COMS20010 — 2020–21 Exam

This exam was actually given as a Blackboard test in a slightly different format to the one used this year (hence "Section 1" and "Section 2"). This means this isn't quite the final version, and since it hadn't gone through the final round of checking at that point there might still be one or two errors lurking in the model answers, but it should be a good guide as to what to expect.

Section 1

1. (5 marks) (Short question.) Mark each of the following statements true or false:
   (a) (1 mark) $n^2 \in O(5n)$.

   Solution: False, $n^2 \in \omega(5n)$.

   (b) (1 mark) $10n^2 + n \in O(n^2)$.

   Solution: True, $10n^2 + n \in O(n^2)$.

   (c) (1 mark) $\log n \in O(n^{1/5})$.

   Solution: True, $\log n \in O(n^{1/5})$.

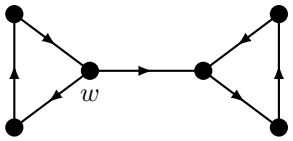
   (d) (1 mark) $n^{65} \in O(n!)$.

   Solution: True, $n^{65} \in O(n!)$.

   (e) (1 mark) $2^{n^2} \in O(2^n)$.

   Solution: False, $2^{n^2} \in \omega(2^n)$.

2. (5 marks) (Short question.) Consider the following graph G and the indicated vertex w.



   Mark each of the following statements true or false:
   (a) (1 mark) G is weakly connected.

   Solution: True, G is weakly connected.

   (b) (1 mark) G is strongly connected.

Solution: False, there is no path from the right triangle to the left triangle.

(c) (1 mark) G has an Euler walk.

Solution: True, starting at v, walking round the left triangle, crossing to the right triangle, walking round the right triangle.

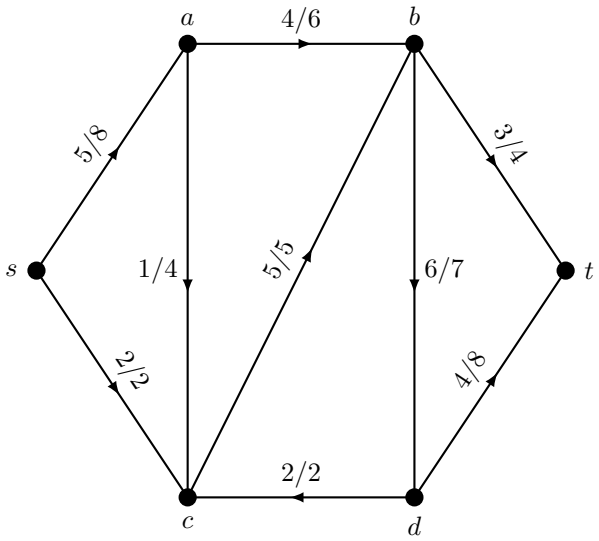
(d) (1 mark) G contains a directed cycle as an induced subgraph.

Solution: True, both triangles are induced directed cycles.

(e) (1 mark) $d^-(w) = 2$.

Solution: False, $d^-(w) = 1$.

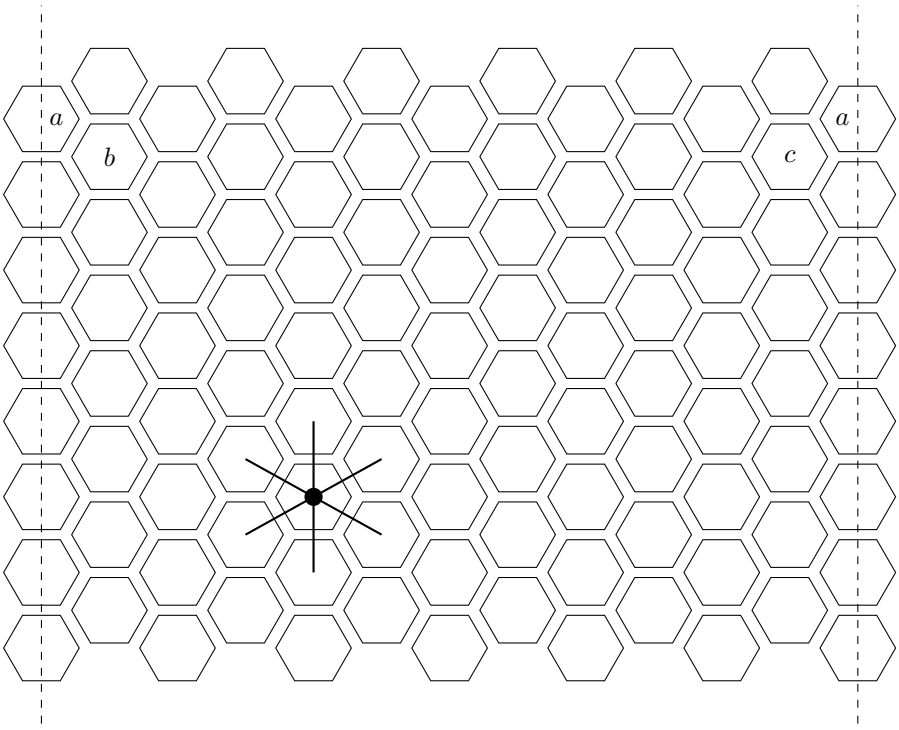
3. (5 marks) (Short question.) Consider the following flow network and flow, with source s and sink t.



List all augmenting paths of the network under the given flow.

Solution: The augmenting paths are *sabt*, *sabdt*, *sabcdt*, *sacdt*, and *sacdbt*.

