1. Homework 11 due Wednesday 12/7 (a day that will live in infamy).

The Cook-Levin Theorem

1. Last class, we began the proof of the Cook-Levin Theorem:

   **Theorem:** SAT is NP-Complete. That is, SAT $\in$ NP and for any $A \in$ NP, $A \leq_p$ SAT.

   I would like to start today with a quick review of the parts of the proof we covered and then complete the presentation.

2. We want to show that for any $A \in$ NP, $A \leq_p$ SAT.

3. All this tells us about $A$ is that for some non-deterministic TM, $M_A$, $A = L(M_A)$ and there is some polynomial function $p(n)$ such that on an input $w$, $M_A$ runs for at most $p(|w|)$ steps (whether it accepts or not).

4. We need to show that given descriptions of $M_A$ and $p(n)$, we can identify a function $f$ which for any input $w$ will produce a boolean expression $\phi$ in such a way that $\phi$ is satisfiable iff $w \in A = L(M_A)$.

5. As a result, rather than really describing $f$, we described a meta-function $F$ such that $F(M_A, p) = f_A$ where $f_A$ it a polynomial-time computable function such that $f_A(w) = \phi$ is satisfiable iff $w \in A$.

6. The approach we took to describe $F$ is based on computation histories. We want $f_A(w)$ to be a formula which encodes the requirements for a table of TM configurations to represent a valid, accepting computation of $M_A$ starting on input $w$ that lasts for less than $p(|w|)$ steps.

7. We are using the fact that a computation history can be organized in a table like:

   ```
   ( S $ 1 0 1 0 $ 1 0 # 1 0 1 # 1 1 1 # )
   ( $ S 1 0 1 0 $ 1 0 # 1 0 1 # 1 1 1 # )
   ( $ 1 0 1 S 0 $ 1 0 # 1 0 1 # 1 1 1 # )
   ( $ 1 0 1 0 S $ 1 0 # 1 0 1 # 1 1 1 # )
   ( $ 1 0 1 0 $ C 1 0 # 1 0 1 # 1 1 1 # )
   ( $ 1 0 1 0 $ # # Z # 1 0 1 # 1 1 1 # )
   ( $ 1 0 1 0 $ # # # C 1 0 1 # 1 1 1 # )
   ```

8. Given this tabular view of computation histories, we can create a set of boolean variables whose values completely describe a computation history. For each cell in the computation history table there will be one boolean variable for each symbol that could appear in that cell. That is, we would have a variable $x_{r,c,s}$ for every row $r$, column $c$ and symbol $s$ in the union of our Turing machine’s tape alphabet $\Gamma$, its state set $Q$, and any other symbols we use to encode the configurations (like the parentheses we use to mark the beginning and end of each configuration) with $x_{r,c,s} = \text{true} \iff s$ appears in the $c$th position of the rth row of the table.

9. We next described a formula $\text{happy}_{r,c}$ that is true iff the variables reflect that a unique symbol is associated with cell $r,c$.

   - We can capture the need for some symbol to be associated with the cell with the formula:
     
     $$\text{assigned}_{r,c} = \bigvee_{s \in \Gamma \cup Q \cup \{(\_)}\} x_{r,c,s}$$

   - We can capture the need for some the assignment to be unique with the formula:
     
     $$\text{unique}_{r,c} = \bigwedge_{s,t \in \Gamma \cup Q \cup \{(\_)}\} (x_{r,c,s} \lor x_{r,c,t})$$
• Combining these we then get:

\[ \text{happy}_{r,c} = \text{assigned}_{r,c} \land \text{unique}_{r,c} \]

10. Capturing the initial configuration is easy. Assuming the input is \( w = w_1w_2...w_n \), we define:

\[ \text{start} = x_{0,0} \land x_{0,1,q_0} \land ( \bigwedge_{1 \leq i \leq n} x_{0,i+1,w_i} ) \land x_{0,n+2} \land ( \bigwedge_{n+2 \leq i \leq p(n)} x_{0,i-} ) \]

11. Making sure that the computation accepts simply require making sure that some cell in the table contains the accept state:

\[ \text{accept} = \bigvee_{1 \leq r,c < p(n)} x_{r,c,q_{\text{accept}}} \]

12. The tricky part is making sure that the configuration in each row of the table yields the configuration in the next row.

• The secret to capturing this is to notice that the contents of consecutive rows in the computation history must be identical except for the symbols before and after the cell holding the state in the earlier row. Therefore, if we verify the correctness of each 2 by 3 block of cells in our table, we can be sure all transitions are handled correctly.

