COMS20010 — Problem sheet 9

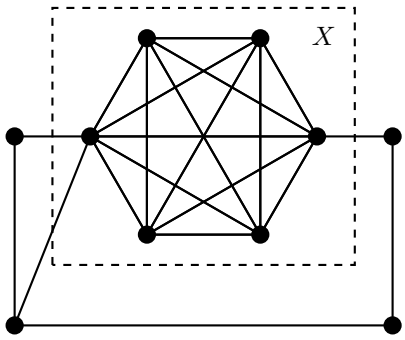
You don't have to finish the problem sheets before the class — focus on understanding the material the problem sheet is based on. If working on the problem sheet is the best way of doing that, for you, then that's what you should do, but don't be afraid to spend the time going back over the quiz and videos instead. (Then come back to the sheet when you need to revise for the exam!) I'll make solutions available shortly after the problem class. As with the Blackboard quizzes, question difficulty is denoted as follows:

- ⋆ You'll need to understand facts from the lecture notes.
- ⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in unfamiliar situations.
- ⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes and also move a bit beyond them, e.g. by seeing how to modify an algorithm.
- ⋆⋆⋆⋆ You'll need to understand and apply facts and methods from the lecture notes in a way that requires significant creativity. You should expect to spend at least 5–10 minutes thinking about the question before you see how to answer it, and maybe far longer. At most 10% of marks in the exam will be from questions set at this level.
- ⋆⋆⋆⋆⋆ These questions are harder than anything that will appear on the exam, and are intended for the strongest students to test themselves. It's worth trying them if you're interested in doing an algorithms-based project next year — whether you manage them or not, if you enjoy thinking about them then it would be a good fit.

If you think there is an error in the sheet, please post to the unit Team — if you're right then it's much better for everyone to know about the problem, and if you're wrong then there are probably other people confused about the same thing.

This problem sheet covers week 10, focusing on complexity theory.

1. [⋆⋆] A *clique* in a graph $G = (V, E)$ is a set of vertices $X \subseteq V$ in which every two vertices are joined by an edge, as in the picture below.



Prove that the following problem is NP-complete under Karp reductions: given a graph $G = (V, E)$ and an integer k, does G contain a clique of size at least k? (**Hint:** To show hardness, try reducing from IS.)

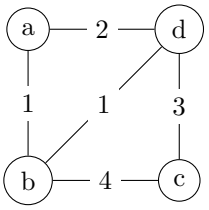
Solution: Let (G, k) be an instance of IS, writing $G = (V, E)$. As in an earlier problem sheet, we define the *complement* of G as the graph $\overline{G} = (V, E')$, where

$$E' = \{\{u, v\} : u, v \in V,\ u \neq v\} \setminus E.$$

In other words, $\overline{G}$ is the graph formed from G by taking the complement of the edge set, removing all the edges and adding all the non-edges. Then by definition, X is an independent set in G if and only if X is a clique in $\overline{G}$. Thus $(\overline{G}, k)$ is an instance of the clique decision problem whose answer is `Yes` if and only if (G, k) is a `Yes` instance of IS. Since we can construct $(\overline{G}, k)$ from (G, k) in polynomial time, we have established a Karp reduction.

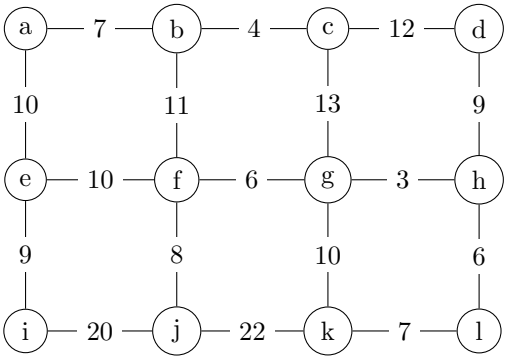
2. Suppose we are given a weighted directed graph G and an integer k, and we want to know whether or not G contains a Hamilton cycle of length at most k — this is called the Travelling Salesman Problem (TSP). Recall that a Hamilton cycle is a cycle which starts and ends at the same vertex, and visits every vertex exactly once.

   (a) [⋆⋆] Does this graph have a Hamilton cycle of length less than 5?



   Solution: No, the only Hamilton cycle in this graph is *abcda*, and this has length 1+2+3+4 = 10.

   (b) [⋆⋆] Consider the following graph.



   Now consider the Hamilton cycle *abcdhlkgfjiea*. Does this graph have a Hamilton cycle of length less than 130? In general, if we're given a Hamilton cycle for G, can we determine its length in time polynomial in the input size?

   Solution: Yes, adding up the weights of the given Hamilton cycle we get

$$7 + 4 + 12 + 9 + 6 + 7 + 10 + 6 + 8 + 20 + 9 + 10 = 108 < 130.$$

   In general, a Hamilton cycle has $|V(G)|$ edges and addition takes constant time, so we can check find the length of a given Hamilton cycle in $O(|V(G)|)$ time. The input size in bits is at least $|V(G)|$ (we need to store the vertices), so this is polynomial in the input size.

(c) [⋆⋆] What does this tell us about TSP in terms of P and NP?

> **Solution:** Given a possible solution, we can verify if it is correct in polynomial time. It's also a decision problem. It follows that this problem is in NP. However, this does not tell us anything about whether or not the problem is in P unless P = NP is resolved.

(d) [⋆⋆] Prove that TSP is NP-complete. (**Hint:** Check out question **??**b).)

> **Solution:** By **??**b), the problem of deciding whether or not a directed graph contains a Hamilton cycle is NP-complete; we give a Karp reduction from HC to TSP. Let G be the directed graph we want to detect a Hamilton cycle in. We define a weighted directed graph G' from G by setting the weight of every edge to 1; this takes polynomial time. Then every Hamilton cycle in G is a weight-$|V(G)|$ Hamilton cycle in G', so $(G', |V(G)|)$ is a TSP instance with the same answer as G and we have a valid Karp reduction.

