NP-Complete Problems

Last Updated September 15th, 2025

1 The Satisfiability (SAT) Problem

A **Boolean formula** $F(x_1, \ldots, x_n)$ over variable set $V = \{x_1, \ldots, x_n\}$ may be represented with a **parse tree** for which i) each internal node is labeled either $\wedge$ or $\vee$, and ii) each leaf node is labeled either x_i or $\overline{x}_i$, for some $i = 1, \ldots, n$. Leaf nodes are also called **literal** nodes, since a formula literal is any variable or its negation. For example, the Boolean formula

$$F(x_1, x_2, x_3) = x_1 \wedge (\overline{x}_2 \vee x_3)$$

may be represented by the **parse tree** shown in Figure 1.

Given Boolean formula F and assignment α over V we may evaluate F using the function $\text{eval}(F, \alpha)$ that returns a value in $\{0, 1\}$. We provide a recursive definition of $\text{eval}(F, \alpha)$ over the set of all Boolean formulas defined over V.

Base Case If F consists of a leaf node labeled with literal l (i.e., x or $\overline{x}$ for some variable x), then

$$\text{eval}(F, \alpha) = \alpha(l).$$

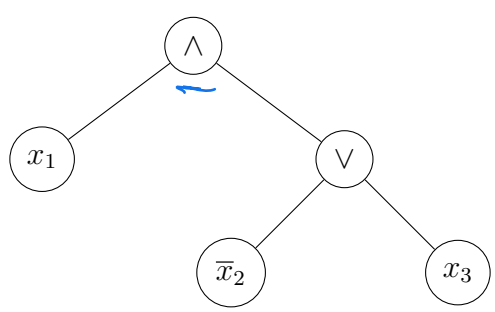


Figure 1: Parse tree for Boolean formula $x_1 \wedge (\overline{x}_2 \vee x_3)$

Recursive Case (And) If the root of F is labeled $\wedge$, and $C_1, \ldots, C_m$ are the root children, then

$$\text{eval}(F, \alpha) = \text{eval}(C_1, \alpha) \wedge \cdots \wedge \text{eval}(C_m, \alpha).$$

Recursive Case (Or) If the root of F is labeled $\vee$, and $C_1, \ldots, C_m$ are the root children, then

$$\text{eval}(F, \alpha) = \text{eval}(C_1, \alpha) \vee \cdots \vee \text{eval}(C_m, \alpha).$$

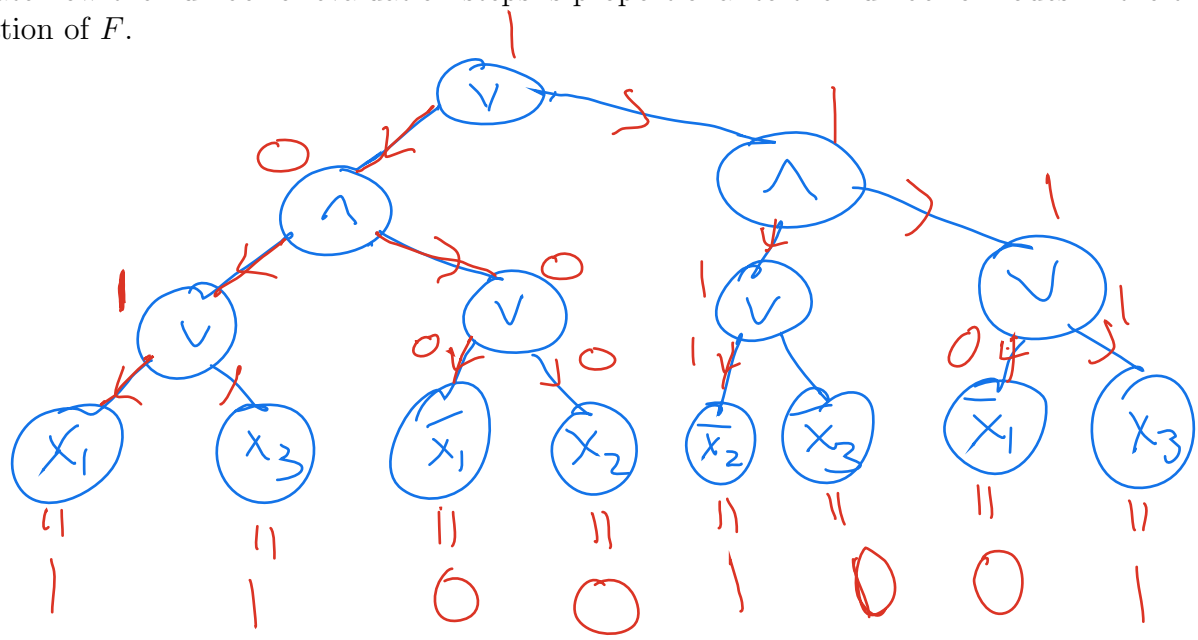
It is worth noting that $\text{eval}(F, \alpha)$ may be computed in $\mathrm{O}(|F|)$ steps, where $|F|$ denotes the number of nodes in formula F.

Example 1.1. Use the recursive definition of `eval` to evaluate the formula

$$F(x_1, x_2, x_3) = ((x_1 \vee x_3) \wedge (\overline{x}_1 \vee x_2)) \vee ((\overline{x}_2 \vee \overline{x}_3) \wedge (\overline{x}_1 \vee x_3))$$

over the assignment $\alpha = (1, 0, 1)$.

Demonstrate how the number of evaluation steps is proportional to the number of nodes in the tree representation of F.



Example 1.2. The **satisfiability problem (SAT)** is the problem of deciding if a Boolean formula $F(x_1, \ldots, x_n)$ evaluates to 1 on some assignment α over the Boolean variables $x_1, \ldots, x_n$. We show that $\texttt{SAT} \in \texttt{NP}$. Let $F(x_1, \ldots, x_n)$ be an instance of $\texttt{SAT}$.

Step 1: define a certificate. **Solution.** α is an assignment over the variables $x_1, \ldots, x_n$.

Step 2: provide a semi-formal verifier algorithm. **Solution.** Evaluate $F(x_1, \ldots, x_n)$ recursively.

//Base Case:

If $F = l$ is a single literal, then return $\alpha(l)$.

//Recursive Case 1

If $F = F_1 \wedge F_2 \wedge \cdots \wedge F_k$, then return

$$\text{eval}(F_1, \alpha) \wedge \text{eval}(F_2, \alpha) \wedge \cdots \wedge \text{eval}(F_k, \alpha).$$

//Recursive Case 2

If $F = F_1 \vee F_2 \vee \cdots \vee F_k$, then return

$$\text{eval}(F_1, \alpha) \vee \text{eval}(F_2, \alpha) \vee \cdots \vee \text{eval}(F_k, \alpha).$$

Step 3: provide size parameters for $\texttt{SAT}$. **Solution.** The size parameter is $|F|$, the number of nodes in F's parse tree.

Step 4: provide the verifier's running time with an explanation. **Solution.** The verifier has running time $\mathrm{O}(|F|)$, since evaluating F can be done by evaluating each node of F's parse tree exactly once.

Therefore, $\texttt{SAT} \in \texttt{NP}$. $\square$

2 NP-Complete Decision Problems

Now that we have an idea about the type of decision problems that belong in NP, we seek a method for providing strong evidence that a decision problem is in some sense one of the *hardest to solve* amongst all problems in NP, and thus is the best candidate for not belonging to class P. Furthermore, if we consider what might constitute a difficult problem amongst a class of problemsthe most difficult would seem to be one to which every other problem in the class can be reduced. Indeed, in the final section of the Mapping Reducibility lecture the following statement was proved.

- If $A \leq_m^p B$ and $B \in \mathtt{P}$ then necessarily $A \in \mathtt{P}$. i.e. B is easy $\Rightarrow$ A is easy
- Therefore, if every problem in NP were polynomial-time reducible to $B \in \mathtt{NP}$, then the $\mathtt{P} =? \mathtt{NP}$ question would hinge on the question of whether B can be solved in polynomial time.

A is hard $\Rightarrow$ B is hard

Definition 2.1. A decision problem B is said to be **NP-complete** iff

1. $B \in \mathtt{NP}$
2. for every other decision problem $A \in \mathtt{NP}$, $A \leq_m^p B$.

$NP \leq_m^p B$

Clique $\leq_T^p$ Max-Clique

"L is CO-NP-Complete"

$\Rightarrow \overline{L}$ is NP-complete $\Rightarrow$

NP-hard

$\Downarrow$

non-decision problems that if efficient an NP-complete problem is efficient

Theorem 2.2. (Cook's Theorem) `SAT` is `NP`-complete.

Outline of a Proof of Cook's Theorem

1. Let $L \in \texttt{NP}$ be an arbitrary decision problem and x an instance of L.
2. Let $v(x, c)$ be the verifier program associated with L.
3. Let $q(|x|)$ denote the running time for $v(x, c)$, where q is a polynomial.
4. Let variables $y_1, \ldots, y_{l(|x|)}$ be a collection of Boolean variables that is capable of encoding any certificate c for verifying x, where l is a polynomial (one of the requirements of a certificate is that its size must be polynomial with respect to $|x|$).
5. It can be shown that any program that runs in a polynomial $q(|x|)$ number of steps and depends on a polynomial $l(|x|)$ number of Boolean variables, can be procedurally converted in polynomial time to a Boolean formula
$$F_{v,x}(y_1, \ldots, y_{l(|x|)}),$$
where
   (a) $F_{v,x}$ is satisfiable iff $v(x, c)$ evaluates to 1 for some certificate c, iff x is a positive instance of L, and
   (b) $|F_{v,x}|$ is bounded in size by some polynomial in terms of $|x|$, the size of x.

   Therefore,
$$L \leq_m^p \texttt{SAT}.$$

3 More NP-Complete Problems

We now build on Cook's theorem to show that a host of other problems are NP-complete (to this date there are several thousand known NP-complete problems from several areas of computer science and mathematics). We do this with the help of the following lemma. Its proof relies on the fact that mapping reducibilities are transitive: if $A \leq_m^p B$ and $B \leq_m^p C$, then $A \leq_m^p C$.

Lemma 3.1. Suppose decision problems A and B are in NP, and A is both NP-complete and polynomial-time reducible to B. Then B is NP-complete.

$$NP \leq_m^p SAT \leq_m^p B$$

$$\Downarrow$$

B is NP-complete

4 The 3SAT Logic Problem

In this Section we introduce the 3SAT logic problem which will which represents one of the more important problems in all of Computer Science.

4.1 Boolean Variable Assignments

Before introducing the 3SAT decision problem, we need to understand the concept of a Boolean variable assignment.

Boolean Variable A variable is said to be **Boolean** iff its domain equal $\{0, 1\}$. We use lowercase letters, such as $x, y, z, x_1, x_2, \ldots$, etc., to denote a Boolean variable.

Assignment An **assignment** over a Boolean-variable set V is a function $\alpha : V \to \{0, 1\}$ that assigns to each variable $x \in V$ a value in $\{0, 1\}$. We may represent α using function notation, or as a labeled tuple.

Example: for the assignment α that assigns 1 to both x_1 and x_2, and 0 to x_3, we may use function notation and write $\alpha(x_1) = 1$, $\alpha(x_2) = 1$, and $\alpha(x_3) = 0$, or we may use tuple notation and write

$$\alpha = (x_1 = 1, x_2 = 1, x_3 = 0),$$

or

$$\alpha = (1, 1, 0),$$

if the associated variables are understood.

Variable Negation If x is a variable, then $\overline{x}$ is called its **negation**.

