NP-Complete Problems So Far

Arrows show reductions discussed in class.
We could construct a polynomial reduction between any pair.

Clicker Question

Suppose we want to reduce Ham-Path to Ham-Cycle. Which of the following statements is true?

A. Trivial Reduction (map to same problem instance): any cycle with all nodes is also a path with all nodes
B. Ham-Path $\not\leq_P$ Ham-Cycle since not all paths are cycles
C. Ham-Path $\not\leq_P$ Ham-Cycle since there are some graphs that have a Hamiltonian path but no Hamiltonian cycle
D. None of the above

Clicker Question

Which of the following graph problems are in NP?

A. Length of longest simple path is $\leq k$
B. Length of longest simple path is $\geq k$
C. Length of longest simple path is $= k$
D. Find length of longest simple path.
E. All of the above.

Numerical problems

Subset Sum decision problem: given $n$ items with weights $w_1, \ldots, w_n$, is there a subset of items whose weight is exactly $W$?

Dynamic programming: $O(nW)$ pseudo-polynomial time algorithm (not polynomial in input length $n \log W$)
Subset Sum Warmup

Does this instance have a solution?

\[
\begin{array}{c|c}
\text{w1} & 1010 \\
\text{w2} & 1001 \\
\text{w3} & 0110 \\
\text{w4} & 0101 \\
\hline
\text{W} & 1111
\end{array}
\]

A. Yes  
B. No

Subset Sum Warmup

For which nonzero values of \(y\) does this instance have a solution?

\[
\begin{array}{c}
10010 \\
10001 \\
01001 \\
01010 \\
00111 \\
00100 \\
\hline
1113y
\end{array}
\]

A. \(y = 1\)
B. \(y = 1, 2\)
C. \(y = 1, 2, 3\)

Subset Sum Warmup

For which nonzero values of \(y\) does this instance have a solution?

\[
\begin{array}{c}
10010 \\
10001 \\
01001 \\
01010 \\
00111 \\
00100 \\
\hline
1112y
\end{array}
\]

A. \(y = 1\)
B. \(y = 1, 2\)
C. \(y = 1, 2, 3\)

Subset Sum Warmup

For which nonzero values of \(y\) does this instance have a solution?

\[
\begin{array}{c}
10010 \\
10001 \\
01001 \\
01010 \\
00111 \\
00100 \\
\hline
1111y
\end{array}
\]

A. \(y = 1\)
B. \(y = 1, 2\)
C. \(y = 1, 2, 3\)

Subset Sum Reduction: Variables

\[
(x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3)
\]

<table>
<thead>
<tr>
<th>Item</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_1)</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(t_2)</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>(t_3)</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>(W)</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Items \(t_i, f_i\) for each \(x_i\); correspond to truth assignment
- Weights \(\Rightarrow\) select exactly one
- (Numbers are base 10)
**Subset Sum Reduction: Clauses**

\[(x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3)\]

<table>
<thead>
<tr>
<th>Item</th>
<th>variable digits</th>
<th>clause digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_1)</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>(f_1)</td>
<td>1 0 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>(t_2)</td>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>(f_2)</td>
<td>0 1 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>(t_3)</td>
<td>0 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>(f_3)</td>
<td>0 0 1</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

\[W | 1 1 1 | \ ? \ ? \ ?\]

- Clause digit equal to 1 iff \(x_i\) assignment satisfies \(C_j\)
- Total for clause digit \(> 0\) iff assignment satisfies \(C_j\)
- Goal: all clause digits \(> 0\). How to set \(W\) to enforce this?
  Total could be 1, 2, 3 for satisfied clause.

**Subset Sum Reduction: Extra Items**

\[(x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3)\]

<table>
<thead>
<tr>
<th>Item</th>
<th>variable digits</th>
<th>clause digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_1)</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>(f_1)</td>
<td>1 0 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>(t_2)</td>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>(f_2)</td>
<td>0 1 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>(t_3)</td>
<td>0 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>(f_3)</td>
<td>0 0 1</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

\[W | 1 1 1 | 3 3 3\]

- Set all clause digits of \(W\) to 3... then add dummy items to increase total by at most two.

