Question 1: Write your name and student number.

Solution: Fred VanVleet, 23

Question 2: The \((0,1)\)-integer programming problem is defined as follows:

\[
\text{IntProg} := \{ (A,c) : \quad A \text{ is an integer } m \times n \text{ matrix for some integers } m \text{ and } n, \\
\quad c \text{ is an integer vector of length } m, \text{ and} \\
\quad \exists x \in \{0,1\}^n \text{ such that } Ax \leq c \text{ (componentwise) } \}.
\]

Prove that the language \(\text{IntProg}\) is in \(\text{NP}\).

Solution: The verification algorithm \(V\) does the following:

- It takes as input
  - an integer \(m \times n\) matrix \(A\), an integer vector \(c\) of length \(m\), and
  - a binary vector \(x\) of length \(n\).

- The verification algorithm computes the vector \(y = Ax\) (which has length \(m\)), then it checks if the \(i\)-th coordinate of \(y\) is at most the \(i\)-th coordinate of \(c\), for all \(i = 1, 2, \ldots, m\). If this is the case, then it returns YES. Otherwise, it returns NO.

We first bound the running time of the verification algorithm: To compute \(Ax\), we have to compute \(m\) times the dot-product of two vectors of length \(n\). Since one such dot-product can be computed in \(O(n)\) time, the vector \(Ax\) can be computed in \(O(mn)\) time. Checking if \(Ax \leq c\) can be done in \(O(m)\) time. Thus, the entire verification algorithm takes time \(O(mn + m) = O(mn)\). We have to show that this running time is polynomial in the length of \((A,c)\) (i.e., in the length of the input for \(\text{IntProg}\)). Since \((A,c)\) contains an \(m \times n\) matrix, its length is at least \(mn\). Therefore, the running time of the verification algorithm is polynomial in the length of \((A,c)\).

The certificate is of course the binary vector \(x\):

\[(A,c) \in \text{IntProg} \quad \text{iff there exists a binary vector } x \text{ of length } n, \text{ such that } Ax \leq c \quad \text{iff there exists a certificate } x \text{ such that } V(A,c,x) \text{ returns YES.}\]

The length of the certificate \(x\) is equal to \(n\), which is \(O(mn)\), i.e., the length of the certificate is polynomial in the length of \((A,c)\).

This shows that \(\text{IntProg} \in \text{NP}\).
Question 3: The *subset sum problem* is defined as follows:

\[ \text{SubsetSum} := \{ (a_1, a_2, \ldots, a_m, b) : m, a_1, a_2, \ldots, a_m, b \text{ are integers and } \exists I \subseteq \{1, 2, \ldots, m\} \text{ such that } \sum_{i \in I} a_i = b \} \]

Prove that \( \text{SubsetSum} \leq_P \text{IntProg} \), i.e., in polynomial time, \( \text{SubsetSum} \) can be reduced to \( \text{IntProg} \).

Solution: We need a function \( f \) such that

- \( f \) maps \((a_1, \ldots, a_m, b)\) to a pair \((A, c)\),
- \((a_1, \ldots, a_m, b) \in \text{SubsetSum} \) if and only if \((A, c) \in \text{IntProg} \),
- \((A, c) \) can be computed in time that is polynomial in the length of \((a_1, \ldots, a_m, b)\).

Consider an input \((a_1, \ldots, a_m, b)\) for \( \text{SubsetSum} \). We define \( A \) to be the \( 2 \times m \) matrix

\[
A = \begin{pmatrix} a_1 & a_2 & \cdots & a_m \\ -a_1 & -a_2 & \cdots & -a_m \end{pmatrix}
\]

and we define \( c \) to be the following vector of length 2:

\[
c = \begin{pmatrix} b \\ -b \end{pmatrix}.
\]

We define \( f(a_1, \ldots, a_m, b) = (A, c) \).

Given \((a_1, \ldots, a_m, b)\), the matrix \( A \) and the vector \( c \) can be computed in \( O(m) \) time, which is polynomial in the length of \((a_1, \ldots, a_m, b)\).

It remains to show that

\((a_1, \ldots, a_m, b) \in \text{SubsetSum} \) if and only if \((A, c) \in \text{IntProg} \).