3. [⋆⋆] You are attempting to form a committee of people to represent Computer Science in an upcoming faculty restructure. A regrettably large number of stakeholders have ideas about what this committee should look like: Denoting the stakeholders by $S_1, \ldots, S_n$, each stakeholder S_i has one list S_i^+ of people they would like to be on the committee, and another list S_i^- of people they would like not to be on the committee. (Either list may be empty.) As in real life, the committee can be arbitrarily large.

   You quickly realise that it is impossible to satisfy everyone's preferences at once, so you decide to try for something easier: you are trying to grant every person at least one of their requests. Thus for every stakeholder S_i, either you have added a person in S_i^+ to the committee, or you have *not* added a person in S_i^- to the committee, or both. For example, if everyone requested that John Lapinskas be added to the committee, then any committee containing John would be a valid committee. Sketch a proof that even so, deciding whether or not a valid committee exists given $S_1^+, \ldots, S_n^+$ and $S_1^-, \ldots, S_n^-$ is an NP-complete problem.

   > **Solution:** The problem is in NP, since given a possible committee we can check whether every stakeholder has had at least one request satisfied in polynomial time. It remains to prove that any problem in NP Karp-reduces to the committee existence problem.
   >
   > By Cook-Levin, it suffices to give a Karp reduction from SAT to the committee existence problem. Let F be an n-variable m-clause instance of SAT, and let $\ell_{i,j}$ be the j'th literal of the i'th OR clause of F; thus
   >
   > $$F = (\ell_{1,1} \vee \cdots \vee \ell_{1,k_1}) \wedge (\ell_{2,1} \vee \cdots \vee \ell_{2,k_2}) \wedge \cdots \wedge (\ell_{m,1} \vee \cdots \vee \ell_{m,k_m})$$
   >
   > for some $k_1, \ldots, k_m$. Let $x_1, \ldots, x_n$ be the variables of F. We define possible committee members $C_1, \ldots, C_n$ and stakeholders $S_1, \ldots, S_m$, associating each committee member C_i with the variable x_i. We then take S_i^+ to be the set of possible members corresponding to all literals which appear un-negated in the i'th clause, and S_i^- to be the set of possible members corresponding to literals which appear negated. For example, if the first clause is $x_1 \vee \neg x_2 \vee x_6$, then $S_1^+ = \{C_1, C_6\}$ and $S_1^- = \{C_2\}$.
   >
   > Committee assignments now correspond bijectively to assignments of values to variables; we take x_i to be true if C_i is on the committee, and vice versa. Moreover, stakeholders correspond bijectively to clauses; stakeholder S_i has at least one request satisfied if and only if the i'th clause is true. Thus valid committees correspond to valid assignments and vice versa. We have therefore given a polynomial-time map from instances of the committee existence problem to instances of SAT which preserves the right answer. This is a Karp reduction, so we're done.

4. In this question we will show that if SAT Karp-reduces to $\overline{\text{SAT}}$, then NP = Co-NP.

(a) [⋆⋆] Suppose we have an oracle for $\overline{\text{SAT}}$ (the complement of SAT). Show how we can use this to construct a Cook reduction from SAT to $\overline{\text{SAT}}$.

Solution: Consider an instance F of SAT. Then $\overline{\text{SAT}}$ with input F is a Yes instance if and only if F is unsatisfiable, i.e. if and only if SAT with input F is a No instance. Thus given an oracle for $\overline{\text{SAT}}$, we can solve $\text{SAT}(F)$ in polynomial time by running the oracle on $\overline{\text{SAT}}(F)$ and returning the opposite answer; this is a valid Cook reduction. It is not, however, a Karp reduction — for the map $F \mapsto F$ to be a Karp reduction, we would need that $\text{SAT}(F)$ and $\overline{\text{SAT}}(F)$ had the same answer.

(b) [⋆⋆] Show that if there is a Karp reduction from SAT to $\overline{\text{SAT}}$, then SAT $\in$ Co-NP and $\overline{\text{SAT}} \in$ NP.

Solution: Recall that SAT $\in$ Co-NP if and only if $\overline{\text{SAT}} \in$ NP, by definition. Thus we must construct a polynomial-time verifier algorithm V for $\overline{\text{SAT}}$, so that F is unsatisfiable if and only if $V(F, w) =$ Yes for some polynomial-sized w.

Suppose a Karp reduction from SAT to $\overline{\text{SAT}}$ exists; given an instance F of SAT, let $\phi(F)$ be the corresponding instance of $\overline{\text{SAT}}$ under this reduction. Then by the definition of a Karp reduction, $\phi(F)$ is unsatisfiable if and only if F is satisfiable; thus F is unsatisfiable if and only if $\phi(F)$ is satisfiable. We can therefore define $V(F, w)$ to compute $\phi(F)$, check whether w is a satisfying assignment for $\phi(F)$ and output the answer.

(c) [⋆⋆] Show that if there is a Karp reduction from SAT to $\overline{\text{SAT}}$, then NP = Co-NP. (**Hint:** Show that NP $\subseteq$ Co-NP and Co-NP $\subseteq$ NP.)

Solution: Suppose such a Karp reduction exists, and let f be any problem in NP; we will show $f \in$ Co-NP. We know there's a Karp reduction from f to SAT, since SAT is NP-complete, so by combining these two reductions we get a Karp reduction ϕ from f to $\overline{\text{SAT}}$. We can now use exactly the same argument from part b) to show that $\overline{f} \in$ NP and hence $f \in$ Co-NP; our verifier $V(x, w)$ for $\overline{f}$ again just checks whether w is a satisfying assignment for $\phi(F)$. Thus NP $\subseteq$ Co-NP.

Now suppose f is any problem in Co-NP; we will show $f \in$ NP. We know $\overline{f}$ is in NP by the definition of Co-NP, and hence $\overline{f}$ is also in Co-NP since we've just shown NP $\subseteq$ Co-NP. Since $\overline{f}$ is in Co-NP, f must be in NP, and we have Co-NP $\subseteq$ NP. Thus NP = Co-NP as required.

