COMS20010 — Mock Exam

Short answer questions

1. (5 marks) (Short question.) Consider the **unoptimised** version of the union-find data structure on a 1000-element set, in which a sequence of n operations takes $\Theta(n \log n)$ time in the worst case. Suppose this data structure is freshly-initialised, so that which each element is currently in its own (singleton) set. What is the least number of `Union` operations that could result in a component of the data structure having depth 4?

   A. 15.

   B. 16.

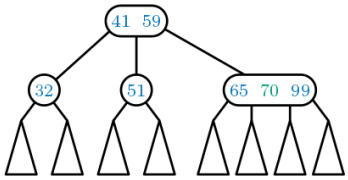
   C. 31.

   D. 32.

   E. None of the above.

> **Solution: A**. Remember, as covered in video 6-2, the depth can only increase when merging two elements of equal depth. To achieve depth 1 we need 1 `Union` operation. To achieve depth 2 we need to merge two trees of depth 2, which takes $2 \cdot 1 + 1 = 3$ `Union` operations. To achieve depth 3 we need to merge two trees of depth 3, which takes $2 \cdot 1 + 1 = 7$ `Union` operations. And likewise, to achieve depth 4 we need $2 \cdot 7 + 1 = 15$ `Union` operations.

2. (5 marks) (Short question.) Consider the following 2-3-4 tree. What will the root contain and how many children will it have after we insert 72 into the tree?



   A. The root will contain $41, 59$ and will have three children.

   B. The root will contain $41, 59, 99$ and will have four children.

   C. The root will contain $59, 70$ and will have three children.

   D. The root will contain $41, 59, 65$ and will have three children.

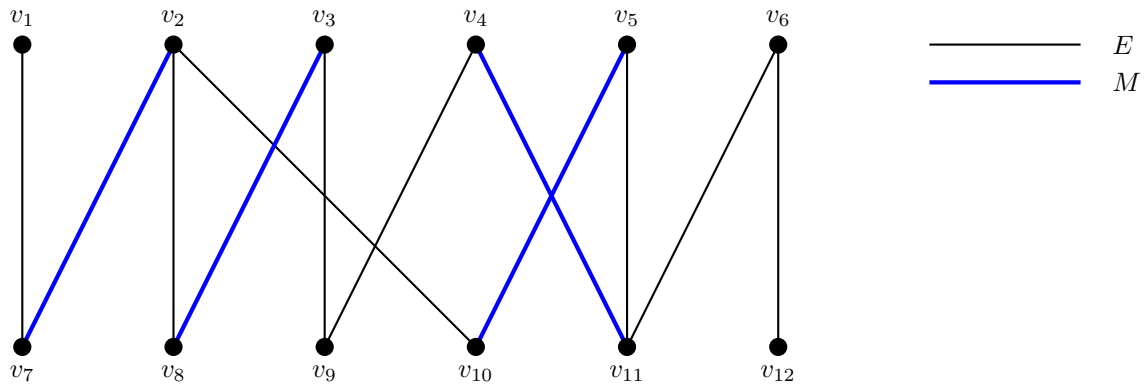
   E. The root will contain $41, 59, 70$ and will have four children.

> **Solution: E**. The insert operation will take the rightmost child from the room, since $72 > 59$. This will take it to $(65, 70, 99)$, a 4-node, which it will split. This sends 70 up to the root, with left child (65) and right child (99). The insertion operation will then descent to (65) and keep going. (Note that the insertion operation does not need to split the root, since not doing so can never result in hitting a 4-node with a 4-node parent on the way down to insert 72. But I wouldn't have listed "The root will contain 59 and have two children" as an option, either.)

3. (5 marks) (Short question.) Mark each of the following statements true or false.
   (a) (1 mark) $40^n \in o(20^n)$.
   (b) (1 mark) $40^n \in O(20^n)$.
   (c) (1 mark) $40^n \in \Theta(20^n)$.
   (d) (1 mark) $40^n \in \Omega(20^n)$.
   (e) (1 mark) $40^n \in \omega(20^n)$.

Solution: A, B and C are false, D and E are true. Remember, we have $40^n = 20^n \cdot 2^n$, and $2^n \in \omega(1)$, so we must have $40^n \in \omega(20^n)$. This immediately implies $40^n \in \Omega(20^n)$, $40^n \notin \Theta(20^n)$, $40^n \notin O(20^n)$ and $40^n \notin o(20^n)$.

4. (5 marks) (Short question.) Consider the bipartite graph $G = (V, E)$ and indicated matching M below.



Which of the following is **not** an augmenting path?

   A. $v_1v_7v_2v_8v_3v_9$.
   B. $v_9v_4v_{11}v_5v_{10}v_2v_7$.
   C. v_6v_{12}.
   D. $v_9v_4v_{11}v_6$.
   E. Either they are all augmenting paths, or more than one of them is not an augmenting path.

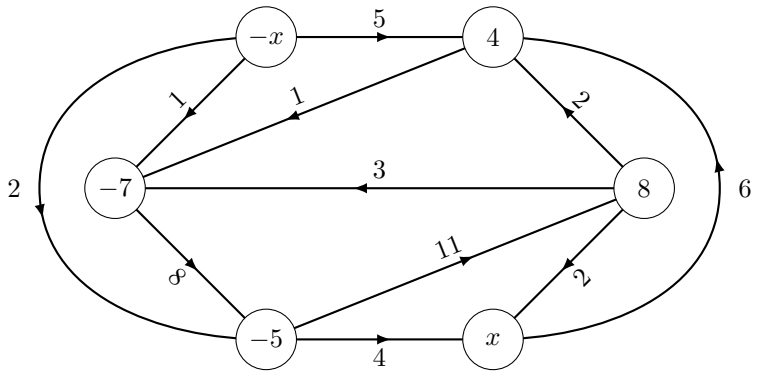
Solution: B. For a path to be alternating, it must begin and end with unmatched vertices, and alternate between matching and non-matching edges. A, C and D achieve this, but B ends with a matched vertex.