Assume that \((a_1, \ldots, a_m, b) \in \text{SubsetSum} \). Then there exists a subset \( I \) of \( \{1, 2, \ldots, m\} \), such that \( \sum_{i \in I} a_i = b \). Let \( x \) be the vector in \( \{0, 1\}^m \), where \( x_i = 1 \) if \( i \in I \) and \( x_i = 0 \) if \( i \notin I \). Then

\[
\sum_{i=1}^{m} a_i x_i = \sum_{i \in I} a_i = b,
\]

which implies that

\[
Ax = \begin{pmatrix} b \\ -b \end{pmatrix} = c.
\]

In particular, we have \( Ax \leq c \) (componentwise), which means that \((A, c) \in \text{IntProg} \).

Conversely, assume that \((A, c) \in \text{IntProg} \). Then there exists a binary vector \( x \) of length \( m \), such that \( Ax \leq c \) (componentwise). This means that

\[
\sum_{i=1}^{m} a_i x_i \leq b
\]
and
\[ \sum_{i=1}^{m} -a_ix_i \leq -b. \]

The latter inequality is equivalent to
\[ \sum_{i=1}^{m} a_ix_i \geq b. \]

It follows that
\[ \sum_{i=1}^{m} a_ix_i = b. \]

Define \( I = \{i : x_i = 1\} \). Then \( \sum_{i \in I} a_i = b \) and, therefore, \( (a_1, \ldots, a_m, b) \in \text{SubsetSum} \).

Question 4: The three-coloring problem is defined as follows:

\[
3\text{Color} := \{G : G \text{ is a graph whose vertices can be colored using three colors such that any two adjacent vertices have distinct colors \}}\}
\]

The clique covering problem is defined as follows:

\[
\text{CliqueCover} := \{(G, k) : G = (V, E) \text{ is a graph whose vertex set } V \text{ can be partitioned into } k \text{ subsets } V_1, V_2, \ldots, V_k \text{ such that each subset } V_i \text{ forms a clique in } G \}\}
\]

(4.1) Prove that the language \( 3\text{Color} \) is in \( \text{NP} \).

(4.2) Prove that the language \( \text{CliqueCover} \) is in \( \text{NP} \).

(4.3) Prove that \( 3\text{Color} \leq_P \text{CliqueCover} \), i.e., in polynomial time, \( 3\text{Color} \) can be reduced to \( \text{CliqueCover} \).

Solution: We start with (4.1). The verification algorithm \( V \) does the following:

- It takes as input
  - a graph \( G = (V, E) \), where \( V = \{v_1, v_2, \ldots, v_n\} \), and
  - a sequence \( c_1, c_2, \ldots, c_n \).
- The verification algorithm
  - checks if each \( c_i \) is an integer in \( \{1, 2, 3\} \),
  - checks if for each edge \( \{v_i, v_j\} \in E \), \( c_i \neq c_j \).

If both these tests are successful, the algorithm returns YES. Otherwise, it returns NO.

It is clear that the running time of the verification algorithm is \( O(n + |E|) \). We have to show that this running time is polynomial in the length of \( G \) (i.e., in the length of the input
for 3COLOR). Obviously, the length of $G$ is equal to $\Theta(n + |E|)$. Therefore, the running time of the verification algorithm is polynomial in the length of $G$.

The certificate is of course the sequence $c_1, c_2, \ldots, c_n$:

\[
G \in 3\text{COLOR} \quad \text{iff there exists a color in } \{1, 2, 3\} \text{ for each vertex such that any two adjacent vertices have distinct colors}
\]

\[
\text{iff there exists a sequence } C = (c_1, c_2, \ldots, c_n) \in \{1, 2, 3\}^n \text{ such that}
\]

\[
\mathcal{V}(G, C) \text{ returns YES.}
\]

The length of the certificate $C$ is equal to $n$, which is $O(n + |E|)$, i.e., the length of the certificate is polynomial in the length of $G$.

This shows that $3\text{COLOR} \in \text{NP}$.

For (4.2), the verification algorithm $\mathcal{V}$ does the following:

- It takes as input
  - a graph $G = (V, E)$, an integer $k$, and
  - a sequence $V_1, V_2, \ldots, V_k$.

