1 Odds and Ends

First, a clarification on the midterm problem 5. Recall that there are \( k \) factions, and \( \leq c \) residents have more than one neighbor whose faction is the same as their faction. We should have also specified in the language that \( c \) and \( k \) are nonnegative integers.

Second, here is a theorem.

**Theorem.** \( A_{TM} \) is NP-hard.

**Proof.** We prove that SAT \( \leq A_{TM} \). Here is a reduction \( f \).

**Description of \( f \):** On input \( \langle \phi \rangle \), construct \( \langle M \rangle \) as follows.

**Description of \( M \):** “On input \( x \), exhaustively search for a satisfying assignment \( a_1, a_2, \ldots, a_k \) to \( \phi \). If it exists, accept. Else, reject.”

Then set \( w = \epsilon \) and return \( \langle M, w \rangle \).

**Runtime:** The only nontrivial part is constructing \( \langle M \rangle \). This involves making a for loop which iterates over all possible assignments to \( \phi \). (We don’t actually run the for loop, which would take exponential time! We just create the turing machine \( M \).) The size of this turing machine will be linear in the size of \( \phi \), since there is one for loop per variable, and then a linear amount of code to check if the assignment makes \( \phi \) true.

**Correctness:**

- If \( \phi \) is satisfiable, then there is a satisfying assingment, so \( M \) will eventually find it. So \( M \) accepts any input \( x \), and therefore \( M \) accepts \( w = \epsilon \).

- If \( \phi \) is not satisfiable, then there is no satisfying assignment, so \( M \) will iterate through all assignments and then reject. So \( M \) does not accept any input, and therefore \( M \) does not accept \( w = \epsilon \).
2 Space Complexity and PSPACE

We define

\[ \text{SPACE}(f(n)) = \{ L \mid \exists \text{ a TM } M \text{ such that} \]
\[ M \text{ is a decider}, L(M) = L, \text{ and} \]
\[ M \text{ uses } O(f(n)) \text{ space on input of size } n. \}\]

\[ \text{NSPACE}(f(n)) = \{ L \mid \exists \text{ an NTM } N \text{ such that} \]
\[ M \text{ is a decider}, L(M) = L, \text{ and} \]
\[ M \text{ uses } O(f(n)) \text{ space on input of size } n. \}\]

(Recall that the amount of space used by a nondeterministic decider is defined to be the amount of space used by the longest branch.) Today we will be concerned with the new complexity class

\[ \text{PSPACE} = \bigcup_{k=1}^{\infty} \text{SPACE}(n^k). \]

3 \( \text{NP} \subseteq \text{PSPACE} \) and other stuff

As a first theorem, we can prove that \( \text{NP} \subseteq \text{PSPACE} \).

**Theorem.** \( \text{NP} \subseteq \text{PSPACE} \).

*Proof.* It is sufficient to show that SAT \( \in \text{PSPACE} \). Here is a polynomial space (but exponential time!) algorithm for SAT:

**On input** \( \phi \): For \( a_1, a_2, \ldots, a_n = 00\ldots0 \) to \( 11\ldots1 \) (all possible truth assignments): if \( a_1, a_2, \ldots, a_n \) satisfies \( \phi \), accept.

Otherwise (if the for loop terminates), reject.

This algorithm is correct because it accepts if there is a satisfying assignment for \( \phi \). It takes polynomial space because we only ever need to keep track of the single string \( a_1, a_2, \ldots, a_n \) (we can iterate through all strings by adding 1 in binary) and it takes polynomial time (and therefore polynomial space, see the next theorem) to check each particular assignment.

**Theorem.** For all functions \( f(n) \), \( \text{TIME}(f(n)) \subseteq \text{SPACE}(f(n)). \)

*Proof.* In \( O(f(n)) \) time, we can only reach \( O(f(n)) \) cells on the tape, since moving right once always takes one unit of time.

**Lemma.** \( \text{SPACE}(f(n)) \subseteq \text{TIME}(2^{O(f(n))}). \)

*Proof.* Consider a Turing machine \( M \), with \( s \) states and input of size \( n \), which uses at most \( f(n) \) space. Let \( c \) be the size of the tape alphabet.

Notice there are \( s \cdot f(n) \cdot c^{f(n)} \) possible distinct configurations of \( M \) on input an \( n \)-bit string; this is because it effectively has a finite tape of size \( f(n) \), so there are \( s \) possible states, \( f(n) \) possible positions of the head, and \( c^{f(n)} \) possible tapes.

If \( M \) were to repeat a configuration, then it would never halt! But since \( M \) is polynomial space, it is a decider, and therefore, \( M \) never reaches the same configuration twice. As every step results in a new configuration, the number of possible configurations therefore bounds the runtime of \( M \). Thus,

\[ \text{TIME}(M) \leq s \cdot f(n) \cdot c^{f(n)} \leq 2^{O(f(n))}, \]

since \( kc^k = 2^{O(k)} \).
What have we proven so far? Define

\[ \text{EXPTIME} = \bigcup_{k=1}^{\infty} \text{TIME} 2^{n^k}. \]

Then we have shown that

\[ P \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXPTIME}. \]

All of these containments are open, except that we do know that \( P \not\subseteq \text{EXPTIME} \) is strict, by what is called the time-hierarchy theorem. We will do the time-hierarchy theorem in a later class period.

