# Reductions and $\mathcal{NP}$–Completeness

## Reductions

If $L_1$, $L_2$ are languages over the alphabets $\Sigma_1$ and $\Sigma_2$, respectively, a *reduction* from $L_1$ to $L_2$ is function $R : \Sigma_1^* \to \Sigma_2^*$ such that $R(w) \in L_2$ if and only if $w \in L_1$. We write $L_1 \leq_R L_2$. We say "$L_1$ reduces to $L_2$."[1] If $R$ is $\mathcal{P}$–TIME, we write $L_1 \leq_{\mathcal{P}} L_2$. Reductions are used often in practice to shortcut calculations. A problem that can be easily reduced to an easy problem is easy.

**Remark 1** *If $L_1 \leq_{\mathcal{P}} L_2$ and $L_2$ is $\mathcal{P}$, then $L_1$ is $\mathcal{P}$.*

**Instances.** A reduction from a problem to another need only be defined on instances of of the first problem. since we can let $R(w) = \lambda$ if $w$ is not an instance. Reductions are typically defined only on instances.

A language $L$ is in the class $\mathcal{NP}$–TIME (or simply $\mathcal{NP}$) if there is a non-deterministic machine $M$ which which accepts $L$ is time which is polynomial in the number of input bits of the given instance of $L$.[2] The computation tree of $M$ given an input $w$ is a binary tree whose height is a polynomial function of $|w|$, the number bits required to write $w$. The input is *accepted* by if $M$ is in an accepting state at at least one leaf of the computation. Thus, $L$ can be decided by a deterministic machine in exponential time, by simply exploring the entire computation tree. But could we determine the answer in polynomial time? That is an unsolved problem, the famous "$\mathcal{P} = \mathcal{NP}$" problem.

### Verification Definition of $\mathcal{NP}$

A language $L$ is $\mathcal{NP}$ if and only if there is some machine $V$ and some integer $k$ such that:

1. For every $w \in L$ there exists a string $c$, called a *certificate* for $w$, such that $V$ accepts the string $(w, c)$ in $O(n^k)$ time, where $n = |w|$.
2. If $w \notin L$ and $c$ is any string, $V$ does not accept the string $(w, c)$.

### $\mathcal{NP}$–Completeness

We define a language $L$ to be $\mathcal{NP}$*–complete* if

1. $L \in \mathcal{NP}$, and
2. Every $\mathcal{NP}$ language reduces to $L$ in polynomial time.

[1]We usually mean that $R$ is recusive, *i.e.* computable.
[2]We can assume that at any step, $M$ has at most two legal choices.

**Theorem 1** *If there is any language which is both $\mathcal{P}$–TIME and $\mathcal{NP}$–complete, then $\mathcal{P} = \mathcal{NP}$.*

*Proof:* Suppose that there is a language $L_1$ which is both $\mathcal{P}$–TIME and $\mathcal{NP}$–complete. Let Let $L_2$ be any $\mathcal{NP}$ language Then $L_2 \leq_{\mathcal{P}} L_1$ by the definition of $\mathcal{NP}$-completeness. Since $L_1$ is $\mathcal{P}$, $L_2$ is $\mathcal{P}$ by Remark rem: P implies P. □

## Boolean Satisfiability

Many $\mathcal{NP}$–COMPLETE problems (languages) have been identified, and the number grows constantly. The first such problem identified is SAT, Boolean satisfiability, proved $\mathcal{NP}$–complete by Theorem **??**, the Cook Levin theorem. Using that theorem and Theorem **??**, thousands (or more) additional $\mathcal{NP}$–complete problems have been found.

Let Bool be the languages of all *Boolean expressions*, defined to be expressions consisting of variables and operators, where all variables have Boolean type and all operators are Boolean. To shorten our notation, we use "+" for *or*, "·" for *and* and "!" for *not.* An *assignment* of a Boolean expression $E$ is an assignment of truth values (there are only two truth values, *true* $= 1$ and *false* $= 0$) to each variable that appears in $E$. An assignment is *satisfying* if given those values, $E$ is *true.* $E$ is *satisfiable* if it has a satisfying assignment, otherwise $E$ is a *contradiction*. For example, $x \cdot !x$ is a contradiction, since its value is false regardless of the assigned value of $x$, while $x \cdot !y$ is satisfiable, because the assignment $x = 1$, $y = 0$ is satisfying. Let SAT $\subseteq$ BOOL be the satisfiable expressions. We also write SAT to be the problem of determining whether $E \in$ SAT. Any satisfying assignment of a $E$ is a certificate which verifies that $E \in$ SAT.