- The verification algorithm
  - checks if each $V_i$ is a subset of $V$,
  - checks if $V_1 \cup V_2 \cup \ldots \cup V_k = V$,
  - checks if $V_i \cap V_j = \emptyset$ for all $i \neq j$,
  - checks if each $V_i$ is a clique in $G$ (this is done by checking if each pair of vertices in $V_i$ is connected by an edge in $G$).

If all these tests are successful, the algorithm returns YES. Otherwise, it returns NO.

We have to show that the running time of the verification algorithm is polynomial in the length of $(V, k)$:

- Check if each $V_i$ is a subset of $V$: this can be done by brute-force in time

\[
O\left(\sum_{i=1}^{k} |V_i| \cdot |V|\right) = O(k|V|^2),
\]

which is polynomial.

- Check if $V_1 \cup V_2 \cup \ldots \cup V_k = V$: this can be done by brute-force in time

\[
O\left(\sum_{i=1}^{k} |V_i| \cdot |V|\right) = O(k|V|^2),
\]

which is polynomial.
• Check if \( V_i \cap V_j = \emptyset \) for all \( i \neq j \): this can be done by brute-force in time

\[
O \left( \binom{k}{2} |V|^2 \right) = O(k^2|V|^2),
\]

which is polynomial.

• Check if each \( V_i \) is a clique in \( G \): this can be done in time

\[
O \left( \sum_{i=1}^{k} |V_i|^2 \right) = O(k|V|^2),
\]

which is polynomial.

Thus, the total running time of the verification algorithm is polynomial in the length of \((G, k)\).

The certificate is of course the sequence \( V_1, V_2, \ldots, V_k \):

\[(G, k) \in \text{CLIQUECOVER} \quad \text{iff there exists a partition of } V \text{ into } k \text{ cliques}
\]

\[
\text{iff there exists a sequence } C = (V_1, V_2, \ldots, V_k) \text{ such that}
\]

\[
\mathcal{V}(G, k, C) \text{ returns YES.}
\]

The length of the certificate \( C \) is equal to \( |V| \), thus it is polynomial in the length of \((G, k)\).

This shows that \( \text{CLIQUECOVER} \in \text{NP} \).

Question (4.3) asks to show that \( 3\text{COLOR} \leq_P \text{CLIQUECOVER} \). For this, we need a function \( f \), such that

• \( f \) maps an input graph \( G \) for \( 3\text{COLOR} \) to an input \( (G', k) \) for \( \text{CLIQUECOVER} \),

• \( G \in 3\text{COLOR} \) if and only if \( (G', k) \in \text{CLIQUECOVER} \),

• \( (G', k) \) can be computed in time that is polynomial in the length of \( G \).

Given a graph \( G \), we define

\[
f(G) = (\bar{G}, 3),
\]

where \( \bar{G} \) denotes the complement of \( G \) (edges become non-edges, and non-edges become edges).

If \( G \) has \( n \) vertices, then \( f(G) \) can be computed in \( O(n^2) \) time, which is polynomial in the length of \( G \).

It remains to show that

\[ G \in 3\text{COLOR} \text{ if and only if } (\bar{G}, 3) \in \text{CLIQUECOVER}. \]

First assume that \( G \in 3\text{COLOR} \). Then the vertices of \( G \) can be colored using the three colors 1, 2, and 3, such that adjacent vertices do not have the same color. For \( 1 \leq i \leq 3 \), let \( V_i \)
be the set of vertices that have color \( i \). Then \( V_1, V_2, \) and \( V_3 \) form a partition of the vertex set of \( G \), which is the same as the vertex set \( \bar{G} \). We claim that each \( V_i \) is a clique in \( \bar{G} \). To show this, consider two arbitrary vertices \( u \) and \( v \) in \( V_i \). Then \( u \) and \( v \) have the same color, thus they are not connected by an edge in \( G \), thus they are connected by an edge in \( \bar{G} \). This shows that \(( \bar{G}, 3) \) \( \in \text{CLIQUECOVER} \).

