COMS20010 — 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) $n \in O(4^n)$.

   Solution: True, $n \in O(4^n)$.

   (b) (1 mark) $\log n \in O(n^{1/3})$.

   Solution: True, $\log n \in O(n^{1/3})$.

   (c) (1 mark) $n^{1.01} + 10^6 n \notin \omega(n^{1.01})$.

   Solution: True, $n^{1.01} + 10^6 n \in \Theta(n^{1.01})$.

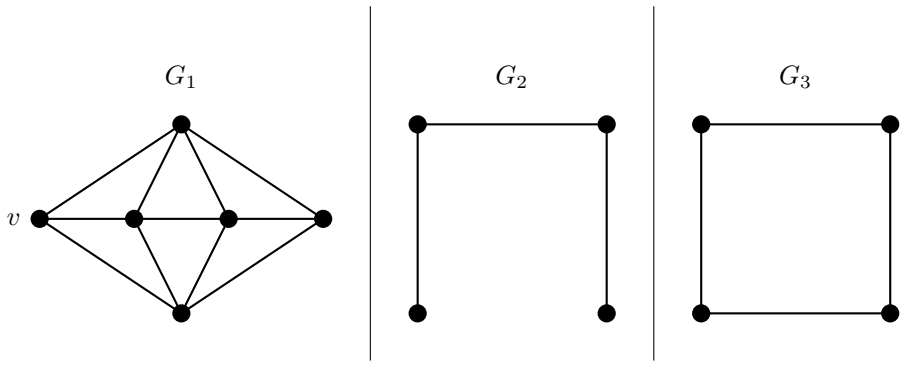
   (d) (1 mark) $mn^2 + m^2 n \in O(m^2 n^2)$ (where neither m nor n is a constant).

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

   (e) (1 mark) In any graph $G = (V, E)$, $|E| \in o(|V|^2)$.

   Solution: False, if G is e.g. a clique then $|E| = |V|(|V| - 1)/2 \in \Theta(|V|^2)$.

2. (5 marks) (Short question.) Consider the following graphs G_1, G_2, and G_3, and the indicated vertex v.



Mark each of the following statements true or false:

(a) (1 mark) G_1 is connected.

Solution: True. There's a path between any two vertices of G_1.

(b) (1 mark) G_3 contains a Hamilton cycle.

Solution: True. In fact, G_3 *is* a Hamilton cycle.

(c) (1 mark) G_1 contains an Euler walk from v back to itself.

Solution: False. G_1 contains an Euler walk from v to the rightmost vertex pictured, but not from v to itself as this would require every vertex to be of even degree and $d(v) = 3$.

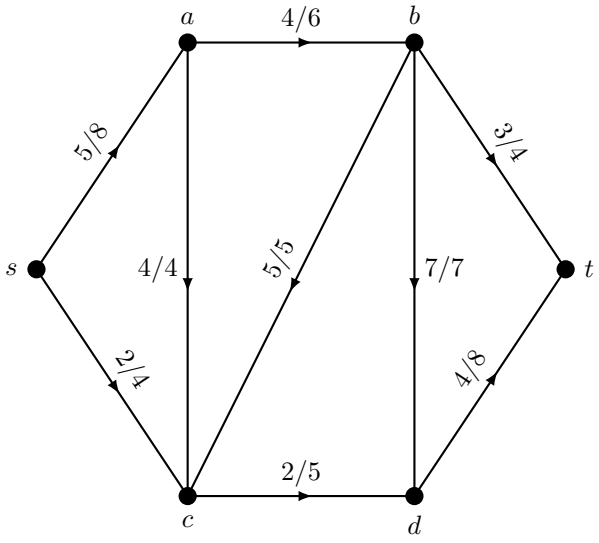
(d) (1 mark) G_1 contains a copy of G_2 as a (not necessarily induced) subgraph.

Solution: True. G_1 contains many paths of length 3, e.g. the path starting at v and proceeding rightwards.

(e) (1 mark) G_1 contains a copy of G_3 as an induced subgraph.

Solution: True. E.g. the outer cycle of G_1 is an induced copy of G_3.

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*, *scabt*, *scbt*, *scdbt*, and *scdt*.

4. (5 marks) (Short question.) In the distant future, the renowned Sol Ballet Corps is putting on a production to emphasise a spirit of love and togetherness between human colonies throughout the solar system. They have 4 dancers from Earth, 5 from Mars, 6 from the moon, and 7 from Ceres. They wish

to arrange a procession in which every pair of dancers **from different worlds** crosses the stage exactly once. For example, one dancer from Earth will cross alongside one dancer from Mars, but not a pair of dancers both from Earth. How many pairs of dancers will need to cross the stage in total? (**Hint:** Try modelling this as a graph and applying the handshaking lemma.)

A. 179.

B. 358.

C. 175.

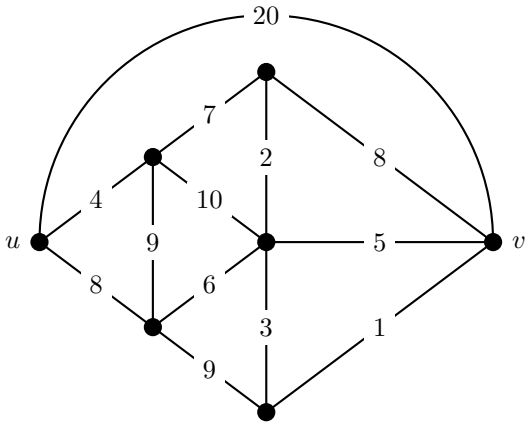
D. 350.

