December 6

Program of the day:

- Questions to O3
- Easy and hard problems, problem reduction
- The classes $\mathcal{P}$, $\mathcal{NP}$, and $\mathcal{NP}$-complete
- The computer as a circuit (to the non-computer scientists)
- CIRCUIT-SAT is $\mathcal{NP}$-complete
Running time

- The running time is the **worst case** number of “simple” operations used to solve a given problem. The running time is measured as function of input length $n$
- We do not care about constants, low-order terms.
- Asymptotic running time
  \[ T = 1000n^2 + 2n^3 + \sqrt{n} + 7 \log n \]
  in this case $T = O(n^3)$
- “Easy” problems are those having polynomial asymptotic running time
- “Difficult” problems are those having superpolynomial running time
Motivation

- The world is filled with $\mathcal{NP}$-complete problems
- It is difficult to distinguish "easy" and "hard" problems
- Reduction (conversion) techniques are generally applicable

Reduction lemma (Wolsey prop. 6.1)

Suppose that $P$ and $Q$ are two problems

- If $Q$ is "easy", and $P$ is not more difficult than $Q$, then $P$ is "easy"

\[
P \leq Q \quad \text{"harder"}
\]

- If $P$ is "difficult", and $P$ is not more difficult than $Q$, then $Q$ is "difficult"

\[
P \leq Q \quad \text{"harder"}
\]
## Easy and hard problems

<table>
<thead>
<tr>
<th>( \mathcal{P} \subseteq \mathcal{NP} )</th>
<th>( \mathcal{NP} )-complete</th>
</tr>
</thead>
</table>
| **Euler cycle**
Given a graph \( G = (V, E) \). Is there a cycle which visits each edge exactly once? | **Hamilton cycle**
Given a graph \( G = (V, E) \). Is there a cycle which visits each node exactly once? |
| **Shortest path**
Given a weighted graph \( G = (V, E, c) \). Is there a path from \( a \) to \( b \) with length \( \leq k \)? | **Longest path**
Given a weighted graph \( G = (V, E, c) \). Is there a path from \( a \) to \( b \) with length \( \geq k \)? |
| **Edge cover**
Given a weighted graph \( G = (V, E) \). Is it possible to choose \( \leq k \) edges so every node is incident to a chosen edge? | **Node cover**
Given a weighted graph \( G = (V, E) \). Is it possible to choose \( \leq k \) nodes so every edge is incident to a chosen node? |
| **Chinese postman**
Given a weighted graph \( G = (V, E, c) \). Is there a cycle which visits each edge at least once and has length \( \leq k \)? | **Traveling salesman**
Given a weighted graph \( G = (V, E, c) \). Is there a cycle which visits each node at least once and has length \( \leq k \)? |
| **2CNF-SAT**
Given a boolean expression in 2CNF-form. Is it possible to assign truth values to variables so expression becomes true? | **3CNF-SAT**
Given a boolean expression in 3CNF-form. Is it possible to assign truth values to variables so expression becomes true? |
Decision problems

Problems are formulated as “yes”-“no” problems

$$\text{KP-OPTIMIZATION}(p, w, c) = \left\{ \max \sum_{j=1}^{n} p_j x_j : \sum_{j=1}^{n} w_j x_j \leq c; \ x_j \in \{0, 1\} \right\}$$

Equivalent decision problem

$$\text{KP-DECISION}(p, w, c, k) = \left\{ \sum_{j=1}^{n} p_j x_j \geq k : \sum_{j=1}^{n} w_j x_j \leq c; \ x_j \in \{0, 1\} \right\}$$

Advantages of decision problems

- “Minimal model”
- Easier to reduce (convert) problems since solution space is the same
- Optimization version can be solved through binary search, using the decision problem
General problem and instance of problem

- KP-DECISION is a general problem
- instance of KP-DECISION is a concrete dataset $X$

<table>
<thead>
<tr>
<th>$j$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_j$</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$w_j$</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

$c = 10$, $k = 9$

Length of input (Wolsey def 6.1)

For a problem instance $X$ the length of input $L(X)$ is the length of binary standard representation

\[ L(X) = (2n + 2)\lceil \log_2(c) \rceil = 10\lceil \log_2(10) \rceil = 40 \]

Important since we measure running time in $L(X)$

- Unary coding
- Binary coding
- Decimal coding
- Fixed integer size
Solving a problem or checking a solution

It is easier to check (verify) a solution than to solve a problem.

Verification

A verification algorithm for a general problem DECISION has input

- An instance $X$
- A certificate $Y$ where $|Y|$ is polynomial in $|X|$

the algorithm outputs "yes" if

- The instance $X$ is a "yes" instance, and $Y$ is a proper solution to the problem.

Wolsey implicitly defines verification in Definition 6.3 as those problems having a "short" proof of the "yes".

Example

LP-DECISION

\[
\begin{align*}
\text{maximize} & \quad cx \geq k \\
\text{subject to} & \quad Ax \leq b \\
& \quad x \geq 0
\end{align*}
\]

Instance $X = (A, b, c, k)$ certificate $Y = x$
Polynomially solvable problems $\mathcal{P}$ (Def 6.4)

$\mathcal{P}$ is the set of decision problems for which an algorithm exists which can answer the problem in polynomial time.

