Introduction to Computational Complexity Theory

Last Updated September 12th, 2023

1 Introduction

Definition 1.1. The **Computational complexity** of a computational problem refers to the minimum amount of resources (e.g. execution steps or memory) needed to solve an instance of the problem in relation to its size.

In other words, given problem A, we seek functions $s(n)$ and $t(n)$ such that, if $\mathcal{A}$ is an algorithm that solves A, then $\mathcal{A}$ will require $\Omega(s(n))$ amount of memory and $\Omega(t(n))$ number of steps to solve an instance of A of size n, where n is the (for simplicity, we're assuming only one) size parameter for A.

In this chapter we focus almost entirely on decision problems. One reason for this is that the vast majority of problems that are of interest to both computing practitioners and complexity theorists are either decision or optimization problems. And we witnessed in Chapter 2 how an optimization problem can be readily translated into a decision problem by introducing a nonnegative integer k that represents a threshold for which it must be decided if the property that is being optimized for the problem instance can achieve the given threshold. For example, an instance of optimization problem `Max Clique` is a simple graph G, and the problem is to find the the largest clique in G. On the other hand, an instance of decision problem `Clique` is a pair (G, k) and the problem is to decide if G has a clique of size k. Notice that an algorithm for solving `Max Clique` immediately yields an algorithm for solving `Clique` (why?). Furthermore, if there is an algorithm for solving `Clique` in $\mathrm{O}(t(n))$ steps, then it can be shown that there is also one for solving `Max Clique` in $\mathrm{O}(\log(n)t(n))$ steps.

2 Problem Size and Size Parameters

Definition 2.1. Given a decision problem A and an instance x of A, |[redacted]| denotes the **size** of x and equals the number of bits needed to encode x in binary. The notation |[redacted]| is often useful when speaking abstractly about a generic decision problem, and an instance x of that problem.

Definition 2.2. Given a decision problem A, a **size parameter** for A is a parameter that may be used to (approximately) represent the size of some aspect of an instance of A. Given an algorithm $\mathcal{A}$ that decides A, its size parameters allow one to describe the number of steps (and/or amount of memory) required by $\mathcal{A}$ as a function of the size parameters.

Example 2.3. The following are some examples of problems, their size parameters, and examples of how those size parameters are used.

`Clique`

1. Instance: $(G = (V, E), k)$
2. Size parameters: $m = |E|$, $n = |V|$
3. Example: verifying that some k vertices form a clique can be done in $\mathrm{O}(n^2)$ steps.

`Subset Sum`

1. Instance: (S, t)
2. Size parameters: $n = |S|$, b is the number of bits needed to write t (we assume that $t \geq s$, for all $s \in S$.
3. Example: verifying that a subset of S sums to t can be done in $\mathrm{O}(nb)$ steps.

`3SAT`

1. Instance: $\mathcal{C}$
2. Size parameters: $m = |\mathcal{C}|$, $n =$ number of variables of $\mathcal{C}$.
3. Example: verifying that an assignment α satisfies $\mathcal{C}$ can be done in $\mathrm{O}(m)$ steps.

`Prime`

1. Instance: n
2. Size parameters: b is the number of bits needed to write n in binary.
3. Example: there is an algorithm that can decide `Prime` using $\mathrm{O}(b^6)$ steps.

3 The Complexity Class `P`

Definition 3.1. A **complexity class** represents a set of decision problems, each of which can be decided by an algorithm that has one or more constraints placed on the resources that it may use when deciding the problem.

Definition 3.2. Decision problem A is a member of complexity class `P` if there is an algorithm that decides A in a polynomial number of steps with respect to the size parameters of A.

Complexity class `P` is considered robust in the sense that its members tend to remain the same from one model of computation to the next (granted, some models of computation are inherently inefficient, and are not appropriate for use in complexity theory).

Example 3.3. Here is a description of some important decision problems that are members of P, some of which required an algorithmic breakthrough before acquiring membership.

Distance between Graph Vertices Given weighted graph $G = (V, E, w)$, vertices $a, b \in V$, and integer $k \geq 0$, is it true that the distance from a to b does not exceed k? Dijkstra's algorithm solves this problem in $O(m \log n)$ steps.

Primality Test Given natural number $n \geq 2$ is n prime? (see "PRIMES is in P", Annals of Mathematics, Pages 781-793 from Volume 160 (2004), Issue 2 by Manindra Agrawal, Neeraj Kayal, Nitin Saxena). The algorithm requires $O(\log^6 n)$ steps.

Linear Optimization Given i) function $f(x) = cx$, for some $1 \times n$-dimensional constant matrix c and $n \times 1$ real-valued matrix x, ii) constant $k \in \mathcal{R}$, iii) $m \times n$ constant matrix A, and iv) $m \times 1$ constant matrix b, is it true that there is an x for which

$$f(x) = cx \leq k,$$

subject to

$$Ax \geq b.$$

Karmarkar's algorithm solves this problem in $O(n^{3.5}L^2 \cdot \log L \cdot \log(\log L))$ steps, where n is the number of problem variables, and L is the number of bits needed to encode an problem instance.

Maximum Flow Given directed network $G = (V, E, c, s, t)$ and integer $k \geq 0$, is there a flow from s to t of size at least k? The Ford Fulkerson algorithm solves this problem in $O(n^3)$ steps.

Perfect Matching Given bipartite graph $G = (U, V, E)$, where $|U| = |V| = n$, does G have a **perfect matching**, i.e. a set of edges $M = \{e_1, \ldots, e_n\} \subseteq E$ such that any two edges $e_i, e_j \in M$ neither share a vertex in U, nor share a vertex in V? The Ford Fulkerson algorithm solves this problem in $O(n^2 + mn)$ steps.

2SAT Given a set of Boolean formulas $\mathcal{C}$, where each formula (called a **clause**) has the form $a \vee b$, where a and b are literals, is there a truth assignment for the variables so that each clause has at least one literal that is assigned `true`? This problem can be solved in $O(m + n)$ steps.

Bitonic Traveling Salesperson given n cities $c_1, \ldots, c_n$, where c_i has grid coordinates (x_i, y_i), and a cost matrix C, where entry C_{ij} denotes the cost of traveling from city i to city j, determine a left-to-right followed by right-to-left Hamilton-cycle tour of all the cities that minimizes the total traveling cost. In other words, the tour starts at the leftmost city, proceeds from left to right visiting a subset of the cities (including the rightmost city), and then concludes from right to left visiting the remaining cities. The problem can be solved in $O(n \log^2 n)$ steps.

Figure 1: Solving an NP problem can be like finding a needle in a haystack.

4 The Complexity Class NP

Definition 4.1. Decision problem A is a member of complexity class NP if there is

1. a set Cert, called the **certificate set**,
2. a decision algorithm V, called the **verifier**, which has the following properties:
   (a) the inputs to V are i) an instance x of A and ii) a certificate $c \in$ Cert
   (b) the output is 1 iff c is a **valid certificate** for x, meaning that c proves that x is a positive instance of A
   (c) V requires a polynomial number of steps with respect to the size parameters of A.

Although, for any given instance x of A and any certificate $c \in$ Cert, the verifier only requires a polynomial number of steps, what makes some NP problems very difficult to solve is that there are usually an exponential number of certificates, and finding a valid one is like finding a "needle-in-a-haystack" because there is no apparent way to avoid having to examine an exponential (in the size parameters of A) number of certificates.

