CSCI 338: Assignment 6 (6 points)

Elliott Pryor

April 28, 2020

This assignment is due on **Monday, April 28, 11:30pm**. It is strongly encouraged that you use Latex to generate a single pdf file and upload it under *Assignment 6* on D2L. But there will NOT be a penalty for not using Latex (to finish the assignment). This is **not** a group-assignment, so you must finish the assignment by yourself.

Problem 1

Problem 7.9 (page 323).

PROOF.

We construct a TM that decides $TRIANGLE$ in polynomial time.

So if there is a triangle between nodes a, b, c then $\exists (a, b), (a, c), (b, c) \in E$. In other words, there is an edge between each vertex. We can check if these edges exist by considering combinations of the edge set. There are $\binom{|E|}{3}$ combinations of the edge set. Let m be the number of edges in G, $m = |E|$. Well $\binom{m}{3} = \frac{m!}{3!(m-3)!} = \frac{m*(m-1)*(m-2)}{6} = \frac{1}{6}m^3 - 3m^2 + 2m$. Then we can iterate the list of combinations until we find one that matches the form $(a, b), (a, c), (b, c)$.

If there is a combination that fits this form, then accept. Otherwise reject.

This runs in polynomial time since the number of combinations is polynomial with respect to the size of the edge set. So we can perform this operation in $O(m^3) = O(|E|^3)$. Therefore, we have an algorithm that decides $TRIANGLE$ in polynomial time, so $TRIANGLE \in P$ by defintion.

$\square$

Problem 2

Problem 7.21 (page 324).

$SPATH = \{< G, a, b, k > | G$ contains a simple path of length at most k from a to $b\}$

$LPATH = \{G, a, b, k > | G$ contains a simple path of at least k from a to $b\}$

Proof.

a) Show that $SPATH \in P$.

We construct a polynomial time deterministic TM S to decide $SPATH$. We run Breadth first search on G starting from node a. We run BFS from a until we reach b. We know that the path found will be the shortest path from a to b. Then we check if the length of this path is less than or equal to k. If so then we accept, otherwise we reject.

This runs in Polynomial time since it is known that BFS runs in polynomial time.

b) Show that $LPATH \in NP - C$

We need to show that $LPATH \in NP$ and $A \leq_p LPATH$ for some NP-Complete problem A.

1) $LPATH \in NP$. We construct a non-deterministic TM R to decide $LPATH$. We nondeterministically select some path from a to b in G. If the length of the path is greater than or equal to k we accept. Otherwise we reject.

   This runs in polynomial time as we only need to compare the length of the path selected. Then by Theorem 7.20, $LPATH \in NP$

2) We reduce the Hamiltonian Cycle problem to $LPATH$, $HAMILTON \leq_p LPATH$. We map $HAMILTONIAN$ input to input for $LPATH$ as follows. We set k to be the number of nodes in G, $k = |V|$. We then set $a = b$ to be some node in G, $a \in G$.

   We show that G contains a Hamiltonian Cycle iff it has an $LPATH$ with length k from a to a. Then if G contains a Hamiltonian Cycle there is a path through G that visits each node once and returns to the start node. So the length of this cycle is $|V|$. Then there is some simple path from a returning to itself, of length k. So $< G, a, a, k > \in LPATH$.

   Then if there is some simple path of length k in G that starts at a and returns to it (eg. $< G, a, a, k > \in LPATH$ then there is a hamiltonian cycle. Because a simple path cannot visit the same vertex twice, we know that it must then visit every vertex in $G \setminus \{a\}$ exactly once, and it visits a twice. So then the path is a hamiltonian circuit as it visits every vertex without repeats.

   Therefore, we have reduced Hamiltonian Cycle (which is known NP in class) to $LPATH$.

Since we have shown both 1) and 2) $LPATH \in NP - Complete$

$\square$

Problem 3

Problem 7.22 (page 324)
$DOUBLE-SAT = \{<\phi>|\phi$ has at least two satisfying assignments $\}$.

Show that $DOUBLE-SAT \in NP-Complete$

Proof.

We need to show $DOUBLE-SAT \in NP$ and $A \leq_p DOUBLE-SAT$ for some NP-Complete problem A.

a) We show $DOUBLE-SAT \in NP$

   We construct a non-deterministic TM S to decide $DOUBLE-SAT$. We non-deterministically select some set of variable assignments X. We then check if X satisfies ϕ. If yes, accept; otherwise, reject.

b) We show that $SAT \leq_p DOUBLE-SAT$.

   We map the input of SAT to $DOUBLE-SAT$ as follows. We construct ϕ' st. $\phi' = \phi \wedge (a \vee \bar{a})$ where a is a new variable $x \notin \phi$.

   We show that $\phi \in SAT$ iff $\phi' \in DOUBLE-SAT$. First, if $\phi \in SAT$ then $\phi' \in DOUBLE-SAT$. Because ϕ is satisfiable, we know there must be at least one set of variable assignments that satisfy ϕ. Then ϕ' has at least two satsifiable assignments because ϕ is satisfiable and $(a \vee \bar{a})$ is satisfied when $a = TRUE$ or $a = FALSE$. So we can select a set of variables that satisfy ϕ and a can be either TRUE or FALSE. So, $\phi' \in DOUBLE-SAT$.