**Theorem 2 (Cook-Levin)** *SAT is $\mathcal{NP}$–complete.*

The proof of Theorem **??** is long, but straightforward. You can find it in books or on the internet.

**Theorem 3** *If $L_1$ is $\mathcal{NP}$-complete and $L_2$ is $\mathcal{NP}$, and there is a polynomial reduction $R_1$ of $L_1$ to $L_2$, hence $L_2$ is $\mathcal{NP}$–complete.*

*Proof:* We need only prove that every $\mathcal{NP}$ language reduces to $L_2$ in polynomial time. Let $L_3 \in \mathcal{NP}$. Since $L_1$ is $\mathcal{NP}$-complete, there is a polynomial time reduction $R_2$ of $L_3$ to $L_1$. The composition $R_2 \circ R_1$ is a polynomial time reduction of $L_3$ to $L_2$. □

Here is a reduction chain of $\mathcal{NP}$–complete problems.

$$SAT \leq_{\mathcal{P}} 3 - SAT \leq_{\mathcal{P}} IND \leq_{\mathcal{P}} SubsetSum \leq_{\mathcal{P}} Partition$$

These problems and reductions are described below.

## $k$-SAT

A Boolean expression is in CNF, *conjuctive normal form* if it is the conjunction (and) of *clauses*, each of which is the disjunction (or) of *terms*, each of which is either a variable or the negation (not) of a variable. CNF $\subseteq$ BOOL is the set of all Boolean expressions written in conjunctive normal form, while $k$-CNF $\subseteq$ CNF is the subset where each clause has at most $k$ terms.

Note that $k$-CNF $\subseteq$ CNF $\subseteq$ BOOL.

We define $k$-SAT $=$ $k$-CNF $\cap$ SAT.

**Theorem 4** *For any $k \geq 3$, $k$-SAT is $\mathcal{NP}$-complete.*

**Theorem 5** *2-SAT is $\mathcal{P}$–TIME.*

We postpone the proofs of Theorems **??** and **??**.

## Independent Set

An instance of the independent set problem, abbreviated IND, is an ordered pair $(G, K)$ where $G$ is a graph and $K$ is a positive integer. We say a set of vertices of $G$ is *independent* if no two are connected by an edge. A solution (certificate) of $(G, K)$ is an independent set of $K$ vertices of $G$, thus IND is $\mathcal{NP}$ by the verification definition of $\mathcal{P}$. We give a polynomial time reduction $R$ of 3-SAT to IND. We define $R$ only on 3-CNF, the language of instances of 3-SAT. By Theorem **??**, IND is $\mathcal{NP}$-complete.

## Subset Sum

An instance of the subset sum problem consists of a sequence of numbers $\sigma = x_1, \ldots x_k$, together with a number $K$. That instance has a solution if there is some subsequence of $\sigma$ whose sum is $K$. Without loss of generality, we assume that the $x_k$ are positive. A subset of sum $K$ is an easily verified certificate, hence subset sum is $\mathcal{NP}$. We give a polynomial time reduction of IND to subset sum, and thus subset sum is $\mathcal{NP}$-complete.

### Partition

An instance of the partition problem is a sequence $\tau = y_1, \ldots y_k$ of positive numbers. A solution to $\tau$ is a subsequence of $\tau$ whose sum is half the sum of the terms of $\tau$, which is an easily verified certificate. We give a polynomial time reduction of subset sum to partition, and thus partition is $\mathcal{NP}$-complete.

Let $E = C_1 * C_2 * \cdots * C_K$ be a Boolean expression in 3-CNF form. We can assume each clause has exactly three terms, since we can pad a clause with duplicate terms. For example, we can replace $x + y$ with $x + x + y$, and $x$ with $x + x + x$.