This is really the point of Karp reductions — the fact that the requirements on them are so strict allows us to use them to investigate questions like NP versus Co-NP. (As a programmer trying to solve a problem though, you're generally happy with Cook reductions.)

5. (a) [⋆⋆] Consider a set $U = \{1, \ldots, n\}$, and subsets $S_1, \ldots, S_m \subseteq U$, and a number $k \in \mathbb{Z}$. The Set Cover problem asks if there exists a collection C of at most k sets in U such that their union covers all of U. Show that VertexCover $\leq_K$ SetCover.

Solution: We begin with an instance of vertex cover, and it's corresponding graph $G = (V, E)$. In order to construct an instance of set cover, first let $U = E$. Then, define each subset S_i as the set of edges incident to vertex i in G. It's easy to see that $S_i \subseteq E$ for all i. This construction can easily be done in polynomial time.

Then, to see correctness, observe that our set cover instance is satisfied if and only if the collection of subsets includes every edge. On mapping $S_i \to i$, this is equivalent to picking at least one end point of each edge which gives us the result.

(b) [⋆⋆] The Set Packing problem is similar to the Set Cover problem, except this time we are asking if there is a subset of at *least* k sets in U such that no two sets in C intersect. Show that

IndependentSet $\leq_K$ SetPacking.

Solution: We begin with an instance on independent set and its corresponding graph $G = (V, E)$. In order to construct an instance of set packing, first let $U = E$. Then, define each subset S_i as the set of edges incident to vertex i in G. It's easy to see that $S_i \subseteq E$ for all i. This construction can easily be done in polynomial time.

Then, to see correctness, observe that our set packing instance is satisfied if and only if the collection of subsets is disjoint. Since the subsets are sets of edges, on mapping $S_i \to i$ this is equivalent to picking vertices which have no incident edges in common, i.e. vertices which are not adjacent. We note that this is precisely the definition of an independent set which gives us the result. Notice that we actually used exactly the same reduction for both Set Cover and Set Packing!

(Another, essentially equivalent, way of doing this reduction would be to take $U = V$ and the subsets to be $\{\{i\} \cup N(i) \colon i \in V(G)\}$.)

(c) [$\star\star$] Now show that IndependentSet $\leq_C$ SetCover and IndependentSet $\leq_C$ SetPacking.

Solution: Given an oracle for Set Packing and an instance (G, k) of Independent Set, we can just turn (G, k) into an instance of Set Packing with the same answer in polynomial time as described above, call our oracle for Set Packing, and then return the answer. We can do the same for Set Cover. Any Karp reduction can be turned into a Cook reduction this way.

6. In this question, we work with a variant of SAT in which variables cannot be negated. Given literals a, b and c, which need not be distinct, an *even clause* $\text{EVEN}(x, y, z)$ evaluates to `True` if and only if either zero or two of x, y and z evaluate to `True`. A *width-3 positive OR clause* is an OR clause of three variables (i.e. **un-negated** literals). A *positive even formula* is a conjunction of even clauses and width-3 positive OR clauses. For example,

$$\text{EVEN}(a, \neg b, c) \wedge \text{EVEN}(a, a, d) \wedge (a \vee b \vee e) \wedge \text{EVEN}(c, d, e).$$

is a positive even formula, but $(\neg a \vee b)$ is not due to both the negated variable and the fact that the clause only contains two variables. The decision problem POS-EVEN-SAT asks whether a positive even formula (given as the input) is satisfiable, in which case the desired output is `Yes`.

(a) [$\star\star$] Give a Karp reduction from POS-EVEN-SAT to SAT and briefly explain why it works.

Solution: Consider an instance of POS-EVEN-SAT. We convert it to CNF form in polynomial time. Any positive OR clause is already in CNF form, and we can write each even clause $\text{EVEN}(x, y, z)$ in CNF form as follows:

$$\begin{aligned}\text{EVEN}(x, y, z) &= \neg\big((x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge \neg z)\big) \\ &= (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z).\end{aligned}$$

(If any variables are repeated more than once within an EVEN clause, we omit the repeats from the corresponding OR clauses.) Then the resulting CNF formula evaluates to `True` if and only if $\text{EVEN}(x, y, z)$ does, so the corresponding SAT instance is satisfiable if and only if the original POS-EVEN-SAT instance is satisfiable, as required.

(b) [$\star\star\star$] Give a Karp reduction from 3-SAT to POS-EVEN-SAT and briefly explain why it works.

Solution: Consider an instance F of 3-SAT with variables $x_1, \ldots, x_n$; we will construct an instance F' of POS-EVEN-SAT in polynomial time. We first add a new variable t and a

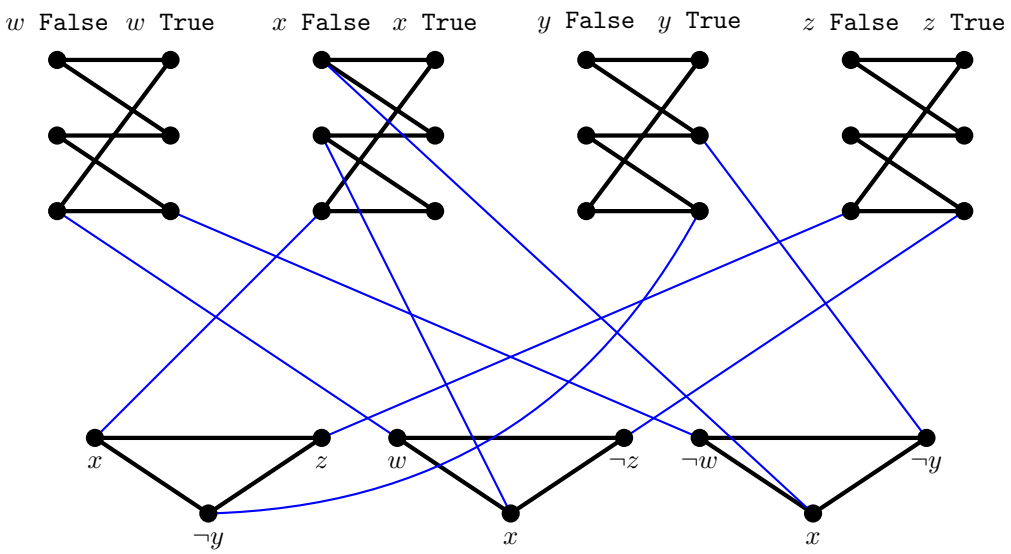
corresponding clause EVEN$(t, t, \neg t)$; observe that t must be `True` in any satisfying assignment. We further add clauses EVEN(x_i, y_i, t) for each $i \in [n]$. Since t must be `True` in any satisfying assignment, it follows that $y_i = \neg x_i$ in any satisfying assignment. Finally, we copy the OR clauses of F into F', replacing each instance of a literal $\neg x_i$ with the corresponding variable y_i. If F' is satisfiable, then $y_i = \neg x_i$ for all i, so $x_1, \ldots, x_n$ form a satisfying assignment for F. Conversely, if F is satisfiable, then we obtain a satisfying assignment for F' on taking $y_i = \neg x_i$ and $t =$ `True`. Thus F is a `Yes` instance of 3-SAT if and only if F' is a `Yes` instance of POS-EVEN-SAT, as required.

