Extra Notes: Parsimonious Reductions

Here are a few examples of parsimonious versions of the reductions we gave in the Polytime Reductions handout. Since we're modifying alredy-existing reductions we'll often refer back to the original reductions and just describe the modifications needed.

Contents

1 Examples

1.1 Boolean Logic and Satisfiability

1.1.1 #3SAT

Definition of #3SAT:

> **Instance:** A Boolean formula ϕ in 3CNF.
>
> **Question:** How many satisfying truth assignments does ϕ have?

Proof that #3SAT is $\#\mathcal{P}$-hard:

See the lecture notes from week 11.

■

1.1.2 #SAT

Definition of #SAT *(Boolean Satisfiability)*:

> **Instance:** A Boolean formula ϕ.
>
> **Question:** How many satsifying truth assignments does ϕ have?

Proof that #SAT is $\#\mathcal{P}$-hard:

The #3SAT problem is a fragment of this one, so its $\#\mathcal{P}$-hardness follows from section 1.1.1.

■

1.1.3 #X1-3SAT

Definition of X1-3SAT:

> **INPUT:** A 3CNF ϕ.
>
> **QUESTION:** How many satisfying truth assignments does ϕ hve in which every clause has exactly one literal set to *true*?

Reduced from: #3SAT (with a 1-1 certifacte matching).

Suppose we are given an input 3CNF ϕ. Note that we can assume using our helpful 3SAT lemma from the first Polytime Reductions Handout that the clauses in ϕ each have three distinct variables.

> *Note that this lemma is itself parsimonious: it preserves the number of satisfying truth assignments.*

We will build a new ϕ' as follows: The first clause will be $(x_F \vee x_F \vee \neg x_F)$, where x_F is a new variable that does not occur in ϕ. We can see that the only way to set the value of x_F and get a satisfying X1-3SAT instance is to set it to F.

> *If you want to preserve the three-distinct-variables property, use three new variables: x_F, y_F, and z_F. Then add the clauses $(x_F \vee y_F \vee z_F) \wedge (\neg x_F \vee \neg y_F \vee z_F) \wedge (x_F \vee \neg y_F \vee \neg z_F)$. If x_F were T, then, by clauses 1 and 3, y_F would have to be both T and F, which is impossible. So $x_F = F$. Similarly, $z_F = F$. But then we have to have $y_F = T$ to satisfy clause 1. And we can see that the truth assignment $x_F = z_F = F, y_F = T$ satisfies these clauses.*
>
> *Either way, we can get an x_F that must be set to F.*

Now, for each clause $c_i = (\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$ of ϕ (recall that the literal ℓ_{i_1} is either a variable or its negation, like x_1 or $\neg x_2$), we introduce seven new variables: $a_i, b_i, c_i, d_i, e_i, f_i$, and g_i. We use these variables to write the following clauses:

$$(x_F \vee \ell_{i_1} \vee a_i) \wedge (a_i \vee b_i \vee c_i) \wedge (c_i \vee \ell_{i_2} \vee d_i) \wedge (d_i \vee e_i \vee f_i) \wedge (f_i \vee \ell_{i_3} \vee x_F) \wedge (b_i \vee d_i \vee g_i).$$

Now, consider the following: If ℓ_{i_1} is set to F, then a_i must be set to T (recall that x_F must be F), and look at the first clause. But if a_i is T, then b_i and c_i are F — just walk right through the clauses and you'll see that the variables are forced.

Similarly, if ℓ_{i_1} and ℓ_{i_2} are both F, then a_i and d_i are both T, while b_i, c_i, e_i, and f_i are all F. If fact, if ℓ_{i_1}, ℓ_{i_2}, and ℓ_{i_3} were all F, we'd get the final $x_F = T$, which is impossible. So there would be no way to satisfy this in X1-3SAT.

On the other hand, here are the possible variable assignments for these clauses using the other seven truth assignments for the ℓ_{i_j}:

ℓ_{i_1}	ℓ_{i_2}	ℓ_{i_3}	a_i	b_i	c_i	d_i	e_i	f_i	g_i
T	T	T	F	T	F	F	T	F	F
T	T	F	F	T	F	F	F	T	F
T	F	T	F	F	T	F	T	F	T
T	F	F	F	F	T	F	F	T	T
F	T	T	T	F	F	F	T	F	T
F	T	F	T	F	F	F	F	T	T
F	F	T	T	F	F	T	F	F	F

All of the assignments on the first five clauses are forced by the ℓ_{i_j} and the x_F for every truth assignment except the T, F, T assignment. You can see this by setting the variables neighbouring a T variable in a clause to F, and by setting a variable to T if it is in a clause with two F literals.

For the T, F, T assignment, use the final clause: it forces at most one of b_i or d_i to be T. If one is T, g_i is F. If both are F, g_i is T.

So all told:

- If we repeat this process for every clause, we will have a reduction from 3SAT to X1-3SAT in which every satisfying truth assignment τ to the input 3SAT instance ϕ will correspond to exactly one satisfying truth assignment τ' for the output X1-3SAT ϕ'.
- If τ_1 and τ_2 are different satisfying truth assignments for ϕ, then τ_1' must agree with τ_1 and τ_2' must agree withτ_2 on the variables common to both ϕ and ϕ'. All of the variables in ϕ also occur in ϕ', and so τ_1' and τ_2' must differ from each other, as well.
- Since every satisfying truth assignment τ' can be projected to an assignment τ for ϕ by removing the x_F and the $a_i, b_i, c_i, d_i, e_i, f_i$, and g_i, we see the converse holds, as well.

So the mapping between certificates is one-to-one.

Finally, we can see that this reduction is polytime: for a k-variable, n-clause input ϕ we output a $(1 + k + 7n)$-variable (or a $(3 + k + 7n)$-variable) ϕ' with $1 + 6n$ (or $3 + 6n$) clauses. Each of these clauses can be computed in polytime, so the overall reduction is also polytime, as required.

■

1.1.4 #NAE-3SAT

Definition of #NAE-3SAT:

> **INPUT:** A 3CNF ϕ.
>
> **QUESTION:** How many satisfying ssignments does ϕ have in which each clause has at least one true and at least one false literal?

Reduced from:

#X1-3SAT

(with a 1-2 certifacte matching).

Suppose we are given an input 3CNF ϕ. We will build a new ϕ' as follows:

Firstly, we will introduce a new variable x_C not in ϕ. For every clause $C_i = (\ell_{i_1}, \ell_{i_2}, \ell_{i_3})$ in ϕ, we add the following new clauses:

$(x_c \vee \ell_{i_1} \vee \ell_{i_2})$,

$(x_c \vee \ell_{i_1} \vee \ell_{i_3})$, and

$(x_c \vee \ell_{i_2} \vee \ell_{i_3})$.

Once we have added one copy of these three clauses for every clause in ϕ, we have our ϕ'.