4. (5 marks) (Short question.) The strategy game Civilization VI is played on a hexagonal map as shown below. The map is effectively a cylinder, with the left and right boundaries are joined together, and the top and bottom boundaries are impassable — thus in the picture below, tile a appears twice, and is adjacent to both tile b and tile c. Units can move between any pair of adjacent tiles, and this is stored in memory as a graph in which there is an edge between two tiles if they are adjacent in the map.



If the map is 90 tiles wide and 56 tiles high, how many edges does the graph have? (**Hint:** Use the handshaking lemma.)

A. 13440.

B. 14829.

C. 14940.

D. 29658.

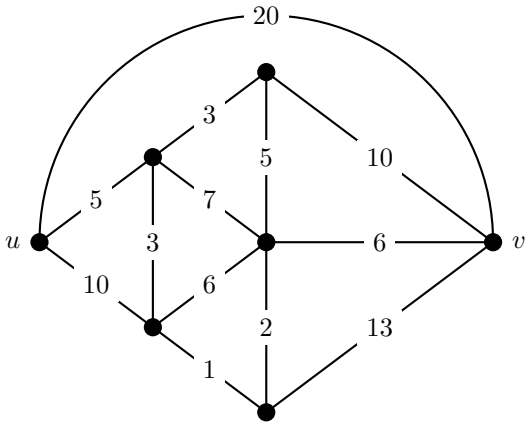
E. None of the above.

Solution: By the handshaking lemma, the total number of edges is half the total degree of the graph. There are 90 tiles of degree 5 and 90 tiles of degree 3, with 45 of each along the top edge and 45 of each along the bottom edge. The remaining $90 \cdot 56 - 90 \cdot 2$ tiles all have degree 6. Thus the total number of edges is

$$\frac{90 \cdot 54 \cdot 6 + 90 \cdot 5 + 90 \cdot 3}{2} = 14940.$$

A is the number of edges in the standard Civilization VI map size of 84×54, just in case someone manages to Google it. B is what you get if you forget the horizontal boundaries wrap. D is what you get if you forget the horizontal boundaries wrap and forget to divide by 2.

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



What is the distance from u to v?

A. 1.

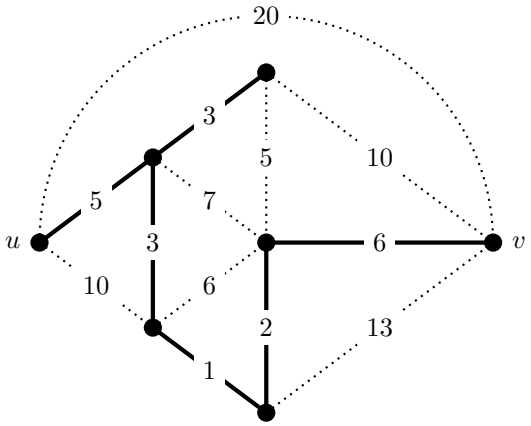
B. 16.

C. 18.

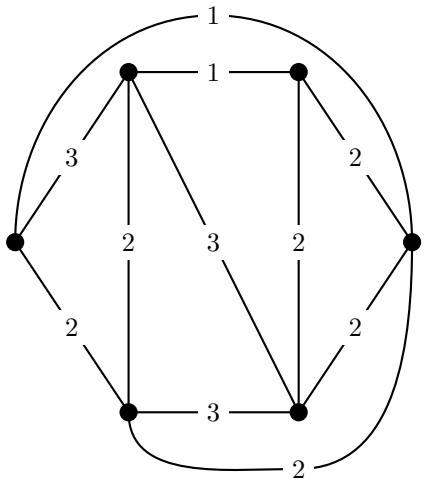
D. 20.

E. None of the above.

Solution: E — None of the above. The correct answer is 17. A shortest-path tree is drawn below.



6. (5 marks) (Short question.) Consider the 11-vertex weighted graph G below.



What is the weight of a minimum spanning tree of G?

A. 12.

B. 13.

C. 14.

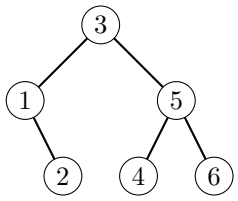
D. G doesn't have a minimum spanning tree.

E. None of the above.

Solution: D — G is disconnected, so it doesn't have any spanning trees at all. A minimum spanning tree of the left component has total weight 8, and a minimum spanning tree of the right component has total weight 6.

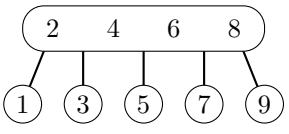
7. (5 marks) (Short question.) Which of the following are valid 2-3-4 trees?

(a) (1 mark)



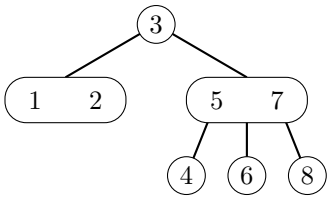
Solution: Invalid. This tree breaks perfect balance, because 1 is a non-leaf 2-node with fewer than 2 children.

(b) (1 mark)



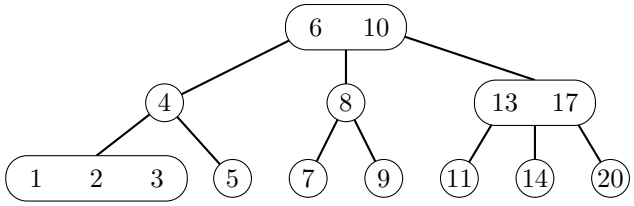
Solution: Invalid. The root is neither a 2-node, nor a 3-node, nor a 4-node.