7. [⋆⋆⋆] Suppose you're working at a hardware store which is trying to put a market research group together. They serve a lot of markets, from woodworking to metalworking to painting, so they'd like their group to be diverse. More specifically, they'd like no two people in the group to have bought the same product from them within the last year. Subject to this, they'd like the group to be as large as possible. They've hired you to develop an algorithm to do this for them. They give you a list of products, a list of customers, and their sales data for the past year in the format of a two-dimensional array L, where $L[i][j]$ is the number of copies of product j bought by customer i. Their desired output is a diverse focus group which is as large as possible. Show that this problem is NP-hard under Cook reductions. (Note: There are **many** more questions like this in Chapter 8 of Kleinberg and Tardós.)

Solution: We reduce from IS. Let (G, k) be an instance of IS, where $G = (V, E)$. Form an instance of the focus group problem by taking the list of products to be E, the list of customers to be V, and the two-dimensional array to have entries given by $L[v][e] = 1$ if v is an endpoint of e and $L[v][e] = 0$ otherwise. Thus two distinct customers u and v have both bought the product e if and only if they are the two endpoints of e, and hence a set C of customers forms a diverse focus group if and only if C is an independent set in G. Our Cook reduction simply runs our focus group oracle on the instance (V, E, L), and returns `Yes` if the resulting focus group has size at least k and `No` otherwise.

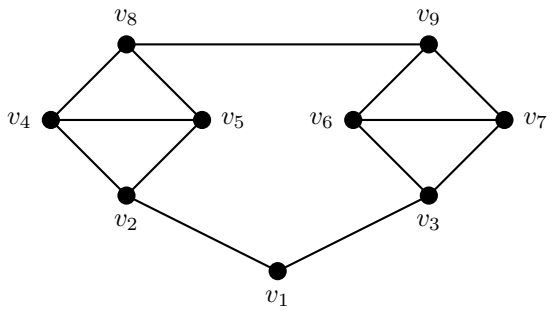
8. (a) [⋆⋆⋆] Prove that IS is NP-complete under Karp reductions even when the input graph is required to have maximum degree at most 3. **Hint:** Try to adapt the NP-completeness proof for IS given in lectures.

Solution: Membership of NP is immediate. To prove hardness, we reduce from 3-SAT. Let a be the maximum number of times any literal appears in the given 3-SAT instance. We take the reduction from 3-SAT to IS given in lectures and change the true/false variable gadgets from edges into even cycles of length $2a$, replacing all edges attached to the "true" vertex of a variable gadget with an edge attached to an even vertices on the corresponding cycle, and all edges attached to the "false" vertex of a gadget with an edge attached to an odd vertex on the corresponding cycle. Here is an example for the formula $(x \vee \neg y \vee z) \wedge (w \vee x \vee \neg z) \wedge (\neg w \vee x \vee \neg y)$, where $a = 3$:



The proof of correctness from lectures still goes through. At most $a/2$ vertices from each length-a cycle corresponding to a variable, and at most one vertex from every triangle corresponding to a clause, can be included in an independent set; thus the size of a maximum independent set is at most $\frac{a}{2} + m$, where m is the number of 3-SAT clauses. In particular, for an independent set to have size $\frac{a}{2} + m$, it must contain one vertex from each clause triangle, and either all the "`False`" vertices or all the "`True`" vertices from each variable cycle. Thus independent sets of size $\frac{a}{2} + m$ correspond to satisfying assignments as in the original gadget.

(b) [⋆⋆⋆⋆] Prove that IS is NP-complete under Karp reductions even when the input graph is required to be cubic — that is, when every vertex has degree *exactly* 3. **Hint:** You may find the following graph useful as a gadget:



Solution: Once again, membership of NP is immediate. Let H be the graph given in the question. The key properties of H are that:

(i) Every vertex in C has degree 3 except v_1, which has degree 2;

(ii) H contains a size-3 independent set including v_1 (e.g. $\{v_1, v_4, v_7\}$);

(iii) H contains a size-3 independent set not including v_1 (e.g. $\{v_2, v_3, v_8\}$);

(iv) Both of the above independent sets are maximum in H. Indeed, let X be an independent set in H. X can include at most one element of $\{v_4, v_5, v_8\}$ and one element of $\{v_6, v_7, v_9\}$. If X contains at most one element of $\{v_1, v_2, v_3\}$ as well, then $|X| \leq 3$; otherwise we must

have $v_2, v_3 \in X$, so $v_1, v_4, v_5, v_6, v_7 \notin X$, and X can contain at most one vertex of $\{v_8, v_9\}$ and again we have $|X| \leq 3$.