Example: Suppose assignment α satisfies $\alpha(x_1) = 0$. Then (extending α to include negation inputs) $\alpha(\overline{x}_1) = 1$.

Literal A **literal** is either a variable or the negation of a variable .

Example: x_1, x_3, $\overline{x}_3$, are $\overline{x}_5$ all examples of literals.

Consistent A set R of literals is called **consistent** iff no variable and its negation are both in R. Otherwise, R is said to be **inconsistent**.

Example: $\{x_1, \overline{x}_2, x_4, \overline{x}_7, \overline{x}_9\}$ is a consistent set, but $\{x_1, \overline{x}_2, x_4, \overline{x}_7, x_7\}$ is an inconsistent set.

Induced Assignment If $R = \{l_1, \ldots, l_n\}$ is a consistent set of literals, then α_R is called the **(partial) assignment induced by R** and is defined by $\alpha(l_i) = 1$ for all $l_i \in R$.

Example: the assignment induced by $R = \{x_1, \overline{x}_2, x_4, \overline{x}_7, \overline{x}_9\}$ is

$$\alpha = (x_1 = 1, x_2 = 0, x_4 = 1, x_7 = 0, x_9 = 0).$$

Definition 4.1. A **ternary disjunctive clause** is a Boolean formula of the form

$$l_1 \vee l_2 \vee l_3,$$

where l_1, l_2, and, l_3 are literals. The clause evaluates to 1 in case at least one of l_1, l_2, or l_3 is assigned 1. In this case we say the clause is **satisfied**. Otherwise it is **unsatisfied**.

Definition 4.2. An instance of the `3SAT` decision problem consists of a set $\mathcal{C}$ of ternary disjunctive clauses. The problem is to decide if there is an assignment α over the variables in $\mathcal{C}$, such that every clause $(l_1 \vee l_2 \vee l_3)$ in $\mathcal{C}$ evaluates to 1 under α. If such an assignment α exists, then it is said to be a **satisfying assignment** and we say $\mathcal{C}$ is **satisfiable**. Otherwise, $\mathcal{C}$ is said to be **unsatisfiable**. Finally, the `3SAT` decision problem is the problem of deciding whether a set $\mathcal{C}$ of ternary clauses is sastisfiable.

Simplified clause notation. In what follows, we often simplify the clause notation by writing each clause $(l_1 \vee l_2 \vee l_3)$ as (l_1, l_2, l_3).

Example 4.3. Provide a satisfying assignment for

$$\mathcal{C} = \{(x_1, x_2, x_3), (\overline{x}_2, x_3, \overline{x}_4), (x_1, x_2, \overline{x}_4), (\overline{x}_1, \overline{x}_3, \overline{x}_4), (\overline{x}_1, x_2, x_4), (\overline{x}_2, x_3, x_4), (x_1, \overline{x}_3, x_4),$$

$$(\overline{x}_2, \overline{x}_3, x_4), (\overline{x}_2, \overline{x}_3, \overline{x}_4)\}.$$

Theorem 4.4. SAT $\leq_m^p$ 3SAT.

Because `3SAT` is a member of `NP`, we have the following corollary.

Corollary 4.5. `3SAT` is an `NP`-complete problem.

Proof. We prove the theorem by making use of what is referred to as the **Tseytin transformation**, named after the Russian mathematician Gregory Tseytin. It is a method for transforming an instance F of `SAT` to an instance $\mathcal{C}$ of `3SAT` that is satisfiability-equivalent to F, meaning that F is satisfiable iff $\mathcal{C}$ is satisfiable. Let $F(x_1, \ldots, x_n)$ be an instance of SAT. Without loss of generality, we may assume that F has a binary parse tree T. Let $n_1, \ldots, n_m$ denote the internal nodes of T, where we assume n_1 correpsonds with the root. We assign a literal to each tree node. If n is a leaf, then the literal assigned to n is the literal l for which n is labeled. If $n = n_i$ is an internal node, then we introduce a new variable y_i and associate

it with n_i.

Thus, the reduction f from `SAT` to `3SAT` is such that $f(F) = \mathcal{C}$, and the variables used in the clauses of $\mathcal{C}$ are precisely $x_1, \ldots, x_n, y_1, \ldots, y_m$. Moreover the clauses of $\mathcal{C}$ are obtained from each internal node. For example, let n_i be an internal node, and suppose it's two children have associated literals l_1 and l_2. If n_i is a $\wedge$-operation, then the goal is to replace the formula

$$y_i \leftrightarrow (l_1 \wedge l_2)$$

with a logically equivalent conjunction of disjunctive clauses. The same is true in the case that n_i is a $\vee$-operation: we must replace

$$y_i \leftrightarrow (l_1 \vee l_2)$$

with a logically equivalent conjunction of disjunctive clauses.

Finally, we add the clause y_1 to assert that formula F evaluates to 1.

To see that $f(F) = \mathcal{C}$ is a polynomial-time reduction, we first note that $\mathcal{C}$ has a number of clauses and variables that is linear in $|F|$. Is is because each formula of the form $y_i \leftrightarrow (l_1 \wedge l_2)$ or $y_i \leftrightarrow (l_1 \vee l_2)$ yields up to six disjunctive formulas. Thus $f(F)$ can be constructed in a number of steps that is linear with respect to $|F|$.

Secondly, if F is satisfiable, then there is an assignment α over $x_1, \ldots, x_n$ for which $F(\alpha) = 1$. Moreover, based on the recursive tree evaluation of

$F(\alpha)$, this computation of $F(\alpha)$ also yields a correpsonding assignment β over the internal y variables in which y_1 is assigned 1. Hence, $\alpha \cup \beta$ is a satisfying assignment for $\mathcal{C}$. Conversely, if $\alpha \cup \beta$ is a satisfying assignment for $\mathcal{C}$, then, since the formulas of $\mathcal{C}$ represent a non-recursive representation for evaluating $F(x_1, \ldots, x_n)$, it follows that α must satisfy F, since it induces a computation of each node of F in which the root node is evaluated to 1, since β must assign $y_1 = 1$ in order to satisfy $\mathcal{C}$. $\square$

Example 4.6. Apply the reduction described in Theorem 4.4 to the Boolean formula

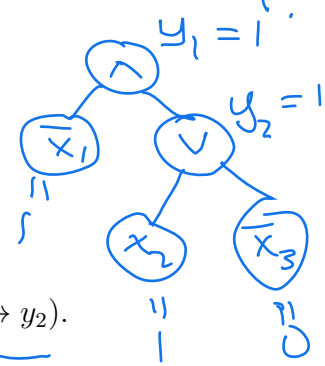
$$F(x_1, x_2, x_3) = \overline{x}_1 \wedge (x_2 \vee \overline{x}_3)).$$

$\alpha = (x_1 = 0, x_2 = 1, x_3 = 1)$ satisfies F.

Solution. We introduce Boolean variables y_1 and y_2, where $y_2 \leftrightarrow (x_2 \vee \overline{x}_3)$, and $y_1 \leftrightarrow (\overline{x}_1 \wedge y_2$. Then $F(x_1, x_2, x_3)$ is satisfiable iff

$$\boxed{y_1 \wedge (y_1 \leftrightarrow (\overline{x}_1 \wedge y_2)) \wedge (y_2 \leftrightarrow (x_2 \vee \overline{x}_3))}$$

is satisfiable. We now convert the latter to a logically-equivalent `3SAT`-formula.



Step 1: replace $\boxed{P \leftrightarrow Q \text{ with } (P \to Q) \wedge (Q \to P).}$

$$y_1 \wedge (y_1 \to (\overline{x}_1 \wedge y_2)) \wedge ((\overline{x}_1 \wedge y_2) \to y_1) \wedge (y_2 \to (x_2 \vee \overline{x}_3)) \wedge ((x_2 \vee \overline{x}_3) \to y_2).$$

Step 2: replace $P \to Q$ with $\overline{P} \vee Q$.

$$y_1 \wedge (\overline{y_1} \vee (\overline{x}_1 \wedge y_2)) \wedge (\overline{(\overline{x}_1 \wedge y_2)} \vee y_1) \wedge (\overline{y_2} \vee (x_2 \vee \overline{x}_3)) \wedge (\overline{(x_2 \vee \overline{x}_3)} \vee y_2).$$

Step 3: apply De Morgan's rule.

$$y_1 \wedge (\overline{y_1} \vee (\overline{x}_1 \wedge y_2)) \wedge (x_1 \vee \overline{y}_2 \vee y_1) \wedge (\overline{y_2} \vee (x_2 \vee \overline{x}_3)) \wedge ((\overline{x}_2 \wedge x_3) \vee y_2).$$

Step 4: distribute $\vee$ over $\wedge$.

$$y_1 \wedge ((\overline{y_1} \vee \overline{x}_1) \wedge (\overline{y_1} \vee y_2)) \wedge (x_1 \vee \overline{y}_2 \vee y_1) \wedge (\overline{y_2} \vee x_2 \vee \overline{x}_3) \wedge ((\overline{x}_2 \vee y_2) \wedge (x_3 \vee y_2)).$$

Step 5: Repeat last literal enough times to make three literals per clause and use clause notation.

$$\{(y_1, y_1, y_1), (\overline{y_1}, \overline{x}_1, \overline{x}_1), (\overline{y_1}, y_2, y_2), (x_1, \overline{y}_2, y_1), (\overline{y_2}, x_2, \overline{x}_3), (\overline{x}_2, y_2, y_2), (x_3, y_2, y_2)\}.$$

For the reduction from `SAT` to `3SAT`, it's fair to ask why it is necessary to add new y-variables and through so many steps to transform F to a set of `3SAT` clauses. For example, why not just map F to

$$\{(\overline{x}_1, \overline{x}_1, \overline{x}_1), (x_2, \overline{x}_3, \overline{x}_3)\}?$$

The problem is that not all formulas are this simple, and some relatively simple formulas may require an exponential number of steps if no new variables are introduced. As an example, consider the formula

$$F(x_1, \ldots, x_{2n}) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \cdots \vee (x_{2n-1} \wedge x_{2n}).$$

This formula is in what is called **disjunctive normal form (DNF)** since it is an OR of AND's. Moreover, to convert it to a logically equivalent formula in **conjunctive normal form (CNF)**, an AND of OR's, would require a number of steps that is exponential with respect to n. This is because it's logically equivalent CNF form has 2^n clauses! Verify this for $n = 2$ and $n = 3$ by repeatedly applying the distributive rule of $\vee$ over $\wedge$.

Although it is seems lengthy even for the simplest of formulas, the reduction method used in Example 4.6 has the advantage of requiring a maximum of $C > 0$ steps per logic operation of F, where C is a constant. This is because all five steps of the procedure require only a constant number of operations. Therefore, the reduction can be completed in $\mathrm{O}(|F|)$ steps which is a linear (and hence a polynomial) number of steps with respect to the size of F.

5 Interdomain Reductions

In this section we look at **interdomain** mapping reductions that reduce a problem from one domain to a problem in a different domain. Some of the different mathematical and computer science domains include the following. include

Mathematics logic, graph theory, algegra and number theory, numerical optimization, geometry

Computer Science machine learning, network design and analysis, data storage and retrieval, cryptography/security, operating systems, automata and languages, programming languages and program optimization.

Of course, as is witnessed by interdomain reductions, different domains are often related in several ways, and thus there is some subjectivity regarding the classifications of problems. Nevertheless, the above mentioned domains are considered separate and vast areas of research. As we'll see in the following examples, interdomain reductions are often the more surprising and clever of all reductions.

