do part of a proof like this or answers questions about about the concepts in proofs like this (polytime reducibility, NP hardness, NP completeness)

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)