Normally, two inputs to an algorithm are usually stated independently of one another. However, in the following examples and exercises, we may define the certificate set in relation to the instance of A being evaluated by the verifier. That way we may tailor the certificate to the particular instance of A being evaluated.

Example 4.2. We show that `Clique` $\in$ `NP`. Let $(G = (V, E), k)$ be a problem instance for `Clique`.

Step 1: define a certificate. Certificate C is a subset of V where $|C| = k$.

Step 2: provide a semi-formal verifier algorithm.

```
Place all edges $e \in E$ into a lookup (e.g. hash) table.

For each $u \in C$,

    For each $v \in C$ with $u \neq v$,
        If $(u, v) \notin E$, then return 0.

Return 1.
```

Step 3: size parameters for `Clique`. $m = |E|$ and $n = |V|$.

Step 4: provide the verifier's running time with an explanation.

Building the lookup table takes $\mathrm{O}(m)$ steps and a single query to the table requires $\mathrm{O}(1)$ steps. Finally, the nested `for`-loops require at most $k^2 = \mathrm{O}(n^2)$ lookup queries for a total of $\mathrm{O}(m + n^2) = \mathrm{O}(n^2)$ steps, which is quadratic in the size of (G, k).

Example 4.3. By repeating the steps of Example 4.2, prove that `Subset Sum` $\in$ `NP`. Let (S, t) be an instance of `Subset Sum`.

Step 1: define a certificate.

A is a subset of S.

Step 2: provide a semi-formal verifier algorithm.

Return $\left(\sum_{a \in A} a = t\right)$.

Step 3: provide size parameters for `Subset Sum`.

b = # of bits of t

$n = |S|$

Step 4: provide the verifier's running time with an explanation.

Complexity of adding all members of A is at most the complexity of adding n b-bit numbers in sequence. Moreover, since target t has at most b bits we may assume that each sum has at most b bits. Thus, $O(nb)$ steps are required which is quadratic in n and b.

Example 4.4. Recall from Exercise 10 of the Computational Problems lecture that the `3-Dimensional Matching (3DM)` decision problem takes as input three sets A, B, and C, each having size n, along with a set S of triples of the form (a, b, c) where $a \in A$, $b \in B$, and $c \in C$. We assume that $|S| = m \geq n$. The problem is to decide if there exists a subset $T \subseteq S$ of n triples for which each member from $A \cup B \cup C$ belongs to exactly one of the triples.

Show that (A, B, C, S) is a positive instance of `3DM`, where $A = \{a, b, c\}$, $B = \{1, 2, 3\}$, $C = \{x, y, z\}$, and

$$S = \{(a, 1, x), \boxed{(a, 2, z)}, (a, 3, z), \boxed{(b, 1, x)}, (b, 2, x), (b, 3, z), (c, 1, x), (c, 2, z), \boxed{(c, 3, y)}\}.$$

Solution. $T = \{(a, 2, z), (b, 1, x), (c, 3, y)\}$ is a 3DM matching for this instance.

Example 4.5. By repeating the steps of Example 4.2, prove that $\texttt{3DM} \in \texttt{NP}$. Let (A, B, C, S) be an instance of $\texttt{3DM}$.

Step 1: define a certificate.

$T \subseteq S$ with $|T| = |A|$.

$t = (t_1, t_2, t_3)$

Step 2: provide a semi-formal verifier algorithm.

```
1. Let H denote a hash table of size 3|A|
       and initially empty.
2. For each $t \in T$,
       If ($t_1 \in H \vee t_2 \in H \vee t_3 \in H$) Return 0.
       Add $t_1, t_2, t_3$ to H
3. Return 1.
```

Step 3: provide size parameters for $\texttt{3DM}$.

$|S| = m \qquad |A| = |B| = |C| = n$

Step 4: provide the verifier's running time with an explanation.

a. Construct H: $O(n)$

b. Iterate through each member of T: $O(n)$

c. Table look up and table insertion are both $O(1)$ for each t_i.

Therefore Running time $= O(n + 6n)$

$= O(n)$.

Example 4.6. By repeating the steps of Example 4.2, prove that 3SAT $\in$ NP. Let $\mathcal{C}$ be an instance of 3SAT.

Step 1: define a certificate.

α is an assignment over variables of $\mathcal{C}$

$c = (l_1, l_2, l_3)$

$c[0] = l_1$ $c[1] = l_2$ etc.

Step 2: provide a semi-formal verifier algorithm.

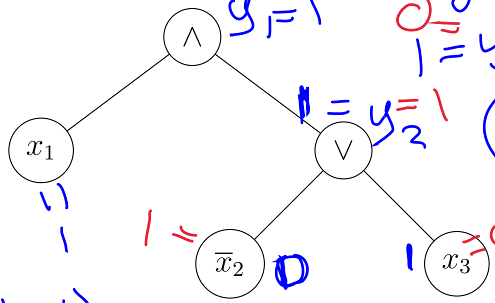
```
For each $c \in \mathcal{C}$,
    If $(\alpha(c[0]) = 0 \wedge \alpha(c[1]) = 0 \wedge \alpha(c[2]) = 0)$, Return 0
Return 1.
```

Step 3: provide size parameters for 3SAT.

$m = |\mathcal{C}|$ $n = \#$ of variables

Step 4: provide the verifier's running time with an explanation.

$O(m)$ since we can check each clause for satisfiability in $O(1)$ steps.
(We're viewing each clause and α as arrays).



$\text{Tseytin}(F) = 1$

$1 = y_1 \wedge ((y_1 \leftrightarrow (x_1 \wedge y_2)) \wedge (y_2 \leftrightarrow (\overline{x}_2 \vee x_3))) = 1$

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

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

$F =$

Tseytin(F) is satisfiable iff F is satisfiable

Figure 2: Parse tree for Boolean formula $x_1 \wedge (\overline{x}_2 \vee x_3)$

4.1 The Satisfiability (SAT) problem is in NP

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 2.

Definition 4.7. assignment over variable set V is a function $\alpha : V \to \{0, 1\}$ that assigns to each variable $x \in V$ a member of $\{0, 1\}$. We may represent α using function notation, or as a labeled tuple. Indeed, 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 $\alpha = (x_1 = 1, x_2 = 1, x_3 = 0)$.

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

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

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

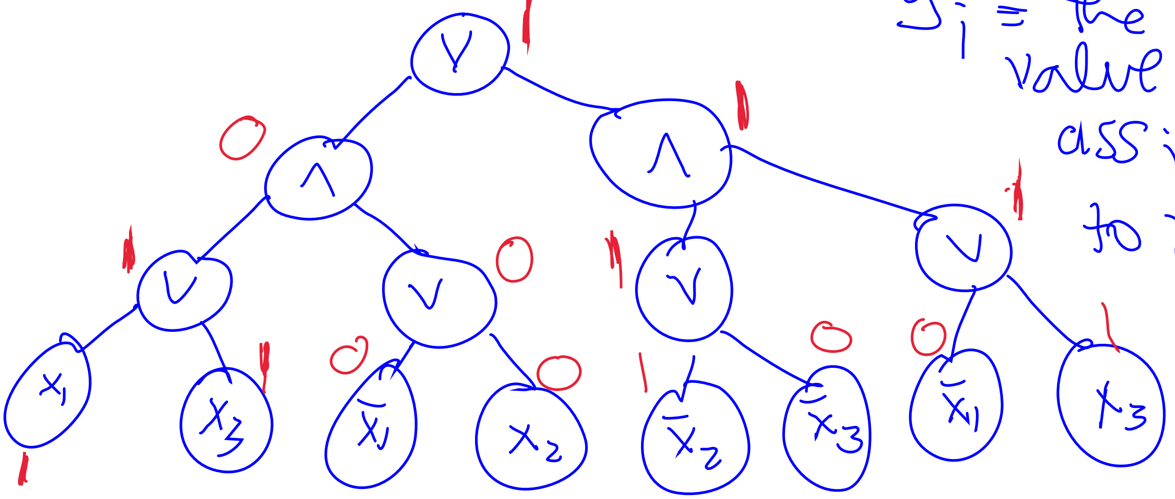
$$\text{eval}(F, \alpha) = \text{eval}(C_1, \alpha) \vee \cdots \vee \text{eval}(C_m, \alpha).$$

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

Example 4.8. 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.



y_i = the value we assign to x_i

$\therefore \quad eval(F, \alpha) = 1$

Example 4.9. 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 $\mathtt{SAT} \in \mathtt{NP}$. Let $F(x_1, \ldots, x_n)$ be an instance of $\mathtt{SAT}$.