E. None of the above.

Solution: A — 179. Model this as a graph G in which each dancer is joined to all dancers from different worlds. Thus dancers from Earth have degree 18, dancers from Mars have degree 17, dancers from the moon have degree 16, and dancers from Ceres have degree 15. The pairs of dancers that cross the stage will be precisely the edges of the graph, and by the handshaking lemma we have

$$|E(G)| = \frac{1}{2} \sum_{v \in V(G)} d(v) = \frac{1}{2}\big(4 \cdot 18 + 5 \cdot 17 + 6 \cdot 16 + 5 \cdot 15\big) = 179.$$

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



What is the distance from u to v?

A. 16.

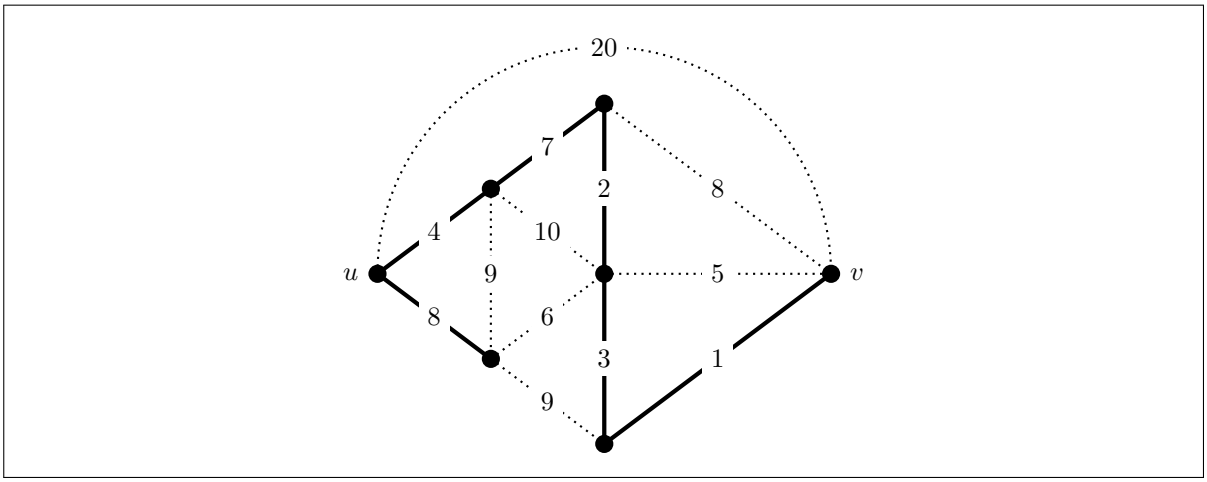
B. 17.

C. 18.

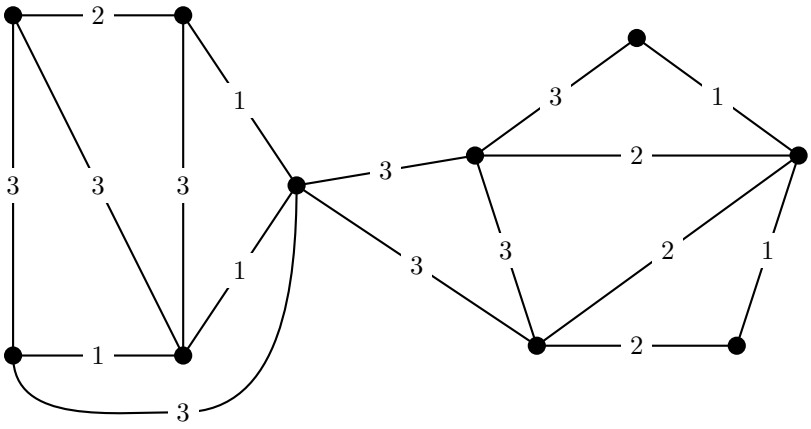
D. 19.

E. None of the above.

Solution: B — 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.

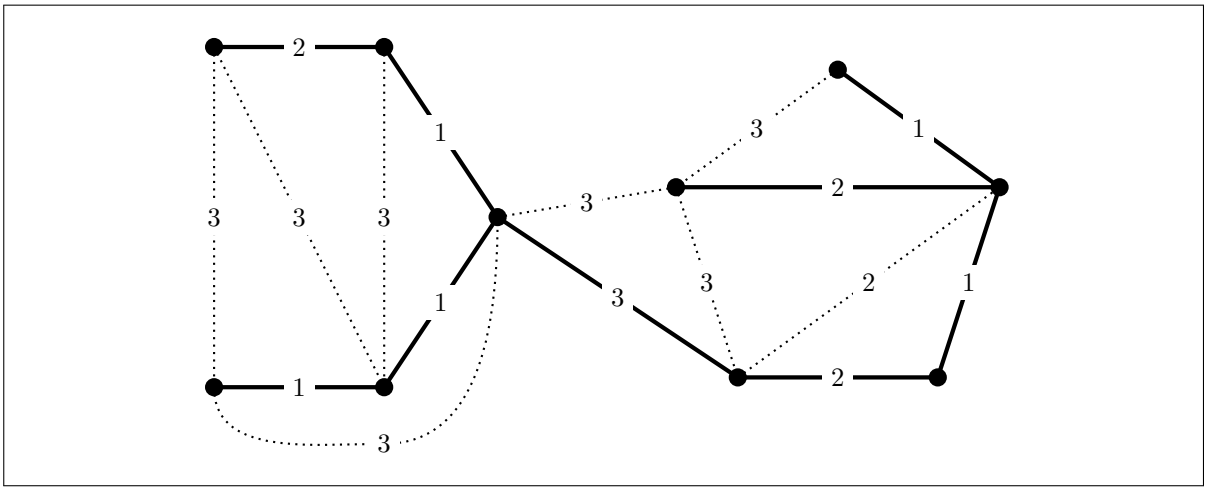
B. 13.