Let $C_i = t_{i,1} + t_{i,2} + t_{i,3}$, the disjunction of three terms. Thus, $E$ contains $3K$ terms.

Define a graph $G_E$ whose vertices are $\{x_{i,j} \ : \ 1 \leq i \leq j, \ j = 1, 2, 3\}$. That is there is a 1-1 correspondence between terms of $E$ and vertices of $G_E$. The vertices $v_{i,1}, v + i, 2, v_{i,3}$ form a 3-clique, a complete subgraph of $G_E$. Besides the clique edges, $G_E$ has contradiction edges, namely an edge from any $v_{i,j}$ to $v_{i',j'}$ if $t_{i,j}$ contradicts $t_{i',j'}$.

**Theorem 6** *$E$ is satisfiable if and only if $G_E$ has an independent set of $K$ vertices.*

We give context-free grammars for both BOOL and 3-CNF.

## Unambiguous Context-free grammar for BOOL

The start symbol is $S$.

$S \to E = E \mid E \Rightarrow E \mid E$
$E \to E + T \mid T$
$T \to T \cdot F \mid F$
$F \to !F \mid V \mid (S)$
$V \to AP$
$P \to AP \mid NP \mid \epsilon$
$A \to a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z$
$N \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

## Ambiguous Context-free grammar for BOOL

It is more practical to give an ambiguous grammar for BOOL, since parse trees are simpler. The start symbol of this grammar is $E$, for "expression." For simplicity, we use the symbol **id** to stand for any identifier.
$E \to E + E$
$E \to E \cdot E$
$E \to !E$
$E \to E \Rightarrow E$
$E \to E = E$
$E \to (E)$
$E \to \textbf{id}$

## Unambiguous Context-free grammar for 3-CNF

The start symbol is $E$.

$E \to E \cdot C \mid C$

$C \to (T + T + T)$

$T \to !V \mid V$

$V \to AP$

$P \to AP \mid NP \mid \epsilon$

$A \to a \mid b \mid c \mid d \mid e \mid f \mid g \mid h \mid i \mid j \mid k \mid l \mid m \mid n \mid o \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z$

$N \to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

**Theorem 7** *3-SAT is $\mathcal{NP}$-complete.*

Theorem **??** is proven by giving a polynomial time reduction of SAT to 3-SAT. We will not give the details of this reduction, but we discuss some of the issues.

Two Boolean expressions are usually said to be equal if they have the same variables and every assignment of those variables satisfies either both expressions or neither expression. For example, the expression $x \cdot (y + z)$ is equal to the expression $x \cdot y + x \cdot z$. It is impossible to find a reduction $R$ of SAT to 3-SAT which maps every boolean expression of BOOL to an equal Boolean expression in 3-CNF. Our reduction must make use of new variables.

Clauses are not independent. For example, let $E$ be the expression in 2-CNF form:

$$(x + y) \cdot (x + !y) \cdot (!x + y) \cdot (!x + !y)$$

Each of the four clauses is satisfiable. In fact, the conjunction of any three of those clauses is satisfiable. But $E$, the conjunction of all four clauses, is not satisfiable.

Let $E_1$ be the Boolean expression $(x + y + z + w) \cdot F$, where $F$ is satisfiable. Clearly $x + y + z + w$ is satisfiable, but $E_1$ may not be. For example, $x$, $y$, $z$, and $w$ could all appear somewhere in $F$, and it could be that the every satisfying assignment of $F$ assigns *false* to all four of those variables. We must find a Boolean expression $E_2$ which is equivalent to $E_1$ in a weaker way, namely that $E_2$ is satisfiable if and only if $E_1$ is satisfiable.

We use the a new variable which must have a name that does not appear in $F$. Let $u$ be that new variable. Now let $E_2$ be the Boolean expression $(x + y + u) \cdot (!u + z + w) \cdot F$. If $F$ is 3-CNF, then $E_2$ is 3-CNF.