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 $\mathtt{SAT}$. **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.

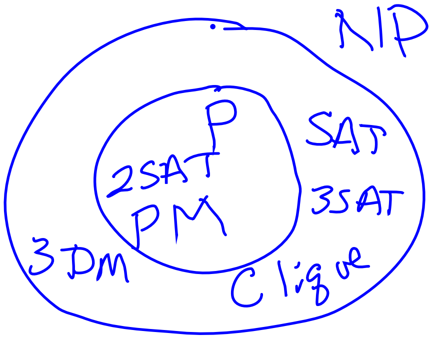
Therefore, $\mathtt{SAT} \in \mathtt{NP}$. □

Theorem 4.10. P ⊆ NP.

Proof. Let $L \in$ P be a decision problem that can be decided in polynomial time. Then there is a polynomial-time computable predicate function $D(x)$ that decides L. We now show that $L \in$ NP by repeating the steps of Example 4.2.

Step 1: define a certificate. **Solution.** Since $L \in$ P, a verifier for L does not require a certificate, since it can run predicate function $D(x)$ in polynomial time to determine if L-instance x is positive. However we must follow the definition of being in NP. So we make a "dummy" certificate set $C = \{1\}$ consisting of a single member which the verifier can ignore.

Step 2: provide a semi-formal verifier algorithm. **Solution.**



```
// A one line program:

Return $D(x)$.
```

Step 3: provide size parameters for L. **Solution.** Since L is a generic problem, we let $|x|$ denote the size of L-instance x.

Step 4: provide the verifier's running time and defend your answer. **Solution.** Since $L \in$ P, predicate function $D(x)$ may be computed in $O(p(|x|))$ steps, for some polynomial p.

Therefore, $L \in$ NP. □

Do there exist decision problems that are in NP but not in P? This would imply that some problems have solutions that can be verified in polynomial time, but not solved in polynomial time. Moreover, NP problems such as Clique, Subset Sum, and 3DM are candidates since, at this writing, polynomial-time algorithms for these problems have yet to be established. But at the same time a proof that such algorithms do not exist has yet to be found.

To better understand the difficulty faced with resolving the P =? NP question, consider the SAT problem. Several algorithms have been designed for solving instances of SAT and, although none of them appear to run in polynomial-time, for some there is no proof that it does *not* run in polynomial time. Moreover, it may be possible to design an "algorithm cocktail" that combines the best SAT algorithms in a way that solves each of the "hard" instances in polynomial time where the polynomial may have a very high degree. In other words, it's possible that two things could be true at the same time:

1. P = NP
2. but some instances of SAT require an exorbitant (albeit polynomial) amount of computational resources to solve, both now and in the forseeable future.

The P=?NP problem is considered one of the most challenging and important in all of computer science and mathematics. The Clay Mathematics Institute is awarding a prize of $1 million dollars to anyone who can resolve this problem.

5 NP-Complete Decision Problems

$(n^2)^3 = n^6$

Now that we have an idea about the type of decision problems that belong in NP, we would like a method for demonstrating that a decision problem is in some sense one of the *hardest* 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 problems, the 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 $B \in \texttt{P}$ then necessarily $A \in \texttt{P}$.
- Thefore, if every problem in ^{N}P were polynomial-time reducible to $B \in \texttt{NP}$, then the P =? NP question would hinge on the question of whether B can be solved in polynomial time.

Definition 5.1. A decision problem B is said to be **NP-complete** iff

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

$\text{NP} \leq_m^p B$

Theorem 5.2. (Cook's Theorem) `SAT` is `NP`-complete.

Outline of a Proof of Cook's Theorem

$c \in C$

1. Let $L \in$ `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,

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

6 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 6.1. 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.

$$\boxed{NP} \leq_m^P A \leq_m^P B \implies$$

B is NP-complete

$A \leq_m^P B$ $\quad$ B is tractable $\implies$ A is tractable

$B \to A$

$\overline{A} \to \overline{B}$

$\underline{A \text{ is intractable}} \implies \underline{B \text{ is intractable}}$

Theorem 6.2. SAT $\le_m^p$ 3SAT.

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 we add a set of disjunctive clauses that are logicaly equivalent to the formula

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

Similarly, if n_i is a $\vee$-operation, then we add a set of disjunctive clauses that are logicaly equivalent to the formula

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

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 $\mathcal{C}$. □

Example 6.3. Apply the reduction described in Theorem 6.2 to the Boolean formula

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

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 $\underline{P \leftrightarrow Q}$ with $\underline{(P \rightarrow Q)} \wedge \underline{(Q \rightarrow P)}$.

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

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

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

Step 3: apply De Morgan's rule.

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

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

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

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

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

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 6.3 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 $O(|F|)$ steps which is a linear (and hence a polynomial) number of steps with respect to the size of F.

Theorem 6.4. The following decision problems are all $\mathtt{NP}$-complete: $\mathtt{Clique}$, $\mathtt{Independent\ Set}$, $\mathtt{Half\ Clique}$, $\mathtt{Subset\ Sum}$, and $\mathtt{Set\ Partition}$.

Proof. All of the following mapping reductions were proved in the Mapping Reducibility lecture.

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

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

and

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

Finally, since

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

by Theorem 6.2, Lemma 6.1 implies that all of the above problems are $\mathtt{NP}$-complete. □

Theorem 6.5. 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 $n - 1$. Sucha a path is called a **(Directed) Hamilton Path (DHP)**. Then `DHP` is `NP` complete.

Proof. The fact that `DHP` is in `NP` is left as Exercise 5. 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 6.6 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 6.6 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.