C. 14.

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

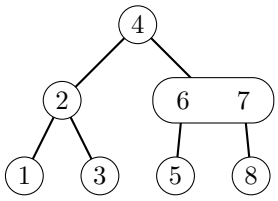
E. None of the above.

Solution: C — 14. One possible minimum spanning tree (of many) is pictured below.



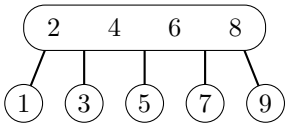
7. (5 marks) (Short question.) Which of the following are valid 2-3-4 trees?
[Implement as jumbled sentence, options "Valid" and "Invalid" for each tree.]

(a) (1 mark)



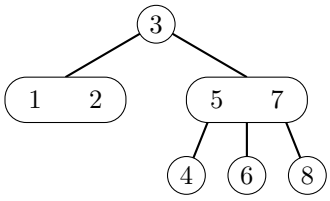
Solution: Invalid. This tree breaks perfect balance, because $(6, 7)$ is a non-leaf 3-node with fewer than 3 children.

(b) (1 mark)



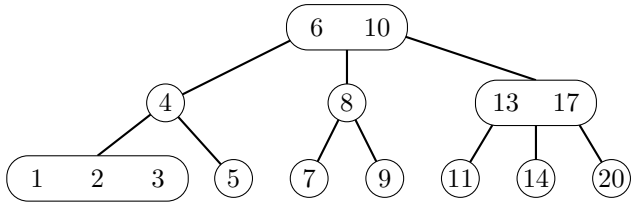
Solution: Invalid. The root is neither a 2-node, nor a 3-node, nor a 4-node.

(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) (4 5 6)

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. A user creates a new instance of the structure with `MakeUnionFind`$(1, \ldots, 100)$, then applies ten `Union` operations to the resulting data structure. Without knowing the values of those `Union` operations, what is the **greatest possible** depth of any tree component of the resulting data structure?

   A. 1.

   B. 2.

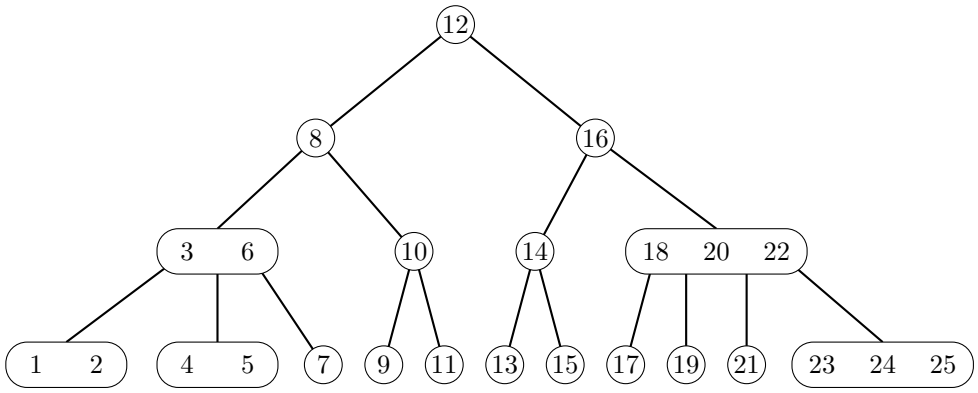
   C. 3.

   D. 4.

   E. None of the above.

Solution: C — 3. Recall the bound on depth from lectures: in order to make a depth-d component, you need to perform a union operation on two depth-$(d-1)$ components. So if you need t union operations to make a depth-$(d-1)$ component, you need $2t+1$ union operations to make a depth-d component. So you need 1 union operation to make a depth-1 component, 3 to make a depth-2 component, 7 to make a depth-3 component, and 15 to make a depth-4 component. We have $7 < 10 < 15$, so the answer is three.

9. (5 marks) (Medium question.) Consider the 2-3-4 tree below:



Let T be the result of first inserting the value 30 and then deleting the value 12, assuming that whenever the deletion algorithm has a choice between a "fuse" operation and a "transfer" operation it chooses to transfer. Calculate T.

(a) (1 mark) What is the depth of T?

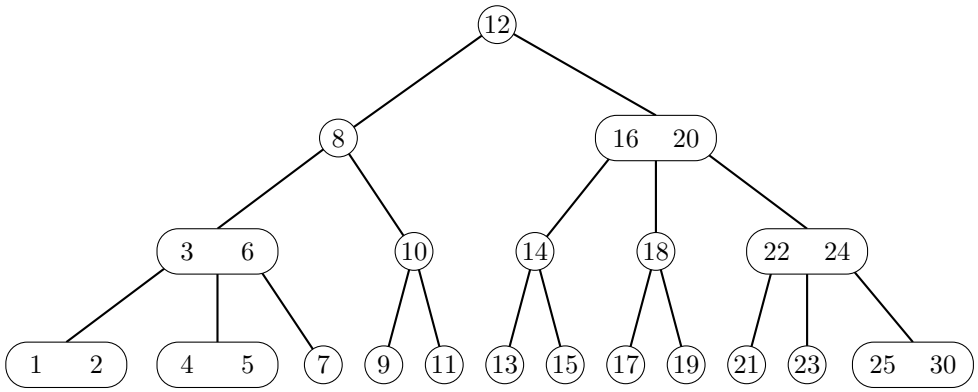
(b) (1 mark) How many children does the root of T have?

