Announcements
1. I will put a new homework online later today, it will be due Wed., Nov. 30. TA hours will be revised appropriate (detail TBA).

Why P?
1. Today, we will start our discussion of the classes P and NP and the open question of whether P = NP.
2. I want to start this out by reflecting a bit on why P is an appropriate measure of “tractable”.
3. In particular, how many of you would consider an algorithm that took \(N^{20}\) steps usable?

Resource Restrictions and Big-O
1. Over the last several weeks, we have happily discussed Turing machine algorithms that required vast amounts of time and tape for relatively simple problems. For the remainder of the semester, we will examine the power of Turing machines when the amount of tape or the number of steps they can use while processing an input is limited.

2. Last class, we talked about a model of computation in which one of the resources the machine could use, memory, was related to the size of its input:

**Definition:** A linear bounded automaton is a restricted type of Turing machine wherein the tape head is not permitted to move more than once for each symbol on the tape containing its input. (Assume that if the TM tries to take to many steps, the input is rejected by default.)

3. We saw that limiting the space a Turing machine could use, limited the number of steps it could use effectively. We concluded that an LBA that ran for more than \(|Q| \times |w| \times |\Gamma|^{|w|}\) steps had to be stuck in an infinite loop.

4. Suppose that we instead decided to place a restriction on the number of steps a TM could take to process its input:

**Definition:** A linear-time bounded automaton is a restricted type of Turing machine wherein the tape head is not permitted to move more than once for each symbol on the tape containing its input. (Assume that if the TM tries to take too many steps, the input is rejected by default.)

5. This restriction changes the beast quite dramatically.
   - Such machines can clearly still recognize all regular languages
   - Such machines cannot recognize even simple context-free languages like \(w#w^R\).
   - They can recognize some weird languages where an interesting prefix is followed by a very long suffix of anything at all that just buys the TM enough steps to verify the prefix.

6. Changing from \(|w|\) steps to \(k \times |w|\) steps does not help much:
   - Still cannot recognize simple context-free languages like \(w#w^R\).

7. On the other hand, changing from \(|w|\) steps to \(|w|^2\) steps makes a big difference. With \(|w|^2\) a TM can recognize
   - \(\{w#w \mid w \in \Sigma^*\}\)
   - \(\{n#w#w_1#w_2#...#w_k \mid n,w,w_i \in \{0,1\}^*, n \leq k, \text{ and } w = w_n\}\)
   - \(\{a^n b^n c^n \mid n \geq 0\} — \text{this language is not context-free.}\)

8. \(|w|^2\) is still not enough to parse arbitrary context-free grammars, but \(|w|^3\) is!

9. This does not explain why P is interesting, but it does provide a handy refresher on and justification for big-O notation:

**Definition:** Let \(f\) and \(g\) be functions \(f, g : \mathcal{N} \to \mathcal{R}^+\). We say that \(f(n) = O(g(n))\) if for some positive integers \(c\) and \(n_0\)
   \[f(n) < cg(n)\]
for all \( n \geq n_0 \). We say that \( g(n) \) is an asymptotic upper bound for \( f(n) \).

10. Basically, when we explore what can be computed by TMs where the amount of space they can use or the number of steps they take is limited to be less than some function of the size of the input, multiplying the function by a constant does not seem to matter. So we only discuss the highest order term of the function that describes the amount of time/space used by an algorithm, ignoring both constant multipliers and lower-order terms.

11. Based on this notion, we can define classes of languages recognized by TMs in numbers of steps bounded asymptotically bounded by any function:

Definition: Let \( t : \mathbb{N} \rightarrow \mathbb{R}^+ \) be a function. Define the \textbf{time complexity class}, \( \text{TIME}(t(n)) \) to be the collection of all languages that are decidable by an \( O(t(n)) \) time Turing machine.

Models and Resources

1. Consider how many steps are required to recognize the languages below on a multi-tape TM:

   - \( \{ w \# w \mid w \in \Sigma^* \} \) — \( O(n) \)
   - \( \{ n \# w \# w_1 \# w_2 \# \ldots \# w_k \mid n, w, w_i \in \{0, 1\}^*, n \leq k, \text{ and } w = w_n \} \) — \( O(n) \)
   - \( \{ w_0 \# w_1 \# w_2 \# \ldots \# w_k \mid \exists i, j \leq k \text{ with } w_i = w_k \} \) — \( O(n \log n) \)

2. Now do the same thing for a single-tape TM. In general, you should recognize that for every step made by a multi-tape TM, a single tape TM may have to make a complete pass over its tape whose length can be as large as the number of steps executed. As a result, the running time will be at most the square of that of a single tape machine.