Conversely, assume that \(( \bar{G}, 3) \) \( \in \text{CLIQUECOVER} \). Then the vertex set of \( \bar{G} \) can be partitioned into three subsets \( V_1, V_2, \) and \( V_3 \), each being a clique in \( \bar{G} \). We give each vertex in \( V_1 \) the color 1, we give each vertex in \( V_2 \) the color 2, and we give each vertex in \( V_3 \) the color 3. We claim that this is a valid three-coloring of \( G \). To show this, let \( \{u, v\} \) be an edge of \( G \), and assume that \( u \) and \( v \) both have the same color \( i \). Then \( u \) and \( v \) are both in \( V_i \). Since \( V_i \) is a clique in \( \bar{G} \), \( u \) and \( v \) are connected by an edge in \( \bar{G} \), thus they are connected by an edge in \( G \), which is a contradiction. We have shown that \( G \) \( \in \text{3COLOR} \).

**Question 5:** You are given

- a sequence \( A = (A_1, A_2, \ldots, A_k) \), where each \( A_i \) is a set consisting of three elements, and
- a sequence \( B = (B_1, B_2, \ldots, B_\ell) \), where each \( B_j \) is a set consisting of two elements.

This pair \((A, B)\) of sequences is called *good* if there exists a set \( T \) such that

- \( T \) contains at least one element of each set \( A_i \), and
- \( T \) contains at most one element of each set \( B_j \).

The *Problem without a Name* is defined as follows:

\[
\text{PROBLEMWITHOUTANAME} := \{(A, B) : \text{the pair } (A, B) \text{ is good}\}.
\]

Prove that the language \text{PROBLEMWITHOUTANAME} is \text{NP}-complete.

*Hint:* You may remember from class that 3\text{Sat} is \text{NP}-complete.

**Solution:**

<table>
<thead>
<tr>
<th>How to prove that a language ( L ) is \text{NP}-complete: If</th>
</tr>
</thead>
<tbody>
<tr>
<td>( L ) is in \text{NP},</td>
</tr>
<tr>
<td>( L' \leq_p L ), and</td>
</tr>
<tr>
<td>( L' ) is \text{NP}-complete,</td>
</tr>
<tr>
<td>then ( L ) is \text{NP}-complete.</td>
</tr>
</tbody>
</table>

We are going to show the following:

- \text{PROBLEMWITHOUTANAME} is in \text{NP}. |
• 3SAT ≤ₚ ProblemWithoutAName.

This will show that ProblemWithoutAName is NP-complete.

We first proof that ProblemWithoutAName is in NP. The verification algorithm \( V \) does the following:

- It takes as input
  - a sequence \( A = (A_1, A_2, \ldots, A_k) \), where each \( A_i \) is a set consisting of three elements,
  - a sequence \( B = (B_1, B_2, \ldots, B_\ell) \), where each \( B_j \) is a set consisting of two elements, and
  - a set \( T \).

- The verification algorithm
  - checks if \( T \) contains at least one element of each set \( A_i \), and
  - checks if \( T \) contains at most one element of each set \( B_j \).

If both these tests are successful, the algorithm returns YES. Otherwise, it returns NO.

A brute-force implementation of this verification algorithm has running time \( O(k|T| + \ell|T|) \), which is polynomial in the length of the pair \( (A, B) \).

The certificate is of course the set \( T \):

\[ (A, B) \in \text{ProblemWithoutAName} \quad \text{iff} \quad \text{there exists a set } T \text{ of size at most } \sum_{i=1}^{k} |A_i| + \sum_{j=1}^{\ell} |B_j| \text{ such that } V(A, B, T) \text{ returns YES.} \]

Observe that we can always take the certificate \( T \) to be a subset of the union of all sets \( A_i \) and all sets \( B_j \). Therefore, we only have to consider certificates \( T \) whose size is at most \( \sum_{i=1}^{k} |A_i| + \sum_{j=1}^{\ell} |B_j| \), which is polynomial in the length of \( (A, B) \). This shows that ProblemWithoutAName ∈ NP.

We next show that 3Sat ≤ₚ ProblemWithoutAName. We need a function \( f \) such that

- \( f \) maps inputs \( \varphi \) for 3Sat to inputs \( (A, B) \) for ProblemWithoutAName,
- \( \varphi \) is satisfiable if and only if \( (A, B) \) is good, and
- given the Boolean formula \( \varphi \), the pair \( (A, B) \), can be computed in time that is polynomial in the length of \( \varphi \).