• Given that the size of the tape alphabet and state set are fixed we know that there are a fixed number \( (|\Gamma| + |Q| + 2)^6 \) of possible configurations of a 2 by 3 subsection of the history table. The number of invalid 2 by 3 subsection must be smaller that this.

• For each invalid configuration, we can produce a boolean expression that verifies that the bad configuration does not appear at a particular point in our table. For example, the subsection:

\[
\begin{array}{ccc}
0 & q_4 & 1 \\
1 & 1 & q_4
\end{array}
\]

would be invalid for any machine (since the symbol 0 which was not under the tape head has changed into a 1).

• If we name this configuration \( b \), then the expression

\[ \text{notbad}_b(r,c) = x_{r,c,0} \lor x_{r,c+1,q_4} \lor x_{r,c+2,1} \lor x_{r+1,c,1} \lor x_{r+1,c+1,1} \lor x_{r+1,c+2,q_4} \]

yields true only if this particular bad configuration does not appear at position \( r,c \).

• Better yet, this formula is a disjunction. So the formula:

\[ \text{notAllBad}(r,c) = \bigwedge_{b \in \text{bad configurations}} \text{notbad}_b(r,c) \]

ensures us that everything is fine at \( r,c \) and is in CNF.

13. With these pieces, we can describe the formula we use to reduce \( A \) to SAT:

\[ \phi = ( \bigwedge_{0 \leq r,c \leq p(n)} \text{happy}(r,c) ) \land ( \bigwedge_{0 \leq r,c \leq p(n)-\epsilon} \text{notAllBad}(r,c) ) \land \text{start} \land \text{accept} \]

Problems on Graphs

1. There are many other problems that are NP-Complete.

2. Given that we know that SAT and 3-SAT are NP-complete, we can show that another problem \( C \) is NP-complete by showing that \( 3-SAT \leq_p C \) or \( SAT \leq_p C \) or \( B \leq_p C \) for any other \( B \) that we have already shown to be NP-complete.
• In general if $B$ is NP-complete, $B \leq_p C \Rightarrow C$ is NP-complete as long as $C \in NP$.

• Basically, $\leq_p$ is transitive. We know that if $B$ is NP-complete then for any $A$, $A \leq_p B$. This means that there exist a polynomial time computable function $f_{A \rightarrow B}$ such that $f_{A \rightarrow B}(w) \in B$ iff $w \in A$. Now, if we learn or can show that $B \leq_p C$ there must also be a polynomial time function $f_{B \rightarrow C}$ such that $f_{B \rightarrow C}(w) \in C$ iff $w \in B$. The function obtained by composing these two functions, $f_{A \rightarrow C}(w) = f_{B \rightarrow C}(f_{A \rightarrow B}(w))$ must be polynomial computable and have the property that $f_{A \rightarrow C}(w) \in C$ iff $w \in A$ which implies that $A \leq_p C$. Thus, all $A$ are polynomial reducible to $C$ which means that $C$ must be NP-complete.

• Given that we showed how to reduce 3-SAT to subset sum before we even showed that 3-SAT was NP-complete, we immediately know that subset sum is NP-complete.

3. Many NP-complete problems involve graphs:
   • k-clique
   • k-colorability
   • vertex cover
   • Hamiltonian path
   • ...

4. As a first example from the realm of graphs, we will consider graph coloring, the problem of determining how many colors are required to color the nodes of a graph in such a way that no two adjacent nodes are assigned the same color.

   • This (seemingly silly) problem has many important applications:
     – If you were building a compiler for a language with $k$ registers and you were generating code for a method with $n > k$ variables you might still hope you could keep all the variables in registers by analyzing the program to figure out which variables were unused at various points of the program. Given the results of such an analysis (commonly called use-def chaining) you would build a graph with one node for each variable and edges between any two variable that are both needed simultaneously at any program point. If this graph is k-colorable, then $k$ registers is enough.

5. Stated as a language

$$k-\text{COLOR} = \{\langle G, k \rangle \mid G \text{ is a graph that can be colored with } k \text{ colors}\}$$

6. We will demonstrate that $k-\text{COLOR}$ is NP-complete by describing a mapping $f(\phi) = \langle G, k \rangle$ that can be computed in polynomial time on a deterministic TM and such that $\phi \in 3\text{-SAT} \iff \langle G, k \rangle \in k\text{-COLOR}$ with $k = 1+ \text{ the number of variables used in } \phi$. 