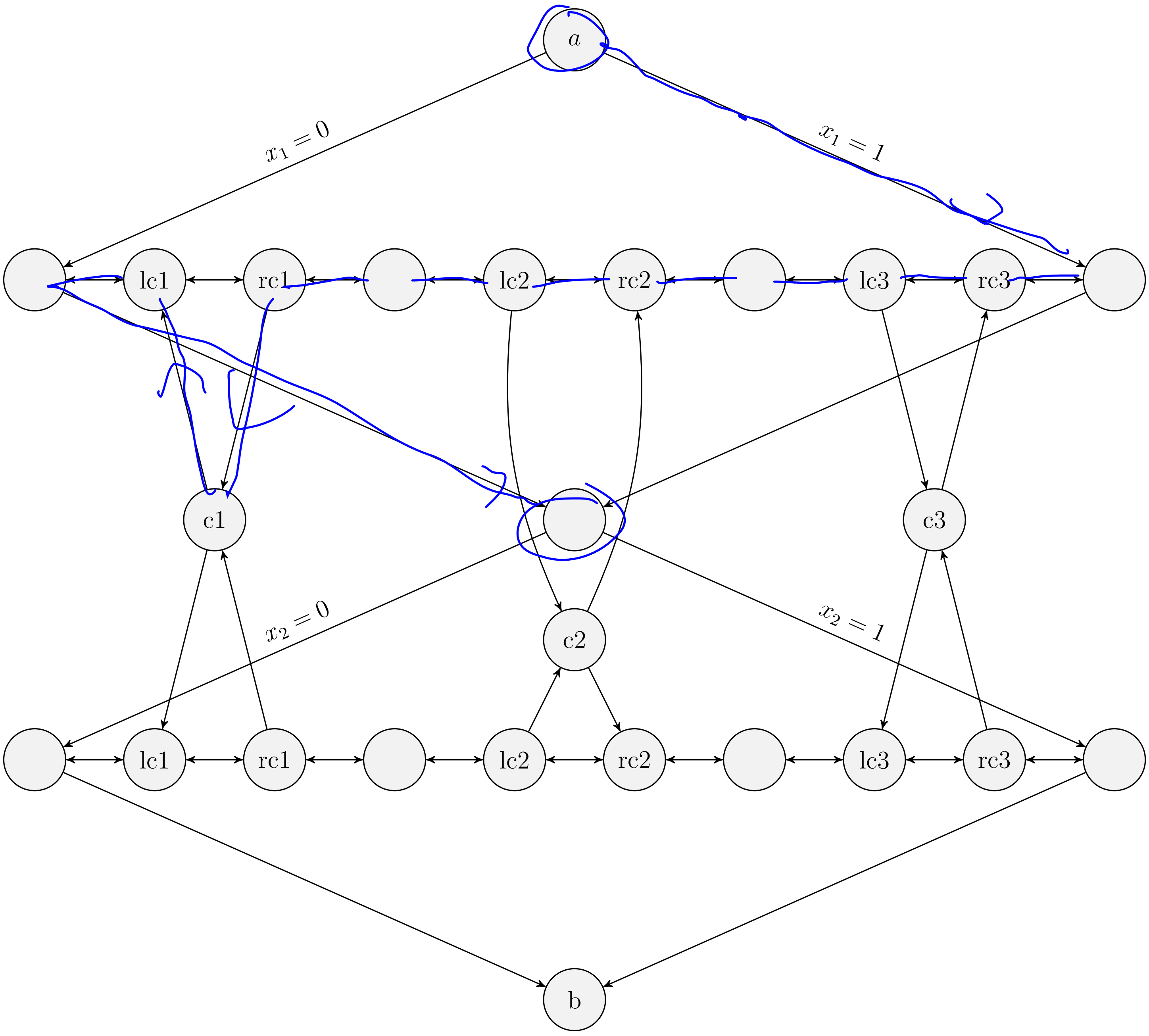
Conversely, suppose P is a DHP in G and let $\Delta(P) = (\delta_1, \ldots, \delta_n)$ be its signature. Then the variable assignment α defined by $\alpha(x_i) = \delta_i$, $i = 1, \ldots, n$, satisfies $\mathcal{C}$. To see this, consider a clause c_j and let D_i be the diamond from where P visits c_j. 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 $\mathcal{C}$. □

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

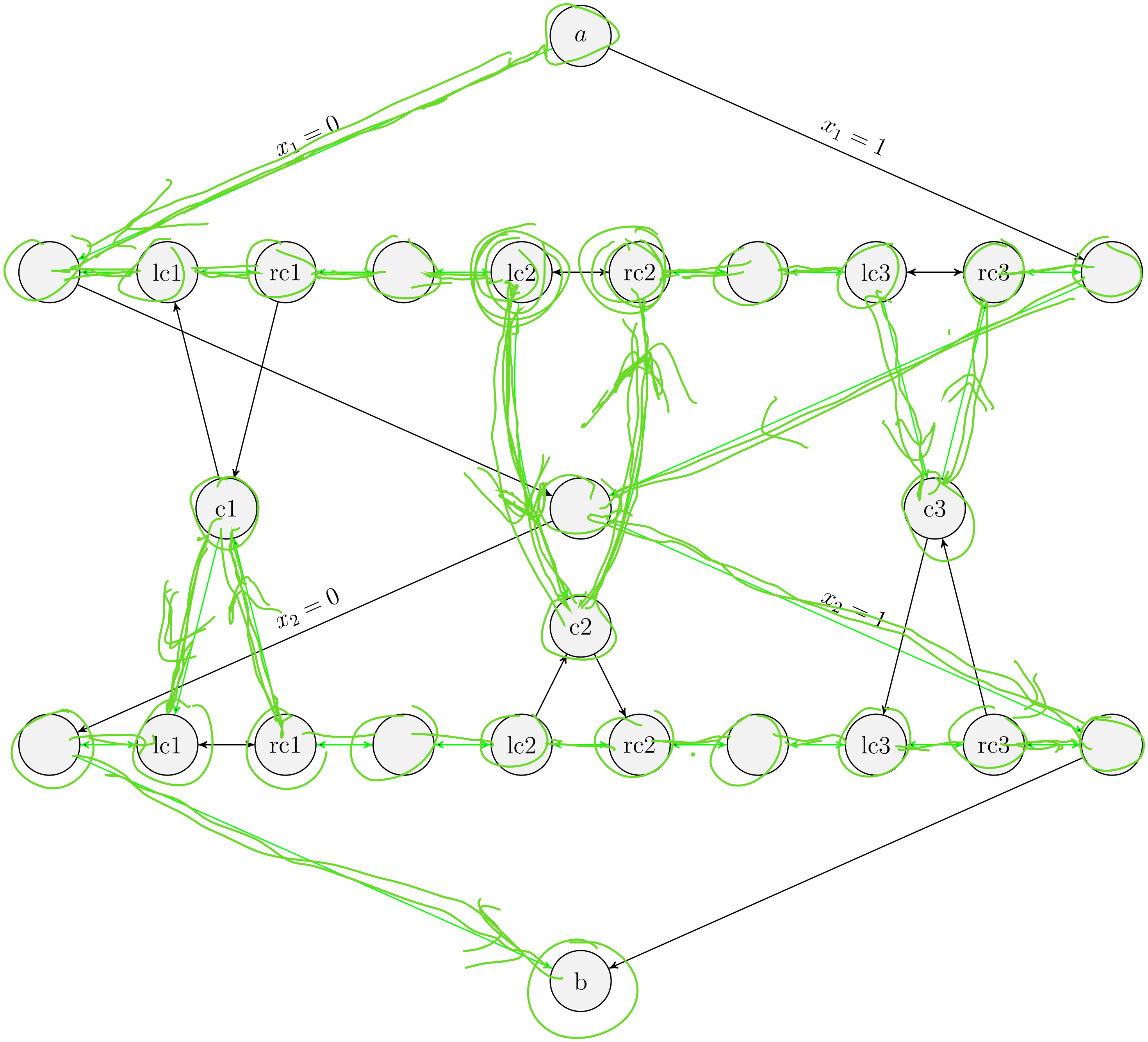
Example 6.6. The following graph shows $f(\mathcal{C})$, where

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

is an instance of `3SAT` and f is the reduction reduction given in Theorem 6.5.