(c) (1 mark) How many 3-nodes does T have?

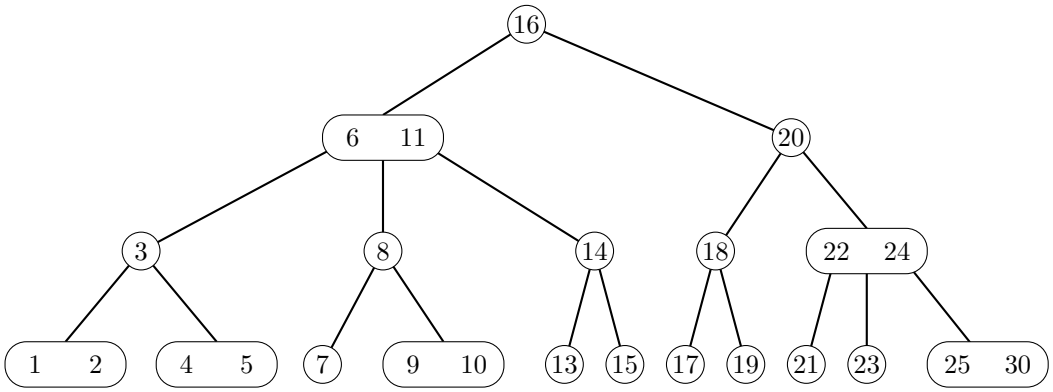
(d) (1 mark) How many 4-nodes does T have?

(e) (1 mark) While inserting 30 and deleting 12, how many fuse, transfer and split operations were performed in total?

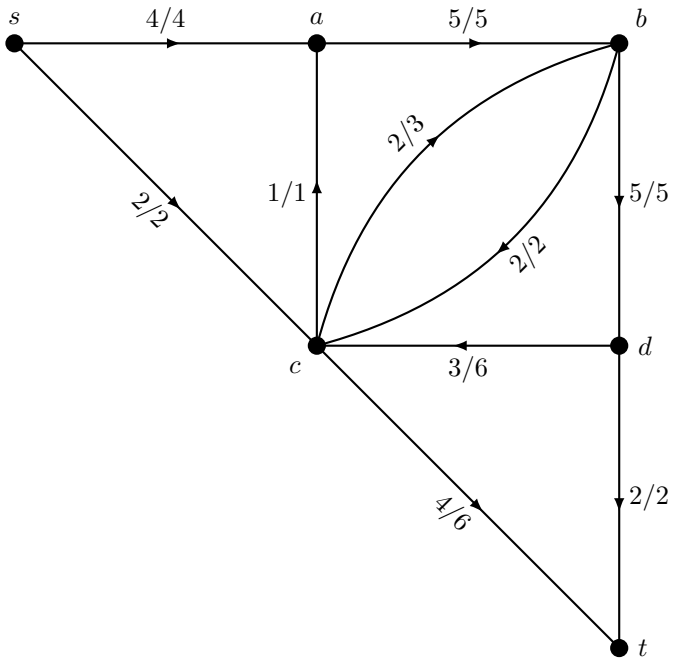
Solution: 3; 2; 6; 0; 5. Inserting 30 requires splitting $(18, 20, 22)$ and $(23, 24, 25)$, then inserting 30 into the newly-formed (25) node. The result looks like this:



Deleting 12 then requires finding the predecessor 11 by traversal, transferring 16 from $(16, 20)$ and transferring 6 from $(3, 6)$ on the way, fusing 9, 10 and 11 together, then deleting 11 from the resulting 3-node and overwriting 12 with 11. (Some of those transfers could have been fuses instead, but recall from the question that when the algorithm has a choice it will choose to transfer.) The result looks like this:



10. (5 marks) (Short question.) Consider the following flow network (G, c, s, t) with the following flow f:



Let $X = \{s, b, d\}$. What is $f^-(X)$?

A. 7.

B. 12.

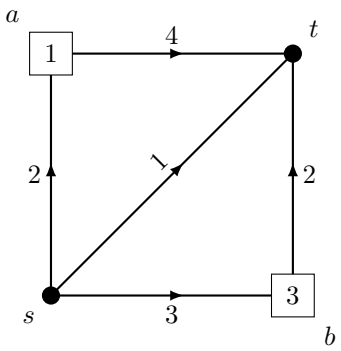
C. -6.

D. $f^-(X)$ is not meaningfully defined.

E. None of the above.

Solution: A — 7. $f^-(X)$ is defined for all sets X, whether they induce connected graphs or not, and is the total flow into X. The flow comes from the edges (a, b) and (c, b); the edge (b, d) does not contribute since it is inside X.

11. (5 marks) (Short question.) Consider the following vertex flow network with source s and sink t:



Which of the following is a valid flow f?