We see that $E_1$ and $E_2$ are not equal, because $E_2$ has more variables that $E_1$, but they are equivalent in the weaker sense defined above: suppose $I_1$ is satisfiable. Then there is an assignment of the variables of $E_1$ such that at least one of the four variables $x$, $y$, $z$, and $w$ is assigned *true*. The same assignment, augmented by an assignment of $u$, satisfies $E_2$. To prove this, we consider two cases.

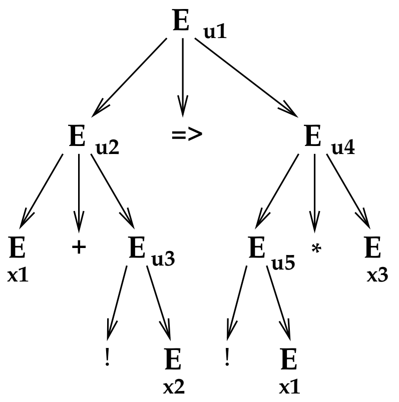
Case I: Either $x$ or $y$ is assigned true. Then assign $u$ the value *false*. The clause $(x + y + u)$ is true because $x$ is true, while the clause $(!u + z + w)$ is true because $u$ is false. The expression $F$ is still true, since the assignments of its variables have not changed.

Case II: Either $z$ or $w$ is assigned true. Then assign $u$ true, making the first clause true. The second clause is also true, and $F$ is still true as before.

On the other hand, suppose $E_1$ is a contradiction. Then $x$, $y$, $z$, and $w$ must all be assigned false. Whatever the assignment of $u$, either the first or the second clause must be false, hence $E_2$ is a contradiction.

## Example

Let $e$ be the Boolean expression $(x1+!x2) \Rightarrow !x1 * x3$. We will compute $R(e)$. Here is the parse tree for $e$, using the ambiguous -free grammar for $L_{\text{bool}}$.



Our next step is to translate the parse tree into the conjunction of clauses:

$$(u2 = x1 + u3) \cdot (u3 = !x2) \cdot (u5 = !x1) \cdot (u4 = u5 \cdot x3) \cdot (u1 = u2 \Rightarrow u4) \cdot (u1)$$

Finally, we translate into 3-CNF form by replacing each of those clauses by the conjunction of CNF clauses of at most three terms:

$(u1) \cdot$
$(u1 + u2) \cdot (u1+!u4) \cdot (!u1+!u2 + u4) \cdot$
$(u2+!x1) \cdot (u2+!u3) \cdot (!u2 + x1 + u3) \cdot$
$(u3 + x2) \cdot (!u3+!x2) \cdot$
$(u5 + x1) \cdot (!u5+!x1) \cdot$
$(u4+!u5+!x3) \cdot (!u4 + u5) \cdot (!u4 + x3)$

If we are required to have exactly three terms in each clause, we can "pad" each clause of length less than three by duplicating terms. For example, we can replace $(u2+!x1)$ with $(u2 + u2+!x1)$, and $(u1)$ with $(u1 + u1 + u1)$.

## Other $\mathcal{NP}$-Complete Problems

The most common method of proving that a given problem (*i.e.*, language) is $\mathcal{NP}$-complete is to use Theorem **??**, where $L_1$ is taken to be a problem (*i.e.*, language) already known to be $\mathcal{NP}$-complete. The problem 3-SAT is one of the more popular choices used for this purpose.[3]

### The Independent Set Problem

Given a graph $G$ an *independent set* of $G$ is defined to be a set $\mathcal{I}$ of vertices of $G$ such that no two members of $\mathcal{I}$ are connected by an edge of $G$. The *order* of $\mathcal{I}$ is defined to be its size, *i.e.*., simply how many vertices it contains.

An instance of the *independent set problem* is $\langle G\rangle\langle k\rangle$, where $G$ is a graph and $k$ is an integer. The question is, "Does $G$ have an independent set of order $k$?"