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.



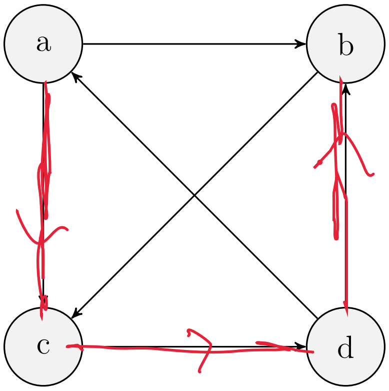
Theorem 6.7. The Hamilton Path (HP) decision problem is the same problem as DHP, but now the edges of graph G are assumed undirected. HP is is NP complete.

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

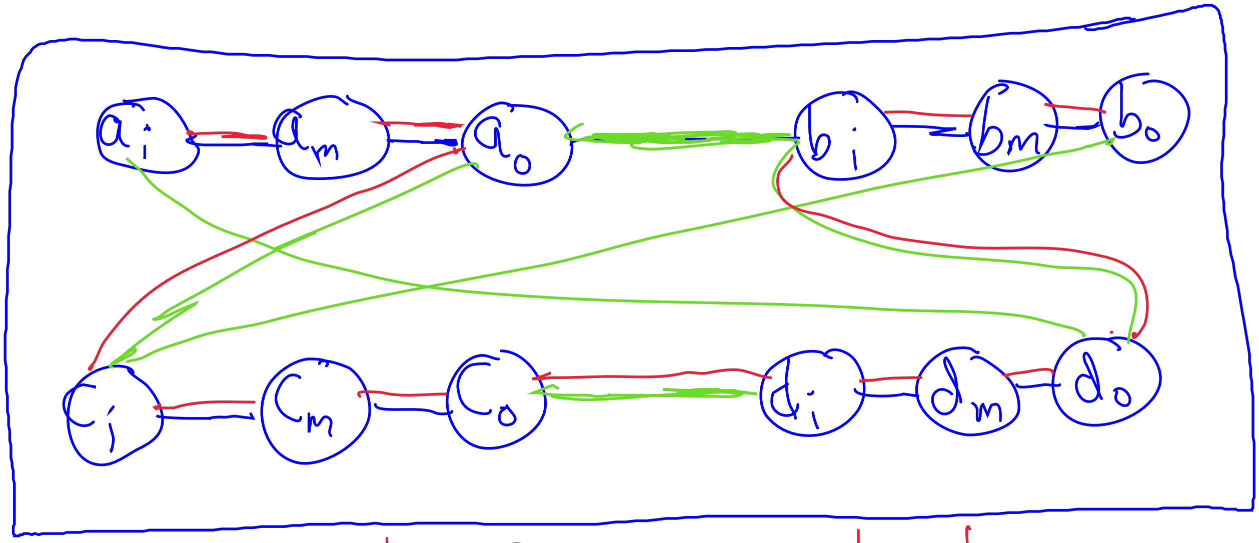
$$(v_{\text{in}}, v_{\text{mid}}), (v_{\text{mid}}, v_{\text{out}}).$$

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



$f(G) =$



$f(G)$ has an HP from a_i to b_o.

We leave it as an exercise to show HC is in NP and to provide a mapping reduction from HP to HC. These exercises show that HC is NP-complete.

Theorem 6.9. 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 $\texttt{TSP} \in \texttt{NP}$. 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

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

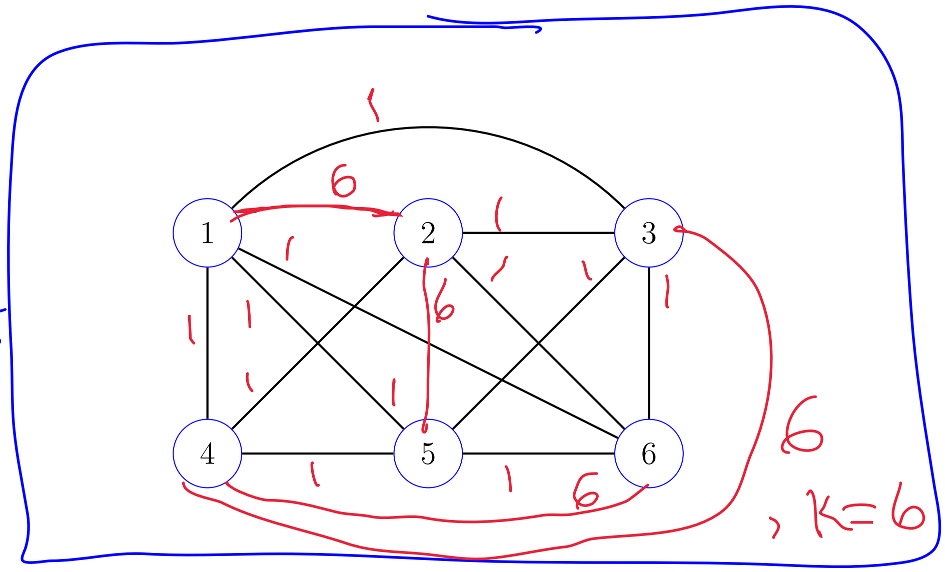
where G' is obtained by taking G and assigning weight 1 to each of its edges. Furthermore, for any $u, v \in V$ for which $(u, v) \notin E$, we add the edge (u, v) to E' and assign it weight n. Thus, G' is a complete weighted graph.

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. □

Example 6.10. Given the graph G shown below, provide $f(G)$, where f is the mapping reduction from `HC` to `TSP`.

$n = |V| = 6$

$f(G) =$



$, k = 6$

The Complexity Class `co-NP`

Given a decision problem L, the **complement** of L, denoted $\overline{L}$, is that decision problem for which a positive (respectively, negative) instance x of $\overline{L}$ is a negative (respectively, positive) instance of L.

Example 6.11. If L is the problem of deciding if a positive integer is prime, then define its complement $\overline{L}$.

Solution. $\text{Composite} = \overline{\text{Prime}}$

n is a positive instanc of Composite iff $n = 1$ or n is a composite number.

Example 6.12. Define the complement of the SAT decision problem.

Solution. Not-SAT

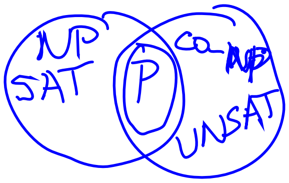
UNSAT

A Logical definition of Co-NP

It is left as an exercise to show that if $L \in$ `P` then $\overline{L} \in$ `P`. On the other hand, it is believed that `NP` and `co-NP` are different complexity classes because each has its own distinct predicate-logic definition.

For example, consider a problem $L \in$ `NP` which has certificate set C and verifier function $v(x, c)$. Then we may logically write that x is a positive instance of L iff

$$\boxed{\exists_{c\in C} v(x, c)}$$



evaluates to 1.

Now consider its complement $\overline{L} \in$ `co-NP`. Then we may logically write that x is a positive instance of $\overline{L}$ iff it is a negative instance of L iff

$V'(x,C) = \neg V(x,C)$

$$\neg \exists_{c\in C} v(x, c) \Leftrightarrow$$

$$\forall_{c\in C} (\neg v(x, c)) \Leftrightarrow$$

$$\boxed{\forall_{c\in C} v'(x, c).}$$

evaluates to 1, where $v'(x, c) = \neg v(x, c)$. In other words, `co-NP` problems are logically defined with a universal predicate-logic statement, while an `NP` problem is logically defined with an existential predicate-logic statement. Thus, logically speaking, these classes seem different in that their problems are complementary to one another.