(c) (1 mark)



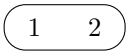
Solution: Invalid. This tree breaks perfect balance, because $(1, 2)$ is a leaf not on the bottom level of the tree.

(d) (1 mark)



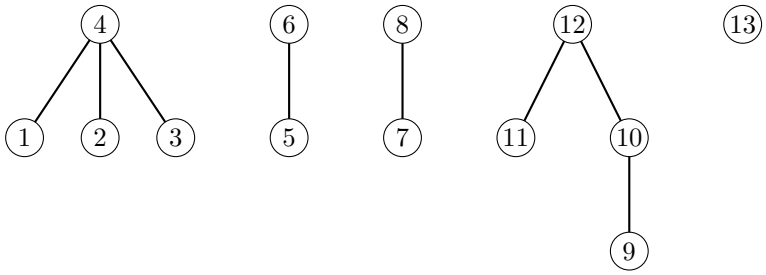
Solution: Valid. The values in a 2-3-4 tree don't have to be consecutive.

(e) (1 mark)



Solution: Valid. A 2-3-4 tree can have any number of levels, even one.

8. (5 marks) (Short question.) Consider the unoptimised version of the union-find data structure, in which a sequence of n operations takes $O(n \log n)$ time rather than $O(n\alpha(n))$ time. Suppose the current state of the data structure is as follows:



(a) (1 mark) What is the result of `Find`(13)?

Solution: 13.

(b) (1 mark) What is the result of `Find`(9)?

Solution: 12, the root of the component containing 9.

(c) (2 marks) Suppose we now apply the operations `Union`(6,8), `Union`(1,7) and `Union`(10,4) to the data structure, in this order. How many components does the data structure now have?

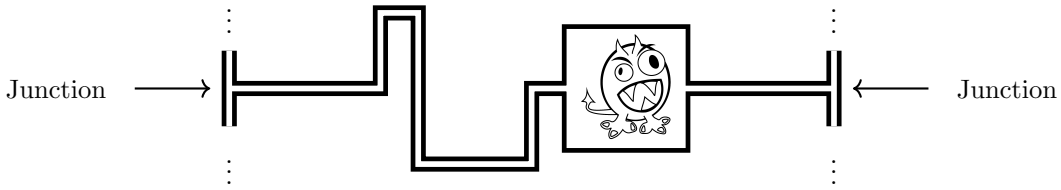
Solution: 2. Each `Union` operation reduces the number of components by one.

(d) (1 mark) Again after applying the `Union` operations above, what is the maximum depth of any component of the data structure? (The answer does not depend on how ties are broken.)

Solution: 3. The exact structure will depend on how ties are broken, but the first `Union` will always merge the middle two components into a depth-2 component, the second `Union` will always merge the first component with this depth-2 component without increasing its depth, and the third `Union` will always merge two depth-2 components to produce a depth-3 component.

9. (5 marks) (Short question.) EvilCo Incorporated designs dastardly mazes for enterprising B-movie villains. In their marketing material, they say that their mazes are so complicated that if you want to get from one point to another without retracing your steps, there's only ever one way to do it. If one of their mazes has j junctions, how many corridors between junctions does it have, and why?

   (Note this does **not** include e.g. dead ends, or the entrance and exit — only corridors between two junctions count. Your answer should be short.)



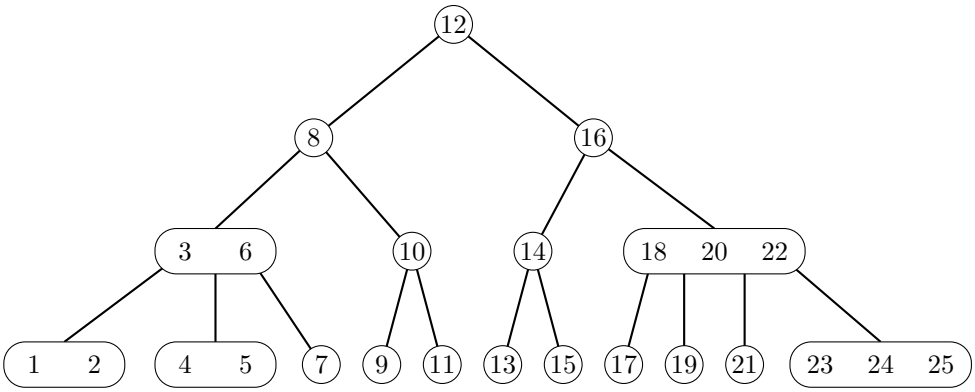
Solution: We can view the maze as a graph, where the vertices are junctions, and two junctions are joined by an edge if there is a direct corridor between them. In this context, the marketing material says that there is a unique path between any two vertices. Hence by the Fundamental Lemma of Trees, the graph must be a tree and contain $j - 1$ edges.

10. (5 marks) (Medium question.) You are the CEO of a sprawling corporation, which contains many overlapping product teams. You wish to set up a *representative meeting* — a meeting containing at least one representative from each product team — which is as small as possible, using the fact that a single person can act as a representative for all of the teams they're a member of. By reducing from vertex cover or otherwise, prove that it is NP-complete (under Karp reductions) to decide whether or not a k-person representative meeting exists, given a list of the teams and the value of k.

Solution: Given a proposed meeting, we can verify that it contains at least one person from each team in polynomial time, so the problem is in NP. It remains to show that the problem is NP-hard under Karp reductions.