5. (5 marks) (Short question.) Which of the following logical formulae is **not** in conjunctive normal form (CNF)?

   A. $(a \vee b \vee c) \wedge (c \vee \neg d)$.
   B. $a \wedge b$.
   C. $a \vee b$.
   D. $(a \vee \neg b) \wedge \neg(c \vee d)$.
   E. Either they are all in CNF, or more than one of them is not in CNF.

Solution: D. A CNF formula is an AND of OR clauses. Both AND and OR clauses can have length 1, as in B and C, but the clauses themselves cannot be negated as in D — only the variables they contain.

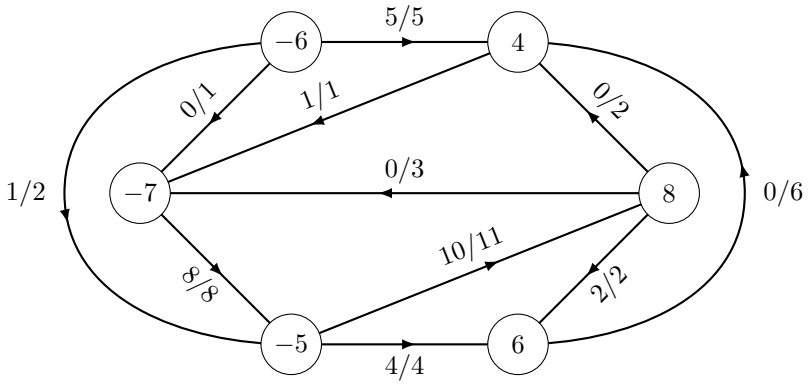
6. (5 marks) (Short question.) Consider the following circulation network.



What is the largest integer value of x such that the network has a valid circulation?

A. 5.
B. 6.
C. 7.
D. There's no valid circulation for any integer value of x.
E. None of the above.

Solution: B. A valid circulation with $x = 6$ is pictured below.



(The process of finding a valid circulation is covered in video 9-2.) The vertex v with demand x has total inward capacity $c^-(v) = 6$, so there is no circulation with $x = 7$ (as this would require too much flow into v).

7. (5 marks) (Short question.) Consider an instance of weighted interval scheduling with interval set $\mathcal{R} = [(1,3),(2,5),(4,9),(0,9),(7,10),(8,10),(11,13),(9,14),(12,15)]$ and weight function w given by

$$w(1,3) = 6, \quad w(2,5) = 6, \quad w(4,9) = 4, \quad w(0,9) = 10, \quad w(7,10) = 3,$$
$$w(8,10) = 3, \quad w(11,13) = 8, \quad w(9,14) = 9, \quad w(12,15) = 4.$$

What is the maximum possible weight of a compatible set?

A. 19.

B. 20.

C. 21.

D. 22.

E. None of the above.

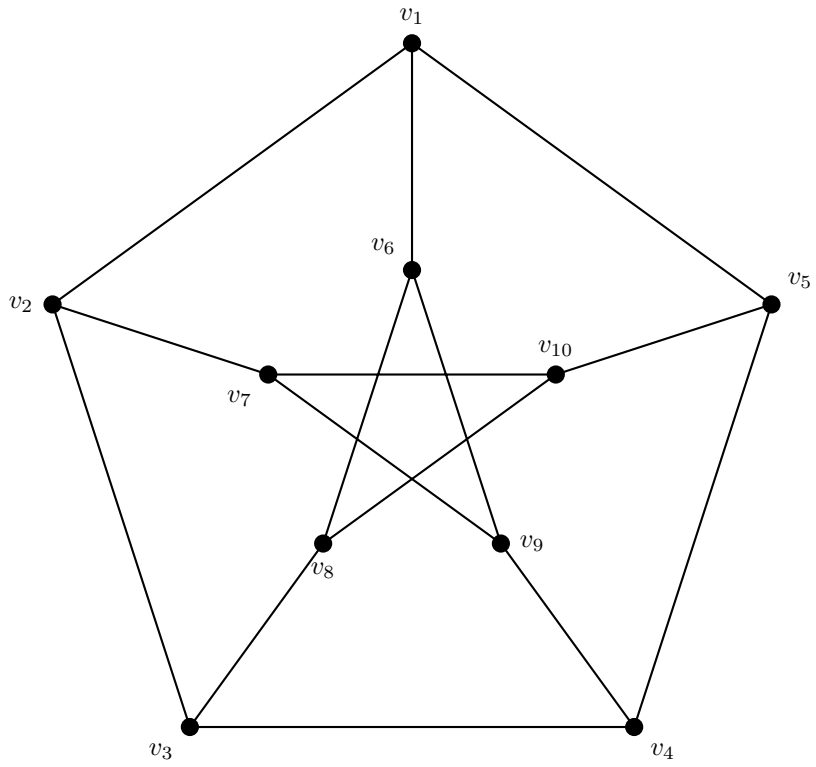
Solution: **A.** Two optimal sets are $\{(0,9),(9,14)\}$ and $\{(1,3),(4,9),(9,14)\}$. In more detail, writing $\mathrm{OPT}(i)$ for the weight of an optimal solution starting from the ith interval sorted in ascending order of finish time, is

$$\mathrm{OPT}(1) = 19, \quad \mathrm{OPT}(2) = 19, \quad \mathrm{OPT}(3) = 19, \quad \mathrm{OPT}(4) = 19, \quad \mathrm{OPT}(5) = 12,$$
$$\mathrm{OPT}(6) = 12, \quad \mathrm{OPT}(7) = 9, \quad \mathrm{OPT}(8) = 9, \quad \mathrm{OPT}(9) = 4.$$