**The language IND** We define IND to be the set of all $\langle G\rangle\langle k\rangle$ such that $G$ has an independent set of order $k$. We prove IND is $\mathcal{NP}$-complete by reduction of 3-SAT to IND. using Theorem **??**

## Subset Sum

Informally, the subset sum problem is whether there is a subset of a given set of items whose total is a given number. Formally, an instance of the subset sum problem is a finite sequence $x_1, \dots x_k$ of non-negative numbers and a single number $B$. This instance is a member of the language $L_{\mathrm{SS}}$ if there is some subsequence of $\{x_i\}$ whose sum is $B$.

We prove subset sum is $\mathcal{NP}$, by reduction of IND to subset sum, using Theorem **??**.

**Theorem 8** *IND is $\mathcal{NP}$ complete.*

*Proof:* Let E $\in$ 3-SAT. Then $e = C_1 \cdot C_2 \cdot \cdots \cdot C_k$, where $C_i = (t_{i,1} + t_{i,2} + t_{i,3})$, where each $t_{i,j}$ is either $x$ or $!x$, where $x$ is a variable.

We now define a graph $G[E] = (V, E)$, where $V = \{v_{i,j} : 1 \le i \le k, 1 \le j \le 3\}$ is the set of vertices of $G[e]$, and $E$ the set of edges of $G[E]$, as follows:

1. For each $1 \le i \le k$, there is an edge from $v_{i,j}$ to $v_{i,j'}$ for all $1 \le j < j' \le 3$. Call these *short* edges.
2. If $t_{i,j} = x$ and $t_{i',j'} = !x$ for some variable $x$, there is an edge from $v_{i,j}$ to $v_{i',j'}$. Call these *long* edges.

[3] The precise definition of the problem described in this handout as 3-SAT differs from book to book, but they are all equivalent.

3. There are no other edges.

Let $R(e) = G[e], k)$ We now show that $R(e) \in \text{IND}$ if and only if $e$ is satisfiable. For each $i$, let $K_i$ be the subgraph of $G[e]$ consisting of the three vertices $v_{i,1}$, $v_{i,2}$, $v_{i,3}$, and the edges connecting them. We call this a *3-clique.*

Suppose $G[e], k \in \text{IND}$. Let $I \subset V$ be an independent set of of size $k$. Since $K_i$ is a 3-clique, and the number of such cliques is equal to $k$, exactly one member of $I$ must lie in each $K_i$.

We define an assignment of $e$. If $v_{i,j} \in I$ and $t_{i,j} = x$ for some variable $x$, assign the value *true* to $x$, while if $t_{i,j} =!x$, assign *false* to $x$. Assign all remaining variables arbitrary Boolean values. This assignment is well-defined, for if $v_{i,j}, v_{i',j'} \in I$ for $i \neq i'$, there can be no edge between those two vertices, which implies that $t_{i,j}$ does not contradict $t_{i',j'}$. Furthermore, each clause has one term which is assigned true, hence each clause is assigned true, and we thus the assignment is satisfying.

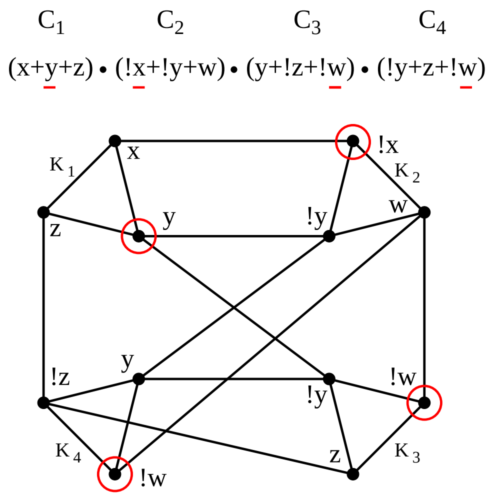
Conversely, suppose that there is a satisfying assignment of $e$. That means each clause $C_i$ must contain one term, say $t_{i,j[i]}$ which is true under the assignment. Let $I = \left\{v_{i,j[i]}\right\} \subseteq V$. No two elements of $I$ are in the same clique $K_i$, hence there is no short edge connecting them, and there can be no long edge connecting them because $v_{i,j[i]}$ and $v_{i',j[i']}$ are both assigned true and hence cannot contradict each other. Thus $I$ is an independent set. □

**Example**

A non-trivial example would have at least eight clauses, but I'll keep it simple. Let $E$ be the 3-CNF expression

$$(x + y + z) \cdot (!x+!y + w) \cdot (y+!z+!w) \cdot (!y + z+!w)$$

Then $k = 4$. The following diagram illustrates $G[E]$. The vertices of $I$ are circled in red. The satisfying assignment shown is $x$ = false, $y$ = true, $w$ = false, while $z$ can be assigned either true or false.