The reductions we study in this section both reduce from the `3SAT` logic problem. Because of the ability to reduce `3SAT` to other problem domains, `3SAT` plays a crucial role in the study of NP-completness which we cover in the next lecture.

The following theorem uses refers to `Clique`, the decsion-problem version of `Max Clique`. In this case, an instance of the `Clique` is a simple graph $G = (V, E)$ and an integer $k \geq 0$. The problem is to decide if there is a subset $C \subseteq V$ of k vertices that are pairwise adjacent.

Theorem 5.1. `3SAT` $\leq_m^p$ `Clique`. Therefore, since `Clique` is an `NP` problem, it is also `NP`-complete.

Proof. Let $\mathcal{C}$ be a collection of m clauses, where clause c_i, $1 \leq i \leq m$, has the form $c_i = l_{i1} \vee l_{i2} \vee l_{i3}$. We now define a mapping $f(\mathcal{C}) = (G, k = m)$ for which G has an m-clique if and only if $\mathcal{C}$ is satisfiable. $G = (V, E)$ is defined as follows. V consists of $3m$ vertices, one for each literal l_{ij}, $1 \leq i \leq m$ and $1 \leq j \leq 3$. Then $(l_{ij}, l_{rs}) \in E$ iff i) $i \neq r$ and ii) l_{ij} is not the negation of l_{rs} (i.e. the two literals are logically consistent).

First assume that $\mathcal{C}$ is satisfiable. Given a satisfying assignment α for $\mathcal{C}$, let l_{ij_i}, $1 \leq i \leq m$, denote a literal from clause i that is satisfied by α, i.e. $\alpha(l_{ij_i}) = 1$. Then, since each pair of these literals is consistent, $(l_{ij_i}, l_{rj_r}) \in E$ for all $i \neq r$. In other words,

$$C = \{l_{1j_1}, l_{2j_2}, \ldots, l_{mj_m}\}$$

is an m-clique for G.

Conversely, assume G has an m-clique. Then by the way G is defined the clique must have the form

$$C = \{l_{1j_1}, l_{2j_2}, \ldots, l_{mj_m}\}$$

where l_{ij_i} is a literal in c_i. This is true since no two literal vertices from the same clause can be adjacent and so each literal vertex in C must come from a different clause. Moreover, by the definition of G, C is a consistent set of literals. Hence the assignment a_C induced by C satisifes every clause of $\mathcal{C}$, since $\alpha_C(l_{ij_i}) = 1$ satisfies c_i, for each $i = 1, \ldots, m$. Therefore, $\mathcal{C}$ is satisfiable.

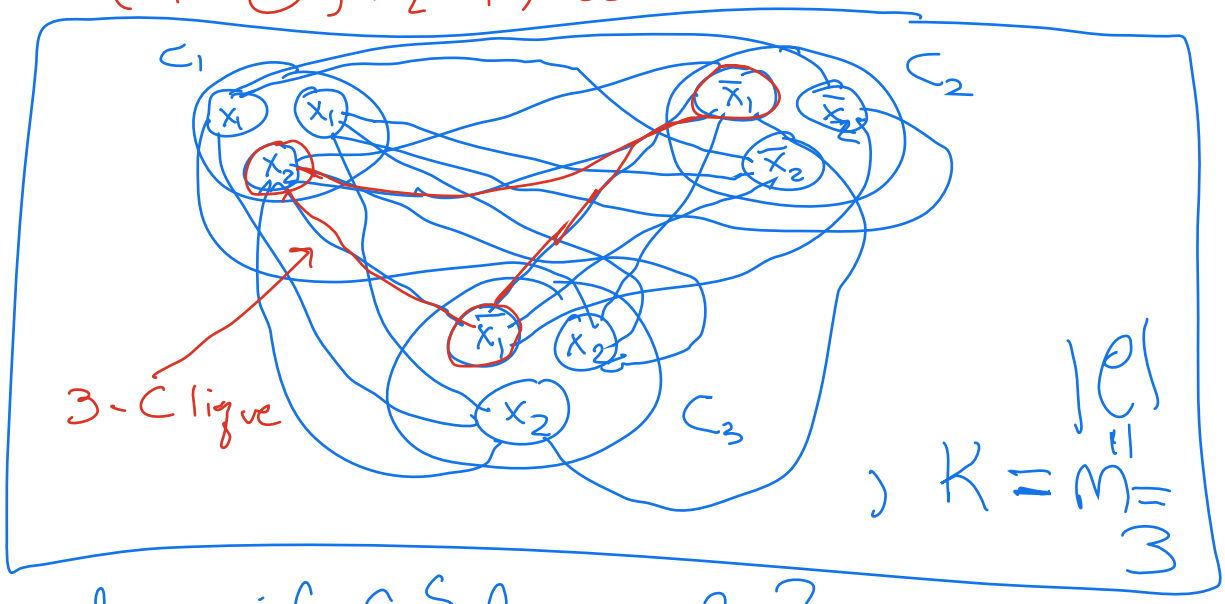
Finally, we must show is that $f(\mathcal{C})$ may be computed via an algorithm whose running time is polynomial in m and n. But this can be done via two nested `for`-loops that iterate through each pair of clauses i and r, $i \neq r$, and identify all consistent pairs of literals (l_{ij_i}, l_{rj_r}). This require $\mathrm{O}(m^2)$ steps. $\square$

Example 5.2. Show the reduction provided in the proof of Theorem 5.1 for input instance

$$\mathcal{C} = \{\overset{C_1}{(x_1, x_1, x_2)}, \overset{C_2}{(\overline{x}_1, \overline{x}_2, \overline{x}_2)}, \overset{C_3}{(\overline{x}_1, x_2, x_2)}\}.$$

$\alpha = (x_1 = 0, x_2 = 1)$ satisfies $\mathcal{C}$

$f(\mathcal{C}) =$



, $K = m = 3$

· Conversely, if $C = \{l_1, \ldots, l_m\}$ is an m-clique for $f(\mathcal{C})$, where $l_i \in C_i$ for every $i = 1, 2, \ldots, m$. Then C is consistent (by definition of $f(\mathcal{C})$) and so α_C satisfies $\mathcal{C}$ since $\alpha_C(l_i) = 1$ and $l_i \in C_i \Rightarrow C_i$ is satisfied, $i = 1, \ldots, m$.

Theorem 5.3. `3SAT` $\leq_m^p$ `Subset Sum`. Therefore, since `SS` is an `NP` problem, it is also `NP`-complete.

Let $\mathcal{C}$ be a collection of m ternary clauses over n variables. The following table provides the reduction $f(\mathcal{C}) = (S, t)$. The table rows correspond to the $(n+m)$-digit integers comprising set S, while t is the integer at the bottom whose first n digits are 1's and whose final m digits are 3's.

$x_1 \Leftrightarrow$

$\overline{x}_1 \Leftrightarrow$

	1	2	3	4	$\cdots$	n	c_1	c_2	$\cdots$	c_m
y_1	1	0	0	0	$\cdots$	0	1	0	$\cdots$	0
z_1	1	0	0	0	$\cdots$	0	0	0	$\cdots$	0
y_2		1	0	0	$\cdots$	0	1	0	$\cdots$	0
z_2		1	0	0	$\cdots$	0	0	0	$\cdots$	0
y_3			1	0	$\cdots$	0	1	1	$\cdots$	0
z_3			1	0	$\cdots$	0	0	0	$\cdots$	1
$\vdots$						$\vdots$	$\vdots$		$\vdots$	$\vdots$
y_n						1	0	0	$\cdots$	0
z_n						1	0	0	$\cdots$	0
g_1							1	0	$\cdots$	0
h_1							1	0	$\cdots$	0
g_2								1	$\cdots$	0
h_2								1	$\cdots$	0
$\vdots$										$\vdots$
g_m									$\cdots$	1
h_m									$\cdots$	1
t	1	1	1	1	$\cdots$	1	3	3	$\cdots$	3

Filler

n $\quad$ m

$S = \{y_1, z_1, \ldots, y_n, z_n, g_1, h_1, \ldots, g_m, h_m\}$

Number y_i correpsonds to literal x_i, while z_i corresponds to literal $\overline{x}_i$. Thus, since the first n digits of t are 1's, we see that, to construct a subset A whose members sum to t, it must either contain y_i or z_i, but not both. Also, the final m digits of y_i (respectively, z_i) indicate which clauses c_i are satisfied by x_i (respectively $\overline{x}_i$).

Now suppose $\mathcal{C}$ is satisfiable via some assignment α. Then the subset A needed to sum to t includes the following numbers. For $1 \leq i \leq n$, if $\alpha(x_i) = 1$, then add y_i to A; otherwise add z_i to A. For $1 \leq j \leq m$, to determine if g_j and/or h_j should be added to A, consider clause $c_j = \{l_{j1}, l_{j2}, l_{j3}\}$ and the sum

$$\sigma_j = \alpha(l_{j1}) + \alpha(l_{j2}) + \alpha(l_{j3}).$$

Since α satisfies $\mathcal{C}$, we must have $\sigma_j \geq 1$.

Case 1: $\sigma_j = 1$. In this case add both g_j and h_j for the c_j-column to sum to 3.

Case 2: $\sigma_j = 2$. In this case add only g_j for the c_j-column to sum to 3.

Case 3: $\sigma_j = 3$. In this case neither g_j nor h_j need to be added since the c_j-column already sums to 3.

. Therefore, it is always possible to find a subset A whose members sum to t.

Conversely, suppose there is a subset $A \subseteq S$ whose members sum to t. Then for each $i = 1, \ldots, n$, either $y_i \in A$ or $z_i \in A$, but not both. This is true since t's first n digits are 1's. Let α be an assignment over the variables of $\mathcal{C}$ such that, for each $i = 1, \ldots, n$ $\alpha(x_i) = 1$ iff $y_i \in A$. Now consider clause c_j, $j = 1, \ldots, m$. We know that the digits of the members of A in the c_j-column sum to 3. Thus, one of the members must either be y_k or z_k for some $k = 1, \ldots, n$. In other words, either $y_k \in A$, $x_k \in c_j$, and $\alpha(x_k) = 1$ or $z_k \in A$, $\overline{x}_k \in c_j$ and $\alpha(\overline{x}_k) = 1$. In either case, we see that α satisfies c_j. Therefore, since $j = 1, \ldots, n$ was arbitrary, α satisfies $\mathcal{C}$.

Finally, notice that each of the $2m + 2n$ integers in S can be constructed in $\mathrm{O}(m + n)$ steps, and so $f(\mathcal{C}) = (S, t)$ can be computed in $\mathrm{O}(m^2 + n^2)$ steps. $\square$

Example 5.4. Show the reduction given in Theorem 5.3 using input instance

$$\mathcal{C} = \{c_1 = (x_1, x_2, x_3), c_2 = (\overline{x}_1, \overline{x}_2, \overline{x}_3), c_3 = (\overline{x}_1, x_2, x_3), c_4 = (x_1, \overline{x}_2, \overline{x}_3)\}.$$

$\alpha = (x_1 = 0, x_2 = 1, x_3 = 0)$

$x_2 = 0$

Solution.