8. (5 marks) (Short question.) Mark the following five statements true or false.
   (a) (1 mark) In all directed graphs $G = (V, E)$, we have $|E| = \sum_{v \in V} d^+(v)$.
   (b) (1 mark) All forests are connected.
   (c) (1 mark) All induced subgraphs of trees are trees.
   (d) (1 mark) Any tree with at least two vertices has at least two leaves.
   (e) (1 mark) Any tree with n vertices has $n - 1$ edges.

Solution: True, false, false, true, true. Statement 1 is the directed form of the handshaking lemma. A forest is any graph without cycles — an example of a disconnected forest is $V = [2]$, $E = \emptyset$. An induced subgraph of a tree need not be connected — for example, taking $V = [3]$, $E = \{\{1,2\},\{2,3\}\}$ and $X = \{1,3\}$, then $G = (V, E)$ is a tree, but $G[X]$ is not connected. Any tree with at least two vertices does have at least two leaves, and this was covered in lectures. Any tree with n vertices has exactly $n - 1$ leaves by the fundamental lemma of trees.

9. (5 marks) (Short question.) Consider the graph below.



Are each of the following sets vertex covers or not?

(a) (1 mark) $\{v_i : i \in [10]\}$;

(b) (1 mark) $\{v_1, v_3, v_7, v_{10}\}$.

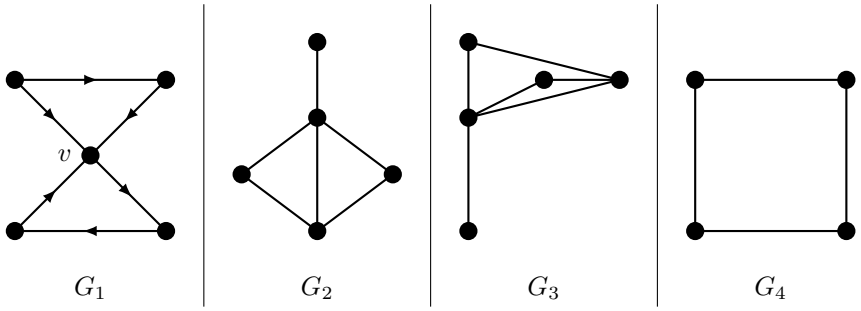
(c) (1 mark) $\{v_i : i \in [5]\}$;

(d) (1 mark) $\{v_1, v_3, v_4, v_7, v_8, v_{10}\}$.

(e) (1 mark) $\{v_2, v_4, v_5, v_6, v_7, v_8\}$.

Solution: Yes, no, no, no, yes. A vertex cover is a set of vertices intersecting every edge in the graph. (b) does not intersect e.g. $\{v_4, v_5\}$. (c) does not intersect e.g. $\{v_7, v_{10}\}$. (d) does not intersect $\{v_6, v_9\}$.

10. (5 marks) (Short question.) We define directed and undirected graphs G_1, G_2, G_3 and G_4 as in the picture below.



Mark each statement true or false.

(a) (1 mark) The in-degree of v in G_1 is 3.

(b) (1 mark) G_1 is weakly connected.

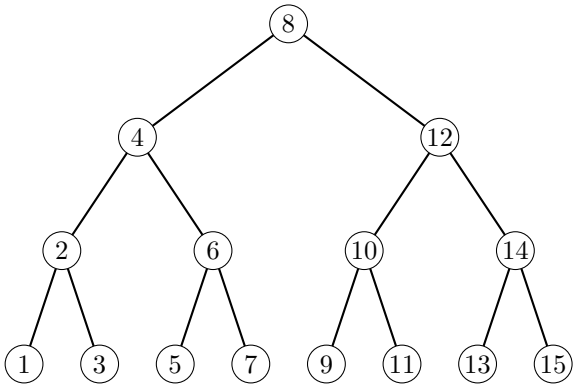
(c) (1 mark) G_2 and G_3 are isomorphic.

(d) (1 mark) Some induced subgraph of G_3 is isomorphic to G_4.

(e) (1 mark) G_2 and G_4 each contain an Euler walk.

Solution: True, true, true, false, true. These all follow immediately from the definitions of the terms.

11. (5 marks) (Medium question.) Consider the effects of deleting 4 from the 2-3-4 tree below.



After deleting 4:

(a) (1 mark) How many 2-nodes are in the tree?

(b) (1 mark) How many 3-nodes are in the tree?

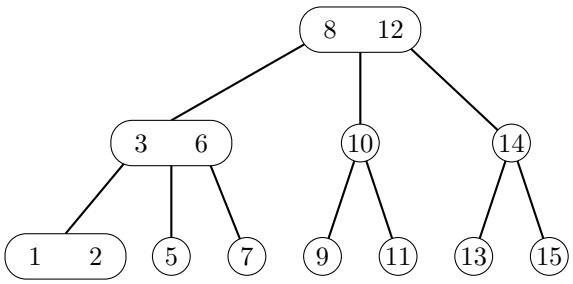
(c) (1 mark) How many 4-nodes are in the tree?