We now reduce from the problem of part (a): IS in which the input graph has maximum degree at most 3. Let (G, k) be an instance of this problem. We reduce to cubic IS by attaching copies of H to each vertex with degree less than 3. More explicitly, let $x_1, \ldots, x_t$ be the set of vertices in (G, k) with degree less than 3, and let $D = \sum_{i=1}^{t}(3 - d(x_i))$. We form an instance $(G', k + 3D)$ of cubic IS by, for each x_i, adding $3 - d(x_i)$ disjoint copies of H to G and joining x_i to their copy of v_1 by an edge. By (i), the resulting instance is cubic.

We claim that any size-s independent set in G corresponds to a size-$(s + 3D)$ independent set in G' and vice versa. Indeed, by properties (ii) and (iii), any size-s independent set in G can be extended to a size-$(s + 3D)$ independent set in G' by adding three vertices from each copy of H. Conversely, by property (iv), any size-$s + 3D$ independent set in G contains at most three vertices from each copy of H, so it can be cut down to a size-s independent set in G by removing $3D$ vertices (prioritising those from copies of H).

(c) [⋆⋆] Is the problem of finding a maximum independent set NP-hard when the input graph has maximum degree 2?

Solution: Probably not. In this case, all components of the input graph G must be paths or cycles (see the proof of Berge's Lemma). The maximum independent set of a k-vertex path or cycle contains $\lfloor k/2 \rfloor$ vertices, so if the components of G are $C_1, \ldots, C_r$, then a size-s independent set exists if and only if $\sum_{i=1}^{r} \lfloor |V(C_i)|/2 \rfloor \geq s$. We can check this in polynomial time with DFS or BFS, so the problem is in P. It is therefore NP-hard if and only if P = NP, which is unlikely.

9. [⋆⋆⋆] In lectures, we proved that integer linear programming was NP-hard by a long chain of reductions. Give an alternative proof via a direct Cook reduction from 3-SAT to integer linear programming. (**Hint:** The idea is similar to the reduction from vertex cover.)

Solution: This is similar to the reduction from vertex cover. Let F be an instance of 3-SAT. Write $C_1, \ldots, C_\ell$ for the OR clauses of F, so $F = C_1 \wedge \cdots \wedge C_\ell$, and $x_1, \ldots, x_n$ for the variables of F. Then we formulate an integer linear programming problem as follows:

$$c_1 + \cdots + c_\ell \to \max, \text{ subject to}$$
$$\text{for all } i,\ c_i \leq \sum_{i\colon x_i \text{ appears un-negated in } C_i} y_i + \sum_{i\colon x_i \text{ appears negated in } C_i} (1 - y_i);$$
$$y_1, \ldots, y_n, c_1, \ldots, c_\ell \geq 0;$$
$$y_1, \ldots, y_n, c_1, \ldots, c_\ell \leq 1;$$
$$y_1, \ldots, y_n, c_1, \ldots, c_\ell \in \mathbb{Z};$$

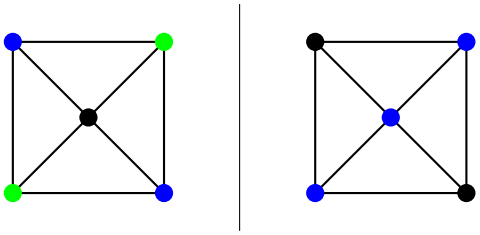
To give an example of the constraints in the second line: if $C_1 = x_1 \vee \neg x_2 \vee x_4$, then the corresponding constraint would be $c_1 \leq y_1 + (1 - y_2) + y_4$.

Assignments of truth values to variables in F now correspond bijectively to assignments of integer values to the variables $y_1, \ldots, y_n$, with an assignment $x_i = \texttt{True}$ corresponding to an assignment of $y_i = 1$ and an assignment of $x_i = \texttt{False}$ corresponding to an assignment of $y_i = 0$. Since the c_i's are bounded above by 1, an optimal solution will have value at most ℓ, with equality if and only if

we can take $c_i = 1$ for all i. Under our correspondence, for given values of $y_1, \ldots, y_n$, we can take $c_i = 1$ if and only if some literal in the clause C_i is `True`. Thus the optimal solution has value ℓ if and only if there's a satisfying assignment for F.

Our reduction therefore constructs the linear programming problem as above, invokes our oracle, and returns `Yes` if the resulting solution has value ℓ and `No` otherwise.

10. For all positive integers k, a *k-colouring* of a graph $G = (V, E)$ is a map $c\colon V \to C$, where C is a set of *colours* with $|C| = k$. We say c is *proper* if $c(u) \neq c(v)$ whenever $\{u, v\} \in E$, i.e. no two adjacent vertices receive the same colour. Two colourings of the same graph, with $C \subseteq \{\text{green}, \text{blue}, \text{black}\}$, are shown below. The 3-colouring on the left is proper, but the 2-colouring on the right is not (because the blue center vertex is adjacent to blue vertices at the corners).



This question is concerned with the following **family** of decision problems. Let k be a positive integer. Then the k-colourability problem asks: given a graph G, does G have a proper k-colouring? Note that k is not part of the input, but part of the problem statement.

(a) [$\star\star$] Give a polynomial-time algorithm for the 2-colourability problem.

Solution: The key point here is that every connected graph has at most two proper 2-colourings. Indeed, for concreteness let us take $C = \{\text{blue}, \text{green}\}$. Let G be a connected graph, and suppose $c\colon V \to C$ is a proper 2-colouring. If $v \in V(G)$ is coloured green, then every vertex in $N(v)$ must be coloured blue. Every vertex in $N(N(v))$ must then be coloured green, every vertex in $N(N(N(v)))$ must be coloured blue, and so on. Since G is connected, this process must eventually terminate, forcing the colour of every vertex in G. Likewise, if $c(v) = \text{blue}$, then every vertex in $N(v)$ must be green and so on, and again the entirety of c is forced.