- Now, consider any satisfying truth assignment τ for ϕ (in Exactly-One-3SAT). For every clause $C_i = (\ell_{i_1}, \ell_{i_2}, \ell_{i_3})$ in ϕ, the truth assignment τ will set exactly one literal in each clause to true. The corresponding clause in ϕ' will also have exactly one true literal (and so at least one false literal as well). If we set $x_C = T$, the three clauses $(x_c \vee \ell_{i_1} \vee \ell_{i_2})$, $(x_c \vee \ell_{i_1} \vee \ell_{i_3})$, and $(x_c \vee \ell_{i_2} \vee \ell_{i_3})$ will each have either one or two true literals (and at least one false literal). That is, the ℓ_{i_1}, ℓ_{i_2}, and ℓ_{i_3} cannot contribute more than one true to any of these three clauses. So the truth assignment $\tau' = \tau \cup \{x_C = T\}$ is exactly one satisfying truth assignment for ϕ' (in NAE-3SAT).

  If, by an abuse of notation, we use the term $\neg\tau$ to refer to the truth assignment that we obtain by negating each of the values assigned in τ (i.e., if τ set $x_1 = T$ and $x_2 = F$, then $\neg\tau$ would set $x_1 = F$ and $x_2 = T$, and so on), we can see that $\tau'' = \neg\tau \cup \{x_C = F\}$ is also a satisfying truth assignment for ϕ' (in NAE-3SAT).

  So we can find two separate satisfying truth assignments τ' and τ'' for ϕ' (in NAE-3SAT) for every one in ϕ (in #X1-3SAT). Furthermore, if τ_1 and τ_2 are distinct satisfying truth assignments to ϕ, then τ'_1, τ'_2, τ''_1, and τ''_2 must also be distinct. So there is at least a one-to-two mapping between the certificates for ϕ and some subset of the certificates for ϕ'. We simply need to argue that every certificate for ϕ' is of this form.

- Consider the satisfying truth assignments τ' for ϕ'.

  We note that the negation $\neg\tau'$ of any satisfying truth assignment τ' for ϕ' is also a certificate for ϕ', and that $\neg\neg\tau' = \tau'$. So we can partition the certificates for ϕ' into pairs $\{\tau', \neg\tau'\}$.

  Consider any one of these pairs of certificates for ϕ', and assume wlog. that τ' is the one in which $x_c = T$. Let τ be the truth assignment that we obtain by removing the variable x_C from the assignment.

Now, we can see that since τ' is a certificate for ϕ' in #NAE-3SAT, each clause $C_i = (\ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3})$ of ϕ must have at least one true and one false literal. We argue that in fact, C_i has two false literals.

If it were otherwise, then two of the literals in C_i would be set to true. We'll consider the case where $\ell_{i_1} = \ell_{i_2} = T$. The other two cases are similar. If it were true that $\ell_{i_1} = \ell_{i_2} = T$, then the clause $(x_C \vee \ell_{i_1} \vee \ell_{i_2})$ would eveluate to $(T \vee T \vee T)$, which would violate the #NAE-3SAT condition. So we know that ℓ_{i_1} and ℓ_{i_2} cannot both be true.

Therefore, τ' (and $\neg\tau'$) are of the form described above, and so every certificate for ϕ' is covered by the one-to-two mapping above.

All told, then, the number of certificates for ϕ' is exactly twice the number of certificates for ϕ.

Finally, we can see that we can construct ϕ' by adding three extra clauses for every clause in ϕ. So if ϕ had n clauses and k variables, then ϕ' would have $4n$ clauses and $k+1$ variables. These new clauses can be efficiently computed from the clauses of ϕ, and so the entire reduction is polytime.

■

1.2 Graph Theory

1.2.1 #CLIQUE

Definition of CLIQUE:

INPUT: A graph $G = (V, E)$ and a number $k \in \mathbb{N}$.

QUESTION: How many size-k cliques does G have?

Reduced from: #3SAT (with a 1-1 certifacte matching).

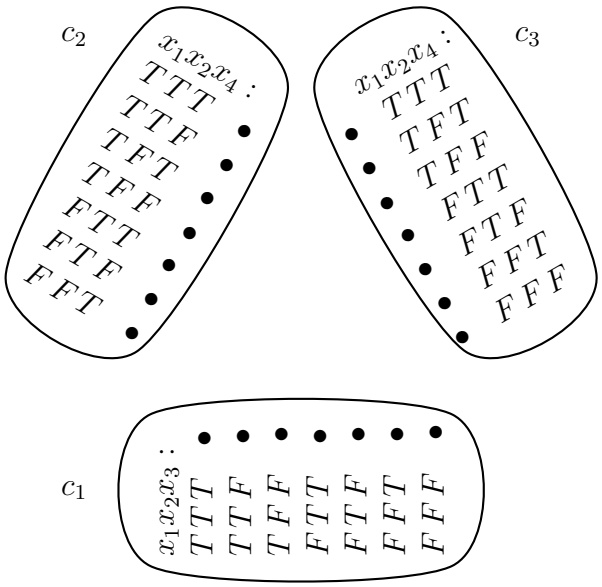
Note: This reduction is derived from the one found in (Arora and Barak, 2009) for INDEPENDENT-SET. *Note that their reduction is also parsiomonious, but is not described as such, since it's in the $\mathcal{NP}$-completeness chapter.*

Suppose that we are given a 3CNF ϕ. We will use this to produce a graph G and a number $k \in \mathbb{N}$.

This reduction will be a lot like the CLIQUE reduction from the polytime reductions chapter, except we'll build groups of seven nodes for each clause instead of groups of three. These seven nodes will correspond to the possible satisfying truth assignments for that clause. We'll connect nodes from different clause groups iff they are consistent in their variable assignments.

As before, no nodes from the same clause group will be connected.

E.g., if we took as an input a $\phi = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$, we'd get the following nodes:



As before, we'll let the resulting graph be G and we'll let k be the number of clauses in ϕ. We'll return $\langle G, k \rangle$.

- We can see that any satisfying assignment to our input ϕ will be associated with exactly one node per clause group. Since all of these nodes will be consistent, they will form a k-clique. We can also see that non-satisfying truth assignments will not give us any k-clique.
- We can also see that if two satisfying truth assignments τ_1 and τ_2 differ on some variable x_i, then the nodes we choose from any widgets for clauses containing x_i or $\neg x_i$ must also differ. So τ_1 and τ_2 each contribute different cliques to G.
- On the other hand, every k-clique must take one node from every clause group. Since these nodes must be consistent with each other, the variable assignment can be extended to form a satisfying truth assignment for ϕ. Since a variable only shows up in G if it shows up in ϕ, there is exactly one such assignment.

So there is a one-to-one relationship between the satisfying truth assignments to ϕ and the k-cliques of G, and so the reduction is parsimonious, as required.

Finally, we see that if ϕ has n clauses and k variables, then G has $7n$ vertices. The vertices, as well as the edges between them, can be calculated efficiently. Therefore the entire reduction is polytime, as required.