(d) (1 mark) How many nodes are in the bottom layer of the tree?

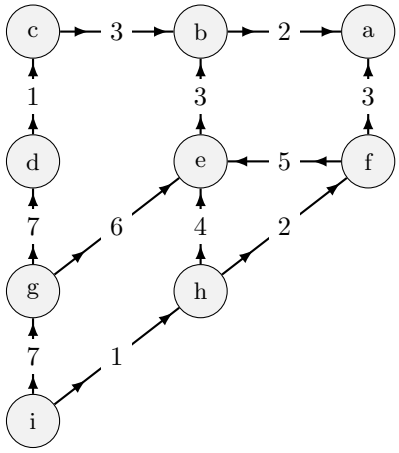
(e) (1 mark) How many total fuse, split and transfer operations occurred in the process of deleting 4?

This question is multiple choice. The available options for each part are 0, 1, 2, 3, 7, 8, 9, and "none of the above".

Solution: 8, 3, 0, 7, 3. The deletion algorithm traverses first to 4, then to the predecessor 3 of 4, fusing two 2-nodes encountered on the way (including the root). It then deletes 3, triggering another fuse operation, and overwrites 4 with 3. The resulting 2-3-4 tree is shown below.



12. (5 marks) (Medium question.) Suppose Dijkstra's algorithm is run on the following graph starting at vertex i. List the vertices in the order the algorithm performs final processing on them (i.e. the order in which their distance from i is finalised).



A. i, h, b, g, e, a, d, f, c

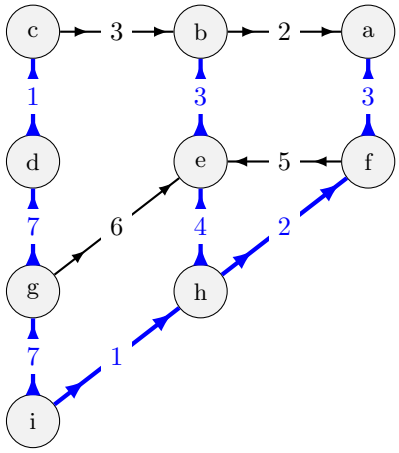
B. i, h, f, e, b, a, g, d, c

C. i, h, f, e, a, g, b, c, d

D. i, h, g, e, a, b, f, d, c

E. i, h, f, e, a, g, b, d, c

Solution: E. See Video 4-4 for a refresher on how Dijkstra's algorithm works. The shortest-path tree is given below.



13. (10 marks) (Medium question.) Suppose that x, y, and z are strings. We say that z is a "shuffle" of x and y if z can be obtained by mixing the characters from x and y in a way that preserves the left-to-right ordering of the characters from x and the characters from y. For example, "LOhamLbuCrgerAT" is a shuffle of "hamburger" and "LOLCAT". Given three strings x, y, and z, you wish to tell whether or not z is a shuffle of x and y.

(a) (8 marks) Fill in the blanks in the incomplete dynamic programming algorithm below. The return value of SHUFFLED(x, y, z) should be `True` if z is a shuffle of x and y, and `False` if it is not.

```
1  begin
2      Let n ← length(x).
3      Let m ← length(y).
4      Let r ← length(z).
5      if n = 0 then
6          If y == z return ____, else return ____.
7      else if m = 0 then
8          If x == z return ____, else return ____.
9      else
10         Let x⁻ ← x[1, ..., n − 1].
11         Let y⁻ ← y[1, ..., m − 1].
12         Let z⁻ ← z[1, ..., r − 1].
13         Return
               ((z[r−1] == x[n−1]) ∧ SHUFFLED(_, _, z1)) ∨ ((z[r−1] == y[m−1]) ∧ SHUFFLED(_, _, z1))
```

Each blank should contain `True`, `False`, x, y, z, x_1, y_1 or z_1. Some terms may appear more than once or not at all.

Solution: In order: True, False, True, False, x^-, y, x, y^-. We break the problem down as a sequence of choices: if z is a shuffle of x and y, then which of x and y does the last character of z come from? The base case, covered by lines 5–9, is that any word z is a shuffle of z and the empty string. x^-, y^- and z^- are x, y and z with their last characters removed, respectively. z can be expressed as a shuffle of x and y whose last character comes from x if and only if the last characters of z and x match, and z^- is a shuffle of x^- and y. Likewise, z can be expressed as a shuffle of x and y whose last character comes from y if and only if the last characters of z and y match, and z^- is a shuffle of x and y^-. This gives rise to the recurrence relation of lines 9–13.

(b) (2 marks) What is the running time of SHUFFLED, if properly memoised? Choose the strongest bound that applies.

A. $O(m + n)$
B. $O(n^2)$.
C. $O(mn)$
D. $O(mnr)$.
E. None of the above.

Solution: C. The only possible recursive calls are to initial segments of x, y and z. Moreover, the number of characters missing from the end of z must be precisely equal to the number of characters missing from the end of x plus the number of characters missing from the end of y — so the z argument is determined by the x argument and the y argument. There are $O(n)$ possible x arguments, and $O(m)$ possibly y arguments, for a total of $O(mn)$ possible recursive calls. The non-recursive parts of each call take $O(1)$ time, so the overall time taken is $O(mn)$. (Note that this is not the same as $O(n^2)$ — for example, if $n = 1$ and m is very large.)