Let (G, k) be an instance of vertex cover. Then we take our employees to be the vertices of G, and our teams to be the edges of G. Thus a set S of k people forms a representative meeting if and only if every edge contains a person in S — that is, if and only if S is a size-k vertex cover of G. We have therefore created an instance of the representative meeting problem in polynomial time whose answer is `Yes` if and only if (G, k) is a `Yes` instance of vertex cover, as required.

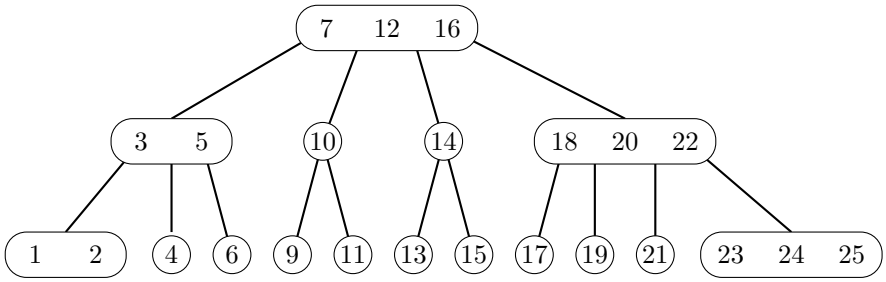
11. (5 marks) (Medium question.) Consider the 2-3-4 tree below:



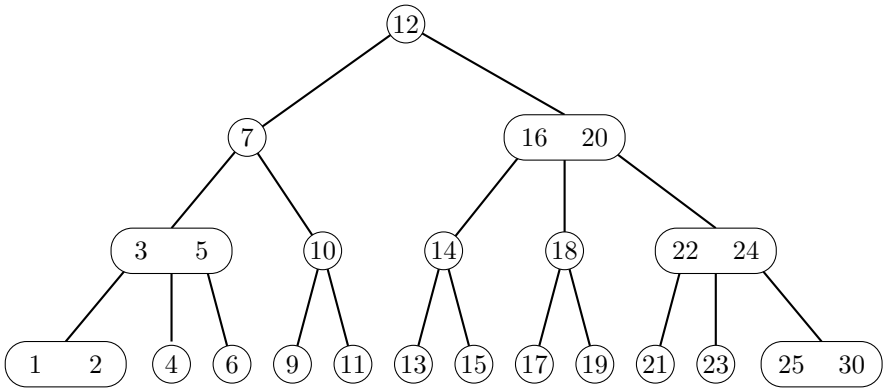
After deleting the value 8 and inserting the value 30:

(a) (1 mark) What is the depth of the resulting tree?

Solution: 3. Deleting 8 involves fusing the root, transferring 5 from $(4, 5)$ to (7), deleting 7 and overwriting 8 with 7. The result looks like this:



Inserting 30 then involves splitting the root, splitting $(18, 20, 22)$, splitting $(23, 24, 25)$ and inserting 30 into the newly formed (25) node. The result looks like this:



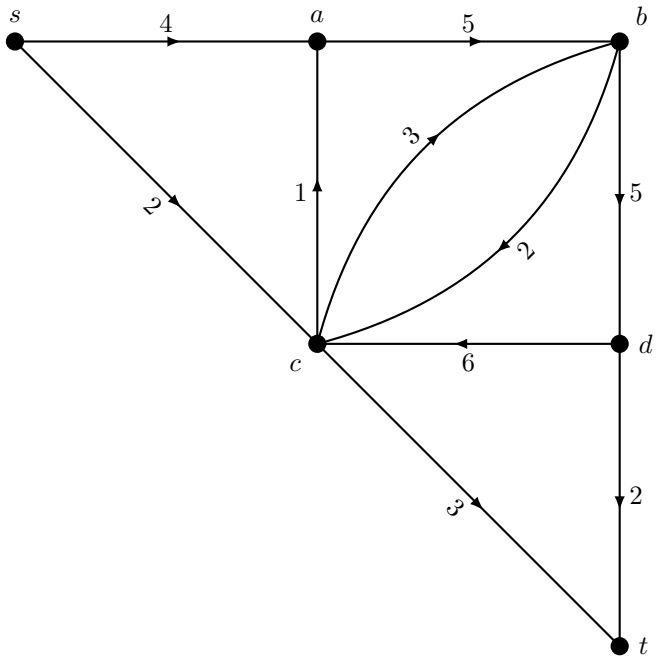
(b) (2 marks) How many 3-nodes does the resulting tree have?

Solution: 5. See above.

(c) (2 marks) While deleting 8 and inserting 30, how many fuse, transfer and split operations were performed in total?

Solution: 5. See above.

12. (5 marks) (Short question.) Consider the following flow network (G, c, s, t):



Let $A = \{s, b\}$ and $B = \{a, c, d, t\}$. What is the capacity $c^+(A)$ of the cut (A, B)?

A. 8.

B. 13.

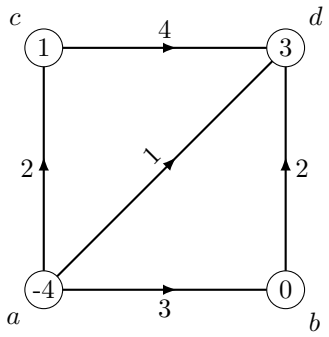
C. 21.

D. (A, B) is not a cut.

E. None of the above.

Solution: B — 13. Cuts do not have to form connected induced subgraphs, and the capacity is the sum of the capacities of all edges leaving A.

13. (5 marks) (Short question.) Consider the following circulation network.



Which of the following is a valid circulation f?

