CSOR W4231: Analysis of Algorithms - Problem Set #6

Sophia Kolak (sdk2147) - sdk2147@columbia.edu

November 10, 2020

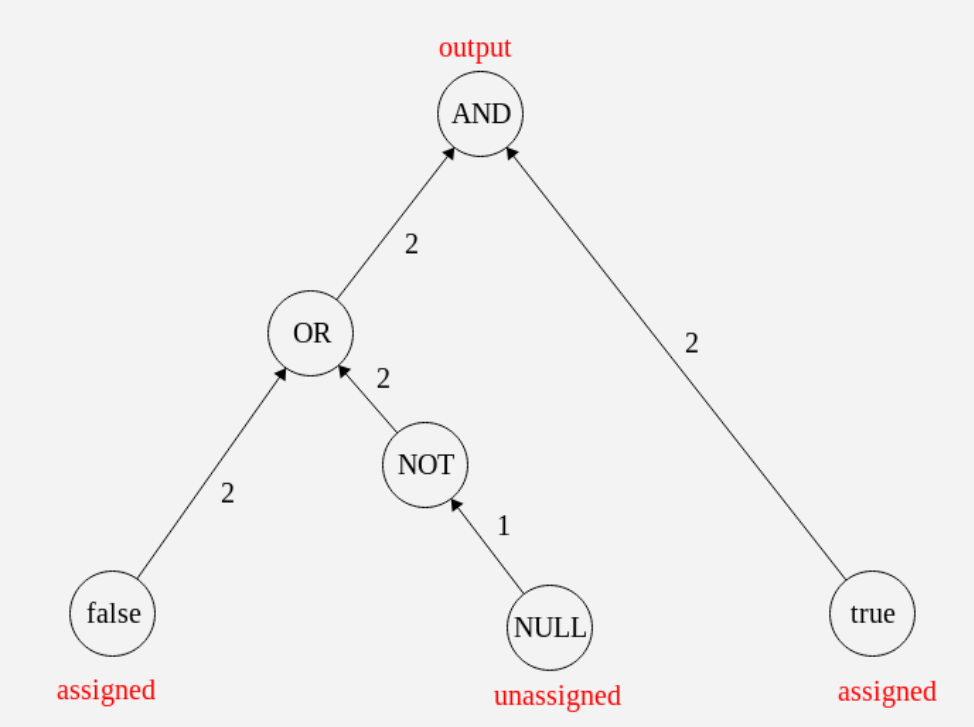
Problem 4: Show that for any decision problem in NP, there is an algorithm that can solve it that runs in time $2^{O(nk)}$,for some constant $k > 0$.

Solution: Our proof comes in two parts. First, we show that all decision problems in NP can be reduced to SAT, specifically, to circuit SAT. Second, we show that there exists a (brute-force) algorithm for circuit sat that runs in $2^{0(n^k)}$ time.

(1) All decision problems in NP can be reduced to circuit SAT: We formulate circuit SAT as a directed acylic graph with the following properties:

1. AND gates have in-degree 2
2. OR gates have in-degree 2
3. NOT gates have in-degree 1
4. assigned bits are labeled either T or F, and have no incoming edges
5. unassigned bits are labeled "NULL", and have no incoming edges

Given this weighting schema, we can evaluate the DAG in topological order until we reach "out", a labeled sink in the DAG. This is where we compute the final value, which is either true with the proper assignment of variables, or false. A simple example circuit SAT graph is provided below for clarity.



Here, if we assign "NULL" to true, the final expression evaluates to true, whereas if we assign it false, the final expression evaluates to false.
It must be the case that circuit SAT reduces to SAT, since we can just re-write any circuit graph as a series of Boolean expressions. This means circuit SAT is also NP-complete.
Now that we have formulated our circuit SAT graphs, we will prove that all NP decision problems can be reduced to circuit SAT. To do this, we will exploit what we know about NP decision problems broadly. By definition, if we are given some potential solution to an NP decision problem, we can verify (or disprove) its correctness in polynomial time. Furthermore, any polynomial time algorithm can be encoded as a Boolean circuit in polynomial time. Our formulation of the problem above shows why this is a polynomial transformation, since decision problems necessarily involve satisfying a set of constraints and outputting an assignment or a determination of impossibility and writing this DAG is just a linear mapping of variables. Thus, all NP decision problems can be formulated as instances of circuit SAT, without exponential blowup.

(2) There exists an algorithm that runs in $2^{O(n^k)}$ for the general case of circuit SAT.
This follows directly from our previous problem definition. Consider the sample graph above. We have 1 unassigned variable, and 2 possible graph outcomes to test. If we take the worst case, brute-force approach to this algorithm, assuming we have n unassigned variables, then there are 2^n possible assignments of variables. If our algorithm simply iteratively tested each possible combination until it either found a solution or

determined that none existed, the algorithm would run in $2^{O(n^k)}$, where $k = 1$. Adding in the cost of the transformation increases the value of k, but not significantly, since we already showed that the cost of transformation was polynomial, and multiplying by a polynomial value will remain bounded by this big O cost. Even with the brute force method, which is a worst case policy (there are clearly more clever ways to approach this problem) we can still obtain the given run-time for any problem that reduces to circuit SAT.

Thus, since all decision problems in NP reduce to circuit SAT, and circuit SAT is known to be solvable in $2^{O(n^k)}$ time, there exists an algorithm with this run-time for all decision problems in NP.