14. (5 marks) (Medium question.) Mark the following five statements true or false.

(a) (1 mark) The problem of finding a minimum vertex cover in a given input graph is in NP.

(b) (1 mark) Every Co-NP-complete problem is the complement of an NP-complete problem (under Karp reductions).

(c) (1 mark) There is a Karp reduction from every problem in NP to 3-SAT.

(d) (1 mark) Every decision problem with a polynomial-time algorithm is in NP.

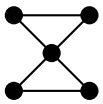
(e) (1 mark) If a problem is NP-complete under Karp reductions, then it is NP-complete under Cook reductions.

Solution: False, true, true, true, true. Finding a minimum vertex cover isn't a decision problem, so it can't be in NP. If X is a Co-NP-complete problem, then for any problem Y in Co-NP, there is a Karp reduction $f_{Y,X}$ mapping instances of Y to instances of X with the same answer in polynomial time. This same Karp reduction also maps instances of $\overline{Y}$ to instances of $\overline{X}$ with the same answer, so since every problem in NP is the complement of a problem in Co-NP, $\overline{X}$ must be NP-complete. 3-SAT is NP-complete as proved in the course, so every problem in NP Karp-reduces to it. The fourth statement says that P $\subseteq$ NP, which is proved in the course. The fifth statement follows from the fact that if $X \leq_K Y$, then $X \leq_C Y$ — the Cook reduction simply applies the Karp reduction to the given instance of X, applies the oracle to the result, and returns the output.

Long answer questions

15. (10 marks) (a) (5 marks) (Short question.) Give an example of an undirected graph with a closed Euler walk but no Hamilton cycle. You do not need to justify your answer.

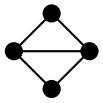
Solution: There are many possible answers, but one of them would be the graph $G = (V, E)$ with $V = [5]$, $E = \{\{1,2\},\{2,3\},\{3,1\},\{3,4\},\{4,5\},\{5,3\}\}$, i.e.:



Here 123451 is a closed Euler walk, but the only cycles in the graph are 1231 and 3453.

(b) (5 marks) (Short question.) Give an example of an undirected graph with a Hamilton cycle but no closed Euler walk. You do not need to justify your answer.

Solution: There are many possible answers, but one of them would be the graph $G = (V, E)$ with $V = [4]$, $E = \{\{1,2\},\{2,3\},\{3,4\},\{4,1\},\{1,3\}\}$, i.e.:



Here 12341 is a Hamilton cycle, but there is no closed Euler walk since 1 and 3 have odd degree.

16. (5 marks) (Short question.) The *Bacon number* of an actor is defined as follows. Kevin Bacon has a Bacon number of zero. The Bacon number of another actor is the shortest chain of co-stars linking them to Kevin Bacon. For example, Elvis Presley was in Change of Habit with Edward Asner, and Edward Asner was in JFK with Kevin Bacon, so Elvis Presley has a Bacon number of at most 2. Elvis was never in any movies with Kevin directly, so he has a Bacon number of exactly 2. Given API access to IMDB, **briefly summarise** an efficient algorithm to output the Bacon number of a given actor by reducing to a problem solved in the course. You do not need to prove your algorithm works.

Solution: Let A be the set of all actors listed on IMDB. Let

$$E = \{i, j : \text{actors } i \text{ and } j \text{ have appeared in a movie together}\}.$$

Then the Bacon number of an actor is precisely their distance from Kevin Bacon in the graph (A, E), which can be queried using the IMDB API. The problem can therefore be solved with $O(|A|^2)$ API queries using breadth-first search.

17. (10 marks) (Short question.) You are running a cattery. You have a large number of cats, and you wish to feed them a diet which meets all their nutritional needs while spending as little money as possible. You are given a list of store-brand foods and supplements, along with their prices and nutritional content per kilo. Express this as a linear programming problem by filling in the blanks below. You may assume all nutritional needs are lower bounds (e.g. "needs at least 200g protein per day").

    The linear programming problem will be of the form

$$\begin{aligned} c_1 x_1 + \cdots + c_t x_t &\to \min, \\ A\vec{x} &\geq \vec{b}, \\ \vec{x} &\geq \vec{0}. \end{aligned}$$

    Here $x_1, \ldots, x_t$ represent ____, $c_1, \ldots, c_t$ are given by ____, the rows of A are given by ____, and $\vec{b}$ is given by ____. This problem ____ an optimal solution.

    Each blank will be filled by one of the following:

    - the costs of each food or supplement per kilo;
    - the nutritional content of each food per kilo;
    - the amounts of each food to give the cats in kilos per day;
    - the total amount of each nutrient the cats require per day;
    - always has;
    - may or may not have;
    - never has.