**Subset Sum Reduction: Final Construction**

\[(x_1 \lor \neg x_2 \lor x_3) \land (\neg x_1 \lor x_2 \lor \neg x_3) \land (\neg x_1 \lor \neg x_2 \lor x_3)\]

<table>
<thead>
<tr>
<th>Item</th>
<th>variable digits</th>
<th>clause digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_1)</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>(f_1)</td>
<td>1 0 0</td>
<td>0 1 1</td>
</tr>
<tr>
<td>(t_2)</td>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>(f_2)</td>
<td>0 1 0</td>
<td>1 0 1</td>
</tr>
<tr>
<td>(t_3)</td>
<td>0 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>(f_3)</td>
<td>0 0 1</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

\[W | 1 1 1 | 3 3 3\]

- Two dummy items per clause ⇒ can increase total by up to 2
- Can make total exactly 3 iff total of non-dummy items is \(> 0\)

**Subset Sum Proof**

- All numbers (including \(W\)) are polynomially long.
- If \(\Phi\) satisfiable,
  - Select \(t_i\) if \(x_i = 1\) in satisfying assignment else select \(f_i\).
  - Take \(y_j, z_j\) as needed.
- If subset exists with sum \(W\)
  - Either \(t_i\) or \(f_i\) is chosen. Assign \(x_i\) accordingly.
  - For each clause, at least one term must be selected, otherwise clause digit is \(< 3\).

**Subset Sum Review**

- All weights have \(n + m\) digits
- Digits 1 to \(n\): For variable \(x_i\), create two items \(t_i, f_i\)
  - Both have \(i\)th digit equal to 1
  - All other items have zero in this digit
  - \(i\)th digit of \(W = 1\) ⇒ select exactly one of \(t_i, f_i\)
- The \(n + j\)th digit corresponds to clause \(C_j\)
  - If \(x_i \in C_j\), set \(n + j\)th digit of \(t_i = 1\)
  - If \(\neg x_i \in C_j\), set \(n + j\)th digit of \(f_i = 1\)
  - Everything else 0.
- Set \(n + j\)th digit of \(W = 3\)
  - Create two "dummy" items \(y_j, z_j\) with 1 in position \(n + j\)
  - Can select dummies to yield total of 3 in position \(n + j\) iff \(C_j\) is satisfied

**Warning**

**Theorem.** \textsc{SubsetSum} is \textsc{NP}-Complete.

But reducing from \textsc{Subset} \textsc{Sum} can be tricky!

- If reducing \textsc{SubsetSum} \(\leq_p X\), reduction needs to be polynomial in \(\log(W)\) (number of digits).
- Also, if \(W\) polynomial in \(n\), dynamic programming still polynomial!
  (this special case is not \textsc{NP}-complete, provided \(P \neq \textsc{NP}\)
Graph Coloring

**Def.** A $k$-coloring of a graph $G = (V, E)$ is a function $f : V \rightarrow \{1, \ldots, k\}$ such that for all $(u, v) \in E$, $f(u) \neq f(v)$.

**Problem.** Given $G = (V, E)$ and number $k$, does $G$ have a $k$-coloring?

Many applications
- Actually coloring maps!
- Scheduling jobs on machine with competing resources.
- Allocating variables to registers in a compiler.

**Claim.** 2-COLORING $\in P$ (equivalent to bipartite testing)

**Theorem.** 3-COLORING is NP-Complete.

Reduction: Clause Gadget

For clause $x_i \lor \neg x_j \lor x_k$

![Clause Gadget Diagram]

Top node can be colored iff not all three $v$-nodes are $F$.

Proof

- Graph is polynomial in $n + m$.
- If satisfying assignment
  - Color $B, T, F$ then $v_1$ as $T$ if $\phi(x_i) = 1$.
  - Since clauses satisfied, can color each gadget.
- If graph 3-colorable
  - One of $v_0, v_1$ must get $T$ color.
  - Clause gadget colorable iff clause satisfied.

**Question.** What about $k$-coloring?

Clicker Question

Which of the following is true?

A: If we can reduce 3-coloring to $k$-coloring, then $k$-coloring is NP-complete
B: $k$-coloring is NP-complete since any 3-coloring is also a $k$-coloring for $k \geq 3$
C: $k$-coloring is not NP-complete since 3-coloring is the hardest case, for $k > 3$ the coloring is easier
D: $k$-coloring is not NP-complete because the 4-color theorem has been proved

NP-Completeness Recap

Types of hard problems:

- Circuit-SAT
- Constraint satisfaction
- Packing
- Indep-Set
- Vertex-Cover
- Set-Cover
- Hamilton-Cycle
- Ham-Path
- Traveling-Salesman
- Circuit-SAT
- Partitioning
- Graph-Coloring
- Subset-Sum
- 0-1 Knapsack
- Numerical
- Sequencing

... any many others. See book or other sources for more examples. You can use any known NP-complete problem to prove a new problem is NP-complete.