We will see that the language SubsetSum is NP-complete:

- SubsetSum = \{ (S, t) \mid S = \{s_1, \ldots, s_n\} is a set of binary integers, t is a binary integer, and there exists a subset \( T \subseteq S \) such that \( \sum_{s_i \in T} s_i = t \) \}

Topics Covered

1. SubsetSum is NP-Complete
2. Perspectives on P, NP, and coNP

1 SubsetSum is NP-Complete

Proof that SubsetSum is NP-Complete  We need to show that SubsetSum is both in NP and NP-hard. To show that it is in NP, consider the following verifier \( V \):

\( V \), on input \( (S, t, w) \):

1. If \( S \) is a set of \( n \) integers, treat \( w \) as an \( n \)-bit string of ones and zeroes.
2. Compute \( \sum_{i=1}^{n} w_i s_i \).
3. If this sum is \( t \), accept. Otherwise, reject.

The verifier \( V \) treats the bits of \( w \) as indicators of which \( s_i \) terms are included in the sum. If there is a way to sum up elements of \( S \) to \( t \), then the corresponding witness has ones in the positions corresponding to these elements, with zeroes elsewhere. \( V \) runs in polynomial time because it linearly processes the elements of \( S \) to compute a sum, and checks for equality with \( t \).

Next, we will show that SubsetSum is NP-hard by a reduction from 3SAT. On input \( \langle \phi \rangle \), suppose \( \phi = C_1 \land \cdots \land C_\ell \) over clauses \( C_1, \ldots, C_\ell \) and variables \( x_1, \ldots, x_k \). For each variable \( x_i \), we introduce two new variables: \( y_i \), which corresponds to \( x_i \) being ‘true’; and \( z_i \), which
corresponds to \( x_i \) being ‘false’. We also introduce “slack” variables \( g_i \) and \( h_i \) for each clause from 1 to \( \ell \). To construct the integers in the set \( S \), we use the rows of following table:

<table>
<thead>
<tr>
<th>( y_1 )</th>
<th>( x_1 )</th>
<th>( \ldots )</th>
<th>( x_i )</th>
<th>( \ldots )</th>
<th>( x_k )</th>
<th>( C_1 )</th>
<th>( \ldots )</th>
<th>( C_j )</th>
<th>( \ldots )</th>
<th>( C_\ell )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_1 )</td>
<td>( \ldots )</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_j )</td>
<td>( \ldots )</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_\ell )</td>
</tr>
<tr>
<td>( z_1 )</td>
<td>1</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>1 iff ( \neg x_1 ) ( \neg ) ( C_1 )</td>
<td>( \ldots )</td>
<td>1 iff ( \neg x_1 ) ( \neg ) ( C_j )</td>
<td>( \ldots )</td>
<td>1 iff ( \neg x_1 ) ( \neg ) ( C_\ell )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_1 )</td>
<td>( \ldots )</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_j )</td>
<td>( \ldots )</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_\ell )</td>
<td></td>
</tr>
<tr>
<td>( y_\ell )</td>
<td>0</td>
<td>( \ldots )</td>
<td>1</td>
<td>( \ldots )</td>
<td>0</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_1 )</td>
<td>( \ldots )</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_j )</td>
<td>( \ldots )</td>
<td>1 iff ( x_1 ) ( \neg ) ( C_\ell )</td>
</tr>
<tr>
<td>( z_\ell )</td>
<td>0</td>
<td>( \ldots )</td>
<td>1</td>
<td>( \ldots )</td>
<td>0</td>
<td>1 iff ( \neg x_1 ) ( \neg ) ( C_1 )</td>
<td>( \ldots )</td>
<td>1 iff ( \neg x_1 ) ( \neg ) ( C_j )</td>
<td>( \ldots )</td>
<td>1 iff ( \neg x_1 ) ( \neg ) ( C_\ell )</td>
</tr>
<tr>
<td>( y_k )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>1</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( h_1 )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( g_1 )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>1</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( h_j )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>1</td>
<td>( \ldots )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( g_j )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>1</td>
<td>( \ldots )</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>( h_\ell )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>0</td>
<td>( \ldots )</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>( t )</td>
<td>1</td>
<td>( \ldots )</td>
<td>1</td>
<td>( \ldots )</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In this table, each \( y_i \) corresponds to the variable \( x_i \) being true in an assignment. There is a 1 in the \( x_i \) column, and 0s in all other variable columns. There is a 1 in the clause \( C_j \) column if setting \( x_i \) to true satisfies \( C_j \), and a 0 otherwise. Similarly, each \( z_i \) corresponds to \( x_i \) being false. For any assignment, we expect to have each variable \( x_i \) be either true or false, so exactly one of \( y_i \) or \( z_i \) is chosen. Each clause is satisfied by zero to three literals, so we use the slack variables to ensure that each clause column adds to three if there is a satisfying assignment. The reduction outputs \( S = \{ y_1, z_1, \ldots, y_k, z_k, g_1, h_1, \ldots, g_\ell, h_\ell \} \) and \( t = 1^{k}3^{\ell} \).