**Theorem 9** *The subset sum problem is $\mathcal{NP}$-complete.*

*Proof:* Trivially, the subset sum problem satisfies the verifiability definition of $\mathcal{NP}$.

We reduce the independent set problem to the subset sum problem. Suppose $\langle G\rangle\langle k\rangle$ is an instance of the independent set problem, where $G$ has $n$ vertices and $m$ edges. Let $e_0, \dots e_{m-1}$ be the edges of $G$ and $v_m \dots v_{n+m-1}$ the vertices of $G$. Define $R(\langle G\rangle\langle k\rangle)$ to be the instance of the subset sum problem $w = (x_0, x_1, \dots x_{n+m-1}, B)$ where

- $x_i = 4^i$ for $0 \le i < m$
- For $m \le i < n+m$, let $J_i = \{j : v_i \text{ is an endpoint of } e_j\}$, then $x_i = 4^m + \sum_{j \in J_i} 4^j$.
- $B = k4^m + \sum_{0 \le j < m} 4^j$

For $0 \le j < m$, $x_j$ corresponds to the edge $e_j$, while for $m \le i < m+n$, $x_i$ corresponds to the vertex $v_i$.

We need to prove $R$ is a reduction of IND to Subset Sum. Suppose $I \subseteq V$ is an independent set of $k$ vertices of $G$. Let

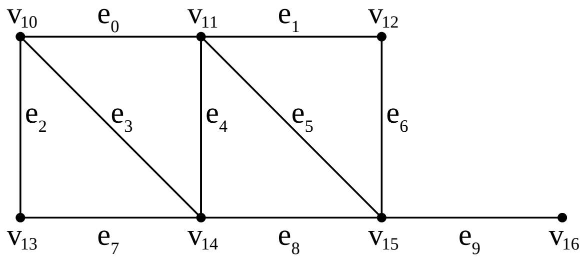
$$S = \{x_j : 0 \le j < m \text{ and no endpoint of } e_j \text{ is in } I\} \cup \{x_i : m \le i < n+m \text{ and } v_i \in I\}$$

We need to show that $\sum S$, the sum of the members of $S$, equals $B$. Since $|I| = k$, the coefficient of $4^m$ is $k$. An $e_j$ is adjacent to either just one member of $I$ or none. If $e_j$ is adjacent to vertex $v_i \in I$, then the coefficient of $4^j$ in $x_i$ is 1, matching that of $B$. Otherwise, $S$ contains $x_j$, which is simply $4^j$, hence $\sum S = B$.

Conversely, suppose there is a subset $S$ of the sequence whose sum is $B$. Let $I = \{v_i : x_i \in S\}$. Since the coefficient of $4^m$ in $B$ is $k$, $I$ must contain exactly $k$ vertices. Since the coefficient of every $x_j$ in $B$ is 1, no two members of the $I$ can be adjacent, hence $I$ is a solution to the subset sum problem. $\square$

## Example

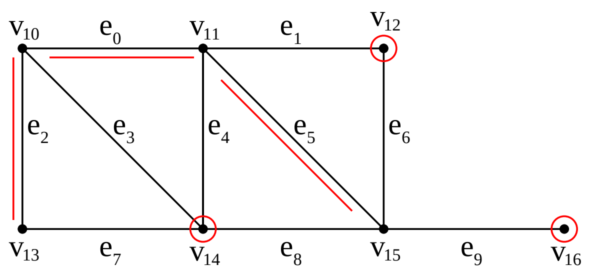
Let $G$ be the graph shown below, and let $k = 3$.