A. $f(s, a) = 1$; $f(s, b) = 3$; $f(s, t) = 1$; $f(a, t) = 1$; $f(b, t) = 2$.

B. $f(s,a) = 1$; $f(s,b) = 1$; $f(s,t) = 1$; $f(a,t) = 1$; $f(b,t) = 1$.

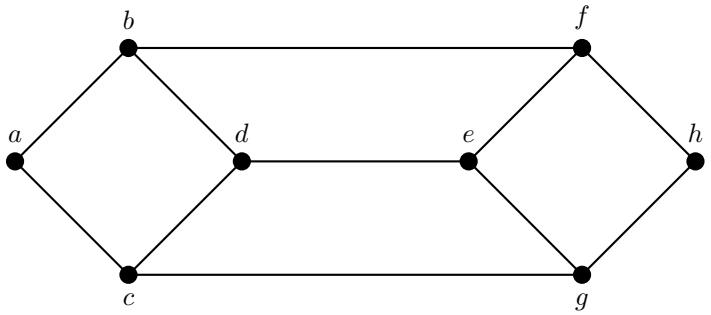
C. $f(s,a) = 2$; $f(s,b) = 2$; $f(s,t) = 2$; $f(a,t) = 2$; $f(b,t) = 1$.

D. $f(s,a) = 1$; $f(s,b) = 3$; $f(s,t) = 1$; $f(a,t) = 1$; $f(b,t) = 3$.

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

Solution: B is a valid flow. A violates conservation of flow at b, C violates the vertex capacity constraint at a, and D violates the edge capacity constraint at (b,t).

12. (5 marks) (Short question.) Consider the graph G shown below, and mark each statement true or false.



A. $\{a,d,e,h\}$ is a vertex cover of G.

Solution: False. The edges $\{b,f\}$ and $\{c,g\}$ are not covered.

B. $\{a,b,c,d,e,f,g,h\}$ is a vertex cover of G.

Solution: True.

C. $\{a,d,e,h\}$ is an independent set of G.

Solution: False. The edge $\{d,e\}$ is included.

D. $\{b,c,f,g\}$ is an independent set of G.

Solution: False. The edges $\{b,f\}$ and $\{c,g\}$ are included.

E. If $X \subseteq V(G)$ is an independent set of G, then $V(G) \setminus X$ is a vertex cover of G.

Solution: True. This is true for any graph G, and was covered in lectures (it's how we proved that IS $\leq_K$ VC in video 10-1).

13. (5 marks) (Short question.) Consider an instance of weighted interval scheduling with interval set $\mathcal{R} = [(1,5),(3,7),(6,10),(4,11),(7,12),(10,14),(11,16),(15,21),(18,23)]$ and weight function w given by

$$w(1,5) = 1, \quad w(3,7) = 3, \quad w(6,10) = 1, \quad w(4,11) = 5, \quad w(7,12) = 7,$$
$$w(10,14) = 6, \quad w(11,16) = 4, \quad w(15,21) = 4, \quad w(18,23) = 6.$$

What is the maximum possible weight of a compatible set?

A. 13.

B. 14.

C. 15.

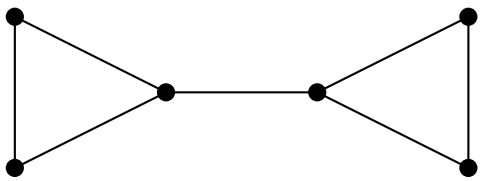
D. 16.

E. None of the above.

Solution: D − 16. The unique optimal set is $\{(3,7),(7,12),(18,23)\}$. In more detail, writing $\mathrm{OPT}(i)$ for the weight of an optimal solution on the first i intervals sorted in ascending order of finish time:

$$\mathrm{OPT}(1)=1, \quad \mathrm{OPT}(2)=3, \quad \mathrm{OPT}(3)=3, \quad \mathrm{OPT}(4)=5, \quad \mathrm{OPT}(5)=10,$$
$$\mathrm{OPT}(6)=10, \quad \mathrm{OPT}(7)=10, \quad \mathrm{OPT}(8)=14, \quad \mathrm{OPT}(9)=16.$$

14. (5 marks) (Medium question.) Consider the problem of finding a minimum vertex cover on the graph below.



As described in lectures, this problem can be reduced to integer linear programming. Here is one such possible reduction, with some parts missing.

$$\begin{aligned}
x_1+x_2+x_3+x_4+x_5+x_6 &\to \text{BLANK subject to:}\\
x_1+x_5 &\geq a,\\
x_1+\text{BLANK} &\geq a,\\
\text{BLANK}+x_2 &\geq a,\\
\text{BLANK}+x_6 &\geq a,\\
x_6+x_4 &\geq a,\\
x_5+x_4 &\geq a,\\
x_3+x_5 &\geq a,\\
0\leq x_1,\ldots,x_6 &\leq a,\\
x_1,\ldots,x_6 &\in \mathbb{N},\\
a &= \text{BLANK}.
\end{aligned}$$

Fill in the blanks to make a valid reduction. Note that the correspondence between variables in the linear program and vertices in the graph is not given. This is not a mistake in the question; you should be able to fill in the blanks without it. (Any valid solution will get full marks.)

Options for blanks: min, max, x_1, x_2, x_3, x_4, x_5, x_6, 0, 1 and 2.