Solution: The completed paragraph is given by:

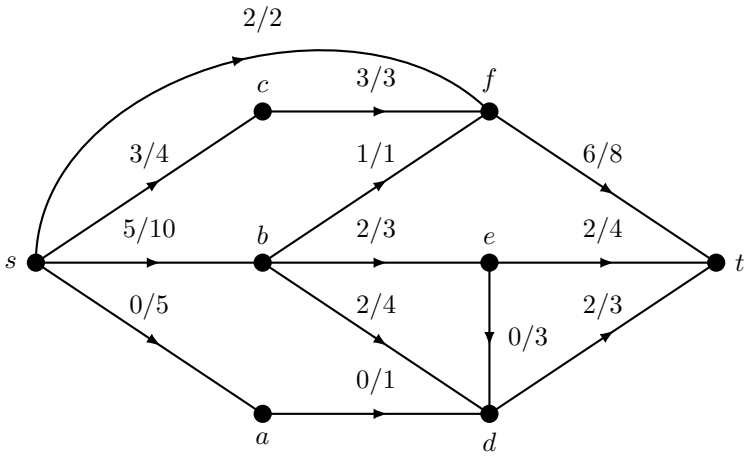
The linear programming problem will be of the form

$$\begin{aligned} c_1 x_1 + \cdots + c_t x_t &\to \min, \\ A\vec{x} &\geq \vec{b}, \\ \vec{x} &\geq \vec{0}. \end{aligned}$$

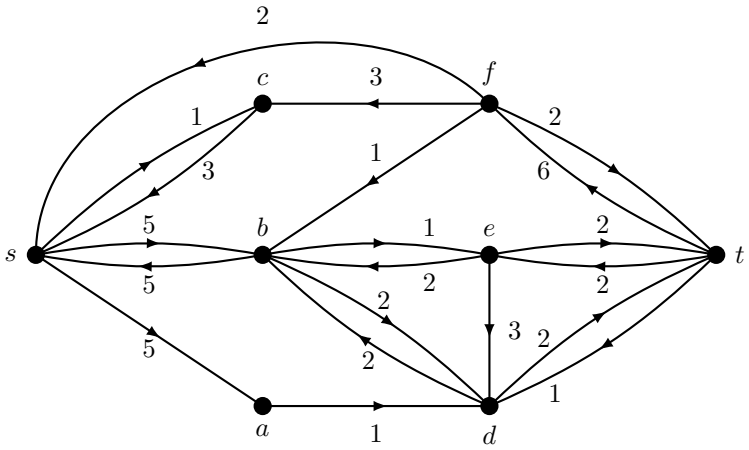
Here $x_1, \ldots, x_t$ represent **the amounts of each food to give the cats in kilos per day**, $c_1, \ldots, c_t$ are given by **the costs of each food or supplement per kilo**, the rows of A are given by **the nutritional content of each food per kilo**, and $\vec{b}$ is given by **the total amount of each nutrient the cats require per day**. This problem **always has** an optimal solution.

To see the last part is true, note that as explained in video 8-2, the only way a problem can fail to have any solutions is if either the objective function is unbounded or if there are no feasible solutions. Here each c_i is implicitly non-negative (no food can ever have a negative cost), so the objective function is bounded below by zero. Moreover, we can satisfy every constraint by taking $x_1, \ldots, x_t$ large enough, so there is at least one feasible solution. (Of course, since we don't currently allow any upper bounds on e.g. caloric intake, this may result in the cats becoming spherical...)

18. (5 marks) (Short question.) Give a list of augmenting paths from s to t in the below flow network, with respect to the given flow.



Solution: ***sbet***, ***sbedt***, ***sbdt***, ***sadt***, ***sadbet***. The residual graph is as shown below.



19. (a) (10 marks) (Medium question.) Consider the following greedy algorithm to find a large independent set in a graph.

Algorithm: GREEDYIS(G, k)

Input : A graph G.
Output: An independent set of G.

```
1 begin
2     Let output ← ∅.
3     foreach v ∈ V(G) do
4         if output ∪ {v} is an independent set then
5             output ← output ∪ {v}.
6     Return output.
```

Fix an integer $\Delta > 0$. If G has n vertices and maximum degree Δ, then how large an independent set is GreedyIS **guaranteed** to output? Your answer should include both an upper bound and a lower bound, and a brief justification of each.

Solution: $n/(\Delta+1)$. A vertex is added to `output` if there is no vertex already adjacent to it in `output` already. Hence each vertex v that GreedyIS adds to `output` prevents at most $d(v) \leq \Delta$ further vertices from being added to `output`. Thus $|\texttt{output}| + \Delta \cdot |\texttt{output}| \geq n$. Rearranging, we obtain $|\texttt{output}| \geq n/(\Delta+1)$ as required. Conversely, if we take G to be a union of cliques of size $\Delta+1$, then any independent set can contain at most one vertex from each clique, so a maximum independent set contains only $n/(\Delta+1)$ vertices.

(b) (5 marks) (Medium question.) Now consider the following refinement of GreedyIS.

Algorithm: BetterGreedyIS(G, k)

Input : A graph G.
Output: An independent set of G.

```
1 begin
2     Sort V(G) in increasing order of degree. Write V(G) = {v_1, ..., v_n}, with d(v_1) ≤ ··· ≤ d(v_n).
      Let output ← ∅.
3     for i = 1 to n do
4         if output ∪ {v_i} is an independent set then
5             output ← output ∪ {v_i}.
6     Return output.
```

Prove that BetterGreedyIS may still output an independent set of size $O(1)$ even if G contains an independent set of size $\Omega(n)$.

Solution: There are many ways of doing this — here's one. Consider an input graph $G = (V, E)$ of the following form. Let t be an arbitrary (large) integer. We have $V = \{a, b\} \cup A \cup B$, where A and B are t-vertex sets. We form E by joining a to every vertex in A, b to every vertex in B, and every vertex in A to every vertex in B. Then $d(a) = d(b) = t$, and $d(v) = t + 1$ for all $v \in A \cup B$. Thus BetterGreedyIS will pick first a, then b, then be unable to pick any other vertices, returning a set of size $2 \in O(1)$. Meanwhile, A is an independent set of size $t = (|V| - 2)/2 \in \Omega(|V|)$.