3. Now do the same thing for a queue machine (like that discussed in the last homework). Again, each step of the single tape TM may take a full pass over the queue. Therefore, the running time may be squared again.

4. The key observation is that for all the variations in our computing model, any bound on one model that is a polynomial remains a polynomial on most other reasonable models.

5. This inspires:

   \textbf{Definition:} \( P \) is the class of language that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words

   \[
P = \bigcup_k \text{TIME}(n^k)
   \]

6. Of course, the biggest variation in model we can make is to switch from a deterministic model to a non-deterministic model. For DFAs and TMs with no limits on the durations of their computations, we have shown nondeterministic modes and deterministic modes are equivalent. We might well wonder whether this is the case for TMs limited by some time bound.

7. As you should already know, it is not known whether this is the case.

8. To allow us to explore this question, we consider the execution time of a nondeterministic TM to be the length of the longest path in its computation tree and say

   \textbf{Definition:} \( \text{NTIME}(t(n)) = \{ L \mid L \text{ is a language decided by an } O(t(n)) \text{ time nondeterministic Turing machine } \} \).

   \textbf{Definition:} \( \text{NP} \) is the class of language that are decidable in polynomial time on a deterministic single-tape Turing machine. In other words

   \[
   \text{NP} = \bigcup_k \text{NTIME}(n^k)
   \]
9. Given that one way to understand nondeterminism is that nondeterministic machines are great guessers, another approach to defining the cost of a nondeterministic algorithm is to concentrate on the effort required to verify that the “guess” was correct.

10. We can therefore define the notion of a verifier and its running time

**Definition:** A *verifier* for a language $A$ is a Turing machine $V$ where

$$A = \{ w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string c} \}$$

We say that $V$ is a polynomial time verifier if it runs in polynomial time in the length of $w$. In this case, we say that $A$ is polynomial verifiable.

**A Member of NP**

1. My favorite example of a problem in NP is definitely the Subset Sum problem.

2. Given a list of numbers like:

   17, 24, 5, 9, 11, 24, 57, 4, 39, 40, 84, 11, 19

   Is there some sublist of these numbers that adds up to exactly N? (for some given N)

3. It should be clear that there is a polynomial time verifier for this problem.

4. It should also be clear how this verifier could be turned into a nondeterministic algorithm for the problem. We would first guess some numbers and then use the verifier to see if we got lucky.

5. At the same time, checking every subset would require exponential work and there is no obvious algorithm that is more efficient.

**3-dimensional Matching**

1. My second favorite problem in NP is the 3-dimensional matching problem.

**Definition:** Given three sets of items, $X, Y, Z$, and a set of allowable triples, $A \subseteq X \times Y \times Z$, we say that $M \subseteq A$ is a *3-dimensional matching* if for any pair of distinct triples $(x_1, y_1, z_1), (x_2, y_2, z_2) \in M$, $x_1 \neq x_2$, $y_1 \neq y_2$, and $z_1 \neq z_2$.

The problem is whether given $X, Y, Z, A \subseteq X \times Y \times Z$, we can find an $M \subseteq A$ of a specified size that forms a valid matching.

2. Those of you who have taken 134 with me over the last few years have seen this problem in the guise of the “Awkward Diners” problem:

   Suppose a set $C$ of dinner companions are at a restaurant serving a set of entrees, $E$, and a set of desserts, $D$ where conveniently $|C| = |E| = |D|$. None of the diners wants to order either the same entree or the same dessert as any of their companions and each diner has his or her own preferences (expressed as a subset of $E \times D$) of combinations of an entree and a dessert they are willing to eat. Given all of their preferences, is there an order for the table that satisfies everyone?

3. If we let $X = C, Y = E, Z = D, A = \{(c, e, d) \mid c \text{ is willing to eat e as an entree with d as a dessert } \}$, then the “Awkward Diners” problem is clearly just a special case of 3-dimensional matching in disguise. In particular, it is the case where $|X| = |Y| = |Z| = |M|$.