Example 6.13. For each of the following problem definitions, provide the complexity class (`P`, `NP`, or `co-NP`) that best fits the problem.

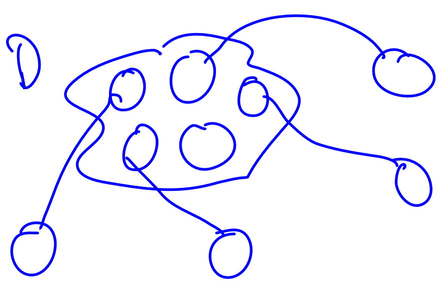
`Tautology` Given a Boolean formula $F(x_1, \ldots, x_n)$ does F evaluate to 1 on all possible 2^n binary input vectors? Co-NP

`Reachability` Given a simple graph $G = (V, E)$ and two vertices $a, b \in V$, does there exist a path in G starting at a and ending at b? P

`Dominating Set` Given a simple graph $G = (V, E)$ and an integer $k \geq 0$, does there exist a set D of k vertices for which every vertex in $V - D$ is adjacent to some vertex in D? NP

`Bounded Cliques` Given a simple graph $G = (V, E)$ and an integer $k \geq 0$, it true that G has no k-cliques?

Co-NP

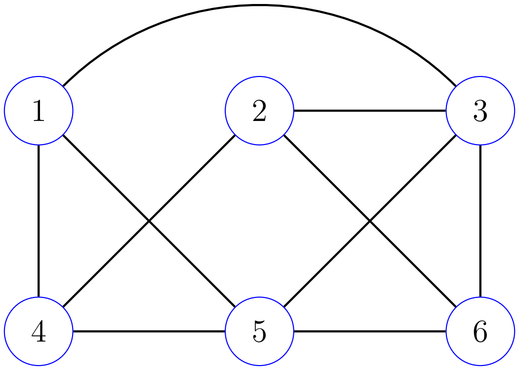


Exercises

1. Let Triangle be the problem of deciding if a simple graph $G = (V, E)$ has a 3-clique. Provide an algorithm that establishes that Triangle is in P. Prove that your algorithm has the requisite running time.

2. Recall the Perm Power decision problem defined in Exercise 20 of Chapter 1. Provide an algorithm that establishes that Perm Power is in P. Prove that your algorithm has the requisite running time.

3. An instance of the 3-Coloring decision problem is a simple graph $G = (V, E)$, and the problem is to decide if the vertices of G can be colored using three colors (red, blue, and green) in such a way that no two adjacent vertices have the same color. In other words, does there exist a function color : $V \to \{\text{red}, \text{blue}, \text{green}\}$, for which, for any edge $(a, b) \in E$, $\text{color}(a) \neq \text{color}(b)$? For example, verify that following graph



admits the 3-coloring

Vertex	Color
1	red
2	green
3	blue
4	blue
5	green
6	red

Prove that 3-Coloring is in NP by completing the following steps.

   a. Define a certificate for 3-Coloring.
   b. Provide a semi-formal algorithm for the 3-Coloring verifier.
   c. Provide size parameters for 3-Coloring.
   d. Provide the verifier's running time and defend your answer.

4. Recall that an instance of Set Partition is a set S of nonnegative integers and the problem is to decide if there are are subsets $A, B \subseteq S$ for which i) $A \cap B = \emptyset$, ii) $A \cup B = S$, and iii)

$$\sum_{a \in A} a = \sum_{b \in B} b.$$

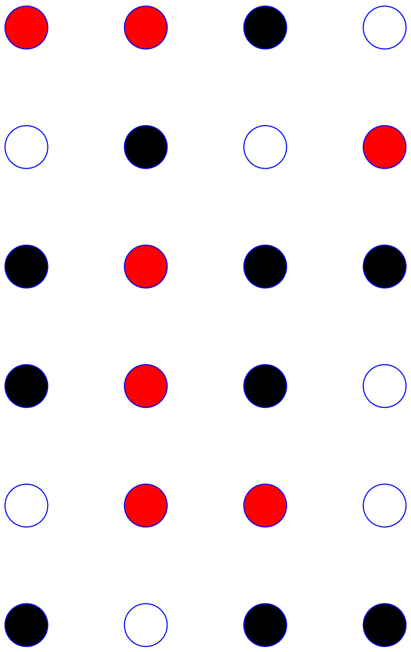
Prove that Set Partition is in NP by completing the following steps.

   a. Define a certificate for `Set Partition`.
   b. Provide a semi-formal algorithm for the `Set Partition` verifier.
   c. Provide size parameters for `Set Partition`.
   d. Provide the verifier's running time and defend your answer.

5. Recall the `DHP` decision problem, where an instance consists of a directed graph $G = (V, E)$ and vertices $a, b \in V$ and the problem is to decide if G has a directed Hamilton path (DHP), Prove that `DHP` is in `NP` by completing the following steps.

   a. Define a certificate for `DHP`.
   b. Provide a semi-formal algorithm for the `DHP` verifier.
   c. Provide size parameters for `DHP`.
   d. Provide the verifier's running time and defend your answer.

6. Consider the `Solitaire` decision problem, where an instance consists of an $m \times n$ grid, and each square in the grid is either empty, has a single red stone, or has a single black stone. The problem is to decide if, for each column, there is a subset of the stones that can be removed so that i) every column has zero or more stones of the same color, and ii) every row has at least one stone placed in it. Show that the following is a positive instance of `Solitaire`.



7. Prove that the `Solitaire` decision problem defined in the previous exercise is in `NP` by completing the following steps.

   a. Define a certificate for `Solitaire`.
   b. Provide a semi-formal algorithm for the `Solitaire` verifier.
   c. Provide size parameters for `Solitaire`.
   d. Provide the verifier's running time and defend your answer.

8. 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

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

Verify that $(\mathcal{S}, m, k)$ is a positive instance of Set Cover, where $m = 9$, $k = 4$, and

$$\mathcal{S} = \{\{1,3,5\}, \{3,7,9\}, \{2,4,5\}, \{2,6,7\}, \{6,7,9\}, \{2,7,9\}, \{1,3,7\}, \{4,5,8\}\}.$$

9. 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 $F(\beta) = 0$.

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

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

12. The transformation used in Example 5 from a SAT formula to an instance of 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.

13. Consider the following 3SAT instance

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

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

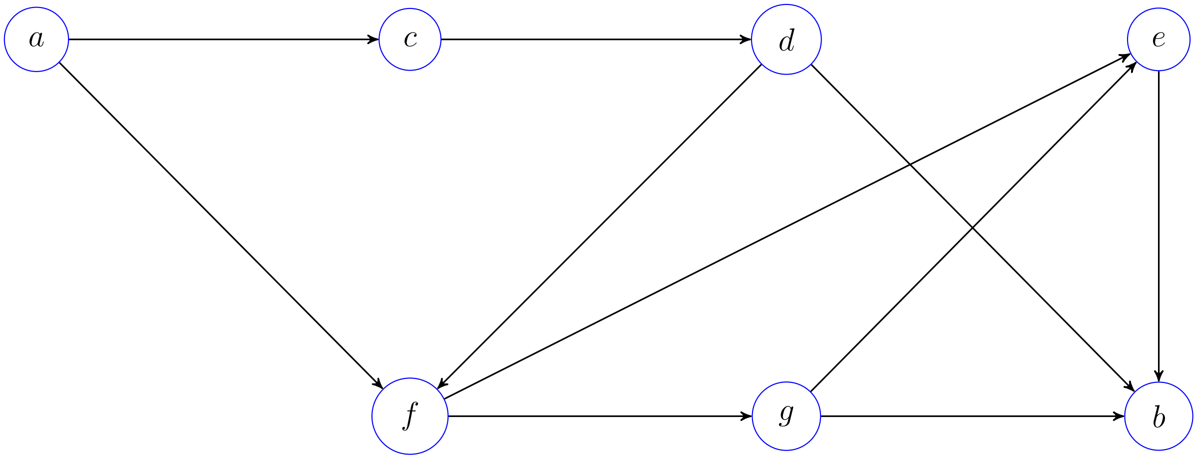
and consider the mapping reduction $f :$ 3SAT $\rightarrow$ DHP described in Theorem 6.5.