   Second, if $\phi' \in DOUBLE-SAT$ then $\phi \in SAT$. Because phi' is satisfiable we know that ϕ must also be satisfiable. We show this by contradiction. Assume ϕ' is satisfiable and ϕ is unsatisfiable. Then by our construction of $phi' = \phi \wedge (a \vee \bar{a})$ if ϕ is unsatisfiable then there would be no set of variable assignments possible to satisfy ϕ', a contradiction. So if ϕ' is satisfiable then ϕ must also be satisfiable.

Since we have shown both 1) and 2) $DOUBLE-SAT \in NP-Complete$ □

Problem 2 BAD

Problem 7.21 (page 324).

Say that two Boolean formulas are equivalent if they have the same set of variables and are true on the same set of assignments to those variables (i.e., they describe the same Boolean function). A Boolean formula is minimal if no shorter Boolean formula is equivalent to it. Let MIN-FORMULA be the collection of minimal Boolean formulas. Show that if P = NP, then $MIN - FORMULA \in P$.

Proof.

We must show that $MIN - FORMULA \in NP$. Then if $P = NP$ $MIN - FORMULA \in P$

We first show that given two boolean fomulae determining if they are equivalent is in NP. We construct a non-deterministic TM S to decide if boolean formulae A, B are equivalent. We non-deterministically select some variable assignment x for the variables in A, B. If $A(x) \neq B(x)$ then we reject, otherwise we accept.

S runs in polynomial time as it would take $O(n)$ time to compute the truth value of A, B given some assignment x where n is the number of variables.

Then we construct a non-deterministic TM R to decide if a boolean formula is minimal. Given input A where A is a boolean formula, we non-deterministically select a boolean fomula B such that B is shorter than A. We then check if B is equivalent to A. If B is equivalent to A we reject, otherwise, we accept.

Then if $P = NP$, R runs in polynomial time. Because if $P = NP$ then the problem of checking if two formulae are equivalent (TM S) can be deterministically solved in polynomial time.

Therefore $MIN - FORMULA \in NP$ by definition. Because $P = NP$, $MIN - FORMULA$ must also be in P, $MIN - FORMULA \in P$

$\square$

Problem 3 BAD

Problem 7.22 (page 324).
Modify the algorithm for context-free language recognition in the proof of Theorem 7.16 to give a polynomial time algorithm that produces a parse tree for a string, given the string and a CFG, if that grammar generates the string.

We modify the algorithm from Theorem 7.16 by adding an additonal table called tree. This table stores the parse trees used to generate the substrings in each spot in table. We index the table with three variables $tree[i, j, A]$ where i, j are indicies in $table$. We need the third index A to identify which rule we are using, as there can be multiple rules stored at each index in $table$.

So during initialization, along the diagonal we would create a tree at $tree[i, i, A]$

$$\begin{tikzcd}
\text{A} \arrow[d, dash] \\
w_i
\end{tikzcd} \tag{1}$$

if $A \to w_i$ is a rule.

Then during the main step, if $A \to BC$ is a rule and $table[i, k]$ contains B and $table[k+1, j]$ contains C, we add a new tree to $tree[i, j, A]$.

$$\begin{tikzcd}
& \text{A} \arrow[dl, dash] \arrow[dr, dash] & \\
tree[i, k, B] & & tree[k+1, j, C]
\end{tikzcd} \tag{2}$$

We would then return $tree[1, n, S]$.

Algorithm 1 Parse Tree

```
function PARSE TREE($s, G$)
    for $i$ in $[1, n]$ do
        for each variable A do
            Test wheterh $A \rightarrow b$ is a rule where $b = w_i$.
            if so, place $A$ in $table(i, i)$ and create tree (1) in $tree(i, i, A)$.
        end for
    end for
    for $l$ in $[2...n]$ do
        for i = 1 to $n - l + 1$ do
            Let $j = i + l - 1$
            for $k = i$ to $j - 1$ do
                for each rule $A \rightarrow BC$ do
                    if table(i, k) contains B and table(k+1, j) contains C then
                        put A in table(i,j)
                        put tree (2) in tree(i,j,A)
                    end if
                end for
            end for
        end for
    end for
    return tree(1, n, S)
end function
```

Problem 4 BAD

Show that if P = NP, a polynomial time algorithm exists that produces a satisfying assignment when given a satisfiable Boolean formula. (Note: The algorithm you are asked to provide computes a function; but NP contains languages, not functions. The P = NP assumption implies that SAT is in P, so testing satisfiability is solvable in polynomial time. But the assumption doesn't say how this test is done, and the test may not reveal satisfying assignments. You must show that you can find them anyway. Hint: Use the satisfiability tester repeatedly to find the assignment bit-by-bit.)

We can find the value assigments for SAT also in polynomial time. We iterate through the list of variables. We assign variable x_i to True. We then test if the formula is satisfiable with this assignment. If it is satsifieable we move on to the next variable. If it is not satisfiable, then x_i must be False because the True assignment violated a necesary constraint. So we assign it to false and move on. We do this for each variable in the set.

Algorithm 2 Parse Tree

```
1: function Parse Tree($X, f$)
2:     for $i$ in $[1, n]$ do
3:         $x_i = True$
4:         if isSatisfiable(X, f) then continue.
5:         else $x_i = False$ then continue.
6:     end for
7:     return $x_1 ... x_n$
8: end function
```