Using this, one polynomial-time algorithm would choose an arbitrary vertex v_1, colour it blue, then run breadth-first search from v_1. Every time we encounter a new vertex, we colour it greedily: if it has a blue neighbour then we colour it green; if it has a green neighbour then we colour it blue; and if it has both blue and green neighbours then we return `No`. This will give a valid 2-colouring of the component containing v_1, if one exists. If there are no uncoloured vertices left in the graph, we return `Yes`. Otherwise, we choose an uncoloured vertex v_2, colour it blue, and repeat the process.

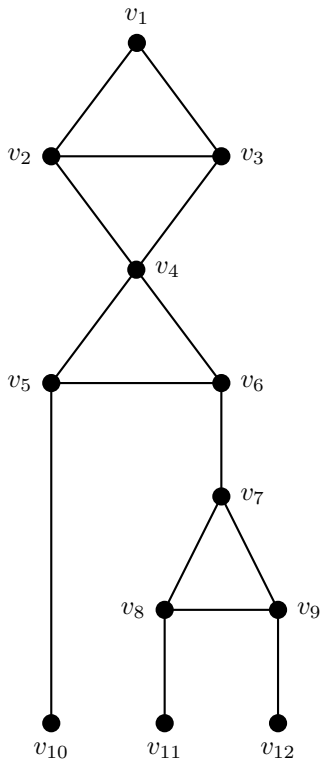
(b) [$\star\star\star$] Next, for all $k > 3$, give a Karp reduction from the 3-colourability problem to the k-colourability problem.

Solution: Let $G = (V, E)$ be an instance of the 3-colourability problem for some $k \geq 4$. We form an instance G' of the k-colourability problem by adding new vertices $x_1, \ldots, x_{k-3}$ joined to each other and every vertex in G by edges. Thus we have $G' = (V', E')$, where

$$V' = V \cup \{x_1, \ldots, x_{k-3}\},$$
$$E' = E \cup \{\{x_i, x_j\}\colon i, j \in [k-3], i \neq j\} \cup \{\{x_i, v\}\colon i \in [k-3], v \in V\}.$$

Any proper k-colouring of G' must colour $x_1, \ldots, x_{k-3}$ with $k-3$ distinct colours, and the whole of V using only the three remaining colours; thus if G' is a `Yes` instance of k-colourability then G is a `Yes` instance of 3-colourability. Conversely, any proper 3-colouring of G can be extended to a proper k-colouring of G' by colouring $x_1, \ldots, x_{k-3}$ arbitrarily with the $k-3$ remaining colours; thus if G is a `Yes` instance of 3-colourability, then G' is a `Yes` instance of k-colourability. We can form G' from G in polynomial time, so this is a valid Karp reduction.

(c) [⋆⋆] In fact, 3-colourability is NP-hard under Karp reductions. One way of proving it is to reduce directly from 3-SAT. A crucial gadget in the standard reduction is as follows:



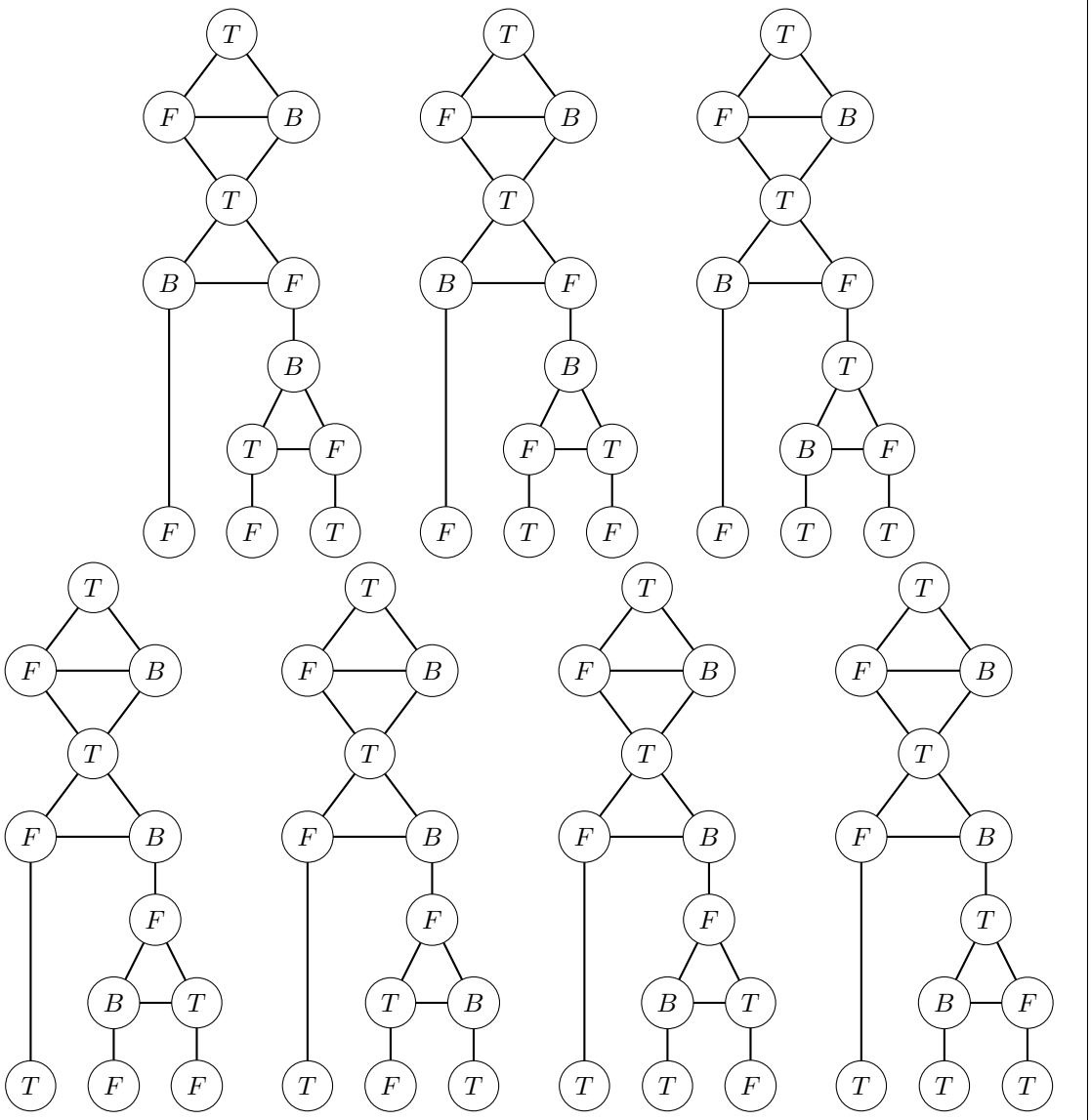
Note that in any proper 3-colouring, the vertices v_1, v_2 and v_3 must receive different colours. Write T for the colour of v_1, F for the colour of v_2, and B for the colour of v_3. Prove that the gadget has the following properties:

- In any proper 3-colouring of the gadget, the vertices v_{10}, v_{11} and v_{12} cannot all be coloured F.
- Any other colouring of v_{10}, v_{11} and v_{12} using only colours T or F can be extended to a proper colouring of the entire gadget.

Hint: There are two ways of proving the second part — a long-but-simple way and a short-but-clever way.

Solution: Suppose for a contradiction that there is a proper 3-colouring in which v_1 is coloured T, v_2 is coloured F, v_3 is coloured B, and v_{10}, v_{11} and v_{12} are all coloured F. Then v_4 must be coloured T; hence v_5 must be coloured B; hence v_6 must be coloured F. Now every vertex in $\{v_7, v_8, v_9\}$ has a neighbour with colour F, so they cannot be coloured F, but they must all receive distinct colours. This is impossible.

The long-but-simple way of proving the second part is to just write down the colourings:



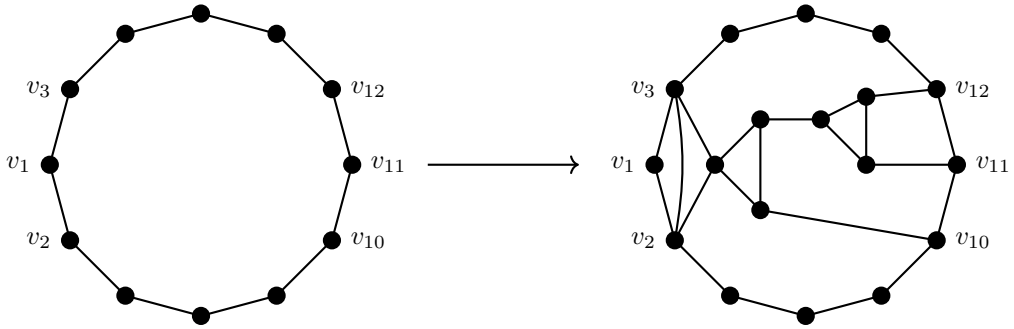
Here's the short-but-clever way. In making a proper colouring, we know v_4 has to be coloured T, since v_2 is coloured F and v_3 is coloured B. If v_{10} is coloured T, then we can colour greedily in the order v_8, v_9, v_7, v_6, v_5 (because in doing so we never colour a vertex that could be adjacent to one vertex each of colours T, B and F), so suppose v_{10} is coloured F. If v_{11} is coloured T, then we can colour greedily in the order v_5, v_6, v_7, v_9, v_8 by the same argument, and if v_{12} is coloured T then we can colour greedily in the order v_5, v_6, v_7, v_8, v_9. So if any of v_{10}, v_{11} and v_{12} are coloured T then we have a proper colouring as required.

(d) [⋆⋆⋆⋆] Using the gadget from the previous part, or otherwise, prove that 3-colourability is NP-complete under Karp reductions.

Solution: Any `Yes` instance of 3-colourability has an associated 3-colouring, which can be verified in polynomial time, so the problem is a member of NP. It remains to prove NP-hardness under Karp reductions.

The key insight here is that the gadget above can be used to add a constraint to colourings

in an existing graph. Suppose we take an existing graph and layer a copy of the gadget on top, adding new vertices for $v_4, \ldots, v_9$ but identifying v_1, v_2, v_3, v_{10}, v_{11} and v_{12} with existing vertices (as in the picture below). Then it has the effect of forbidding all 3-colourings in which v_{10}, v_{11} and v_{12} receive colour F, while allowing all 3-colourings in which they receive some other combination of F and T, as part 3c implies these can be extended to proper 3-colourings of the entire gadget. This is a common design pattern in gadget construction; the vertices v_1, v_2, v_3, v_{10}, v_{11} and v_{12} are colloquially known as the *terminals* of the gadget.



We will use this property to simulate our clauses. Let F be a width-3 CNF formula with variables $x_1, \ldots, x_n$. To form our instance G of the 3-colourability problem, we proceed as follows:

- Add a triangle with new vertices B, T and F.
- For each variable x_i, add vertices X_i and $\neg X_i$, and add the triangle $X_i(\neg X_i)B$ to the edge set.
- For each clause C_i, add a copy of the gadget above, identifying v_1 with T, v_2 with F, v_3 with B, and v_{10}, v_{11} and v_{12} with the vertices corresponding to the literals in C_i; for example, if the clause reads $x_2 \wedge x_3 \wedge \neg x_5$, then identify v_{10}, v_{11}, and v_{12} with X_2, X_3 and $\neg X_5$. Thus for each C_i we have added only six new vertices (corresponding to $v_4, \ldots, v_9$).

In the graph we have constructed after the second bullet point, we see that proper 3-colourings correspond almost bijectively to assignments of values to variables. Indeed, in any proper 3-colouring, write B for the colour of B, T for the colour of T, and F for the colour of F. Then each vertex X_i must receive one colour from $\{T, F\}$, and each vertex $\neg X_i$ must receive the other; this corresponds to the assignment which maps x_i to `True` if X_i receives colour T and $\neg X_i$ receives colour F, and to `False` if X_i receives colour F and $\neg X_i$ receives colour T. This correspondence is a bijection up to the choice of colours for B, F and T.