To analyze this reduction, first note that it takes polynomial time (although we won’t prove this in detail). For correctness, we must show that \( \langle \phi \rangle \) is in 3SAT if and only if \( \langle S, t \rangle \) is in \text{SubsetSum}.

1. First suppose \( \langle \phi \rangle \) is in 3SAT, with a satisfying assignment \( a_1, \ldots, a_k \). Construct the set \( T \) as follows:
1. Initialize $T$ to be empty.

2. For $i$ from 1 to $k$:
   (a) If $a_i$ is ‘true’, add $y_i$ to $T$.
   (b) Else, add $z_i$ to $T$.

3. For $i$ from 1 to $\ell$:
   (a) If the sum so far in position $C_i$ is 1, add both $g_i$ and $h_i$ to $T$.
   (b) Else, if the sum so far in position $C_i$ is 2, add only $g_i$ to $T$.

By construction, we can sum rows of the table such that there is a 1 in each $x_i$ position and a 3 in each $C_i$ position. Thus, $\sum_{s_i \in T} s_i = t$.

2. For the other direction, suppose $T \subseteq S$ such that $\sum_{s_i \in T} s_i = t$. Construct a truth assignment $a_1, \ldots, a_k$ as follows:

   $$a_i = \begin{cases} 'true' & \text{if } y_i \in T \\ 'false' & \text{if } z_i \in T \end{cases}$$

Exactly one of $y_i$ and $z_i$ is in $T$ because the corresponding digit in the target $t$ is 1. The assignment $a_1, \ldots, a_k$ satisfies each clause $C_j$ because the corresponding digit in $t$ is 3. We can use at most two slack variables $g_j$ and $h_j$ to sum to 3, so at least one $y_i$ or $z_i$ has a 1 in the $C_j$ column, indicating that it satisfies the clause $C_j$. Thus, there is a satisfying assignment for $\phi$.

This shows that $\langle \phi \rangle$ is in 3SAT if and only if the constructed $\langle S, t \rangle$ is in SubsetSum. Since 3SAT is NP-complete, SubsetSum is NP-hard and thus NP-complete. ■

Notes on the Reduction  There are a few observations and modifications we can make for this construction:

1. First, our constructed set $S$ is a multiset because $g_i = h_i$ for all $i$ from 1 to $\ell$. To convert $S$ to a set, we can add another $\ell$ columns and slack variables. Each $g_i$ has a 0 in all $\ell$ of these positions. Each $h_i$ has a 1 in the $i$th of these $\ell$ columns, and 0s elsewhere. The $\ell$ new variables are of the form $e_i$ where the first $k + \ell$ digits are 0, and the $i$th of the final $\ell$ digits is a 1. We then add $\ell$ 1s to the end of the target $t$. By doing so, we ensure that every row represents a unique integer.

2. A different way to convert $S$ to a set is to use 2s instead of 1s for each row $h_i$, and change the 3s in the target to 4s.
3. Second, the base with which we construct the table matters. All digits are 0 or 1, except for the 3s in the target. We should use a base of at least 4, so that modular arithmetic does not change the correlation between the number of 1s in a column and the corresponding digit in $t$. We will generally consider numbers to be in decimal.