$x_1 \Leftrightarrow$

$\overline{x}_1 \Leftrightarrow$

$\overline{x}_2 \Leftrightarrow$

	1	2	3	c_1	c_2	c_3	c_4
y_1	1	0	0	1	0	0	1
z_1	1	0	0	0	1	1	0
y_2		1	0	1	0	1	0
z_2		1	0	0	1	0	1
y_3			1	1	0	1	0
z_3			1	0	1	0	1
g_1				1	0	0	0
h_1				1	0	0	0
g_2					1	0	0
h_2					1	0	0
g_3						1	0
h_3						1	0
g_4						0	1
h_4						0	1
t	1	1	1	3	3	3	3

e.g. x_1 satisfies c_4

Then

$A_\alpha = \{z_1, y_2, z_3, g_1, h_1, g_2, g_3, g_4, h_4\}$

$\sum_{a \in A_\alpha} a = t$

Theorem 5.5. An instance of the `Directed Hamilton Path (DHP)` decision problem is a directed graph $G = (V, E)$ and two vertices $a, b \in V$. The problem is to decide of G possesses a directed simple path from a to b and having length $n - 1$. Such a a path is called a **(Directed) Hamilton Path (DHP)**. Then `DHP` is `NP` complete.

Proof. The fact that `DHP` is in `NP` is an exercise from the Computational Complexity lecture. We show a polynomial-time mapping reduction from `3SAT`. Let $\mathcal{C}$ be a collection of m ternary clauses over n variables. We proceed to define $f(\mathcal{C}) = (G = (V, E), a, b)$, a directed graph $G = (V, E)$ along with two vertices $a, b \in V$ so that G has a Hamilton path from a to b iff $\mathcal{C}$ is satisfiable.

G is defined as follows (see the graph in Example 5.6 for a specific image of the following general description). G has m clause vertices $c_1, \ldots, c_m$ and n *diamond subgraphs*, one corresponding to each variable x_i, $1 \leq i \leq n$. Diamond subgraph D_i consists of a top vertex t_i, bottom vertex b_i, left vertex l_i, and right vertex r_i, along with edges

$$(t_i, l_i), (t_i, r_i), (l_i, b_i), (r_i, b_i).$$

In addition, there is a row of $3m - 1$ vertices that connect l_i with r_i:

$$lc_{i1}, rc_{i1}, s_{i1}, lc_{i2}, rc_{i2}, s_{i2} \ldots, lc_{im}, rc_{im}.$$

The s_{ij} vertices, $j = 1, \ldots, m - 1$, are called *separators*, while the lc_{ij} and rc_{ij} pairs, $j = 1, \ldots, m$, correspond with each of the m clauses and are used for making round-trip excursions to each of the clause vertices. Every vertex in the row is bidirectionally adjacent to both its left and right neighbor, i.e.,

$$(l_i, lc_{i1}), (lc_{i1}, rc_{i1}), (rc_{i1}, s_{i1}), \ldots, (s_{i(m-1)}, lc_{im}), (lc_{im}, rc_{im}), (rc_{im}, r_i) \in E,$$

as well as the reversals of each of these edges. Finally, if x_i is a literal of clause c_j, then edges $(rc_{ij}, c_j), (c_j, lc_{ij})$ are added. On the other hand, if $\overline{x}_i$ is a literal of clause c_j, then edges $(lc_{ij}, c_j), (c_j, rc_{ij})$ are added.

Finally, for $1 \leq i \leq n - 1$ the edges (b_i, t_{i+1}) are added to connect the n diamond subgraphs, and $a = t_1$, while $b = b_n$. We leave it as an exercise to show that $f(\mathcal{C}) = (G = (V, E), a, b)$ can be constructed in time that is polynomial with respect m and n. It remains to prove that $\mathcal{C}$ is satisfiable iff $f(\mathcal{C}) = (G = (V, E), a, b)$ has a DHP from a to b.

Claim. Suppose P is a DHP from a to b. Then, for all $i = 1, \ldots, n - 1$, P must visit every vertex in D_i before moving to a later diamond D_j, $j > i$.

Proof of Claim. Suppose by way of contradiction that P is a DHP from a to b, and let D_i be the first diamond where the path moves from D_i to D_j, for some $j > i$, without having visited every vertex in D_i. The only way this can happen is if P moves from either vertex lc_{ik} or rc_{ik} in D_i to clause vertex c_k, and then from there moves to either vertex lc_{jk} or rc_{jk} in D_j. In other words, the clause vertex c_k acts as a bridge between the two diamonds. Without loss of generality, assume that P is moving from left to right through D_i, then moves from lc_{ik} to c_k, followed by moving to either lc_{jk} or rc_{jk}. Now consider vertex rc_{ik}. The only vertex that has yet to be visited and can reach rc_{ik}

is separator vertex s_{ik}. Thus, rc_{ik} must immediately follow s_{ik} in P. But then there are no other vertices that can be visited after rc_{ik} since both lc_{ik} and s_{ik} have been visited, which contradicts that P is a DHP from $a = t_1$ to $b = b_n$. A similar argument holds if instead the path moves from rc_{ik} to c_k.

By the above claim, we see that, when forming a DHP, there is at most one direction (left-to-right or right-to-left) that a path can move through a diamond D_i and be able to visit some clause vertex c. Moreover, based on how G was defined, the direction is left-to-right (respectively, right-to-left) iff $\overline{x}_i$ (respectively, x_i) is a literal of c. For some path P that traverses through all the diamonds (and perhaps some of the clause vertices), starting at a and finishing at b, let $\Delta(P) = (\delta_1, \ldots, \delta_n)$ denote a binary vector, where δ_i denotes the direction that P takes (0 = left-to-right, 1 = right-to-left) through diamond D_i. We'll call $\Delta(P)$ the *signature* of P. As an example, consider the path P shown in Example 5.6 that is highlighted in green. Then its signature is $\Delta(P) = (0, 1)$ since it moves left-to-right in the x_1 diamond, and right-to-left in the x_2 diamond.

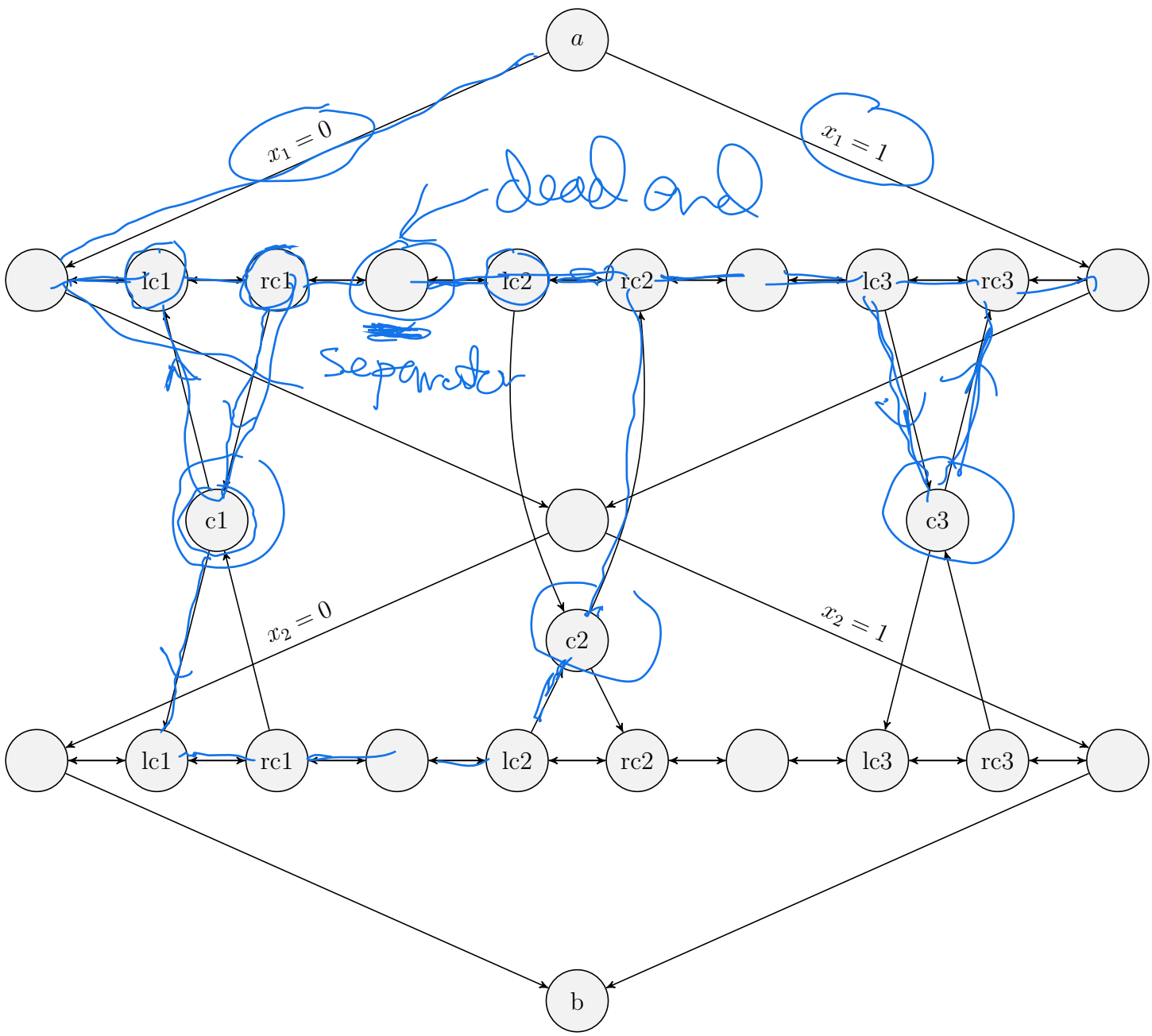
Now suppose $\mathcal{C}$ is satisfiable via satisfying assignment α. Then there is a path P for which P i) has signature $\Delta(P) = (\alpha(x_1), \ldots, \alpha(x_n))$, ii) visits every clause vertex exactly once, and iii) is a DHP from a to b. To see this, consider a clause c_j and let i be the least index for which $\alpha(x_i)$ satisfies c_j. Then if, for example, $\alpha(x_i) = 1$, then x_i is a literal of c_j and, by the way in which G was defined, P may move from right to left in D_i and visit c_j via the sequence rc_{ij}, c_j, lc_{ij}. Therefore, every clause clause vertex gets visited exactly once and P is a DHP from a to b.