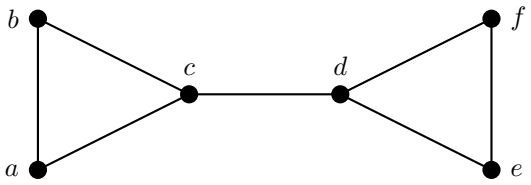
Solution: In order, the blanks read: min, x_3, x_4, x_2, 1. This is the only valid solution. The unique correct reduction is as follows.

$$x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \to \text{min subject to:}$$

$$\begin{aligned} x_1 + x_5 \geq a, \quad & x_1 + x_3 \geq a, \quad & x_4 + x_2 \geq a, \quad & x_2 + x_6 \geq a, \\ x_6 + x_4 \geq a, \quad & x_5 + x_4 \geq a, \quad & x_3 + x_5 \geq a, \quad & \\ 0 \leq x_1, \ldots, x_6 \leq a, \quad & x_1, \ldots, x_6 \in \mathbb{N}, \quad & a = 1. & \end{aligned}$$

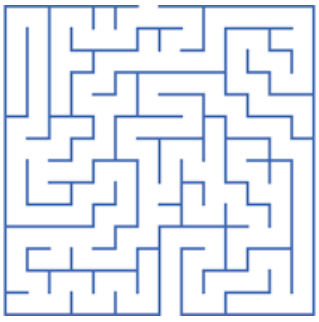
The first and last blank are immediate from the form of the reduction from lectures, and the middle three blanks should make the equations $x_i + x_j \geq 1$ correspond to the edges $\{x_i, x_j\}$ of the graph. The difficult part of the question is working out which variables should correspond to which vertices. Here is one way of deriving that correspondence.

Consider the labelled version of the graph below.



From the parts of the equation you're given, x_5 must correspond to a degree-3 vertex with neighbourhood $\{x_1, x_3, x_4\}$; let's say x_5 corresponds to c, since c and d are equivalent by symmmetry. Then x_4 sends out an edge to $x_6 \notin N(x_5)$, so x_4 must correspond to d. Then x_1 and x_3, as the other neighbours of x_5, must correspond to a and b in some order, so since $\{a, b\}$ is an edge we require $x_1 + x_2 \geq 1$. Moreover, x_2 and x_6 must correspond to e and f in some order, so we must have $x_2 + x_6 \geq 1$ and $x_4 + x_2 \geq 1$.

15. (10 marks) (Short question.) Consider a maze, as shown below.



Explain very briefly how you would find the shortest path from the entrance to the exit in $O(n \log n)$ time, where n is the number of junctions in the maze. You may assume that as part of the input, you have access to:

- A list J of all junctions in the maze plus the entrance and exit;
- A list C of all corridors directly connecting two junctions, along with their lengths.

You may assume $|C| = O(n)$; this is true for any 2D maze. You may use any algorithm covered in the course without explaining its implementation, and you do not need to prove your algorithm works.

Solution: Form a weighted graph whose vertices are given by J, whose edges are given by C, and whose weights are given by the corridor lengths. Then apply Dijkstra's algorithm to find a shortest path from the entrance to the exit. The running time is $O((|J| + |C|) \log |J|) = O(n \log n)$.

16. (10 marks) In a graph $G = (V, E)$, we say a vertex $v \in V$ is *critical* if G is connected and $G - v$ has exactly two components, i.e. if deleting v and all its edges from G splits G into two parts.

    (a) (5 marks) (Short question.) For $n \geq 3$, what is the **least** number of edges an n-vertex graph can have while still having a critical vertex, and why?

    Solution: By the fundamental lemma of trees, any n-vertex graph with fewer than $n-1$ edges must be disconnected, so the answer is at least $n - 1$. A length-$(n-1)$ path has n vertices, $n-1$ edges, and all vertices other than the endpoints are critical, so the answer is at least $n-1$. Putting the two statements together, the answer is exactly $n - 1$.

    (b) (5 marks) (Medium question.) For $n \geq 3$, assuming n is odd, what is the **greatest** number of edges an n-vertex graph can have while still having a critical vertex, and why? You may wish to use the following algebraic fact, which you do not need to prove: For all $a, b, t > 0$ satisfying $a + b \leq t$ we have $a^2 + b^2 \leq t^2/2$, i.e. $a^2 + b^2$ is maximised when $a = b = t/2$.

    Solution: Suppose $G = (V, E)$ is an n-vertex graph with a critical vertex v, and let $G - v$ be the graph formed by removing v from G. Then since v is critical, $G - v$ has two components; call these C_1 and C_2, let $c_1 = |V(C_1)|$, and let $c_2 = |V(C_2)|$, so that $c_1 + c_2 = n - 1$. There are no edges between C_1 and C_2, at most $\binom{c_i}{2}$ edges internal to each C_i, and at most $n - 1$ edges between v and the rest of the graph, so we have

    $$|E(G)| \leq \binom{c_1}{2} + \binom{c_2}{2} + n - 1 = \frac{c_1^2 + c_2^2 - c_1 - c_2}{2} + n - 1.$$

    We have $c_1 + c_2 = n - 1$, so it follows that

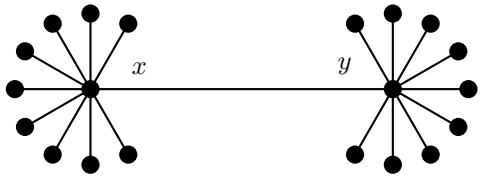
    $$|E(G)| \leq \frac{c_1^2 + c_2^2}{2} + \frac{n-1}{2}.$$

    Moreover, by the algebraic fact applied with $a = c_1$, $b = c_2$ and $t = n - 1$, we have $c_1^2 + c_2^2 \leq (n-1)^2/2$. It follows that

    $$|E(G)| \leq \frac{(n-1)^2}{4} + \frac{n-1}{2} = \frac{n^2 - 1}{4}.$$

    We can achieve equality in all these bounds by taking C_1 and C_2 to be complete graphs, taking v joined to every vertex in C_1 and C_2, and taking $c_1 = c_2 = (n-1)/2$. So the greatest possible number of edges is $(n^2 - 1)/4$.