A. $f(a, b) = 2$; $f(a, c) = 0$; $f(a, d) = 1$; $f(b, d) = 2$; $f(c, d) = 0$.

B. $f(a, b) = 2$; $f(a, c) = 2$; $f(a, d) = 0$; $f(b, d) = 1$; $f(c, d) = 1$.

C. $f(a,b) = 3$; $f(a,c) = 1$; $f(a,d) = 0$; $f(b,d) = 3$; $f(c,d) = 0$.

D. $f(a,b) = 1$; $f(a,c) = 2$; $f(a,d) = 1$; $f(b,d) = 1$; $f(c,d) = 1$.

E. None of the above, or more than one of the above.

Solution: D is a valid circulation. A violates the demand constraints at a and c. B violates the demand constraints at b and d. C violates the capacity constraint at (b,d).

14. (5 marks) (Short question.) Mark each of the following statements true or false.

(a) (1 mark) The problem of outputting a size-k vertex cover in a graph G if one exists or `No cover` otherwise, given G and k as inputs, is NP-hard.

Solution: True. There is an immediate reduction from (the decision version of) vertex cover, the NP-completeness of which was proved in lectures..

(b) (1 mark) The problem of outputting a size-k vertex cover in a graph G if one exists or `No cover` otherwise, given G and k as inputs, is in NP.

Solution: False. This is not a decision problem, so it is not a member of NP.

(c) (1 mark) Because SAT is an NP-complete problem, it is never practical to solve a 1,000-variable instance of SAT.

Solution: False. Discussed highly-efficient industrial SAT-solvers in lectures, and the 1,000-variable instance could just be e.g. $x_1 \wedge x_2 \wedge \cdots \wedge x_{1000}$.

(d) (1 mark) If we had working quantum computers, we know how to use them to solve NP-complete problems in polynomial time.

Solution: False. Explicitly discussed limitations of quantum computers in lectures.

(e) (1 mark) If a problem has instance size n and parameter k, then an algorithm with running time $O(n^{\sqrt{k}})$ shows that the problem is fixed-parameter tractable.

Solution: False. FPT time requires running time $O(g(k)n^C)$ for some computable function g and some constant C.

15. (5 marks) (Short question.) Consider an instance of weighted interval scheduling with interval set $\mathcal{R} = [(1,5),(4,6),(2,11),(6,13),(11,16),(14,19),(14,21),(18,21),(20,25)]$ and weight function w given by

$$w(1,5) = 3, \quad w(4,6) = 5, \quad w(2,11) = 7, \quad w(6,13) = 3, \quad w(11,16) = 3,$$
$$w(14,19) = 4, \quad w(14,21) = 6, \quad w(18,21) = 5, \quad w(20,25) = 3.$$

What is the maximum possible weight of a compatible set?

A. 13.

B. 14.

C. 15.

D. 16.

E. None of the above.

Solution: C − 15. An optimal set is $\{(2, 11), (11, 16), (18, 21)\}$. In more detail, writing OPT(i) for the weight of an optimal solution on the first i intervals in ascending order of finish time, we have:

$$\text{OPT}(1) = 3, \quad \text{OPT}(2) = 5, \quad \text{OPT}(3) = 7, \quad \text{OPT}(4) = 8, \quad \text{OPT}(5) = 10,$$
$$\text{OPT}(6) = 12, \quad \text{OPT}(7) = 15, \quad \text{OPT}(8) = 15, \quad \text{OPT}(9) = 15.$$

The two compatible sets with weight 15 are $\{(4, 6), (6, 13), (14, 19), (20, 25)$ and $\{(2, 11), (11, 16), (18, 12)\}$.

16. (5 marks) (Short question.) You are captain of a spaceship in the far future, and you wish to traverse the galaxy from Caldari Prime to Molea Cemetary, hopping from planet to planet on the way while spending as few resources and making as much profit through trade on the way as possible. Your crew is troublesome — they require that each stop brings you strictly closer to your destination, or they will mutiny, and on arrival at a new planet they will go on shore leave and consume any goods left in your hold which you do not immediately sell. You are given a list of planets, their locations, and the (possibly negative) cost of travelling from one planet directly to another. Which of the following algorithms would be the best fit for this situation?

    A. Breadth-first search.

    B. Dijkstra's algorithm.

    C. The Bellman-Ford algorithm.

    D. Depth-first search.

    E. There is no unique best choice from the above options.

Solution: C — The Bellman-Ford algorithm. The situation can be modelled as a weighted directed graph on the set of planets. There is an edge from planet X to planet Y if Y is strictly closer to Molea Cemetary than X, and the weight of this edge is the cost of travelling from X to Y. You seek a shortest path from Caldari Prime to Molea Cemetary. By the triangle inequality, the graph contains no cycles at all and hence no negative-weight cycles, so the Bellman-Ford algorithm will work. Breadth-first search and depth-first search don't handle weighted edges, and Dijkstra's algorithm doesn't handle edges with negative weights.

17. (10 marks) (Short question.) You have been placed in charge of overseeing plywood distribution in the USSR. You are given a list of production facilities $F_1, \ldots, F_n$ and destinations $D_1, \ldots, D_m$. Each facility F_i can produce at most p_i units of plywood per day, and each destination D_j requires at least r_j units of plywood per day. The cost of moving one unit of plywood from facility F_i to destination D_j is $c_{i,j}$, and you may assume this scales linearly (so that e.g. the cost of moving half a unit is $c_{i,j}/2$). You wish to ensure that the requirements of each destination are met while spending as little on transportation as possible. Formulate this as a linear programming problem. (Your answer does not have to be in standard form, and you do not have to justify it.)