Conversely, suppose P is a DHP in G and let $\Delta(P) = (\delta_1, \ldots, \delta_n)$ be its signature. Then the variable assignment α defined by $\alpha(x_i) = \delta_i$, $i = 1, \ldots, n$, satisfies $\mathcal{C}$. To see this, consider a clause c_j and let D_i be the diamond from where P visits c_j. Then by the way G was defined, P can either visit c_j by moving left-to-right, in which case $\overline{x}_i$ is a literal of c_j and $\alpha(x_i) = \delta_i = 0$ satisfies c_j, or by moving right to left, in which case x_i is a literal of c_j and $\alpha(x_i) = \delta_i = 1$ satisfies c_j. In either case α satisfies c_j and, since j was arbitrary, we see that α satisfies $\mathcal{C}$. $\square$

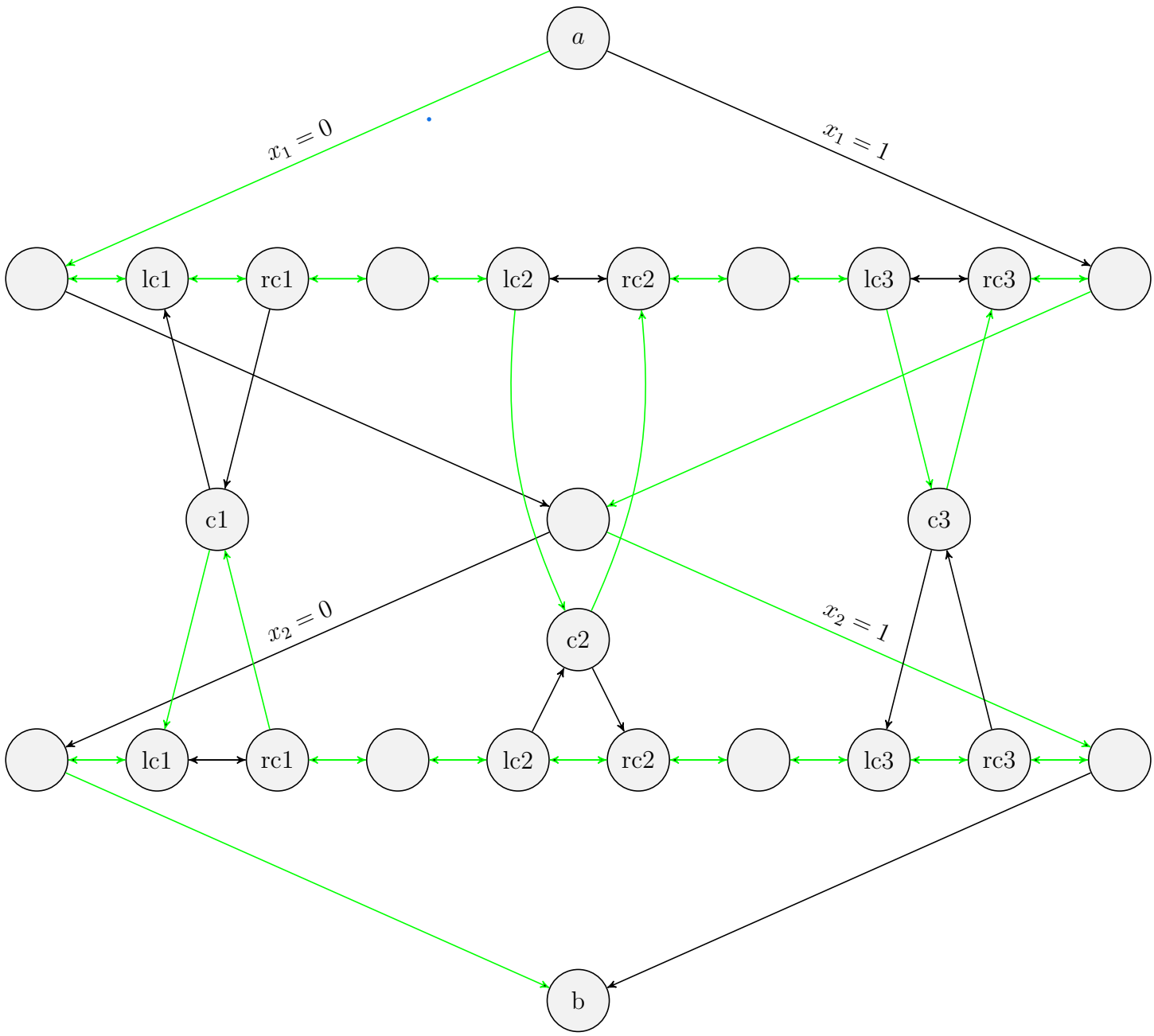
Example 5.6. The following graph shows $f(\mathcal{C})$, where

$$\mathcal{C} = \{c_1 = (x_1, x_2, x_2), c_2 = (\overline{x}_1, \overline{x}_2, \overline{x}_2), c_3 = (\overline{x}_1, x_2, x_2)\}$$

is an instance of `3SAT` and f is the reduction reduction given in Theorem 5.5.



Notice that $\mathcal{C}$ is satisfiable via assignment $\alpha = (x_1 = 0, x_2 = 1)$. Therefore, $f(\mathcal{C})$ must have a DHP from a to b. In fact, α gives directions for the path: go left in the x_1-diamond, right in the x_2-diamond, and visit a clause vertex if i) it has yet to be visited and ii) the clause is satisified by the direction of movement through the diamond. The figure below shows such a DHP in green.



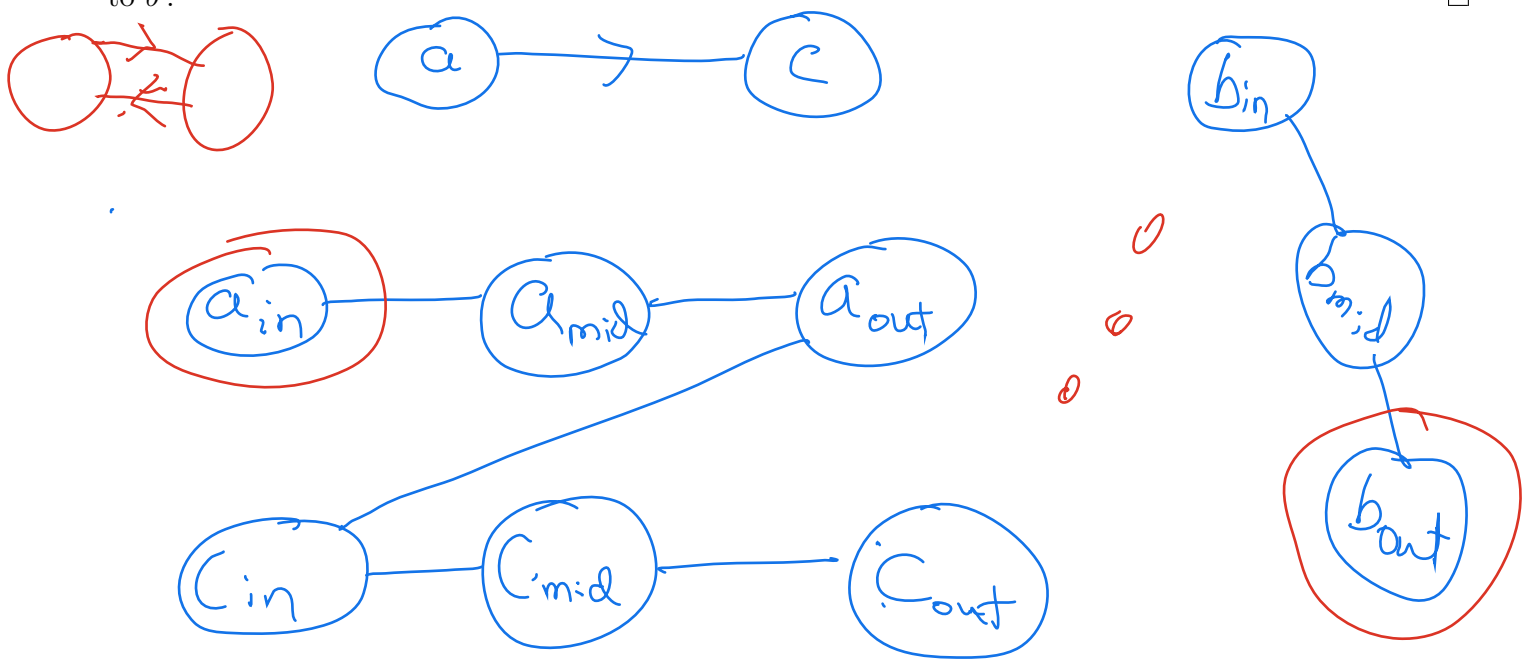
6 The "Package Delivery Problem" is NP-Complete

Theorem 6.1. The `Hamilton Path (HP)` decision problem is the same problem as `DHP`, but now the edges of graph G are assumed undirected. `HP` is is `NP` complete.

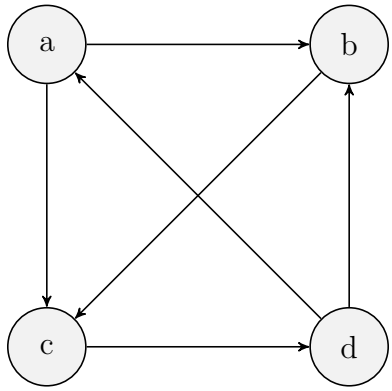
Proof. The proof that `HP` is in `NP` is almost identical to that of showing it for `DHP`. Furthermore, we may map reduce `DHP` to `HP` via the function $f(G = (V, E), a, b) = (G' = (V', E'), a', b')$. To get G' from G, we convert each vertex $v \in V$ to three vertices in V': v_{in}, v_{mid} andn v_{out}. Also we add add to E' the undirected edges

$$(v_{\text{in}}, v_{\text{mid}}), (v_{\text{mid}}, v_{\text{out}}).$$

Also, for each directed edge $(u, v) \in E$, we add to E' the edge $(u_{\text{out}}, v_{\text{in}})$. Finally, $a' = a_{\text{in}}$ while $b' = b_{\text{out}}$. We leave it as an exercise to show that G has a DHP from a to b iff G' has an HP from a' to b'. □



Example 6.2. Given the graph G shown below, provide $f(G, a, b)$, where f is the mapping reduction from `DHP` to `HP`.



Definition 6.3. An instance of the `Hamilton Cycle (HC)` decision problem is a simple graph $G = (V, E)$ and the problem is to decide if G has a **Hamilton Cycle**, i.e. a cycle that has length $n = |V|$ and visits every vertex in V.

Theorem 6.4. `Hamilton Cycle` is an `NP` problem and `HP` $\leq_m^p$ `HC`, making `HC` `NP`-complete.

Proof. `Hamilton Cycle` is an `NP` problem since, given an instance $G = (V, E)$, a certificate would be a permutation $v_1, v_2, \ldots, v_{n-1}, v_n$ of the vertices in V. A verifier can then check that each pair $(v_i, v_{i+1}) \in E$, for each $i = 1, \ldots, n-1$, and also check that $(v_n, v_1) \in E$. This can be done in $\mathrm{O}(m + n)$ steps.

We now provide a mapping reduction from `HP` to `HC`. Let (G, a, b) be an instance of `HP`, where $G = (V, E)$ is a simple graph and $a, b \in V$ with $a \neq b$. Then $f(G, a, b) = G'$, where G' is obtained from G by adding an additional vertex u to G, as well as the undirected edges (b, u) and (u, a). To show that this reduction is valid, suppose (G, a, b) is a positive instance of `HP`. Then there is a Hamilton path P in G that starts at a, ends at b, and visits every vertex in G. Hence, $C = P, u, a$ is a Hamilton cycle for G', since, once b is reached from a via path P, u is the only remaining vertex to visit, followed by returning to a. Hence, G' is a positive instance of `HC`. Conversely, assume G' is a positive instance of `HC` and, without loss of generality, assume the cycle begins at a and u is *not* the next visited vertex. Therefore, u has to be the final visited vertex and b has to be the second-to-last visited vertex before the cycle gets completed. Hence, $C = a, \ldots, b, u, a$ which means that the part of C that is the path from a to b must be a Hamilton path from a to b. Therefore, (G, a, b) is a positive instance of `HP`. $\square$

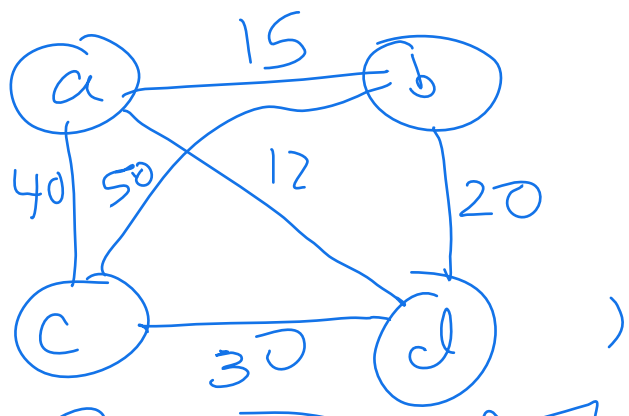
Theorem 6.5. An instance of the `Traveling Salesperson (TSP)` decision problem is a complete weighted graph $G = (V, E, w)$ and a number k, and the problem is to decide if there exists a Hamilton cycle in G whose edge weights sum to a value that does not exceed k. Then `TSP` is `NP`-complete.