17. (10 marks) (a) (5 marks) (Short question.) How large is a maximum matching for the graph below? Briefly explain your reasoning.



Solution: Since edges of a matching are disjoint, any matching must contain at most one edge incident to x and at most one edge incident to y, so a maximum matching contains at most two edges. Conversely, any two edges incident to x and y form a matching as long as neither of them is $\{x, y\}$, so a two-edge matching exists and a maximum matching contains at least two edges. Thus a maximum matching contains exactly two edges.

(b) (5 marks) (Medium question.) Give an example of a graph with no perfect matching in which every vertex has degree 16, and briefly explain why it has no perfect matching. Your graph does not have to be bipartite, and you do not have to draw it.

Solution: One example is the complete graph on 17 vertices, in which every vertex is joined to every other vertex by an edge. The edges in a matching are disjoint, so any matching covers only an even number of vertices; since 17 is odd, it follows that no matching can cover all 17 vertices.

18. (10 marks) (Medium question.) After making some questionable life choices, you are developing marketing materials for a company selling stock trading advice to the general public. To illustrate the power of good financial advice, you want to know how much money someone could theoretically have made on Gamestop shares if they had initially invested in one share and then sold and re-bought a limited number of times with perfect foreknowledge. ("With just five smart trades at the right moments, you could have earned over five hundred squillion pounds! Trade the smart way. Trade with StonksCo!")

You have access to a list $p_1, \ldots, p_n$ of Gamestop's share prices at the start of each day of historical trading; you may assume the price stays constant throughout the day and is unaffected by your theoretical trades, and that you can buy fractions of shares. You are also given a number k; this is the maximum number of trades you are allowed to make after buying the initial share on day 1. The only possible trades you need to consider are selling all your shares (when the price is higher) or reinvesting all your money into buying new shares (when the price is lower). Your goal is to find the maximum possible profit, in either stocks or cash. Fill in the blanks of the following dynamic programming algorithm to solve the problem.

Here, $H[i, j]$ will contain the maximum amount of money available from owning a single share at the start of day j with i trades remaining. You should infer the meaning of $N[i, j]$ from the algorithm.

```
1  begin
2      Initialise H and N to empty two-dimensional arrays indexed by {0,...,k} × {1,...,n}.
3      Initialise H[i, n] = p_n for all i ≤ k.
4      Initialise N[i, n] = p_n for all i ≤ k.
5      Initialise H[0, j] = p_n for all j ≤ n.
6      Initialise N[0, j] = p_j for all j ≤ n.
7      for i = 0 to k do
8          for j = n − 1 to 1 do
9              Let H[i, j] = max{BLANK, max{N[i − 1, k] : k ≥ j + 1}}.
10             Let N[i, j] = max{BLANK, max{ BLANK / BLANK · BLANK[i − 1, k] : k ≥ j + 1 }}.
11     Return H[1, k] − p_1.
```

Options for blanks: p_1, p_i, p_j, p_k, p_n, H, N.

Solution: In order, the blanks read: p_n, p_j, p_j, p_k, H. The full algorithm is:

```
1  begin
2      Initialise H and N to empty two-dimensional arrays indexed by {0,...,k} × {1,...,n}.
3      Initialise H[i,n] = p_n for all i ≤ k.
4      Initialise N[i,n] = p_n for all i ≤ k.
5      Initialise H[0,j] = p_n for all j ≤ n.
6      Initialise N[0,j] = p_j for all j ≤ n.
7      for i = 0 to k do
8          for j = n − 1 to 1 do
9              Let H[i,j] = max{p_n, max{N[i−1,k]: k ≥ j+1}}.
10             Let N[i,j] = max{p_j, max{(p_j/p_k) · H[i−1,k]: k ≥ j+1}}.
11     Return H[1,k] − p_1.
```

$N[i,j]$ is the maximum profit available given enough money to buy one share on day j with i trades remaining. With no trades remaining or no days remaining, your profit will be whatever you currently have — either the final price of the share you're holding, or the value of the money you're holding. Thus, for all $i \leq k$ and $j \leq n$:

$$H[i,n] = H[0,j] = p_n, \qquad N[i,n] = p_i; \qquad N[0,j] = p_j.$$

If you are holding a share on day j with at least one trade remaining, then you can either hold onto it until day n (with value p_n) or use a trade to sell it at some subsequent day k; thus for all $i > 0$ and $j < n$, we have

$$H[i,j] = \max\{p_n, \max\{N[i-1,k] \colon k \geq j+1\}\}.$$

If you are holding enough money to buy a single share on day j (i.e. £p_j), then you can either hold onto it until day n (with value p_j) or use a trade to buy shares on some subsequent day k. In the latter case, you will be able to buy p_k/p_j shares, so you will get p_k/p_j times as much profit as you would from a single share. Thus for all $i > 0$ and $j < n$, we have

$$N[i,j] = \max\{p_j, \max\{(p_j/p_k) \cdot H[i-1,k] \colon k \geq j+1\}\}.$$

The correct algorithm simply implements these recurrence relations.

Section 2

19. (10 marks) (Medium question.) Consider the following greedy algorithm `heuristicIS`, intended to find a maximum independent set in an arbitrary graph in polynomial time.

```
Input  : A graph G with vertex set {1, 2, ..., n}.
Output: An independent set X of G which we hope is maximum.
begin
    Let X ← ∅.
    while V(G) ≠ ∅ do
        Let δ ← min{d(v): v ∈ V}.
        Let x be the lowest-numbered vertex with d(x) = δ.
        X ← X ∪ {x}.
        G ← G − x − N(x), the graph with vertex set V(G) \ (N(x) ∪ {x}) and edge set
          {e ∈ E(G): e ∩ (N(x) ∪ {x}) = ∅}.
    Return the value of X.
```

(a) (1 mark) Why, without looking at the implementation of `heuristicIS`, can we be almost certain that it doesn't actually return a maximum independent set for all input graphs?