4. Our SubsetSum reduction relies on a binary encoding of the integers in $(S, t)$. We can convert decimal to binary numbers in linear time. However, converting from decimal to unary takes exponential time with respect to the input length. In fact, UnarySubsetSum is in P. That is, if $S$ and $t$ are represented in unary, we can solve UnarySubsetSum in polynomial time using dynamic programming. The only change is the representation of the input; an algorithm might take exponential time with respect to binary integers but polynomial time with respect to the same integers in unary.

2 Perspectives on P, NP, and coNP

Recall that coNP consists of languages whose complements are in NP. Some examples of languages in coNP are Tautology, SAT$^C$, and 3-Uncolorable. Two big open questions are whether P = NP, and whether NP = coNP. Similar to notions of NP-completeness, we can also define coNP-completeness:

- A language $L$ is **coNP-complete** if (1) $L \in$ coNP and (2) for all $A \in$ coNP, $A \leq_p L$.
  
  This second condition is called **coNP-hardness**.

**Lemma 1** If $A \leq_p B$ then $A^C \leq_p B^C$.

**Proof of Lemma 1** We have seen this lemma previously in the context of NP-completeness.

**Lemma 2** If $L$ is NP-complete then $L^C$ is coNP-complete.

**Proof of Lemma 2** First, note that $L^C \in$ coNP because $L \in$ NP by the definition of NP-completeness. Let $A$ be any language in coNP. Then $A^C \leq_p L$, as $L$ is NP-hard. By Lemma 1, $A \leq_p L^C$. This holds for any $A \in$ coNP, so $L^C$ is coNP-hard. Hence, $L^C$ is coNP-complete. ■

**Lemma 3** If $A \leq_p B$ and $B \in$ NP, then $A \in$ NP.

**Proof of Lemma 3** If $B \in$ NP, then there exists a poly-time NTM whose language is $B$. Let $f$ be the poly-time computable function mapping instances of $A$ to instances of $B$. Since $A \leq_p B$, there exists $f$ such that $x \in A$ if and only if $f(x) \in B$. We can construct
a poly-time NTM for $A$ as follows: “On input $x$, compute $f(x)$. Run the NTM for $B$ on $f(x)$ and return the same result.” As there is a poly-time NTM whose language is $A$, $A \in \text{NP}$.  

**Lemma 4** If there exists a language $L$ such that $L \in \text{coNP}$ and $L$ is NP-complete, then $\text{NP} = \text{coNP}$.  

**Proof of Lemma 4** Suppose there exists $L$ which is both in coNP and is NP-complete. To show that $\text{NP} = \text{coNP}$, we need to show that (1) $\text{NP} \subseteq \text{coNP}$ and (2) $\text{coNP} \subseteq \text{NP}$.

1. Let $A \in \text{NP}$. Then $A^C \in \text{coNP}$ and $A \leq_p L$, so $A^C \leq_p L^C$. Because $L^C \in \text{NP}$, $A^C \in \text{NP}$ by Lemma 3. Then $A \in \text{coNP}$, proving that $\text{NP} \subseteq \text{coNP}$.

2. Let $A \in \text{coNP}$. Then $A^C \in \text{NP} \subseteq \text{coNP}$ by the previous result. Since $A^C \in \text{coNP}$, we have $A \in \text{NP}$. Thus, $\text{coNP} \subseteq \text{NP}$.

Therefore, if such a language $L$ exists, $\text{NP} = \text{coNP}$.  

We’re not sure if $P = \text{NP}$ or if $\text{NP} = \text{coNP}$. There are three possible situations to consider:

1. $P = \text{NP}$, in which case several complexity classes collapse to one.
2. P ≠ NP and NP = coNP, where NP and coNP collapse to a single class.

3. P ≠ NP and NP ≠ coNP, where the classes are all disjoint. Without proof either way, many computer scientists think that this is the most likely case.