4. For example, we might have $C = \{ \text{Beth, Carl, Diana, Evan, Fred, Gina, Harry } \}, E = \{ \text{Sea Scallops, Pumpkin Ravioli, Roast Chicken, Shortribs, Roast Skate, Flying Pig, Lobster } \},$ and $D = \{ \text{Key Lime Pie, Apple Tart, Black Forest Cake, Tangerine Sherbet, Cheesecake, Tiramisu, Creme Brule } \}$. Beth might only be willing to eat sea scallops with key lime pie, shortribs with key lime pie, roast skate with tangerine sherbet, or roast chicken with tangerine sherbet. Carl might only be willing to eat ... and so on.

5. It should be clear that we can efficiently verify a proposed solution to this problem and therefore it is in NP.

**Polynomial Time Reducibility**
1. Hopefully, those of you who saw the subset-sum and “Awkward Diners” previously in CS 134 remember the punch line. In some sense, the subset sum problem and the 3-dimensional matching problem where $|X| = |Y| = |Z| = |M|$ are the same problem.

2. To see that this is the case, first number the elements of the sets $X$, $Y$, and $Z$. That is, assuming all of the sets involved are of size $n$ we would write $X = \{x_0, x_1, x_2, \ldots, x_{n-1}\}$, $Y = \{y_0, y_1, y_2, \ldots, y_{n-1}\}$, and $Z = \{z_0, z_1, z_2, \ldots, z_{n-1}\}$.

3. Now, we can associate each triple $(x_i, y_j, z_k) \in A$ with the number $b^i + b^{n+j} + b^{2n+k}$. That is, we will associate each triple with a number written in base $b$ as $3n$ digits $000\ldots1000\ldots1000\ldots1000$ where the last 1 appears $i$ positions from the end, the second 1 appears $n + i$ positions from the end and the first 1 appears $2n + k$ from the end.

4. If there is a subset $M$ of triples from $A$ that forms a valid 3-dimensional matching, then each element of $X$, $Y$, and $Z$ will appear in exactly 1 triple. Therefore, for any position $m$, exactly one of the numbers $b^i + b^{n+j} + b^{2n+k}$ associated with these triples will have a 1 at position $m$. Therefore the sum of the numbers corresponding to the triples will be $3n$ digits long and of the form $1111\ldots111 = b^{3n} - 1$.

5. On the other hand, given the numbers associated with the triples in $A$, if we can find a subset of these numbers that add up to $b^{3n} - 1$, then the corresponding set of triples must form a valid matching $M$. (To be careful, we should choose $b$ to be large enough that carries between columns cannot occur during the addition.)

6. What we have shown is clearly a many-to-one reduction of 3-dimensional matching to subset sum. That is:

$$3\text{-dim matching} \leq_m \text{subset sum}$$

7. The reduction we have described, however, is more than computable. It is easy to compute. In particular, the number of numbers we must generate from the 3-dimensional matching problem and the number of digits in each number is linear in the length of the description of the original problem. As a result, we can build a Turing machine to compute the appropriate subset sum problem in some polynomial number of steps as a function of the input size (the size of the original matching problem).

8. This leads to:

**Definition:** We say that $A$ is polynomial-time reducible to $B$ (written $A \leq_p B$) if and only if there exists a polynomial time function $f : \Sigma^* \rightarrow \Sigma^*$ such that $w \in A$ if and only if $f(w) \in B$.

9. In particular, we now can say:

$$3\text{-dim matching} \leq_p \text{subset sum}$$

10. Just as many-to-one reducibility allowed us to draw conclusions about whether a language was decidable or recognizable, polynomial reducibility allows us to make statements about membership in $P$ (and eventually $NP$).

**Theorem:** If $A \leq_p B$ and $B \in P$ then $A \in P$.

and conversely

**Theorem:** If $A \leq_p B$ and $A \notin P$ then $B \notin P$.

11. Thus, we now know that if subset sum is in $P$ then 3-dimensional matching (at least looking for complete matchings) is in $P$. Also, if 3-dimensional matching is not in $P$ then subset sum must not be in $P$ either.