Solution: Let $x_{i,j}$ be the number of units of plywood moved from facility F_i to destination D_j each day. We wish to minimise $\sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j} x_{i,j}$. Our constraints are:

- We cannot transport a negative amount of plywood: $x_{i,j} \geq 0$ for all i, j.

- Each factory F_i can produce at most p_i plywood per day: $\sum_{j=1}^{m} x_{i,j} \leq p_i$ for all $i \in [n]$.
- Each destination D_j needs at least r_j plywood per day: $\sum_{i=1}^{n} x_{i,j} \geq d_j$ for all $j \in [m]$.

18. (10 marks) (Medium question.) You are given a directed graph G with vertex set $\{v_1, \ldots, v_n\}$ which has the following property: for any edge $(v_i, v_j) \in E(G)$, we have $i < j$. That is, edges are always directed from vertices with lower indices to vertices with higher indices. You wish to find the length of a **longest** path from v_1 to v_n, returning `Null` if no such path exists — recall that the length of a path is the number of **edges** it contains. Fill in the blanks of the following dynamic programming algorithm.

```
1  begin
2      Initialise A to an empty array indexed by 1, ..., n.
3      Set A[n] ← BLANK.
4      for i = n − 1 to 1 do
5          Let S ← {j : v_j ∈ N^+(v_i)}.
6          if A[j] = Null for all j ∈ S then
7              Set A[j] ← BLANK.
8          else
9              Let X be the BLANK of the non-Null values of {A[j] : j ∈ S}.
10             Set A[i] ← X BLANK BLANK.
11     Return A[BLANK].
```

Options for each blank: 0, 1, `Null`, $+$, $-$, maximum, minimum, none of the above.

Solution: First BLANK: 0. $A[i]$ should contain the length of a longest path from v_i to v_n, and the longest path from v_n to itself has length zero.

Second BLANK: `Null`. If v_i has no outgoing edges, then there are no paths from v_i to v_n.

Third BLANK: maximum. Fourth BLANK: $+$. Fifth BLANK: 1. We are trying to find the length of a longest path from v_i to v_n. The longest path must start with an edge to v_j for some $v_j \in S$; thus the length is equal to $\max\{A[j] : j \in S\} + 1$.

19. (10 marks) (Medium question.) Given three distinct literals x, y and z, a *ONE clause* $\text{ONE}(x, y, z)$ evaluates to `True` if and only if **exactly one** of x, y and z evaluates to `True`. A *lonely formula* is a conjunction of ONE clauses, e.g.:

$$\text{ONE}(a, b, c) \wedge \text{ONE}(\neg a, b, \neg c) \wedge \text{ONE}(b, c, \neg d).$$

The decision problem LONELY-SAT asks whether a ONE formula given as part of the input is satisfiable. There is a valid Karp reduction from 3-SAT to LONELY-SAT which replaces each OR clause $(x \vee y \vee z)$ of a 3-SAT instance with a LONELY-SAT gadget containing four new variables a, b, c and d. Which of the following gadgets will work?

A. $\text{ONE}(x, a, b) \wedge \text{ONE}(y, b, c) \wedge \text{ONE}(z, c, d)$.

B. $\text{ONE}(\neg x, a, b) \wedge \text{ONE}(y, b, c) \wedge \text{ONE}(\neg z, c, d)$.

C. $\text{ONE}(x, a, b) \wedge \text{ONE}(\neg y, b, c) \wedge \text{ONE}(z, c, d)$.

D. $\text{ONE}(\neg x, a, \neg b) \wedge \text{ONE}(y, b, c) \wedge \text{ONE}(\neg z, \neg c, d)$.

E. None of the above, or more than one of the above.

Solution: Gadget B works because:

- If x, y and z are all `False`, then the first clause forces $a = b =$ `False`, the third clause forces $c = d =$ `False`, and then the second clause is unsatisfiable.
- If y is `True`, then the gadget is satisfied on taking $b = c =$ `False`, $a = x$ and $d = z$.
- If y is `False`, then at least one of x or z must be true — wlog by symmetry we may assume it is x. Then the gadget is satisfied on taking $a = c =$ `False`, $b =$ `True` and $d = z$.

Gadget A doesn't work because when x, y and z are set to `False` in the original instance, the gadget is satisfiable by e.g. setting $a = c =$ `True` and $b = d =$ `False`.

Gadget C doesn't work because when x, y and z are set to `True` in the original instance, the gadget is unsatisfiable — the first clause forces $a = b =$ `False`, the third clause forces $c = d =$ `False`, and then the second clause is unsatisfiable.

Gadget D doesn't work because when y is `True` and (e.g.) x is `False`, then the second clause forces $b = c =$ `False`, which makes the first clause unsatisfiable.

Section 2

20. (10 marks) (Medium question.) Let t be a natural number. Let G be an n-vertex graph with $2t$ vertices of odd degree and $n - 2t$ vertices of even degree. Prove, by induction on t or otherwise, that we can write $E(G)$ as a **disjoint** union of edge sets

$$E(G) = E(P_1) \cup \cdots \cup E(P_t) \cup E(H),$$

where $P_1, \ldots, P_t$ are paths in G and H is a graph whose vertices all have even degree.

Solution: Base case: If $t = 0$, then we can simply take $H = G$.

Inductive step: Suppose the result holds for all $t \leq t_0$ for some $t_0 \geq 0$; then we must prove it holds for $t = t_0 + 1$.