Proof. We leave it as an exercise to show that `TSP` $\in$ `NP`. To show it is `NP`-complete, we map reduce `HC` to `TSP`. Let $G = (V, E)$ be an instance of `HC`, where $n = |V| \geq 3$. Define $f : \texttt{HC} \to \texttt{TSP}$ by

$$f(G = (V, E)) = (G' = (V, E', w), n),$$

where G' is obtained by taking G and assigning weight 1 to each of its edges. Furthermore, for any $u, v \in V$ for which $(u, v) \notin E$, we add the edge (u, v) to E' and assign it weight n. Thus, G' is a complete weighted graph.

We see that f is computable in $\mathrm{O}(n^2)$ steps which is the number of steps needed to construct a complete graph over n vertices. Also, if G has an HC, then G' has an HC having cost n. Conversely, if G' has an HC with a cost of at most n, then this HC must only use unit-weight edges, which means it only uses edges in E. Therefore, G has an HC. $\square$



$K = 115$

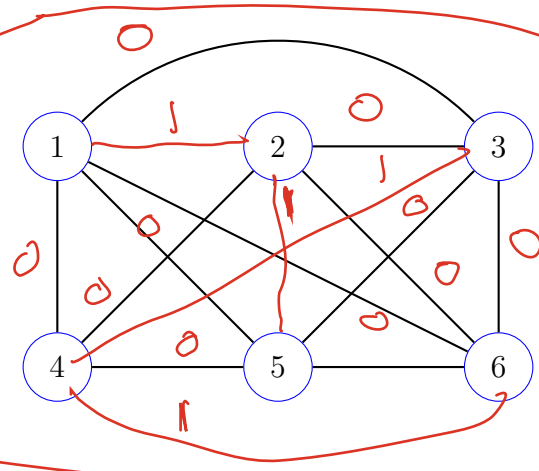
$C = \boxed{a, b, d, c}, a$

$\text{cost}(C) = 15 + 20 + 30 + 40 = 105 < 115.$

Example 6.6. Given the graph G shown below, provide $f(G)$, where f is the mapping reduction from `HC` to `TSP`.

HC instance G

G:



: f(G)

, K=0

7 Summary

The following decision problems are all NP-complete: Clique, Independent Set (IS), Subset Sum (SS), Set Partition (SP), Hamilton Path (HP), Vertex Cover (VC), Half Vertex Cover (HVC), Half Clique, SAT, 3SAT, Directed Hamilton Path (DHP), Hamilton Cycle (HC), Traveling Salesperson (TSP). Below are the chains of mapping reductions that establish each problem as being NP-complete.

$$\texttt{SAT} \leq_m^p \texttt{3SAT}$$

$$\texttt{3SAT} \leq_m^p \texttt{Clique} \leq_m^p \texttt{Half Clique},$$

$$\texttt{Clique} \leq_m^p \texttt{Independent Set},$$

$$\texttt{3SAT} \leq_m^p \texttt{Subset Sum} \leq_m^p \texttt{Set Partition}.$$

$$\texttt{3SAT} \leq_m^p \texttt{DHP} \leq_m^p \texttt{HP} \leq_m^p \texttt{HC} \leq_m^p \texttt{TSP}.$$

Also, Exercise 3 yields

$$\texttt{Independent Set} \leq_m^p \texttt{Vertex Cover} \leq_m^p \texttt{Half Vertex Cover}.$$

NP-Completeness Core Exercises

1. Given Boolean formula

$$F(x_1, x_2, x_3, x_4) = \overline{x}_1 \wedge (x_2 \vee (\overline{x}_3 \wedge (x_1 \vee x_2) \wedge \overline{x}_4)),$$

   draw its parse tree and provide two assignments α and β for which $F(\alpha) = 1$ and $F(\beta) = 0$.

2. Provide the three 3SAT clauses whose conjunction is logicaly equivalent to the Boolean formula $x \leftrightarrow (y \vee z)$.

3. Provide the three 3SAT clauses whose conjunction is logicaly equivalent to the Boolean formula $x \leftrightarrow (y \wedge z)$.

4. The transformation used in the reduction from `SAT` formula to `3SAT` is referred as the **Tseytin transformation**, named after the Russian mathematician Gregory Tseytin. Apply the Tseytin transformation to the formula

$$F = x_1 \vee (\overline{x}_2 \wedge (x_3 \vee \overline{x}_1))$$

   to obtain an instance of `3SAT`.

5. Consider `3SAT` instance

$$\mathcal{C} = \{c_1 = (x_1, \overline{x}_2, \overline{x}_3), c_2 = (\overline{x}_1, x_2, x_3), c_3 = (x_1, x_2, x_3)), c_4 = (\overline{x}_1, \overline{x}_2, x_3)\},$$

   consider the polynomial-time mapping reduction $f(\mathcal{C}) = (G, k)$ from `3SAT` to `Clique` described in Theorem 5.1.

   a. How many edges does G have?
   b. What is the value of k? Does G have a k-clique? Explain and provide one if your answer is "yes".

6. For the polynomial-time reduction f from `3SAT` to `Clique` described in Theorem 5.1, if an instance $\mathcal{C}$ of `3SAT` has 536 clauses and 243 variables, then, given $(G, k) = f(\mathcal{C})$, how many vertices does G have? Provide a good upper bound on G's size (i.e. number of edges). What is the value of k?

7. For the polynomial-time reduction f from `3SAT` to `Clique` described in Theorem 5.1, how does the reduction change if we reduce from `4SAT` instead of `3SAT`. Repeat the previous problem but with the reduction coming from an instance of `4SAT`.

8. Consider `3SAT` instance

$$\mathcal{C} = \{c_1 = (\overline{x}_1, \overline{x}_2, \overline{x}_3), c_2 = (\overline{x}_1, x_2, \overline{x}_3), c_3 = (x_1, \overline{x}_2, x_3)), c_4 = (x_1, \overline{x}_2, x_3)\},$$

   consider the polynomial-time mapping reduction $f(\mathcal{C}) = (S, t)$ from `3SAT` to `Subset Sum` described in Theorem 5.3.

   a. Draw S and t as a table of values.
   b. Does S have a subset that sums to t? Explain and provide one if your answer is "yes".

9. For the reduction f : `3SAT` $\rightarrow$ `SS` from `3SAT` to `Subset Sum` provided in Theorem 5.3, if an instance $\mathcal{C}$ of `3SAT` has 275 clauses and 57 variables, then how many integers does the set S have, where $f(\mathcal{C}) = (S, t)$? What is the value of the target integer t?

10. For the reduction f : `3SAT` $\rightarrow$ `SS` from `3SAT` to `Subset Sum` provided in Theorem 5.3, suppose an instance $\mathcal{C}$ of `3SAT` has 57 clauses and 10 variables. Assuming $f(\mathcal{C}) = (S, t)$, what is the size of the smallest subset of S that could possibly sum to the target value t. Explain. What is the greatest size? Explain.

11. Consider the following `3SAT` instance

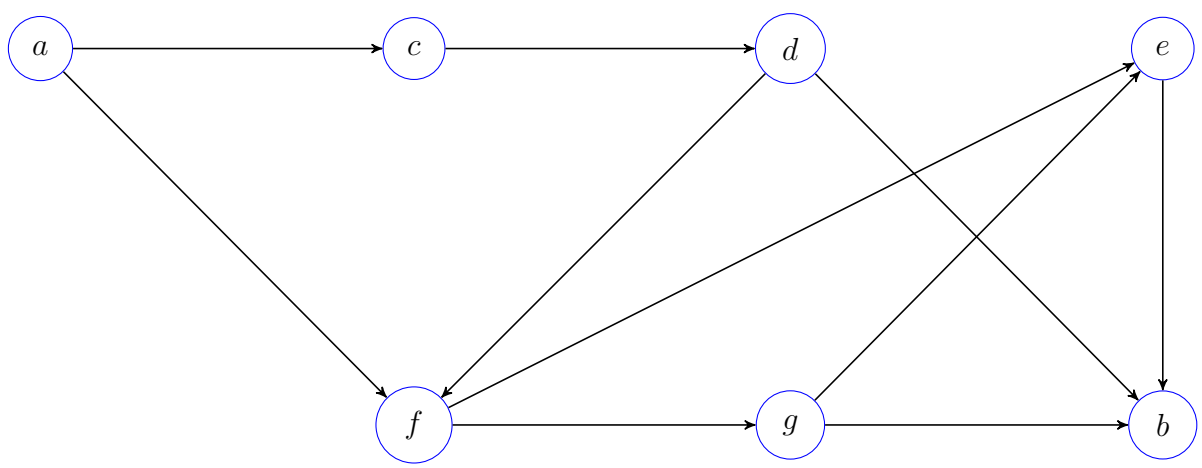
$$\mathcal{C} = \{c_1 = (\overline{x}_1, \overline{x}_4, x_5), c_2 = (x_2, \overline{x}_3, \overline{x}_5), c_3 = (\overline{x}_1, \overline{x}_2, \overline{x}_4), c_4 = (x_1, x_3, x_4), c_5 = (x_2, x_3, \overline{x}_5),$$

$$c_6 = (x_1, \overline{x}_3, x_5), c_7 = (\overline{x}_2, \overline{x}_3, x_4), c_8 = (\overline{x}_1, x_4, x_5), c_9 = (\overline{x}_2, \overline{x}_4, \overline{x}_5)), c_{10} = (x_1, \overline{x}_4, x_5)\},$$

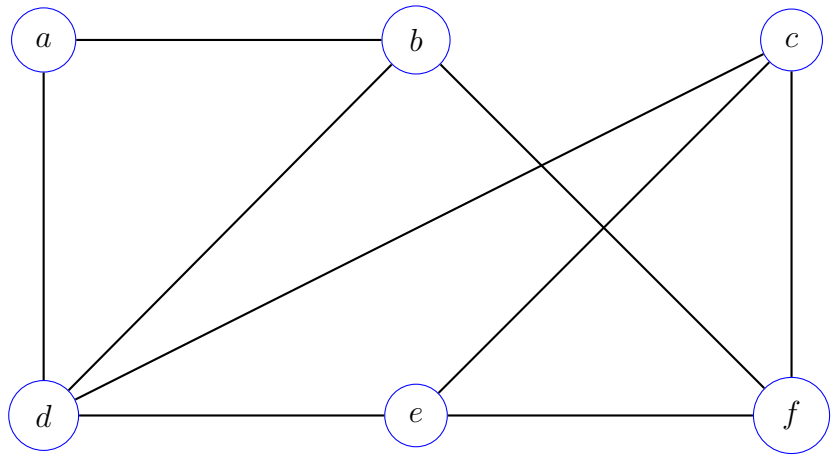
and consider the mapping reduction f : `3SAT` $\rightarrow$ `DHP` described in Theorem 5.5.

   a. Verify that $\mathcal{C}$ is satisfiable by finding a satisfying assignment.
   b. Consider $f(\mathcal{C}) = (G, a, b)$ and let P be the directed Hamilton path from a to b that is guaranteed by the mapping reduction. Indicate the direction (left or right) that the path takes in each of the diamonds.
   c. For each clause c, what it the earliest diamond from which c can be visited along path P?