■

1.2.2 #INDEPENDENT-SET

Definition of IS:

> **INPUT:** A graph $G = (V, E)$ and a number $k \in \mathbb{N}$.
>
> **QUESTION:** How many independant sets of size k does G have?

Reduced from: #CLIQUE (with a 1-1 certifacte matching).

> *Note: The reduction from* #3SAT *to* #INDEPENDENT-SET *given in (Arora and Barak, 2009) is also parsiomonious, as was stated in the* #CLIQUE *section.*

The reduction given in the first polytime reductions handout is easily seen to be parsiomonious. We leave the details as an exercise.

□

1.2.3 #VERTEX-COVER

Definition of #VERTEX-COVER:

> **INPUT:** A graph $G = (V, E)$ and a number $k \in \mathbb{N}$.
>
> **QUESTION:** How many vertex covers of size k does G have?

Reduced from: #INDEPENDENT-SET (with a 1-1 certifacte matching).

The reduction given in the first polytime reductions handout is easily seen to be parsiomonious. We leave the details as an exercise.

□

1.2.4 #HAM-PATH

Definition of #HAM-PATH:

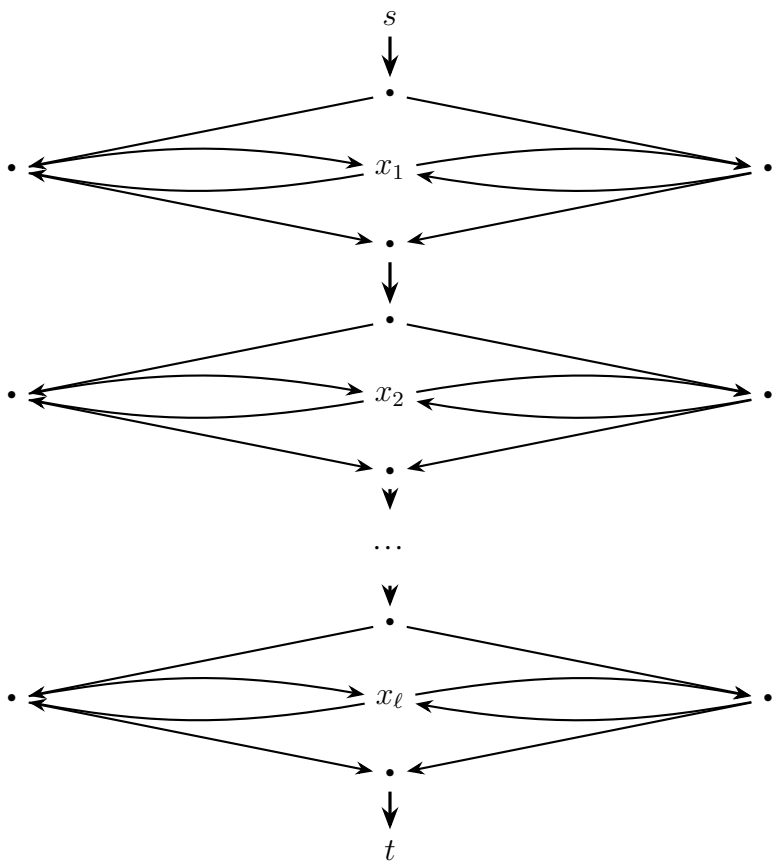
> **INPUT:** A directed graph $G = (V, E)$, two distinct nodes s and t in V.
>
> **QUESTION:** How many Hamiltonian paths from s to t does G have?

Reduced from: #3SAT (with a 1-1 certifacte matching).

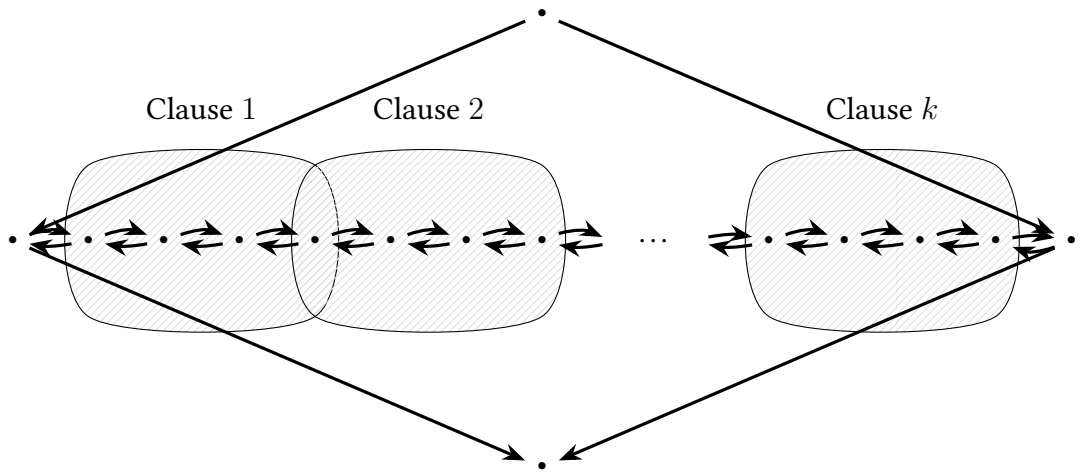
Let a 3CNF formula ϕ be given. We will assume, using our helpful 3SAT lemma from the first Polytime Reductions handout that no clause of ϕ contains repeated literals.

> *We again note that this doesn't change the number of satisfying truth assignments for* ϕ.

We'll start with the reduction from the polytime reductions chapter — that is, we'll create a set of diamond widgets for the variables like so:



and we'll split the internal nodes of the diamond widgets like so:



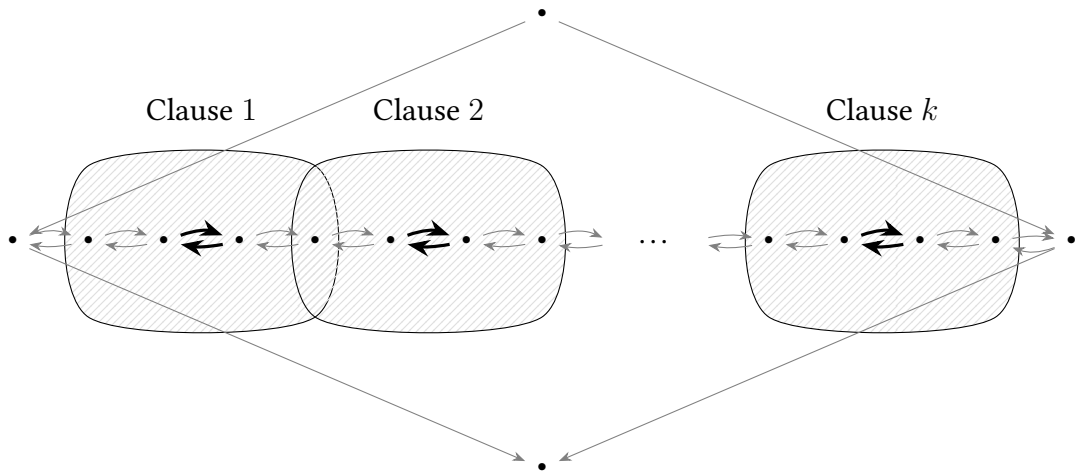
Up to this point everything is parsimonious, so we're OK. But the problem is with the clause node. If a truth assignment satisfies one literal of a clause there'll be one way to go through the node, but if all three literals are true there'll be three. That's what we need to fix.