### 4 Savitch’s Theorem

**Theorem.** Let \( f : \mathbb{N} \to \mathbb{R}^+ \) be any function, such that \( f(n) \geq n \) for all \( n \). Then

\[ \text{NSPACE}(f(n)) \subseteq \text{SPACE}(f(n)^2). \]

**Proof.** Given an NTM \( N \), and input \( w \), we always know what the start configuration is:

\[
\begin{array}{cccccccc}
  w_1 & w_2 & w_3 & \cdots & w_n & \swarrow & \cdots \\
\end{array}
\]

Given \( N \) that erases its tape and returns its head to the beginning of the tape before accepting, we also always know what the accept configuration looks like:

\[
\begin{array}{cccccccc}
  \swarrow & \swarrow & \swarrow & \cdots & \swarrow & \swarrow & \cdots \\
\end{array}
\]

Consider as a subprocedure a decider for \( \text{CANYIELD}_N(c_1, c_2, t) \), which accepts if \( N \) can get from configuration \( c_1 \) to \( c_2 \) in exactly \( t \) steps. (We will fill in the details for this subprocedure a bit later.) Then, let \( N \) be an \( f(n) \)-space non-deterministic decider for \( L \). Without loss of generality, \( N \) erases its tape and returns its head to the beginning of the tape before accepting. Construct the following equivalent deterministic TM \( M \):

**Description of \( M \):** “On input \( w \), set \( c_1 \) to be the start configuration of \( N \) on input \( w \), and set \( c_2 \) to be the accept configuration of \( N \). If \( \text{CANYIELD}_N(c_1, c_2, 2^{cf(n)}) \) is true, accept. Else, reject.”

If \( N \) accepts \( w \), then it will do so in at most \( 2^{cf(n)} \) steps in the shortest case, since the shortest case will never repeat a configuration. So \( M \) accepts \( w \) if and only if \( N \) accepts \( w \).

To analyze the space used by \( M \), we need to construct the decider for \( \text{CANYIELD}_N \).

**Decider for \( \text{CANYIELD}_N \):** Recall that after a NTM halts, at each step it just stays in the same configuration (the accepting configuration).

- **If** \( t = 1 \), then if \( c_1 = c_2 \) = the accept configuration, accept. Otherwise, if \( N \)'s transition function allows to get from \( c_1 \) to \( c_2 \), accept. Else, reject.
- **Else**, for every \( c_m \) that encodes a configuration of \( N \) using \( f(n) \) space: Run \( \text{CANYIELD}(c_1, c_m, t/2) \) and \( \text{CANYIELD}(c_m, c_2, t/2) \). If both accept, accept.

If this for loop over all \( c_m \) terminates, reject.

**Analysis:** The runtime of \( \text{CANYIELD}_N \) is ridiculously bad. However, we are not actually doing that badly on space. If \( S(t) \) is the amount of space we are using, then

\[
S(t) = c \cdot f(n) + S(t/2)
\]

\( c \) space needed to store guess for \( c_m \)
and $S(1) = 1$. So this expands out to imply

$$S(t) = c \cdot \log t \cdot f(n).$$

What then is the runtime of $M$? All we did was run $\text{CAN\textsc{YIELD}}(c_1, c_2, 2^{cf(n)})$. So this takes runtime

$$S(2^{cf(n)}) = c \cdot \log(2^{cf(n)}) \cdot f(n) = c^2 f(n)^2.$$

Which proves the theorem.

This theorem is important mainly because it implies the following interesting corollary:

**Corollary.** $\text{PSPACE} = \text{NPSPACE}$.

### 5 An example PSPACE language

**Theorem.** The language

$$\text{ALL}_{\text{NFA}} = \{ \langle A \rangle : A \text{ is an NFA and } L(A) = \Sigma^* \}$$

is in PSPACE.

**Proof.** To decide $\text{ALL}_{\text{NFA}}$, we could always just

1. Convert the NFA to a DFA.
2. Switch the accepting and rejecting states.
3. Check if there is a path from $q_0$ to a rejecting state.

But, this is not polynomial time or polynomial space! Step 1 takes exponential time and exponential space.

Instead of converting the NFA to a DFA, note that the minimum path from $q_0$ to a rejecting state, if it exists, is at most $2^{\# \text{ of states in } A}$. So here is a better algorithm. On input the description of a DFA $\langle A \rangle$,

1. Nondeterministically guess $w$ of length $2^{\# \text{ of states in } A}$, one symbol at a time.
2. Keep track of a subset of $A$, which can be thought of as a set of pebbles on various states in the NFA. Each time we guess a new symbol, advance the pebbles in simulating $A$ on input $w$, so that the set of pebbles marks all possible states the NFA could be in on this particular input. (If there is more than one pebble on a particular state, we always throw the extra pebble(s) away.)
3. If we reach a set of states that are all rejecting, i.e. every pebble is on a rejecting state, accept.

So all we have to keep track of is the currently guessed symbol and the pebbles. This is a nondeterministic polynomial-space algorithm to decide $\text{ALL}_{\text{NFA}}$. But $\text{PSPACE} = \text{NPSPACE}$, so this can be converted to a deterministic polynomial-space algorithm. Then $\text{PSPACE} = \text{coPSPACE}$, so $\text{ALL}_{\text{NFA}}$ is polynomial space also.
Diagram of complexity classes we have seen