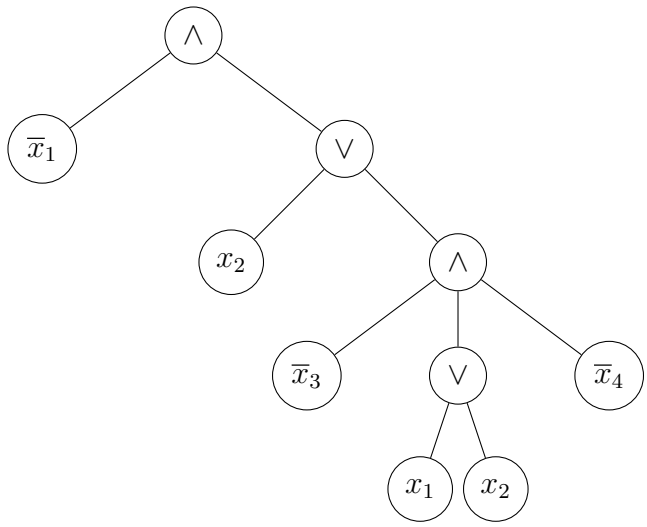
12. For the graph G below, compute $f(G, a, b) = (G', a', b')$ where f is the mapping reduction from `DHP` to `HP`. Draw G' and verify that it has an HP from a' to b', since G has a DHP from a to b.



13. For the graph G below, compute $f(G) = (G', k)$ where f is the mapping reduction from `HC` to `Traveling Salesperson`. Draw G', provide k, and verify that G' has a Hamilton cycle whose cost does not exceed k.


a
b
c
d
e
f

Solutions to NP-Completeness Core Exercises

1. 

2. We have
$$x \leftrightarrow (y \vee z) \Leftrightarrow (x \rightarrow (y \vee z)) \wedge ((y \vee z) \rightarrow x) \Leftrightarrow$$
$$(\overline{x} \vee y \vee z) \wedge (\overline{y} \vee x) \wedge (\overline{z} \vee x) \Leftrightarrow$$
$$(\overline{x} \vee y \vee z) \wedge (\overline{y} \vee x \vee x) \wedge (\overline{z} \vee x \vee x).$$

3. We have
$$x \leftrightarrow (y \wedge z) \Leftrightarrow (x \rightarrow (y \wedge z)) \wedge ((y \wedge z) \rightarrow x) \Leftrightarrow$$
$$(\overline{x} \vee y) \wedge (\overline{x} \vee z) \wedge (\overline{y} \vee \overline{z} \vee x) \Leftrightarrow$$
$$(\overline{x} \vee y \vee y) \wedge (\overline{x} \vee z \vee z) \wedge (\overline{y} \vee \overline{z} \vee x).$$

4. We have
$$y_1 \wedge (y_1 \leftrightarrow (x_1 \vee y_2)) \wedge (y_2 \leftrightarrow (\overline{x}_2 \wedge y_3)) \wedge (y_3 \leftrightarrow (x_3 \vee \overline{x}_1)).$$
Then use the previous two excercises to convert each double-arrow equivalence to CNF.

5. We must count the pairs of consistent vertices between vertex groups $c_1 - c_2$, $c_1 - c_3$, $c_1 - c_4$, $c_2 - c_3$, $c_2 - c_4$, $c_3 - c_4$. The number of pairs of consistent vertices sums to
$$6 + 7 + 7 + 8 + 8 + 7 = 43$$
edges total. Also, $k = |[\mathcal{C}| = 4$ and G does have a 4-clique since $\alpha = (x_1 = 1, x_2 = 1, x_3 = 1)$ satisfies $\mathcal{C}$ (a positive instance of 3SAT must map to a positive instance of `Clique`). One such clique is $C = \{x_1, x_2, x_1, x_3\}$, where the i th literal listed in the set comes from clause c_i, $i = 1, 2, 3, 4$.

6. We have $f(\mathcal{C}) = (G = (V, E), k)$ where $k = 536$, $|V| = 536 \times 3 = 1608$ vertices, and at most
$$\frac{9(536)(535)}{2} = 1,290,420$$
edges (yikes!).

7. If `4SAT` were used instead of `3SAT`, then every vertex group would have 4 (instead of 3) vertices. Then have $f(\mathcal{C}) = (G = (V, E), k)$ where $k = 536$, $|V| = 536 \times 4 = 2144$ vertices, and at most

$$\frac{16(536)(535)}{2} = 2,294,080$$

edges (yikes!).

8. We have the following table that describes the members of S and t.

	1	2	3	c_1	c_2	c_3	c_4
y_1	1	0	0	0	0	1	1
z_1	1	0	0	1	1	0	0
y_2		1	0	0	1	0	0
z_2		1	0	1	0	1	1
y_3			1	0	0	1	1
z_3			1	1	1	0	0
g_1				1	0	0	0
h_1				1	0	0	0
g_2					1	0	0
h_2					1	0	0
g_3						1	0
h_3						1	0
g_4						0	1
h_4						0	1
t	1	1	1	3	3	3	3

Also, since $\alpha = (x_1 = 0, x_2 = 1, x_3 = 1)$ satisfies $\mathcal{C}$ and a positive instance of 3SAT must map to a positive instance of `Subset Sum`, (S, t) is a positive instance and

$$A = \{z_1, y_2, y_3, g_1, h_1, g_2, g_3, h_3, g_4, h_4\}$$

sums to $t = 1,113,333$ (verify!).

9. $|S| = 2(57) + 2(275) = 664$, and $t = 1 \cdots 13 \cdots 3$ has 57 1's and 275 3's.

10. The smallest subset of S that could possibly sum to t has a size equal to 10, since either y_i or z_i must be selected (but not both) for $i = 1, \ldots, 10$. This corresponds with every literal of every clause being satisfied by some assignment. On the other hand, the largest subset could have size equal to $2(57) + 10 = 124$. This would be the case where, in addition to selecting a, y_i or z_i, both filler numbers would also be selected for every clause. This corresponds with exactly one literal of every clause being satisfied by some assignment.

11. We have the following.

    a. $\alpha = (x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 1)$ satisfies $\mathcal{C}$.

    b. Since $\mathcal{C}$ is a positive instance of `3SAT`, $f(\mathcal{C}) = (G, a, b)$ is a positive instance of DHP. Moreover, according to α from part a, the path has the following signature: move right in diamond 1, move right in diamond 2, move left in diamond 3, move left in diamond 4, move right in diamond 5.

c. The earliest diamond from which clause c can be visited corresponds with the least index i for which the assigned value to x_i from α satisfies c. For c_1, this would be diamond 4, since $\overline{x}_4$ satisfies c_1 and x_1 does not. Similarly, for c_2 it is diamond 2, since x_2 satisfies c_2.

12.

13.

Additonal Exercises

1. Recall the reduction from `DHP` to `HP` described in Theorem 6.1. Suppose this reduction only used v_{in} and v_{out} for each vertex, but did not use v_{mid}. Show by example that the mapping reduction is no longer valid. In other words, including v_{mid} is essential.

2. Consider the following practical application of a mapping reduction from `HP` to `HC`.

   a. Rakesh's logistics project requires that he determine whether or not a particular undirected graph $G = (V, E)$ has a Hamilton Path from vertex a to vertex b. Moreover, his colleague Jennifer has implemented a function that takes as input a simple undirected graph $G = (V, E)$ and returns 1 iff G has a Hamilton Cycle. Jennifer says to Rakesh, "you may use my function to get your answer, just make sure to add an edge (if one doesn't already exist) that connects a with b". In other words, Jennifer has provided Rakesh with a way to reduce his `HP` problem instance to an instance of `HC`. Give an example that shows that the answer returned by Jennifer's function might not coincide with the answer to Rakesh's original problem. Conclude that Jennifer's reduction does not work.

   b. What modification should Jennifer have asked Rakesh to make to his graph so that her program's answer would be sure to coincide with the answer to Rakesh's original problem?

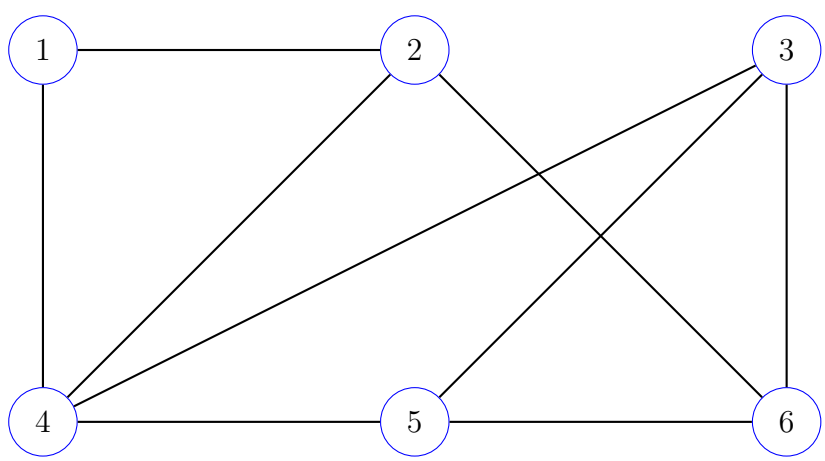
3. Provide a polynomial-time mapping reduction from `Independent Set` to `Vertex Cover`. Defend your reduction: i) establish that it is computable in a polynomial number of steps with respect to the size parameters of `IS`, and ii) argue that a positive (respectively, negative) instance of `IS` maps to a positive (respectively, negative) instance of `Vertex Cover`. Hint: if C is a vertex cover for $G = (V, E)$ of size k, what can you say about the subset of vertices $V - C$?

4. Recall that an instance of `Set Cover` is a triple $(\mathcal{S}, m, k)$, where $\mathcal{S} = \{S_1, \ldots, S_n\}$ is a collection of n subsets, where $S_i \subseteq \{1, \ldots, m\}$, for each $i = 1, \ldots, n$, and a nonnegative integer k. The problem is to decide if there are k subsets $S_{i_1}, \ldots, S_{i_k}$ for which

$$S_{i_1} \cup \cdots \cup S_{i_k} = \{1, \ldots, m\}.$$

   Prove that `Set Cover` is an `NP`-complete problem. Hint: reduce from `Vertex Cover`.

5. For graph G shown below and $k = 3$ Compute $f(G, k)$, where f is the mapping reduction from `VC` to `Set Cover` from the previous exercise.



6. Let `Double-SAT` be the problem of deciding if a Boolean formula has at least two satisfying assignments. Provide a polynomial-time reduction from SAT to `Double-SAT`.

7. Let $\mathcal{C}$ be a 3-CNF formula. A $\neq$-assignment to $\mathcal{C}$ is a truth assignment that satisfies $\mathcal{C}$, but in such a way that every clause of $\mathcal{C}$ has at least one literal set to true, but also has one literal set to false.

   a. Prove that, if a is a $\neq$-assignment, then so is its negation $\overline{a}$, where the negation of an assignment is the assignment that is obtained by negating each assignment value of a.

   b. Let $\neq$-`SAT` be the problem of deciding if a 3-CNF Formula has a $\neq$-assignment. Prove that `3SAT` is polynomial-time mapping reducible to $\neq$`SAT`, by mapping each clause c_i of the form $(l1 \vee l_2 \vee l_3)$ to the two clauses

   $$(l1 \vee l_2 \vee z_i) \text{ and } (\overline{z}_i \vee l_3 \vee b),$$

   where z_i is a newly introduced variable specific to c_i, and b is a single new "global" variable.