Under this lens, then, the effect of adding the gadget corresponding to clause C_i is to forbid assignments in which C_i is false, and only those assignments. Indeed, by part 3c, any proper 3-colouring of the graph can be extended into a proper 3-colouring of the gadget unless v_{10}, v_{11} and v_{12} all recieve colour F — in other words, unless their corresponding literals all evaluate to `False`. Thus any 3-colouring of our graph must correspond to an assignment in which all clauses are true, i.e. a satisfying assignment, and vice versa. Since we constructed G in polynomial time, we have given a valid Karp reduction as required. A more detailed version of this argument is contained in Chapter 8.7 of Kleinberg and Tardós' book *Algorithm Design*.

11. (a) [★★★★] Prove that the problem of deciding whether or not a **directed** graph contains a (directed) Hamilton cycle is NP-complete under Karp reductions. **Hint:** One valid approach involves replacing variables by path gadgets with edges in both directions, where travelling one direction on the path corresponds to an assignment of true and travelling the other direction corresponds to an assignment of false, and replacing clauses with individual vertices.

Solution: Given a directed graph G and a bit string w, it is easy to decide whether w represents a directed Hamilton cycle of G in polynomial time, so the problem is a member of NP. It remains to prove NP-hardness.

Let F be an instance of 3-SAT. Let $x_1, \ldots, x_n$ be the variables it contains (in some arbitrary order). Write $F = C_1 \wedge C_2 \wedge \cdots \wedge C_k$, where each C_i is a disjunctive clause; and write each $C_i = c_{i,1} \vee \cdots \vee c_{i,3}$, where each $c_{i,j}$ is either $x_{k_{i,j}}$ or $\neg x_{k_{i,j}}$ for some variable $x_{k_{i,j}}$. We construct an instance G of the directed Hamilton cycle problem as follows.

For each variable x_i of F, add a bidirectional $(3k+2)$-edge path $P_i = p_0 p_1 \ldots p_{3k}$ to G, which we write as $p_{i,1} \ldots p_{i,k+1}$. Add directed edges from both endpoints of each P_i to both endpoints of P_{i+1} for all $i \in [n-1]$, and add directed edges from both endpoints of P_n to both endpoints of P_1. For each clause C_j, add a vertex y_j to G. If x_i appears in C_j as an un-negated variable, then add the edges $(p_{i,3j+1}, y_j)$ and $(y_j, p_{i,3j+2})$ to G. If x_i appears in C_j as a negated variable, then add the edges $(p_{i,3j+2}, y_j)$ and $(y_j, p_{i,3j+1})$ to G.

Any assignment S of F corresponds to a unique cycle W in G as follows. Let $\overline{P}_i$ be the path formed by reversing P_i. Let W be the (non-Hamilton) cycle formed by concatenating the paths $Q_1 \ldots Q_k$, where $Q_i = P_i$ if x_i is true under S, and $Q_i = \overline{P}_i$ if x_i is false under S. Then W covers every vertex except $y_1, \ldots, y_k$.

If in addition S is satisfying, then W can be extended into a Hamilton cycle W' as follows. Each clause C_i must contain some true literal $c_{i,j}$, with corresponding variable x_j. If x_j is true under S, replace the edge $(p_{j,3i+1}, p_{k,3i+2})$ by the pair of edges $(p_{j,3i+1}, y_i)$ and $(y_i, p_{j,3i+2})$ in W'; otherwise, replace the edge $(p_{j,3i+2}, p_{k,3i+1})$ by the pair of edges $(p_{j,3i+2}, y_i)$ and $(y_i, p_{j,3i+1})$ in W. Then W is a Hamilton cycle.

Moreover, it can be shown that every Hamilton cycle in G is necessarily of this form. The key facts are:

- If a Hamilton cycle enters a clause vertex y_i from a path P_j, then it must immediately return to P_j.
- If a Hamilton cycle enters a path P_i going in one direction, then it must continue in that direction with occasional detours to clause vertices.

G can be computed from F in polynomial time, so we have given a Karp reduction as required.

(b) [⋆⋆⋆] Prove that the problem of deciding whether or not an **undirected** graph contains a Hamilton cycle is NP-complete under Karp reductions. **Hint:** One valid approach uses small vertex gadgets.

Solution: As before, given a graph G and a bit string w, it is easy to decide whether w represents a Hamilton cycle of G in polynomial time, so the problem is a member of NP. It remains to prove hardness.

Let $G = (V, E)$ be a directed n-vertex graph. We define a corresponding instance $f(G) = (V', E')$ of UndirHC as follows. We take V' to be a set of $3|V|$ vertices which we denote by $V' = \bigcup_{v \in V} \{v_1, v_2, v_3\}$. We define E' from E as follows:

- join each vertex $v_1 \in V'$ to v_2 and v_3;
- for each directed edge $(u, v) \in E$, join u_2 to v_3.

We can certainly compute $f(G)$ in polynomial time, so it suffices to prove that $f(G)$ contains a Hamilton cycle if and only if G does. This holds because the edge (u, v) is present in G if and only if the path $u_1 u_2 v_3 v_1$ is present in $f(G)$. So if G contains a Hamilton cycle $v^1 \ldots v^n$, then $v_1^1 v_2^1 v_3^2 v_1^2 v_2^2 v_3^3 v_1^3 \ldots v_1^n v_2^n v_3^1$ is a Hamilton cycle in $f(G)$. Conversely, if $f(G)$ contains a

Hamilton cycle, then it must be divided into subpaths of the form $u_1u_2v_3v_1$, corresponding to a Hamilton cycle in G. Indeed, u_1 has degree 2, so it must send an edge to u_2 within the cycle; u_2 only sends edges to u_1 and vertices of the form v_3, so it must send its second cycle edge to some v_3; and v_3 is adjacent to v_2, which has degree 2, so v_3 must send its second cycle edge to v_2 (or v_2 cannot be included in the cycle).