The way we'll fix this is to have three nodes for each clause, instead of one. We'll only be able to visit the first of these clauses if the clause is satisfied, but we'll have to visit all three. In addition, we'll set it up so that there's only ever one path through the widget.

We'll do this as follows: we have an ordering on the widgets (we can generally assume this, it's how we got the order of the diamonds). So for any clause there'll be a first, second, and

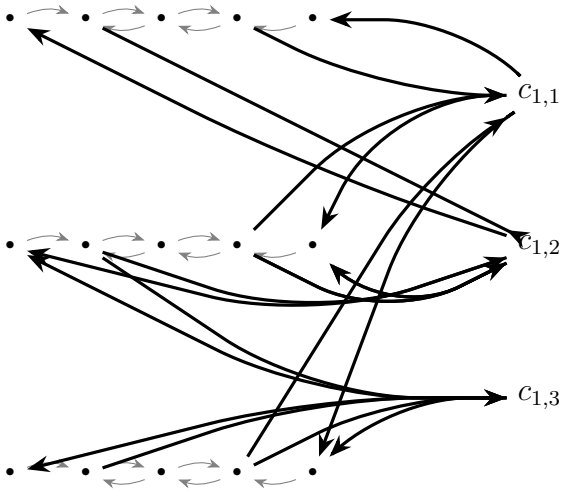
third variable. We'll show how to connect the clause $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$, the other possibilities are similar.

For each of the variables, we'll take the central loop of the clause section of the corresponding variable widget:



We'll separate these into five nodes as follows:

We'll do this for all three variables, and connect them to the clause widget as follows:



Note that we have removed some of the left-right and right-left arrows. Other clauses are analogous. Recall that the left-to-right arrows on the variables indicate that that variable is set to false, while the right-to-left arrows correspond to true.

Now, we can see that since this is for the clause $(\neg x_1 \vee \neg x_2 \vee \neg x_3)$, there should be exactly one path through these nodes for every truth assignment other than *T T T*, which will have no path.

- Firstly, we can see that for the truth assignments *F F F*, *F F T*, *F T F*, and *F T T* (the assignments for which x_1 is set fo false), we cannot visit the $c_{1,2}$ widget from x_1. If we tried this, we wouldn't be able to reach the center node of the x_1 widget and leave again.

  On the other hand, the only way to reach the right-hand side of the widget will be to pass through and visit the $c_{1,1}$ node. Furthermore, the $c_{1,1}$ node will be the only node we can use to access the right node of x_1, and so the path through x_1 is a left-to-right path that passes through $c_{1,1}$.

  Similarly, the path through x_3 will have to be a path (either left-to-right or right-to-left, depending on the variable assignment) that goes through $c_{1,3}$.

  Once we know that these paths are fixed, we can see that the path through x_2 must visit $c_{1,2}$.

- Secondly, if x_1 is set to true, then the right-to-left path through the x_1 widget cannot pass through $c_{1,1}$ without bloacking the middle x_1 node, and must pass through $c_{1,2}$. Since $c_{1,2}$ is th eonly node that will have access to the leftmost node in x_1, the x_1 path must be a right-to-left path through $c_{1,2}$.

- If x_1 is set to true and x_2 is set to false, then the path through x_2 must be a left-to-right path that cannot pass through $c_{1,3}$ (at least, not without blocking the middle x_2 node), and so must visit $c_{1,1}$. Since $c_{1,2}$ is already used, there is no way to access the right side of x_2 except by passing through the $c_{1,2}$ node normally. So the path through x_2 is a left-to-right path that goes through c_2.

  From there, we can see that the x_3 path must pass through $c_{1,3}$, just like in the $x_1 = T$ case.

- If x_1 and x_2 are both true, then, as we have said, the path through x_1 must be a right-to-left path that goes through $c_{1,2}$. Similarly, the only path available for x_2 will be a right-to-left path through $c_{1,3}$.

  This means that the only path available for x_3 will be the left-to-right path that passes through $c_{1,1}$. This can only occur if x_3 is set to true.

So we see that there is exactly one path for every truth assignment other than T T T, and that T T T has no path, as required.

If we repeat this process for every clause, we can see that:

- Every satisfying truth assignment for ϕ contributes exactly one Hamiltonian path to G.

- If τ_1 and τ_2 are different satisfying truth assignments for ϕ, then their Hamiltonian paths must differ on the way they pass through the variable diamond widgets, and so they must contribute different Hamiltonian paths.

- Every Hamiltonian path P in G must pass through each variable diamond widget, and for each widget must take a left-to-right or right-to-left path. The direction of these paths induces a truth assignment τ for ϕ, and we can see that P must be the unique Hamiltonian path contributed by τ.

So there is a one-to-one relationship between the satisfying truth assignments for ϕ and the Hamiltonian paths in G.

Finally, we see that this construction adds 11 vertices and 22 edges per clause beyond the original reduction, and that we can tell how to place these vertices and edges with just a simple lookup. So the overall construction is still polytime.