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?

14. 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.



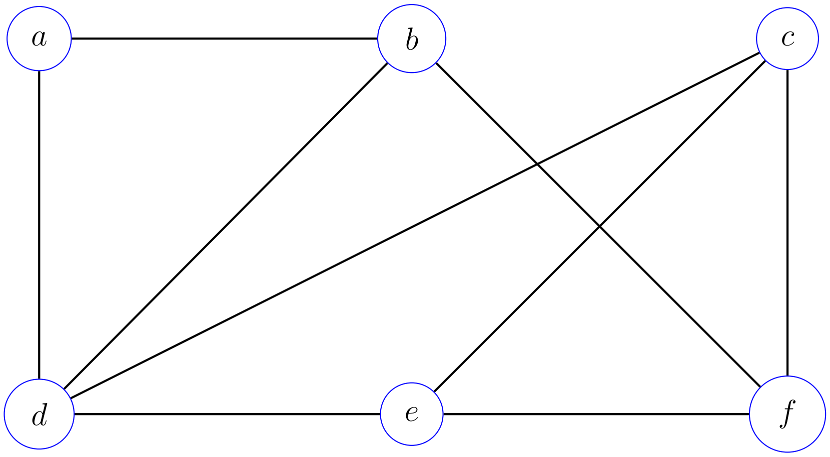
15. Recall the reduction from DHP to HP described in Theorem 6.7. Suppose this reduction only used v_{in} and v_{out} for each vertex, but did not use v_{mid}. Show by example that the mapping reduction is no longer valid. In other words, including v_{mid} is essential.

16. 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?

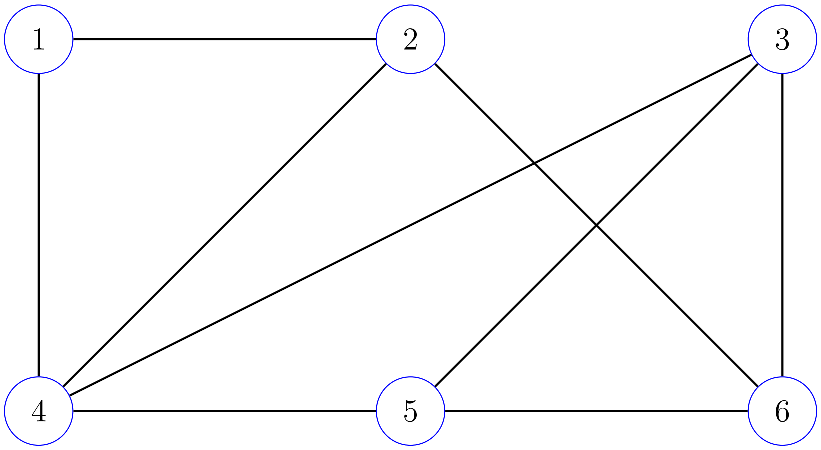
17. For the graph G below, compute $f(G) = (G', k)$ where f is the mapping reduction from HC to Traveling Salesperson. Draw G', provide k, and verify that G' has a Hamilton cycle whose cost does not exceed k.



18. 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 $V - C$?

19. Prove that `Set Cover` (see Exercise 8) is an `NP`-complete problem. Hint: reduce from `Vertex Cover`.

20. 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.



21. 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`.

22. 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.

23. 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.

Exercise Solutions

1. We have the following algorithm.

```
Name: has_triangle
Input: simple graph $G = (V, E)$.
Output: true iff $G$ has a triangle.
Add each edge $e \in E$ to a lookup table.
For each $u \in V$,
    For each $v \in V$ with $v \neq u$,
        For each $w \in V$ with $w \neq u$ and $w \neq v$,
            If $(u, v) \in E$, and $(u, w) \in E$, and $(v, w) \in E$, Return 1.
Return 0.
```

   Each of the three nested loops makes at most $n = |V|$ iterations, for a total of $\mathrm{O}(n^3)$ iterations. Therefore, `Triangle` is in `P`.

2. The size parameters for `Perm Power` are n and $\log t$, where n is the permutation size. Thus, we must decide if $q = p^t$ in a polynomial number of steps in n and $\log t$. The key idea is to only compute power-of-two powers of p, namely $p, p^2, \ldots, p^{2^j}$, where j is the largest integer for which $2^j \leq t$. We can do this via the multiplications

$$p^2 = p \cdot p, p^4 = p^2 \cdot p^2, \ldots p^{2^j} = p^{2^{j-1}} \cdot p^{2^{j-1}},$$

   for a total of $j - 1$ multiplcations, where each multiplication requires $\Theta(n)$ steps. But $2^j \leq t$ implies that $j \leq \log t$. After computing all the power-of-two powers of p, we may multiply selected powers-of-two powers to obtain p^t and then compare it with q. This can all be accomplished in $\Theta(n \log t)$ which is quadratic in n and $\log t$.

   The following algorithm represents the above idea.

```
Name: pow
Inputs: i) permutation $p$, ii) positive integer $t$.
Output: $p^t$.
Initialize: prod = $p$.
Initialize: ans = $I$, where $I = (1\ 2 \cdots n)$ is the identity permutation.
While $t > 0$,
    If $t$ mod $2 = 1$,
        ans = ans $\times$ prod.
    $t = t/2$.
    prod = prod $\times$ prod.
Return ans.
```

3. The following establishes `3-Coloring` is in `NP`.

a. Certificate C is a vector of length n, where the i th vector component, $i \geq 1$, is one of `red`, `blue`, `green`.

b. The following is the verifier algorithm.

```
Inputs: i) simple graph $G = (V, E)$ ii) certificate $C$ which is an $n$-dimensional color vector, and determines the color of the $i$ th vertex.
Output: true iff $C$ does not color two adjacent vertices with the same color.
For each $e = (u_i, v_j) \in E$,
    If $C_i = C_j$, Return 0.
Return 1.
```

c. Size parameters: $m = |E|$, $n = |V|$.

d. Algorithm analysis: the algorithm requires $O(m)$ steps since it iterates once through the set of edges, and accessing a vertex's color from vector C can be done in constant time. Therefore, `3-Coloring` is in `NP`.

4. The following establishes `SP` is in `NP`. Assume S is an instance of `SP`.

a. Certificate A is a subset of S.

b. The following is the verifier algorithm.

```
Inputs: i) set $S$ of nonnegative integers, ii) certificate $A \subseteq S$.
Output: true iff the members in $A$ sum to the members not in $A$ (i.e. in $B = S - A$).
Initialize: $B = S - A$
Return $(\sum_{a \in A} a = \sum_{b \in B} b)$.
```

c. Size parameters: $n = |S|$, m is a bound on the maximum number of bits required by any number in S.

d. Algorithm analysis: the algorithm requires $O(mn)$ steps since it requires making at most n additions with numbers that are at most m-bits each. Therefore, `SP` is in `NP`.