20. (5 marks) (Medium question.) Let (G, w) be a connected weighted graph with non-negative weights, and let T be result of running Kruskal's algorithm on G. Let $w'(x, y) = \log(w(x, y) + 1)$ for all edges $\{x, y\} \in E(G)$. Is T a minimum spanning tree of (G, w')? Briefly explain your answer.

Solution: Yes. Observe that by exponentiating both sizes, for all $x \geq 0$, $\log(1 + x) \leq \log(1 + y)$ if and only if $1 + x \leq 1 + y$ if and only if $x \leq y$. Thus Kruskal's algorithm will consider the edges of (G, w') in the same order as the edges of (G, w), and it will pick the same spanning tree for both graphs.

21. (5 marks) (Medium question.) The Travelling Salesman Problem (TSP) is defined as follows. We are given a list of n cities and, for each unordered pair $\{i, j\}$ of distinct cities, the cost $c(i, j)$ of travelling between i and j. (These costs can be arbitrary positive integers.) We are also given an integer k. We must output `Yes` if there is a way to travel to each city **exactly once**, then return back to the starting point, with total cost at most k. We call this a *round trip*. Otherwise, we must output `No`.

As an example, the input may be the set of cities {Amsterdam, Baghdad, Cairo, Dublin}, the cost function

$$c(\text{Amsterdam}, \text{Baghdad}) = 175, \quad c(\text{Amsterdam}, \text{Cairo}) = 95, \quad c(\text{Amsterdam}, \text{Dublin}) = 24,$$
$$c(\text{Baghdad}, \text{Cairo}) = 140, \quad c(\text{Baghdad}, \text{Dublin}) = 250, \quad c(\text{Cairo}, \text{Dublin}) = 122,$$

and the integer $k = 500$. The round trip from Amsterdam to Baghdad to Cairo to Dublin and back to Amsterdam costs $175 + 140 + 122 + 24 = 461$. So there is a round trip with cost at most 500, and the desired output is `Yes`.

Prove that the Travelling Salesman Problem is NP-complete (under Karp reductions). You may use the fact that the problem HC of deciding whether or not a graph contains a Hamilton cycle is NP-complete.

Solution: The desired output is `Yes` if and only if there is a round trip with cost at most k. Given a sequence of cities, we can verify this property in polynomial time — we simply need to check the sequence for duplicates and sum up the costs. By the definition of the NP class, it follows that the Travelling Salesman Problem is in NP.

Let $G = (V, E)$ be an instance of HC, i.e. an undirected n-vertex graph. We define a corresponding instance $f(G)$ of TSP as follows. Let the set of cities be V. For each distinct $i, j \in V$, let

$$c(i, j) = \begin{cases} 1 & \text{if } \{i, j\} \in E, \\ n + 1 & \text{otherwise.} \end{cases}$$

Let $k = n$. Then it is easy to compute (V, c, k) in polynomial time, and (V, c, k) is a `Yes`-instance of TSP if and only if there is a round trip of cost at most n. This holds if and only if every successive pair of cities in the round trip has cost 1, which holds if and only if the round trip is a Hamilton cycle in G. Thus f is a Karp reduction from HC to TSP.

22. (5 marks) (Long question.) Consider the following problem, which we call Approx-SAT. The input is a logical formula F in conjunctive normal form. The desired output is `Yes` if there is an assignment of truth values to variables which satisfies at least two thirds of F's OR clauses, and `No` otherwise. Prove that Approx-SAT is NP-complete under Karp reductions.

Solution: Given a proposed assignment, we can easily check whether it satisfies at least two thirds of F's OR clauses, so Approx-SAT is in NP. It remains to show that Approx-SAT is NP-hard under Karp reductions. We do so by giving a Karp reduction from SAT to Approx-SAT. Let F be an arbitrary instance of SAT, with k OR clauses. Then we form an instance F' of Approx-SAT by introducing new variables $x_1, \ldots, x_k$, and taking

$$F' = F \wedge x_1 \wedge (\neg x_1) \wedge x_2 \wedge (\neg x_2) \wedge \cdots \wedge x_k \wedge (\neg x_k).$$

This can obviously be done in polynomial time. Now, F' is a `Yes` instance of Approx-SAT if and only if there is an assignment satisfying at least $2k$ of its $3k$ OR clauses. *Any* assignment satisfies exactly k of the new (trivial) OR clauses, and F only has k OR clauses to begin with, so this can only happen if F is satisfied. Hence F' is a `Yes` instance of Approx-SAT if and only if F is a `Yes` instance of SAT, as required.

23. (5 marks) (Long question.) Let $G = (V, E)$ be a bipartite graph with bipartition (A, B). Let a and b be positive integers, and suppose that every vertex in A has degree a and every vertex in B has degree b. What is the average degree $\frac{1}{|V|} \sum_{v \in V} d(v)$ of G's vertices, in terms of a and b? (Note that your answer should **not** depend on $|A|$ or $|B|$.)