How do we get this function \( f \)? Observe the following:
• $\varphi$ is an input for 3SAT, thus $\varphi$ has clauses, each consisting of exactly three literals.
• The sets in the sequence $A$ all have size three.
• If $\varphi$ is satisfiable, then for each clause, at least one of the literals must be true.
• $\varphi$ is composed of variables $x_j$, each of which can be TRUE or FALSE, i.e., it can take two values.
• The sets in the sequence $B$ all have size two.
• In any assignment of truth-values to the variables, each variable has exactly one value, in particular, it has at most one value.

This leads to the following function $f$: Consider an input

$$\varphi = C_1 \land C_2 \land \ldots \land C_m$$

for 3SAT, and denote the variables in $\varphi$ by $x_1, x_2, \ldots, x_n$.

For each $i$ with $1 \leq i \leq m$, we define

$$A_i := \text{the set consisting of the three literals in the clause } C_i.$$

If, for example, $C_i = x_1 \lor \neg x_2 \lor \neg x_3$, then $A_i = \{x_1, \neg x_2, \neg x_3\}$.

For each $j$ with $1 \leq j \leq n$, we define

$$B_j := \{x_j, \neg x_j\}.$$

Since each set $A_i$ has size three, and each set $B_j$ has size two, we obtain a valid input for PROBLEM WITHOUT ANAME. If we denote the $A_i$-sequence by $A$ and the $B_j$-sequence by $B$, then we have obtained the function $f$:

$$f(\varphi) = (A, B).$$

The time to compute $f(\varphi)$ is $O(m + n)$, which is polynomial in the length of $\varphi$. It remains to show that $\varphi$ is satisfiable if and only if $(A, B)$ is good.

First assume that $\varphi$ is satisfiable. Then, there exist truth-values for $x_1, \ldots, x_n$, such that $\varphi$ is true. We define the set $T$ in the following way:

• For each variable $x_j$: If $x_j$ is true, then we include $x_j$ in $T$. If $x_j$ is false, then we include $\neg x_j$ in $T$.

We claim that this set $T$ proves that $(A, B)$ is good:

• For each $i$, the $i$-th clause $C_i$ is true. Therefore, at least one of the literals in $C_i$ is true.
- If this literal is of the form $x_j$, then $x_j$ is true and, thus, $x_j \in T$. Since $x_j \in A_i$, we know that $T$ contains at least one element of $A_i$.

- If this literal is of the form $\neg x_j$, then $\neg x_j$ is true and, therefore, $x_j$ is false. Thus, $\neg x_j \in T$. Since $\neg x_j \in A_i$, we know that $T$ contains at least one element of $A_i$.

- For each $j$, the variable $x_j$ is true or false, but not both. Therefore, $T$ contains exactly one element of $B_j$. In particular, $T$ contains at most one element of $B_j$.

To prove the other direction, we assume that $(A, B)$ is good. Then, there exists a set $T$ that contains at least one element of each $A_i$ and at most one element of each $B_j$. We give truth-values to the variables in the following way:

- For each $j$ with $1 \leq j \leq n$: If $T \cap B_j = \{x_j\}$, then we give $x_j$ the value true; if $T \cap B_j = \{\neg x_j\}$, then we give $x_j$ the value false; if $T \cap B_j = \emptyset$, then we give $x_j$ the value true or false (you can choose).

- Observe: in this way, we do not give any variable both the value true and false.

We claim that, for these truth-values, the formula $\varphi$ is true. In order to prove this, we must show that each clause $C_i$ is true.

- Consider any clause $C_i$. This clause consists of three literals, and these literals form the set $A_i$. Since $T$ contains at least one element of $A_i$, $T$ contains at least one of these three literals.

  - If this literal is of the form $x_j$, then $x_j \in T$. Since $x_j$ is also in $B_j$, we must have $T \cap B_j = \{x_j\}$. Therefore, $x_j$ is true, which implies that $C_i$ is true.

  - If this literal is of the form $\neg x_j$, then $\neg x_j \in T$. Since $\neg x_j$ is also in $B_j$, we must have $T \cap B_j = \{\neg x_j\}$. Therefore, $x_j$ is false, which implies that $\neg x_j$ is true. Thus, $C_i$ is true.