5. The following establishes `DHP` is in `NP`. Assume $(G = (V, E), a, b)$ is an instance of `HP`, where $a \in V$ is the start vertex and $b \in V$ is the end vertex.

a. For simplicity, assume $V = \{1, \ldots, n\}$, $a = 1$, and $b = n$. Certificate P is a permutation of the numbers $1, \ldots, n$.

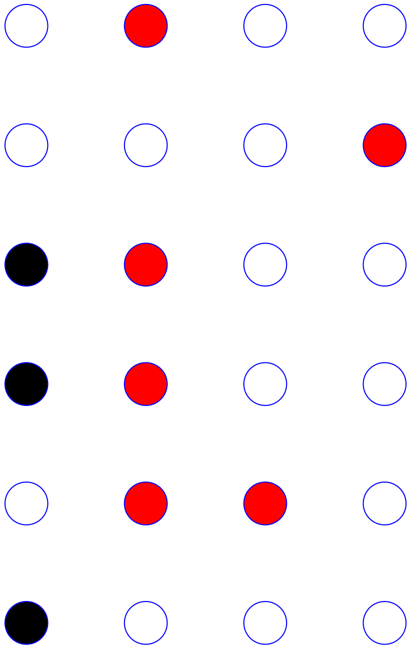
b. The following is the verifier algorithm.

```
Inputs: i) simple graph $G = (V, E)$, ii) certificate $P$, a permutation of the numbers $1, \ldots, n = |V|$.
Output: true iff $P$ forms a valid Hamilton path in $G$.
If $P(1) \neq 1$ or $P(n) \neq n$, Return 0.
Store the edges of $G$ in a lookup table.
For $i = 1, \ldots, n - 1$
    If $(P(i), P(i + 1) \notin E$, Return 0.
Return 1.
```

c. Size parameters: $n = |V|$, $m = |E|$.

d. The algorithm requires $O(m+n)$ steps since it requires $O(m)$ steps to build the lookup table and then $O(n)$ to make sure each pair $(P(i), P(i+1))$ is an edge of G. Therefore, `HP` is in `NP`.

6. Remove all the red stones from column 1, and all the black stones from columns 2-4. Notice that every row has a stone and every column has stones of the same color.



7. The following establishes `Solitaire` is in `NP`. Assume $m \times n$ matrix M is an instance of `Solitaire`, where the entries for M are either 0 (empty), 1 (black), or -1 (red).

a. Certificate R is an n-dimensional vector $(R_1, \dots, R_n)$, where $R_j \subseteq \{1, \dots, m\}$ indicates those row values where a stone in column j is to be removed.

b. The following is the verifier algorithm.

Inputs: i) $\{-1, 0, 1\}$-matrix M, ii) certificate vector $R = (R_1, \dots, R_n)$ of subsets of $\{1, \dots, m\}$.

Output: `true` iff R is a legal and winning prescription for which stones are to be removed in each column.

```
//Check for columns that have stones of different colors.
For each $j = 1, \dots, n$,
    For each $i_1 = 1, \dots, m$ for which $i_1 \notin R_j$,
        For each $i_2 = 1, \dots, m$ for which $i_2 \notin R_j$,
            If $M[i_1, j]M[i_2, j] = -1$, Return 0. //A red and black stone each remain in column $j$.
//Check for rows having no stones.
For each $i = 1, \dots, m$,
        If $\forall j(M[i,j] = 0 \vee i \in R_j)$, Return 0. //Row $i$ has no stones.
    Return 1.
```

c. Size parameters: m: number of rows of M. n: number of columns of M.

d. Algorithm analysis: We assume that all membership queries to each R_j set, $j = 1, \ldots, m$, requires $\mathrm{O}(1)$ steps. This can be accomplished if we represent each R_j as an array of size m. Checking if any columns have different colored stones after the removals have been made requires at most $\mathrm{O}(m^2n)$ steps, while checking if any row is empty requires $\mathrm{O}(mn)$ steps for a total of $\mathrm{O}(m^2n)$ steps. Therefore, `Solitaire` is in `NP`.

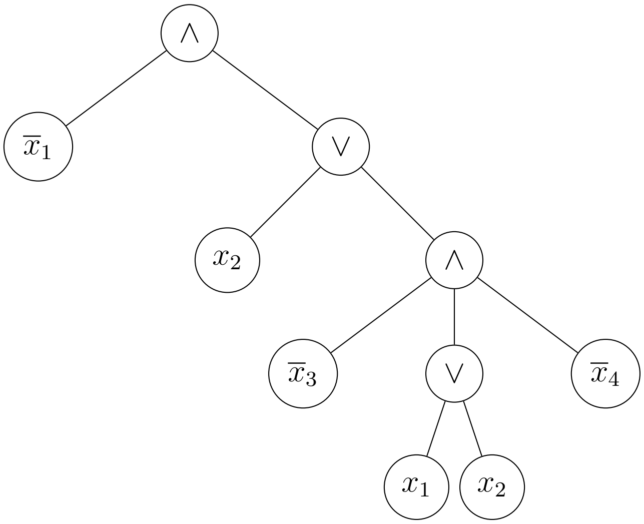
8. The sets

$$\{\{1,3,5\},\{2,6,7\},\{2,7,9\},\{4,5,8\}\}$$

satisfy

$$\{1,3,5\} \cup \{2,6,7\} \cup \{2,7,9\} \cup \{4,5,8\} = \{1,2,\ldots,9\}.$$

9.



10. We have

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

11. We have

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

12. 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.

13. 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.

14.

15.

16.

17.

18. 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 $n - k$.

19. 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 $n = |V|$.

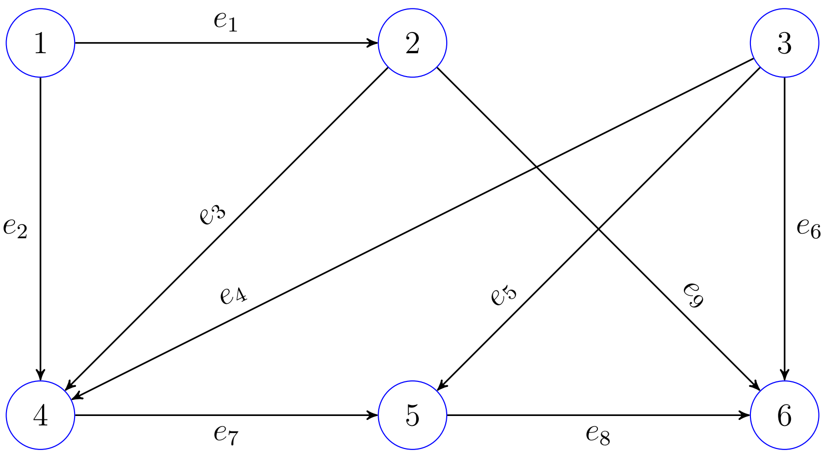
20. We first label each of G's edges as is shown below. Note: please ignore the directions on each edge. This is a technical error. Then $f(G,k) = (\mathcal{S}, m = 8, k = 3)$ is the instance of `Set Cover` for which

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

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.



21. $f(F) = \hat{F}$, 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.

22. 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`.

23. 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`.