Solution: Double-counting the number of edges in G gives $|E| = a|A| = b|B|$. Thus

$$\frac{1}{|V|}\sum_{v\in V} d(v) = \frac{a|A| + b|B|}{|V|} = \frac{2a|A|}{|V|}.$$

Moreover, we have $|V| = |A| + |B|$, so since $a|A| = b|B|$ we have $|V| = (1 + a/b)|A|$; hence

$$\frac{1}{|V|}\sum_{v\in V} d(v) = \frac{2a|A|}{(1 + \frac{a}{b})|A|} = \frac{2ab}{a + b}.$$

24. (5 marks) (Long question.) Let $G = (V, E)$ be a graph. A *dominating set* in G is a subset $X \subseteq V$ such that for all $v \in V \setminus X$, we have $\{x, v\} \in E$ for some $x \in X$. In other words, every vertex of G is either contained in X or joined to X by an edge (or both). *DS* is the problem which asks: Given a graph G and an integer k, does G contain a dominating set of size at most k? Give a Karp reduction from VC to DS and briefly explain why it works. (**Hint:** Among other things, you will need to insert a vertex into the middle of each of G's edges.)

Solution: Let (G, k) be an instance of VC. Then we form an instance (H, k) of DS as follows. First, remove all degree-zero vertices of G. Next, replace each edge $e = \{x, y\} \in E$ by a triangle $\{\{x, v_e\}, \{v_e, y\}, \{x, y\}$, where v_e is a new vertex specific to e. Then we claim that for all k, G has a vertex cover of size at most k if and only if H has a dominating set of size at most k, so the map from (G, k) to (H, k) is the claimed Karp reduction.

Suppose $X \subseteq V$ is a vertex cover of G with $|X| \leq k$. If X contains any degree-zero vertices, we may remove them to form a new vertex cover $X' \subseteq X$. Then every edge of G has at least one endpoint in X', so every vertex v_e in H is adjacent to a vertex in X'. Moreover, every vertex in H is incident to at least one edge, and hence either in X' or adjacent to a vertex in X'. Thus X' is a dominating set of H of size at most k.

Conversely, suppose $Y \subseteq V$ is a dominating set of H with $|Y| \leq k$. If Y contains any vertices $v_e \notin V$, then we may replace them with one of v_e's endpoints to form a new dominating set Y' with $|Y'| \leq |Y|$. Then $Y' \subseteq V$, and every vertex v_e is adjacent to a vertex in Y', so Y' is a vertex cover of G of size at most k.

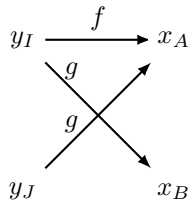
(Another approach would be to form H by replacing each edge of G by a length-2 path rather than a triangle, and adding a new vertex adjacent to every vertex in V. Then G will have a vertex cover of size at most k if and only if H has a vertex cover of size at most $k + 1$. There are probably other approaches that work as well.)

25. (5 marks) (Long question.) You are given two sets $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ of real values, and one list $e_1, \ldots, e_n \geq 0$ of tolerances. You wish to determine whether or not there is a map $f\colon \{y_1, \ldots, y_n\} \to \{x_1, \ldots, x_n\}$ such that each y_i is mapped to exactly one x_j, and if y_i is mapped to x_j then $x_j - e_j \leq y_{i_j} \leq x_j + e_j$. In other words, you wish to know whether $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$ are the same set up to additive error given by $e_1, \ldots, e_n$. Sketch an $O(n^2)$-time greedy algorithm for this problem, and prove using an exchange argument that it works.

[This one is a bit harder than anything in the exam — sorry... Still good practice though.]

Solution: Sort $y_1, \ldots, y_n$ in increasing order — say $y_1 \leq \cdots \leq y_n$. For each $i \in [n]$, let $X_i = \{x_j \colon x_j - e_j \leq y_i \leq x_j + e_j\}$ be the set of values x_j which y_i could be mapped to. Set $f(y_1)$ to the $x_j \in X_1$ with $x_j + e_j$ smallest. Set $f(y_2)$ to the $x_j \in X_2 \setminus \{f(y_1)\}$ with $x_j + e_j$ smallest, set $f(y_3)$ to the $x_j \in X_3 \setminus \{f(y_1), f(y_2)\}$ with $x_j + e_j$ smallest, and so on. If at any point there are no available candidates, output `No map`, otherwise output `Map`. Certainly if this algorithm returns `Map` then it has succeeded in building a valid map; it remains to show that if a valid map exists, this algorithm will return `Map`.

Suppose a valid map g exists. Let f be the (possibly partial) function found by our algorithm, and suppose $f \neq g$. Let $I = \min\{i \colon f(y_i) \neq g(y_i)\}$ be the first point at which g diverges from our algorithm. Write $f(y_I) = x_A$, and $g(y_I) = x_B$. Since $f(y_I) \neq g(y_I)$, we must have $x_A = f(y_I) = g(y_J)$ for some J. Moreover, since $f(y_i) = g(y_i)$ for all $i < I$, we must have $J > I$. See the picture below.



We will remap $g(y_J)$ to x_B and $g(y_I)$ to x_A. We know $x_A \in X_I$ since $f(y_I) = x_A$, so it suffices to show $x_B \in X_J$. By the way we chose $f(y_I)$, we have $x_B + e_B \geq x_A + e_A$, and since g is a valid map we have $x_A + e_A \geq y_J$; hence $x_B + e_B \geq y_J$. Finally, since $J > I$ and the y_i's are sorted, we have $y_J \geq y_I$, and since g is a valid map we have $y_I \geq x_B - e_B$; hence $y_j \geq x_B - e_B$ also. By repeating this process, we can turn g into f, and hence f must be a valid map and our algorithm will return `Map`.