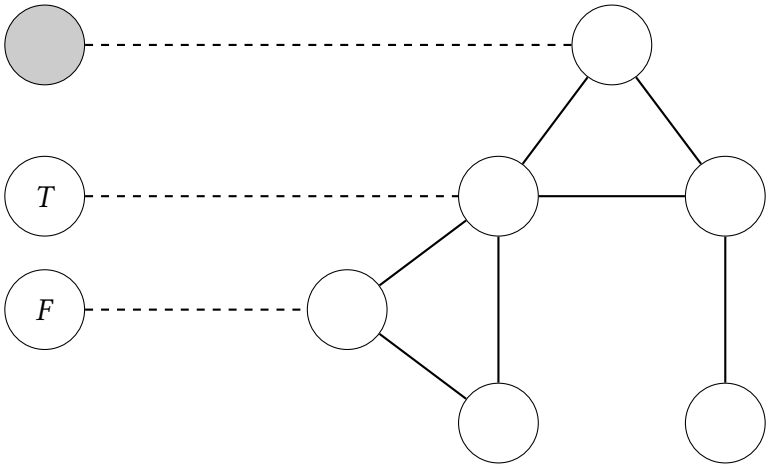
■

1.2.5 #3COL

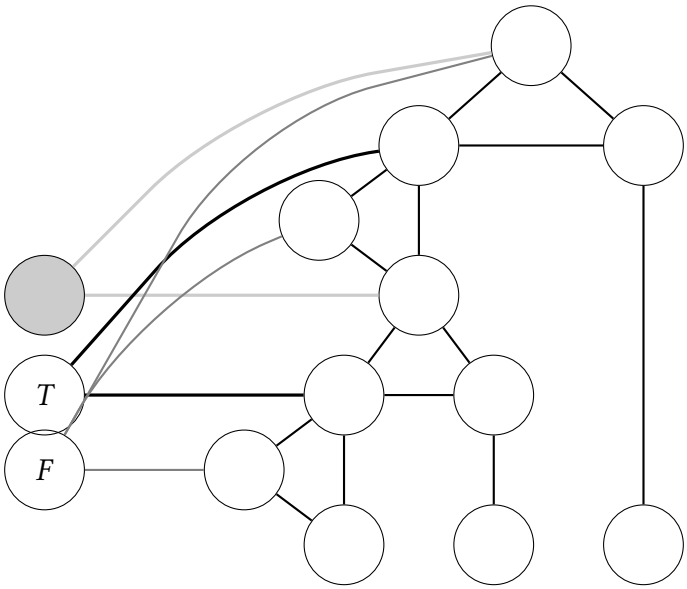
Definition of #3COL:

> **INPUT:** A graph $G = (V, E)$.
>
> **QUESTION:** How many 3-colourings does G have?

Reduced from: #3SAT (with a 1-6 certifacte matching).
We'll start with the reduction from the polytime reductions handout. In particular, the palette and variable widgets will be exactly the same. We'll change things by using the following for the *or*-widget:

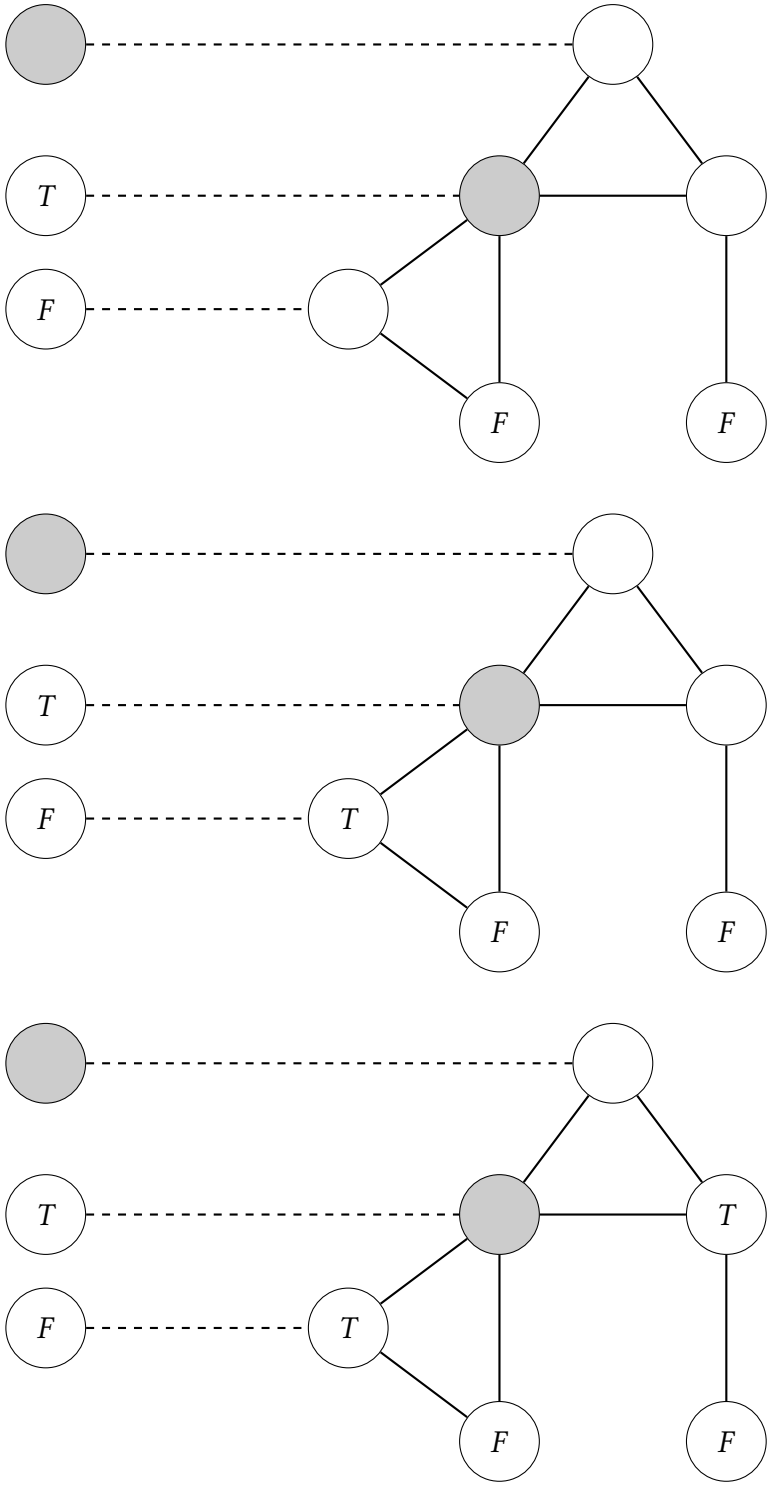


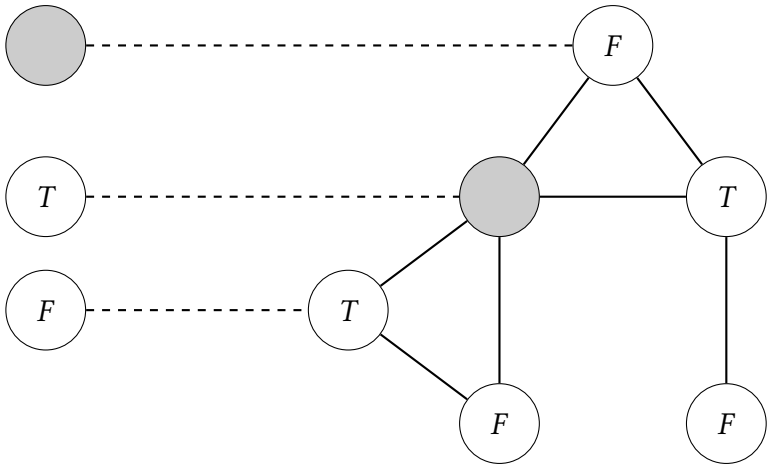
The whole clause widget now looks like this:



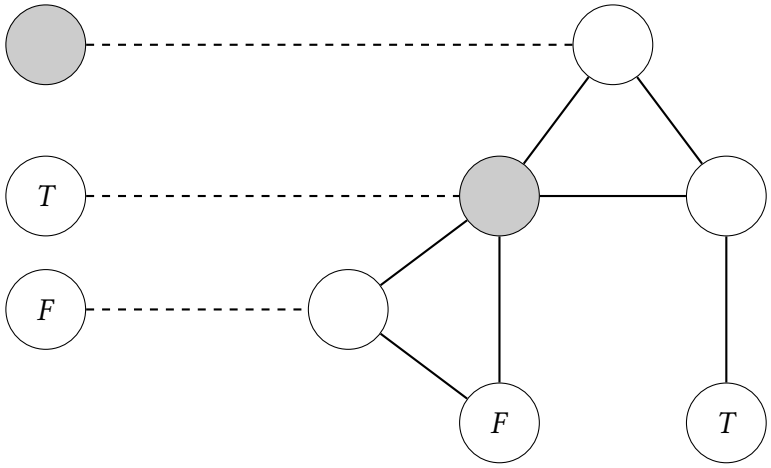
Now, if we input the four possible inputs to the *or*-widget, we get these possible colourings:

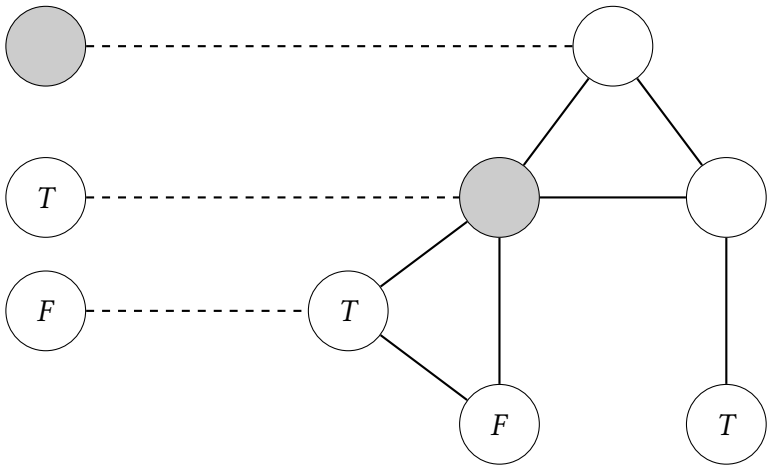
- If the input is *F F*, the colouring must be:

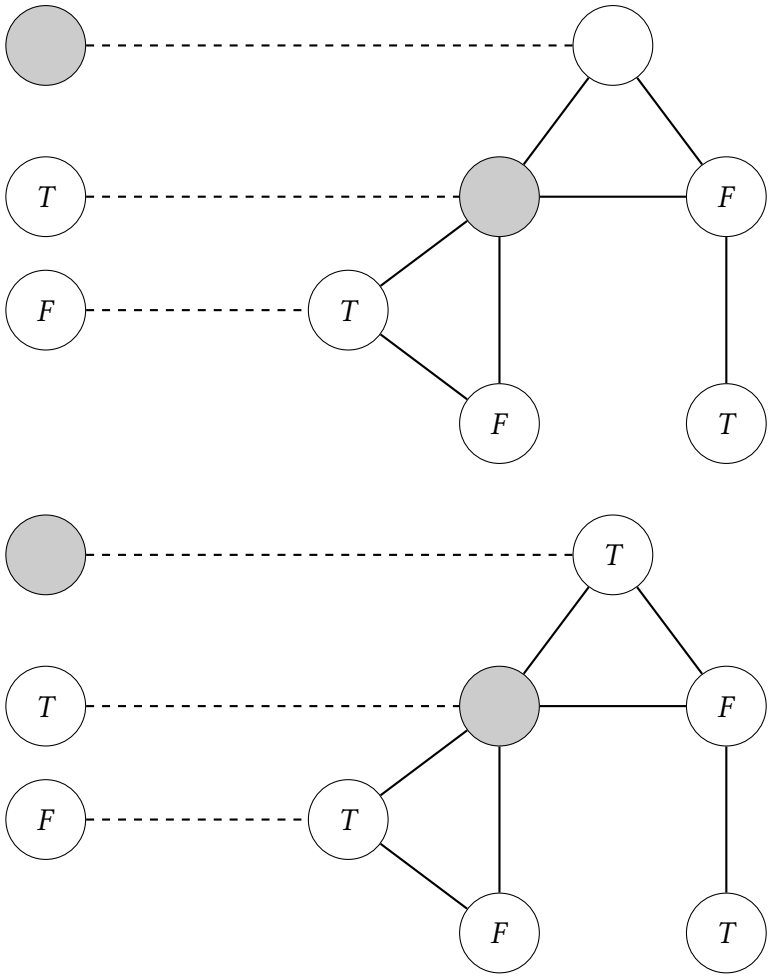




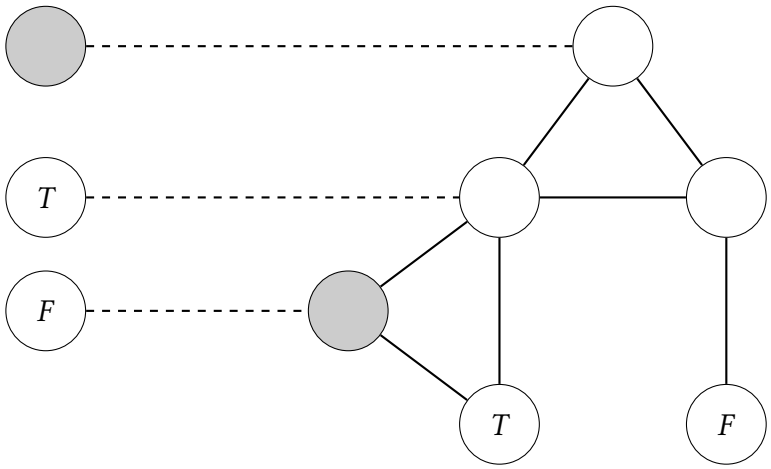
- If the input is *F T*, the colouring must be:

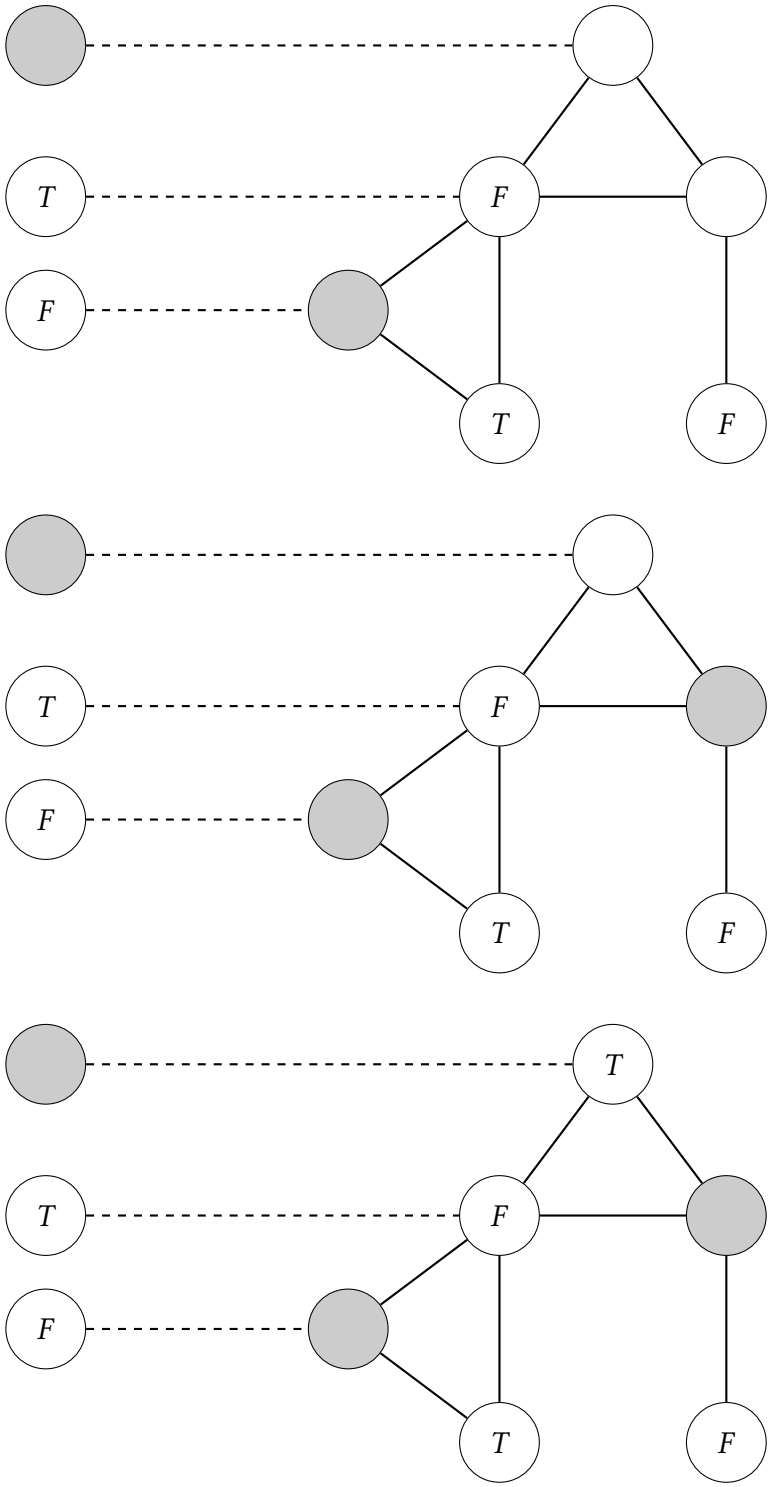




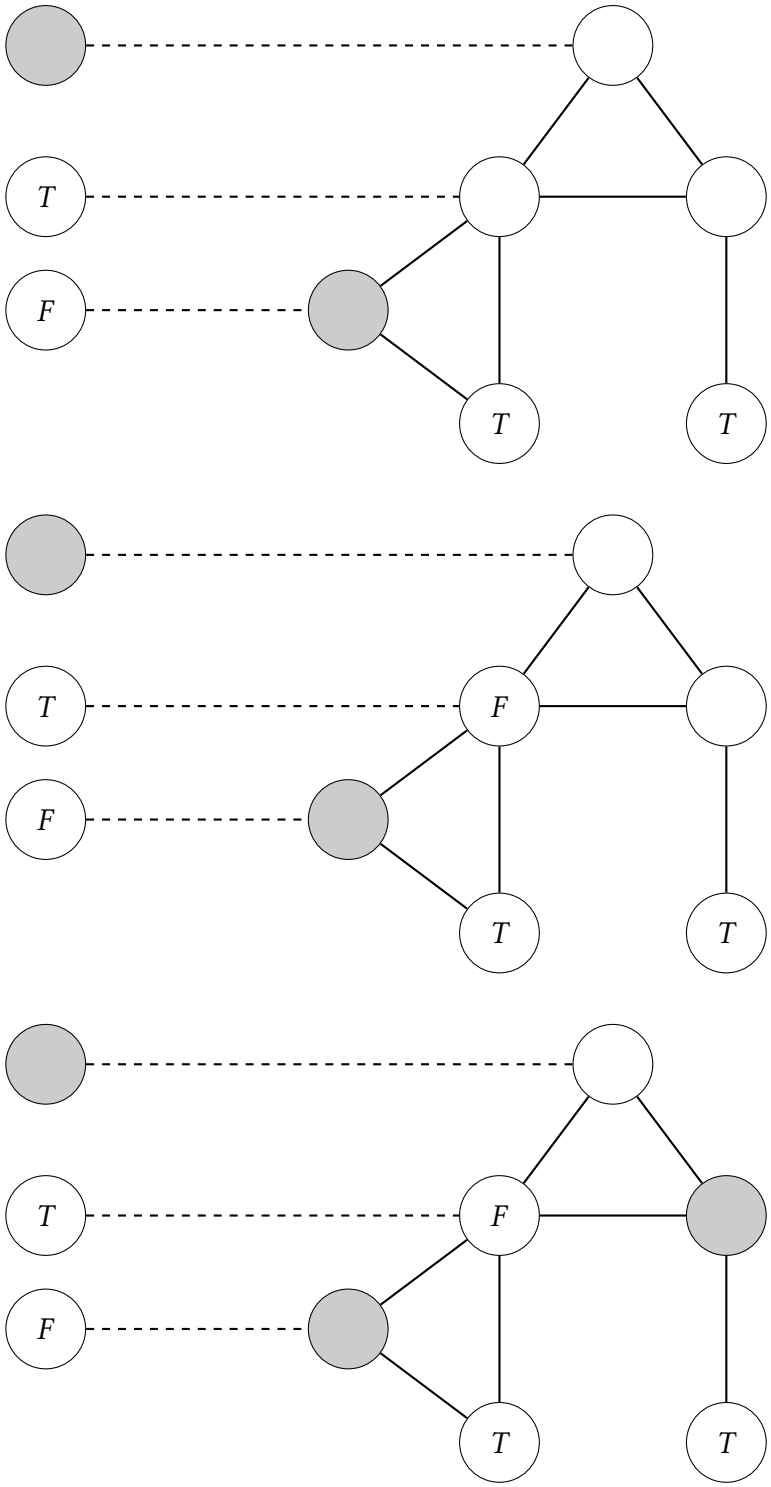


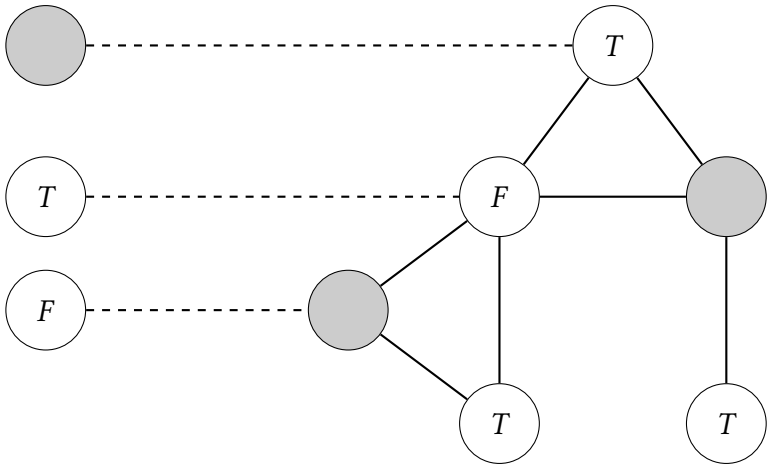
- If the input it $T\ F$, the colouring must be:





- And finally, if the input it T T, the colouring must be:


T
F
T
T
T
F
F
T
T
T
F
F
T
T



I've filled in each of these colourings one node at a time, and you can see that in every step there was only one choice for that node. So there is only one way to fill in the *or*-widget, and this method will force the top node to be the Boolean or of the inputs.

If we apply this construction to every clause, we can see that:

- Any truth assignment τ that satisfies the input ϕ will contribute one possible 3-colouring to G (times 6 — one for each arrngement of the palette)
- If τ_1 and τ_2 are distinct satsifying truth assignments for ϕ, then they will contribute 12 colourings to G: The T colour will be the one attached to the output of the clauses, and the *red* colour will be colour of the palette node attached to both of the x_i and $\neg x_i$ variable nodes. But given these colours, the colours of the variable nodes will match the τ_1 and τ_2, and so will not match. In particular, any τ that does not satisfy ϕ will contribute no 3-colourings.
- Finally, any 3-colouring of G is one of the six contributions from some τ: If we label the palette node attached to both of the x_i and $\neg x_i$ variable nodes as *red* and the colour of the clause outputs as *T*, then we can read a τ from the colours of the variable widgets. Since, by construction, the clause output colours match the logical output of the clauses in ϕ, we can see that τ is a satsifying truth assignment for ϕ.

So the total number of 3-colourings will be six times the number of satisfying assignments for ϕ.

Finally, we can see that theis constuction only adds two nodes and eight edges per clause, so the entire construction still takes $\mathcal{O}(k + \ell)$ time.

■

1.3 Sets, Partitions, and Orderings

1.3.1 #SUBSET-SUM

Definition of #SUBSET-SUM:

> **INPUT:** A (multi)set S of non-negative integers, and a number $t \in \mathbb{N}$.
>
> **QUESTION:** How many subsets $S' \subseteq S$ are there such that $\sum_{x \in S'} x = t$?
>
> *Our reduction will actually produce a set S rather than a multiset, and so this problem remains $\#\mathcal{P}$-complete even when S is a proper set.*

Reduced from: #3SAT (with a 1-1 certifacte matching).

Suppose we are given an input 3CNF ϕ. We will assume, using our helpful 3SAT lemma from the first Polytime Reductions handout that no clause of ϕ contains repeated literals.

> *We again note that this doesn't change the number of satisfying truth assignments for ϕ.*

We'll write the variables of ϕ as $x_1, \ldots, x_\ell$ and its clauses as $c_1, \ldots, c_k$.

Recall the reduction in the polytime reductions chapter, where we built a table:

	1	2	3	...	ℓ	c_1	c_2	...	c_k
x_1	1	0	0	...	0	1	0	...	0
$\neg x_1$	1	0	0	...	0	0	0	...	0
x_2	0	1	0	...	0	0	1	...	0
$\neg x_2$	0	1	0	...	0	1	0	...	0
x_3	0	0	1	...	0	1	0	...	0
$\neg x_3$	0	0	1	...	0	0	1	...	1
$\vdots$				$\vdots$				$\vdots$	
x_ℓ	0	0	0	...	1	0	0	...	1
$\neg x_\ell$	0	0	0	...	1	0	0	...	0
d_1						1	0	...	0
d'_1						1	0	...	0
d_2						0	1	...	0
d'_2						0	1	...	0
$\vdots$								$\vdots$	
d_k						0	0	...	1
d'_k						0	0	...	1
t	1	1	1	...	1	3	3	...	3

We'll build another table here. The only difference between that table and this one is in the dummy variables and in the final t. This time, our final table is:

	1	2	3	...	ℓ	c_1	c_2	...	c_k
x_1	1	0	0	...	0	1	0	...	0
$\neg x_1$	1	0	0	...	0	0	0	...	0
x_2	0	1	0	...	0	0	1	...	0
$\neg x_2$	0	1	0	...	0	1	0	...	0
x_3	0	0	1	...	0	1	0	...	0
$\neg x_3$	0	0	1	...	0	0	1	...	1
$\vdots$				$\vdots$				$\vdots$	
x_ℓ	0	0	0	...	1	0	0	...	1
$\neg x_\ell$	0	0	0	...	1	0	0	...	0
d_1						1	0	...	0
d'_1						2	0	...	0
d_2						0	1	...	0
d'_2						0	2	...	0
$\vdots$								$\vdots$	
d_k						0	0	...	1
d'_k						0	0	...	2
t	1	1	1	...	1	4	4	...	4

As before, the elements of the set S are the vaules we get by reading the rows of this table as numbers (in base 10). The t is the bottom row. Now, se can see that:

- If τ is any satisfying truth assignment for ϕ, we can build a subset S' of S by choosing the corresponding variables in our table. For each clause c_i, the sum in the column (the digit) corresponding to c_i will have a sum between 1 and 3. If the value is 1, we can bring it to 4 by adding the elements d_i and d'_i to S'. If the sum is 2, we add d'_i. If the sum is 3, we add d_i. Moreover, we can see that these are the only ways to add the values d_i and d'_i to obtain the final vaule of t. In particular, if a τ is not a satisfying assignment then at least one c_i has a sum of 0. There will be no way to add the values d_i and d'_i to S' to brinth the sum up to t.
- If the satisfying truth assignments τ_1 and τ_2 differ on variable x_1, then their corresponding subsets S'_1 and S'_2 differ on those values as well. So they contribute different subsets to our #SUBSET-SUM problem.
- If S' is a certifacte for $\langle S, t\rangle$, then for each variable x_i, S' must contain exactly one of the two elements corresponding to x_i and $\neg x_i$. So we can read a truth assignment τ from S'. Clearly, S' must be the unique subset contributed to #SUBSET-SUM problem by τ.

So there is a one-to-one correspondence between the satisfying trith assignments for ϕ and the certificate subsets of $\langle S, t\rangle$.

Finally, the total table size is $(\ell + k)^2$, which is clearly polynomial in the size of ϕ. Each entry in the table is a 0, a 1, or a 3, and each value can be found with at worst a polynomial time lookup. The string value of t, $\langle t\rangle$ is $1^\ell 3^k$, which can also be found in polynomial time. So all told, this construction takes polynomial time.

■

References

Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.