8. An instance $(S, \mathcal{C})$ of `Set Splitting` is a finite set S and a collection of subsets $\mathcal{C} = \{C_1, \ldots, C_m\}$ of S. The problem is to decide whether or not S can be partitioned into two sets A and B such that

   a. $S = A \cup B$
   b. $A \cap B = \emptyset$
   c. $C_i \cap A \neq \emptyset$, for all $i = 1, 2, \ldots, m$
   d. $C_i \cap B \neq \emptyset$, for all $i = 1, 2, \ldots, m$.

   Prove that `Set Splitting` is `NP`-complete. Hint: map reduce from $\neq$-SAT.

Solutions to Additonal Exercises

1.

2. The problem is that Rakesh's graph may not have a Hamilton path from a to b but still could have a Hamilton cycle (give an example of such a graph). So, if he simply connects a to b, then his graph will still be a negative instance of `HP`, but will be a positive instance of `HC` and Jennifer's algorithm will return an incorrect answer. To remedy this, Rakesh should add a new vertex v' to his graph, along with edges (a, v') and (b, v'). Then, if his original graph has a Hamilton path P from a to b, then P, v', a yields a Hamilton cycle for his new graph. Conversely, if his new graph has a Hamilton cycle, then the cycle must use the edges (a, v') and (b, v') since they are the only edges incident with v'. In fact, the cycle can be written $C = a, v', b, \ldots, a$, where $b, \ldots, a$ represents a Hamilton path from b to a, and so the reversal of this path gives a Hamilton path from a to b. Therefore, Rakesh's graph will be positive for `HP` iff his modified graph is positive for `HC` and the mapping reduction is now valid.

3. Let $G = (V, E)$ be a simple graph. The key insight is that G has an independent set I of size k iff it has a vertex cover of size $|V| - k$. This is because every edge $e \in E$ is incident with at least one vertex that is *not* in I. If this were false, then there would be an edge that is only incident with vertices in I which would imply that two vetices in I are adjacent, contradicting the independence of I. Therefore, the mapping reduction is simply,

$$f(G = (V, E), k) = (G = (V, E), n - k),$$

where $n = |V|$. Assuming n is provided as part of the input, then f is computable in $\mathrm{O}(\log n)$ steps since we only need to subtract k from n. Finally, based on the above reasoning, G has a k-independent iff $f(G, k) = (G, n - k)$ has a vertex cover of size $n - k$.

4. We leave it as an exercise to prove that `Set Cover` is in `NP`. We now provide a polynomial-time mapping reduction from `Vertex Cover` to `Set Cover`. Let $(G = (V, E), k)$ be an instance of `VC`. Without loss of generality assume $V = \{1, 2, \ldots, n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$, where each edge e_i is of the form (k, j) for some $k, j \in V$. We map this instance to the `Set Cover` instance $(\mathcal{S}, m, k)$, where $\mathcal{S} = \{S_1, \ldots, S_n\}$, and, for each $i = 1, 2, \ldots, n$,

$$S_i = \{j | \text{edge } e_j \text{ is incident with vertex } i\}.$$

In words, S_i is the set of all (indices of) edges that are covered by vertex i. Based on these definitions and the definition of the `Set Cover` decision problem, instance $(\mathcal{S}, m, k)$ is a positive instance of `Set Cover` iff there are sets $S_{i_1}, \ldots, S_{i_k}$ for which

$$S_{i_1} \cup \cdots \cup S_{i_k} = \{1, \ldots, m\}.$$

But this is equivalent to there being k vertices of G that cover all the edges in E. In other words, $(G = (V, E), k)$ is a positive instance of `VC` iff $(\mathcal{S}, m, k)$ is a positive instance of `Set Cover`.

Finally, the mapping reduction requires $\mathrm{O}(m + n)$ steps to construct the sets in $\mathcal{S}$ from $G = (V, E)$, where $m = |E|$ and $n = |V|$.

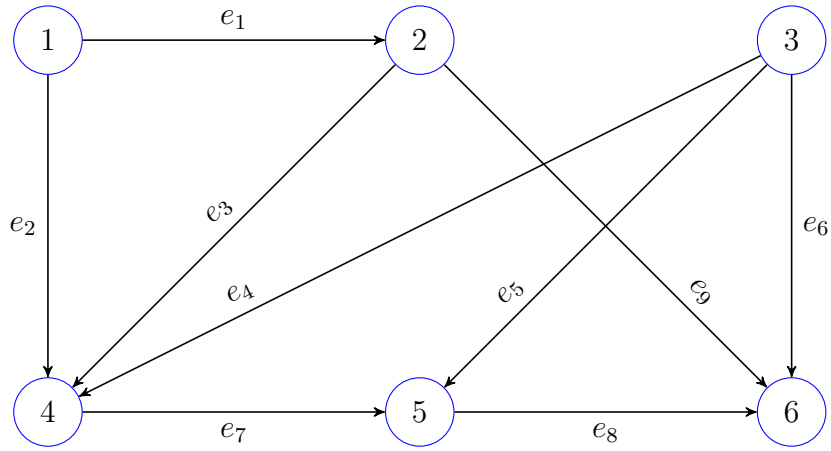
5. We first label each of G's edges as is shown below. Note: please ignore the directions on each edge. This is a technical error. Then $f(G, k) = (\mathcal{S}, m = 8, k = 3)$ is the instance of `Set Cover` for which

$$\mathcal{S} = \{S_1, \ldots, S_6\}$$

and

$$S_1 = \{e_1, e_2\}, S_2 = \{e_1, e_3, e_9\}, S_3 = \{e_4, e_5, e_6\}, S_4 = \{e_2, e_3, e_4, e_7\}, S_5 = \{e_7, e_5, e_8\}, S_6 = \{e_6, e_8, e_9\}.$$

Then G has a vertex cover of size $k = 3$ iff $(\mathcal{S}, m = 9, k = 3)$ has a cover of size $k = 3$, since the sets in the cover correspond with vertices in G, and the members of the sets correspond with the edges of G.



6. $f(F) = \hat{F}$, where $\hat{F} = F \wedge (z \vee \overline{z})$, where z is a variable that does not appear in F. Then F is satisfiable iff $\hat{F}$ has two satisfying assignments (one that sets $z = 1$, the other that sets $z = 0$). In other words, F is a positive instance of SAT iff $f(F)$ is a positive instance for Double-SAT. Note also that $f(F)$ can be computed in polynomial time, since we simply add two nodes to F's tree.

7. a. Given $\neq$-`SAT` instance $\mathcal{C}$ and $\neq$-assignment α that satisfies $\mathcal{C}$, for any $c \in \mathcal{C}$ we have $\alpha(l_1) = 0$ and $\alpha(l_2) = 1$, where l_1 and l_2 are literals of c. Thus, $\overline{\alpha}(l_1) = 1$ and $\overline{\alpha}(l_2) = 0$ and so, since c was arbitrary, $\overline{\alpha}$ is also a $\neq$-assignment.

   b. Let $\mathcal{C}$ be a satisifiable instance of `3SAT` having variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$. Let α be a satisfying assignment for $\mathcal{C}$. We can get a $\neq$-assignment for $f(\mathcal{C})$ over the variables $x_1, \ldots, x_n, z_1, \ldots, z_m, b$, by extending α to $\hat{\alpha}$ in the following way. First, assign $\hat{\alpha}(b) = 0$. Next, consider clause $c_i = (l_1, l_2, l_3)$ of $\mathcal{C}$.

   Case 1: $\alpha(l_1) = \alpha(l_2) = 0$. Then $\alpha(l_3) = 1$, so assign $\hat{\alpha}(z_i) = 1$. In this case, $\hat{\alpha}$ is a $\neq$ assignment for

   $$(l1 \vee l_2 \vee z_i) \text{ and } (\overline{z}_i \vee l_3 \vee b).$$

   Case 2: $\alpha(l_1) = 1$ or $\alpha(l_2) = 1$. In this case assign $\hat{\alpha}(z_i) = 0$. Then $\hat{\alpha}$ is a $\neq$ assignment for

   $$(l_1 \vee l_2 \vee z_i) \text{ and } (\overline{z}_i \vee l_3 \vee b).$$

   Thus, assigning $b = 0$ and the z's in the above manner show that a positive instance of `3SAT` maps to a positive instance of $\neq$-`SAT`.

   Conversely, now assume that $f(\mathcal{C})$ has a $\neq$-assignment $\hat{\alpha}$. Without loss of generality, we may assume that $\hat{\alpha}(b) = 0$ (why?). Let $c_i \in \mathcal{C}$ be arbitrary, where $c_i = (l_1, l_2, l_3)$.

   Case 1: $\hat{\alpha}(z_i) = 1$. Then $\hat{\alpha}(\overline{z}_i) = 0$ which forces $\hat{\alpha}(l_3) = 1$.

   Case 2: $\hat{\alpha}(z_i) = 0$. Then either $\hat{\alpha}(l_1) = 1$ or $\hat{\alpha}(l_2) = 1$.

   Finally, let α denote the restriction of $\hat{\alpha}$ to the variables $x_1, \ldots, x_n$. Then the above two cases imply that, for each $c_i \in \mathcal{C}$, α assigns a literal of c_i to 1. Therefore, $\mathcal{C}$ is a positive instance of `3SAT`.

8. We leave it as an exercise to prove that `Set Splitting` is in `NP`. Given $\mathcal{C}$, an instance of $\neq$`SAT`, let $x_1, \ldots, x_n$ be its variables, and $c_1, \ldots, c_m$ its clauses. Then $f(\mathcal{C}) = (S, \hat{\mathcal{C}})$ consists of set $S = \{x_1, \overline{x}_1, \ldots, x_n, \overline{x}_n\}$, while $\hat{\mathcal{C}}$ consists of sets $c_1, \ldots, c_m$, along with $\{x_1, \overline{x}_1\}, \ldots, \{x_n, \overline{x}_n\}$. Then if α is a $\neq$-assignment for $\mathcal{C}$, let A be the subset of literals l for which $\alpha(l) = 1$. For

example, if $\alpha = (x_1 = 1, x_2 = 0, x_3 = 1)$, then $A = \{x_1, \overline{x}_2, x_3\}$. Furthermore, let $B = \overline{A}$ be the complement of all the literals of A. For example, if $A = \{x_1, \overline{x}_2, x_3\}$, then $B = \{\overline{x}_1, x_2, \overline{x}_3\}$. Clearly, $A \cup B = S$, and both A and B intersect each of the sets of C. Indeed, since α is a $\neq$-assignment, each clause c will contain a literal of A and a literal of B, while each set $\{x_i, \overline{x}_i\}$ will have either $x_i \in A$ and $\overline{x}_i \in B$, or $x_i \in B$ and $\overline{x}_i \in A$.

Conversely, if there exist A and B for which $S = A \cup B$, and A and B intersect all of the clauses and literal sets $\{x_i, \overline{x}_i\}$, then A and B must both be consistent sets of literals, since, for any variable x_i, neither can possess both x_i and $\overline{x}_i$. Let α_A be the assignment over $\{x_1, \ldots, x_n\}$ induced by A, and β_B the assignment induced by B. Then, β is the complement of α. Moreover, since both α and β satisfy each clause (since A and B intersect each clause), we see that α is a $\neq$-assignment for $\mathcal{C}$.

Finally, notice that the reduction can be performed in polynomial time, since the number of sets in $\hat{\mathcal{C}}$ is $m + n$, which is linear in m and n, the size parameters of $\neq$-`SAT`.