We first form graphs G' and H^- by greedily removing cycles from G and adding them to H^- until no cycles remain. Since every vertex in a cycle has even degree, G' still has exactly k vertices of odd degree, and every vertex in H^- has even degree. Now let v be a vertex of odd degree in G', and let P be a path formed by starting from v and extending greedily until no further extensions are possible. (In other words, P is a maximal path with v as an endpoint.) Let x be the other endpoint of P, so that $N(x) \subseteq V(P)$. Since there are no cycles in G, x can only send one edge into $V(P)$, to its direct predecessor; thus $d(x) = 1$. Form G'' by removing P's edges from G; then G'' has $2t - 2 = 2(t-1)$ vertices of odd degree. By induction, we can write the edges of G'' as a disjoint union

$$E(G'') = E(P_1) \cup \cdots \cup E(P_{t-1}) \cup E(H^+),$$

where $P_1, \ldots, P_{t-1}$ are paths and H^+ is a graph whose vertices have even degree. Thus we have

$$E(G) = E(P_1) \cup \cdots \cup E(P_{t-1}) \cup E(H^+) \cup E(P) \cup E(H^-),$$

and so we are done on taking $P_t = P$ and $H = H^+ \cup H^-$.

Alternative solution: Use handshaking to show that each component has an even number of vertices of odd degree, then take P to be a path between two vertices of odd degree in the same component.

21. (10 marks) **Important:** The two parts of this question are completely independent — they have been put together for technical Blackboard reasons. Doing part a) will not help you with part b).

(a) (5 marks) (Long question.) Suppose that a connected n-vertex, m-edge graph G (given in adjacency-list form) contains two vertices x and y which are distance $d(x, y) \geq n - c$ apart, for some integer $c \geq 1$. Give an algorithm which, given G, x, and an integer k as input, decides whether or not G contains a k-vertex clique in time $O(m + n + c^k)$. Briefly explain why it works and why it runs in the claimed time. (You do not need to give full pseudocode — an informal description is fine.)

Hint: You may find it helpful to use a BFS tree.

Solution: If $k \in \{1, 2\}$, then the algorithm simply returns `Yes`, as any connected graph on at least two vertices contains both a vertex and an edge.

Otherwise, run BFS starting at x to obtain a BFS tree T rooted at x in time $O(m + n)$. Since $d(x, y) \geq n - c$, T must contain at least $n - c + 1$ layers. It follows that all but at most c layers $L_1, \ldots, L_t$ contain only a single vertex, and that $L_1, \ldots, L_t$ between them contain at most c vertices. Since a vertex can only send edges to adjacent layers, it follows that every vertex in G has degree at most $c + 1$. Moreover, since $k \geq 3$, every k-clique in G must contain at least one vertex from $L_1, \ldots, L_t$.

The algorithm therefore simply takes each vertex $v \in L_1, \ldots, L_t$, and checks whether each size-$(k-1)$ subset of $N(v)$ forms a k-clique together with v. There are $O(c)$ total vertices in $L_1, \ldots, L_t$, and there are $\binom{c+1}{k-1} \leq (c+1)^{k-1}$ subsets to check, so overall this takes $O(c^k)$ time.

(b) (5 marks) (Long question.) Let $n \geq 8$ be an even integer. Using Dirac's theorem or otherwise, prove that any n-vertex graph whose vertices all have degree at least $\frac{n}{2} + 3$ must contain a spanning subgraph whose vertices all have degree exactly 5. (Your answer shouldn't need to be more than a paragraph or two.)

Solution: By Dirac's theorem, G contains a Hamilton cycle C. Since n is even, writing $C = x_1 x_2 \ldots x_n$, we have that $M = \{\{x_{2i-1}, x_{2i}\} : i \in [n]\}$ is a perfect matching. Since M has degree 1 in every vertex, the minimum degree of $G - M$ is at least $\frac{n}{2} + 2$, and so by Dirac's theorem it contains a Hamilton cycle C_1. Since C_1 has degree 2 in every vertex, the minimum degree of $G - M - C_1$ is at least $\frac{n}{2}$, and so once again by Dirac's theorem it contains a Hamilton cycle C_2. Then M, C_1 and C_2 are edge-disjoint spanning subgraphs of G whose vertices all have degree 1, 2 and 2 respectively, so $M \cup C_1 \cup C_2$ is a spanning subgraph of G whose vertices all have degree 5.

22. (10 marks) **Important:** The two parts of this question are completely independent — they have been put together for technical Blackboard reasons. Doing part a) will not help you with part b).

(a) (5 marks) (Long question.) You are teaching a class of schoolchildren $C_1, \ldots, C_n$ in the pre-COVID era. The end of term is approaching, and you have bought them a large tub of chocolates — you plan to give them four chocolates each, and eat the rest yourself. However, there are many types $T_1, \ldots, T_k$ of chocolate in the tub, and not every child likes every type of chocolate — a given child C_i is willing to accept only chocolates from a set $S_i \subseteq \{T_1, \ldots, T_k\}$. The tub is also not bottomless — for each $i \in [k]$, there are only $t_i > 0$ chocolates of type T_i. Give an algorithm which, given $T_1, \ldots, T_k$, $t_1, \ldots, t_k$ and $S_1, \ldots, S_n$, assigns four chocolates to every child in polynomial time if possible and returns `Impossible` otherwise.