We show the reduced instance of the subset sum problem, where the $x_i$ are written in base 4.

$$\begin{array}{rcr}
x_0 & = & 1 \\
x_1 & = & 10 \\
x_2 & = & 100 \\
x_3 & = & 1000 \\
x_4 & = & 10000 \\
x_5 & = & 100000 \\
x_6 & = & 1000000 \\
x_7 & = & 10000000 \\
x_8 & = & 100000000 \\
x_9 & = & 1000000000 \\
x_{10} & = & 10000001101 \\
x_{11} & = & 10000110011 \\
x_{12} & = & 10001000010 \\
x_{13} & = & 10010000100 \\
x_{14} & = & 10110011000 \\
x_{15} & = & 11101100000 \\
x_{16} & = & 11000000000 \\
B & = & 31111111111
\end{array}$$

The graph below shows an independent set $I$ of vertices of order 3, together with the set of edges which are not adjacent to members of $I$.



Finally, we show the members of the subsequence corresponding to the chosen vertices and edges.

$$\begin{array}{rcr}
x_0 & = & 1 \\
x_2 & = & 100 \\
x_5 & = & 100000 \\
x_{12} & = & 10001000010 \\
x_{14} & = & 10110011000 \\
x_{16} & = & 11000000000 \\
\hline
B & = & 31111111111
\end{array}$$

## Subset Sum and Partition

The subset sum problem can be shown to be $\mathcal{NP}$-complete by reducing IND to subset sum. We can then show that the partition problem is $\mathcal{NP}$-complete by reducing Subset Sum to Partition.

Recall that an instance of the Subset Sum problem is a number followed by a sequence of positive numbers, followed by one number, $K$, that is, $(x_1, x_2, \ldots x_m, K)$, and that instance is in $L_{\text{subs}}$ if there is some subsequence of $x_1, \ldots x_m$ whose total is $K$. Let $L_{\text{subs}}$ be the language of all instances of the problem which have a solution.

Similarly, an instance of the Partition problem is a sequence of positive numbers, namely $\langle y_1, \ldots y_\ell \rangle$, and there is a solution to that instance if and only if there is some subsequence of $y_1, \ldots y_\ell$ whose sum is half the total, *i.e.* $\frac{1}{2}\sum_{j=1}^{\ell} y_j$. Let $L_{\text{part}}$ be the language of all instances of the problem which have a solution.

The partition problem is trivially in the class $\mathcal{NP}$, since the solution, if any, can be verified in polynomial time.

### Reduction

Given an instance $I_K = (x_1, x_2, \ldots x_m, K)$ of Subset Sum, let $A = \sum_{i=1}^{m} x_i$. Let $R(I_K) = I_P$ be the following instance of Partition:

$$I_P = (x_1, \ldots x_m, K+1, A-K+1)$$

That is, $I_P = (y_1, \ldots y_\ell)$ where $\ell = m+2$, $y_i = x_i$ for $i \leq m$, $y_{m+1} = K+1$, and $y_{m+2} = A-K+1$.

We need to prove that this reduction works, that is, that $I_P \in L_{\text{part}}$ if and only if $I_K \in L_{\text{subs}}$. There are thus two directions to the proof.

Note that the sum of the items of $I_P$ is $2A+2$, and half of that is $A+1$.

Suppose that $I_K \in L_{\text{subs}}$. Then there is a subsequence of $\{x_i\}$ whose total is $K$. The items of this subsequence, together with $A-K+1$, total $A+1$, and thus $I_P \in L_{\text{part}}$.

Conversely, suppose some subsequence $S$ of $K+1, x_1, \ldots x_m, L+1$ has total $A+1$. That subsequence cannot contain both $K+1$ and $A-K+1$, since their total exceeds A+1. Similarly, and by symmetry, $S$ must contain either $K+1$ or $A-K+1$. Without loss of generality, $S$ contains $A-K+1$. The remaining members of $S$ constitute a subsequence of $x_1, \ldots x_m$ whose total is $K$, and we are done.

Conversely, suppose there is a subset $B$ of $\{x_i\}$ whose total is $K$. Then $B \cup \{A-K+1\}$ is a subset of the sequence $\{y_j\}$ whose total is $A+1$, and we are done.