Last Time

• Mapping Reducibility
  – Formalized definition
  – Mapping reducibility
  – “We can solve A with a solver for B”
Today

• A quick, high-level coverage of the material in Chapter 7.
  – How do we measure time complexity?
  – How does that measurement apply to algorithms?
  – Can we apply the idea of reductions to time complexity analysis?
  – What are P, NP, and NP-Completeness?
Measuring Complexity

\[ A = \{0^k1^k \mid k \geq 0\} \]

• How much time does a single-tape TM take to decide A?
  
  M = “On input string w:
    1. Scan across the tape and reject is a 0 is found to the right of a 1.
    2. Repeat if both 0s and 1s remain on the tape:
       3. Scan across the tape, crossing off a single 0 and a single 1.
      4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, reject. Otherwise, if neither 0s nor 1s remain on the tape, accept.

• Assuming input length \( n \), it will take \( 2n \) steps for 1 (across and back), \( 0.5n^2 \) steps for 2-3 (across and back on each pass, at most \( n/2 \) passes), and \( n \) steps for 4. The \( n^2 \) term dominates for large \( n \). Thus, we say this TM has time complexity \( O(n^2) \).
# Time Complexity Classes

- $O(1)$ - constant time
- $O(\log(n))$ - logarithmic time
- $O(n)$ - linear time
- $O(n \times \log(n))$ - “n log n” time
- $O(n^2)$ - quadratic time
- $O(n^3)$ - cubic time
- $O(n^n)$ - exponential time
- $O(n!)$ - factorial time

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>GOOD</td>
</tr>
<tr>
<td>$O(\log(n))$</td>
<td>GOOD</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>GENERALLY OK</td>
</tr>
<tr>
<td>$O(n \times \log(n))$</td>
<td></td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>GENERALLY OK</td>
</tr>
<tr>
<td>$O(n^3)$</td>
<td>GENERALLY OK</td>
</tr>
<tr>
<td>$O(n^n)$</td>
<td>OFTEN BAD</td>
</tr>
<tr>
<td>$O(n!)$</td>
<td>OFTEN BAD</td>
</tr>
</tbody>
</table>
Definitions

- **Asymptotic notation or big-O notation** – A notation showing the highest-order term of the running time of an algorithm.

- **Time complexity class** – Class TIME(t(n)) is a class of all languages that are decidable by an O(t(n)) time Turing machine.
Some Time Complexity Classes

- **Class P** – The class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine.

- **Class NP** – The class of languages that have polynomial time verifiers, and are decidable in polynomial time on a non-deterministic single-tape Turing machine.

- **Verifier** – An algorithm $V$ for language $A$, where $A = \{w \mid V$ accepts $\langle w, c \rangle$ for some string $c\}$. 
The PATH Problem

\[ \text{PATH} = \{(G, s, t) | G \text{ is a directed graph that has a directed path from } s \text{ to } t\} \]

- **Theorem:** \( \text{PATH} \in P \)
- **Proof:** We prove this theorem by presenting a polynomial time algorithm that decides \( \text{PATH} \).

\[ M = \text{"On input } (G, s, t), \text{ where } G \text{ is a directed graph with nodes } s \text{ and } t:\]

1. Place a mark on node \( s \).
2. Repeat the following until no additional nodes are marked:
   3. Scan all the edges of \( G \). If an edge \((a, b)\) is found going from a marked node \( a \) to an unmarked node \( b \), mark node \( b \).
   4. If \( t \) is marked, accept. Otherwise, reject.”

- Steps 1 and 4 are only run once. Step 3 runs at most \( m \) times, where graph \( G \) has \( m \) nodes. Thus, the total number of steps is \( 1+1+m = O(m) \), which is polynomial time.
The HAMPATH Problem

\[ HAMPATH = \{ \langle G, s, t \rangle | G \text{ is a directed graph with a Hamiltonian path from } s \text{ to } t \} \]

- **Theorem:** \( HAMPATH \in NP \)
- **Proof:** We prove this theorem by presenting a polynomial time algorithm that decides \( HAMPATH \) on an NTM.

\( N = \) “On input \( \langle G, s, t \rangle \), where \( G \) is a directed graph with nodes \( s \) and \( t \):

1. Write a list of \( m \) numbers, \( p_1, \ldots, p_m \), where \( m \) is the number of nodes in \( G \). Each number in the list is nondeterministically selected to be between \( 1 \) and \( m \).
2. Check for repetitions in the list. If any are found, reject.
3. Check whether \( s = p_1 \) and \( t = p_m \). If either fail, reject.
4. For each \( i \) between \( 1 \) and \( m-1 \), check whether \( (p_i, p_{i+1}) \) is an edge of \( G \). If any are not, reject. Otherwise, accept.”

- All steps run in nondeterministic polytime. Now we need to show that a verifier exists that runs in deterministic polytime:

\( V = \) “On input \( \langle w, c \rangle \), where \( w \) and \( c \) are strings:

1. Simulate input \( w \) on NTM \( N \) and do same.”

\( \qed \)
Is P = NP?

• Probably not, but we haven’t proven it yet.
• If you prove it, you’ll be set for life.
• (Don’t waste your time trying to prove it.)
NP-Completeness

- **NP-Completeness** – There exist certain problems in NP whose individual complexity is related to that of the entire class.
  - (In other words, each problem in this NP-Complete class is mapping-reducible to each other problem in this NP-Complete class.)
The Base Problem

• **Satisfiability Problem** – A Boolean formula is satisfiable if some assignment of 0s and 1s to the variables in the formula makes the formula evaluate to 1.

\[ \phi = (\bar{x} \land y) \lor (x \land \bar{z}) \]

• If we set \( x = 0 \), \( y = 1 \), and \( z = 0 \), \( \phi \) evaluates to 1.

\[ SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable Boolean formula} \} \]
Back to Reductions

• A function $f : \Sigma^* \rightarrow \Sigma^*$ is a **polynomial time computable function** if some polynomial time Turing machine $M$ exists that halts with just $f(w)$ on its tape, when started on any input $w$.

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

• **Proof:** Let $M$ be the polytime algorithm deciding $B$, and $f$ be the polytime algorithm reduction from $A$ to $B$. A polytime algorithm deciding $A$ is:

  $N = \text{"On input } w:\$
  
  1. Compute $f(w)$.
  2. Run $M$ on input $f(w)$ and output same.”

• **Corollary:** If $A \leq_p B$ and $A \in NP$, then $B \in NP$. 

Two NP-Complete Problems

\[ 3SAT = \{ \langle \phi \rangle \mid \phi \text{ is a satisfiable 3-cnf formula}\} \]
\[ CLIQUE = \{ \langle G, k \rangle \mid G \text{ is an undirected graph with a } k-\text{clique}\} \]

- **Theorem:** 3SAT is polytime reducible to CLIQUE.
- **Proof:** On board.
NP-Completeness

• **NP-Completeness** – There exist certain problems in NP whose individual complexity is related to that of the entire class.
  
  – (In other words, each problem in this NP-Complete class is mapping-reducible to each other problem in this NP-Complete class.)

• A language $B$ is **NP-Complete** if it satisfies two conditions:
  
  1. $B$ is in NP.
  2. Every $A$ in NP is polytime reducible to $B$. 
Any Questions?

HOMEWORK (due 11/20)
7.1, 7.5, 7.6, 7.7, 7.8, 7.9, 7.10,

2nd Edition: 7.21, 7.24, 7.27
3rd Edition: 7.22, 7.26, 7.29