Solution: We construct a bipartite graph as follows. We assign each child C_i four vertices $c_{i,1}, \ldots, c_{i,4}$, each one representing a choice of chocolate. We assign each type T_i of chocolates t_i vertices $x_{i,1}, \ldots, x_{i,t_i}$, each one representing an available chocolate. We join two vertices $c_{i,j}$ and $x_{\ell,m}$ if and only if $T_\ell \in S_i$ — that is, if and only if child C_i is willing to eat chocolates

of type T_ℓ. A matching of size $4n$ in this graph corresponds to giving four chocolates to every child:

- Each edge $\{c_{i,j}, x_{\ell,m}\}$ in the matching corresponds to giving a chocolate of type T_ℓ to child C_i;
- The requirement that matching edges are disjoint in vertices $c_{i,j}$ corresponds to the requirement that each child receives at most four chocolates;
- The requirement that there are $4n$ edges in the matching corresponds to the requirement that each child receives at least four chocolates (since this can only occur if every vertex $c_{i,j}$ is matched);
- The requirement that matching edges are disjoint in vertices $x_{i,j}$ corresponds to the requirement that no more than t_i chocolates of each type are given out.

We can therefore simply find the maximum matching in the graph with e.g. Ford-Fulkerson, and return either the corresponding assignment of chocolates (if it has size $4a$) or `Impossible` (otherwise).

(b) (5 marks) (Long question.) Your company has been tasked with producing a set of widgets $w_1, \ldots, w_t$, using a 3D printer to make the parts. Each widget w_i must be first printed in time p_i, then assembled in time a_i. You have enough workers to be able to assemble as many widgets as you like in parallel, but you can only print one widget at a time. You would like to find an order of printing the widgets which allows you to finish assembling them all as quickly as possible. Give a greedy algorithm which, given $p_1, \ldots, p_t$ and $a_1, \ldots, a_t$, will output an optimal widget order in $O(t \log t)$ time. Give a careful proof that it works (e.g. using a greedy-stays-ahead or exchange argument).

Solution: The algorithm sorts the widgets in decreasing order of assembly time, so that (after sorting) $a_1 \geq a_2 \geq \cdots \geq a_t$, then outputs the result.

We prove this works using an exchange argument. Let $w_1, \ldots, w_t$ be the output of our algorithm, and let $u_1, \ldots, u_t$ be an optimal widget ordering. For convenience, let p'_i be the printing time of u_i and let a'_i be the assembly time of a_i. Let $I = \min\{i \colon w_i \neq u_i\}$ be the first point of divergence of our algorithm's ordering from $u_1, \ldots, u_t$. Let J be the index in $[t]$ such that $w_I = u_J$, and note that $J > I$ since $w_i = u_i$ for all $i \leq I$ and $w_I \neq u_I$. Form $u'_1, \ldots, u'_t$ from $u_1, \ldots, u_t$ by removing u_J from the ordering and re-inserting it at the I'th position, so that

$$u'_i = \begin{cases} u_i & \text{if } i < I \text{ or } i > J; \\ u_J & \text{if } i = I; \\ u_{i-1} & \text{if } I < i \leq J. \end{cases}$$

The overall effect is that for all $I < i \leq J$, u'_j will finish p'_J time units later than u_{j-1} did; u_I will finish earlier than u_J did; and all other widgets will finish at the same time as before. However, because our algorithm chose u_J before any other element of $\{u_I, \ldots, u_t\}$, the assembly time of u_J must be at least as large as any other widget in that set. Since u_J could not start to be assembled until after it finished printing, it follows that in $u_1, \ldots, u_t$, u_J finished at least p'_J later than any widget in $\{u_I, \ldots, u_{J-1}\}$. Hence no element of $u'_I, \ldots, u'_J$ finishes later than u_J did, and so the overall finishing time of $u'_1, \ldots, u'_t$ is at least as early as the overall finishing time of $u_1, \ldots, u_t$.

We have therefore made $u_1, \ldots, u_t$, an ordering with the earliest possible finishing time, closer to $w_1, \ldots, w_t$ without making its finishing time any earlier. By repeating the process up to t

times, we can turn $u_1, \ldots, u_t$ into $w_1, \ldots, w_t$ without making its finishing time earlier, proving that the finishing time of $w_1, \ldots, w_t$ is as early as possible.

23. (10 marks) (Medium question.) Consider the following greedy algorithm for SAT.

Input : A CNF formula F with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$.
Output: A satisfying assignment if F is satisfiable, `No` otherwise.

```
1  begin
2      for i = 1 to n do
3          Let a_i be the number of clauses C_j containing x_i.
4          Let b_i be the number of clauses C_j containing x_i such that every other literal in C_j has
             already been set to False.
5          Let c_i be the number of clauses C_j containing ¬x_i.
6          Let d_i be the number of clauses C_j containing ¬x_i such that every other literal in C_j has
             already been set to False.
7          if b_i > 0 and d_i > 0 then
8              Return No.
9          else if b_i > 0 then
10             Set x_i to True.
11         else if d_i > 0 then
12             Set x_i to False.
13         else if a_i ≥ c_i then
14             Set x_i to True.
15         else
16             Set x_i to False.
17     Return the values of x_1, ..., x_n.
```

Give a counterexample to show that this algorithm does not work, and briefly explain it.

Solution: The following formula is a counterexample:

$$(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (\neg x_1 \vee x_4).$$

In any satisfying assignment, x_1 must be `False`, or one of the clauses $(\neg x_1 \vee \neg x_4)$ or $(\neg x_1 \vee x_4)$ would be false. However, the algorithm will set x_1 to `True`, since $a_1 = c_1 = 2$ and $b_1 = d_1 = 0$. Thus the algorithm will return `No`.

It remains only to note that there is a valid satisfying assignment (so the algorithm's answer is incorrect), namely $x_1 =$ `False`, $x_2 = x_3 = x_4 =$ `True`.