Polynomially verifiable problems $\mathcal{NP}$ (Def 6.3)

$\mathcal{NP}$ is the set of decision problems for which a polynomial time verification algorithm exists.

Example

- Assignment problem is in $\mathcal{P}$ and $\mathcal{NP}$
- LP is in $\mathcal{P}$ and $\mathcal{NP}$
- IP is in $\mathcal{NP}$
Reduction, (Def 6.5)

A decision problem DECISION1 can be reduced (converted) to a decision problem DECISION2 in polynomial time if

- We can find a mapping $f$ which for any instance $X \in$ DECISION1 returns an instance $f(X) \in$ DECISION2
- If $X$ is a “yes” instance then $f(X)$ is a “yes” instance (if and only if)
- The mapping $f$ runs in polynomial time measured in $L(X)$

We write

$$\text{DECISION1} \leq_p \text{DECISION2}$$

Example

\text{ASSIGNMENT-DECISION} \leq_p \text{LP-DECISION}

Assignment problem in decision form

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ 
\text{subject to} & \quad \sum_{i=1}^{n} x_{ij} = 1 \\ & \quad \sum_{j=1}^{n} x_{ij} = 1 \\ & \quad x_{ij} \in \{0, 1\}
\end{align*}
\]

Since constraint matrix is TU, we can formulate problem as an LP-problem
The most difficult \(\mathcal{NP}\)-problems

A problem \(Q\) is \(\mathcal{NP}\)-complete if

1. \(Q \in \mathcal{NP}\)
2. \(\forall R \in \mathcal{NP} : R \leq_p Q\)

The class of \(\mathcal{NP}\)-complete problems is denoted \(\mathcal{NPC}\)

It is difficult to prove (2). We will only show it for CIRCUIT-SAT, and then use transitive properties to show (2) for other problems

\(\mathcal{NPC}\) is our clue to the \(\mathcal{NP}=\mathcal{P}\) question

- If a problem \(Q \in \mathcal{NP}\) cannot be solved in pol. time
- If a problem \(Q \in \mathcal{NPC}\) can be solved in pol. time
The computer as a circuit

The computer is composed of gates

<table>
<thead>
<tr>
<th>NOT</th>
<th>AND</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x1</th>
<th>x2</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Example: a latch

A latch can be seen as a simple “memory”
Circuit satisfiability

CIRCUIT-SAT is the following problem

CIRCUIT-SAT: Given a circuit, is it possible to assign values \{0, 1\} to the input gates \(x_1, \ldots, x_n\) so that the output \(z\) becomes 1

Circuit satisfiability is in \(\mathcal{NP}\)

Certificate \(Y = \{x_1, \ldots, x_n\}\) does not work
Circuit satisfiability is in $\mathcal{NP}$

Certificate $Y = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ is better

Verification algorithm: check all gates, input and output. Runs in polynomial time.
The computer as a circuit

Running a verification algorithm \( A \) on a computer

- **CPU, central processing unit**
  Can make an arithmetic-logic operation in each clock cycle (+,-,and,or,if)
  \( constant \ number \ of \ gates \)

- **RAM, random access memory**
  Can store algorithm \( A \), instance \( X \), certificate \( Y \), variables \( V \), output \( Z \)
  \( |A| \ polynomial \ in \ X \)
  \( |Y| \ polynomial \ in \ X \)
  \( |V| \ polynomial \ in \ X \)

- **OUTPUT**
  Value \( \{0, 1\} \) is written to RAM, where hardware shows it on screen
  \( constant \ number \ of \ gates \)

A polynomial algorithm \( A \) cannot be longer than \( |X|^k \).
A polynomial algorithm \( A \) cannot use more space (variables) than \( |X|^k \)
Circuit satisfiability is $\mathcal{NP}$-complete

```
+-----------------+-----------------+-----------------+-----------------+---------+
| algorithm A     | instance X      | certificate Y   | variables V     | RAM    |
| y_1, y_2, \ldots, y_n |
```

```
+-----------------+-----------------+-----------------+-----------------+---------+
| algorithm A     | instance X      | certificate Y   | variables V     | RAM    |

```

```
+-----------------+-----------------+-----------------+-----------------+---------+
| algorithm A     | instance X      | certificate Y   | variables V     | RAM    |

```

```
+-----------------+-----------------+-----------------+-----------------+---------+
| algorithm A     | instance X      | certificate Y   | variables V     | RAM    |

```

```
+-----------------+-----------------+-----------------+-----------------+---------+
| algorithm A     | instance X      | certificate Y   | variables V     | RAM    |

```

```
+-----------------+-----------------+-----------------+-----------------+---------+
| algorithm A     | instance X      | certificate Y   | variables V     | RAM    |

```

```
+-----------------+-----------------+-----------------+-----------------+---------+
| algorithm A     | instance X      | certificate Y   | variables V     | RAM    |
| z               |
```

15