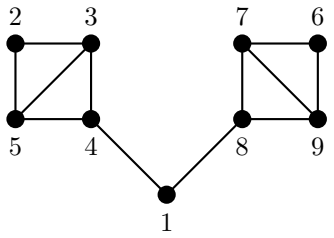
Solution: If `heuristicIS` worked then it would allow us to solve IS, the decision problem which asks whether G contains an independent set of a given size, in polynomial time. We proved in lectures that this problem is NP-complete, so this would imply P = NP. This is quite unlikely to be true, and incredibly unlikely to be proved in such a simple fashion even if it is true.

(b) (4 marks) Give an example of a graph G and a *key vertex* $v \in V(G)$ such that every vertex in G has degree at least two, and such that every maximum independent set of G contains v.

Solution: One example is given by a size-4 clique missing an edge: $V(G) = \{1, 2, 3, 4\}$, $E(G) = \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}, \{1, 3\}\}$, $v = 2$. Indeed, 134 is a triangle so any independent set containing 2 can have at most one vertex, but $\{2, 4\}$ is an independent set containing two vertices.

(c) (5 marks) Using your answer to part (b) or otherwise, give an example of a graph on which `heuristicIS` fails to return a maximum independent set. Explain why your example works.

Solution: Form G as follows. Let G_1 and G_2 be two copies of your answers to part (b), and let t be the size of a maximum independent set in G_1. Let v_1 be the key vertex of G_1, and let v_2 be the key vertex of G_2. Add a new vertex w joined to both v_1 and v_2. Set $w = 1$, so that `heuristicIS` will pick w to add to the independent set first. Then any independent set in G_1 not containing v_1 must be of size at most $t - 1$, and likewise for G_2, so `heuristicIS` must return a set of size at most $2(t-1)+1 = 2t-1$. But G contains at least one independent set of size $2t$, formed by combining maximum independent sets of G_1 and G_2. See the picture below for an example with $t = 2$, $v_1 = 4$ and $v_2 = 8$.



20. (10 marks) (a) (5 marks) (Long question.) Let $G = (V, E)$ be a connected graph with edge weights given by $w: E \to \mathbb{R}$. You may assume that every edge gets a different weight. Let C be a cycle in

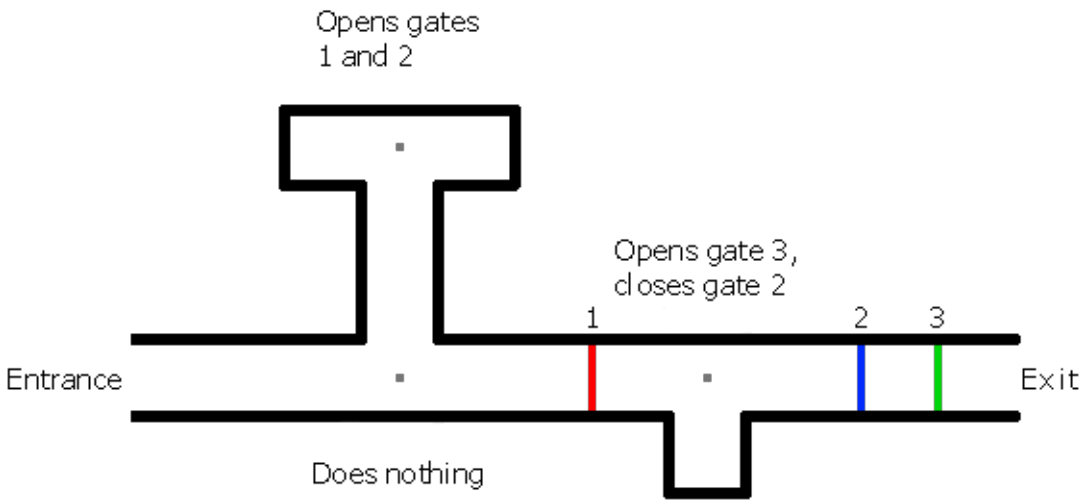
G, and let e be the highest-weight edge in C. Prove that no minimum spanning tree of G contains e. You may assume any consequence of the Fundamental Lemma of Trees without proof as long as you state it clearly.

Solution: Write $f = \{x_1, x_2\}$, and suppose that T is a spanning tree of G which contains e. By the Fundamental Lemma of Trees, $T - e$ has two components; call these T_1 and T_2, with $x_1 \in V(T_1)$ and $x_2 \in V(T_2)$. Imagine walking along C the long way from x_1 to x_2 (i.e. along the path which does not use e). We start in $V(T_1)$ and end in $V(T_2)$, so at some point we must cross from T_1 to T_2. Let f be this crossing edge with one vertex in T_1 and one vertex in T_2. We have $w(f) < w(e)$ since f is on C. Form another tree T' by removing e from T and replacing it with f. We know that T' is a spanning tree by the Fundamental Lemma of Trees, since it has $n - 1$ edges and it's connected, and the total weight of T' is less than that of T since $w(f) < w(e)$. Hence T was not a minimum spanning tree of G.

(b) (5 marks) (Medium question.) Using the result stated in part (a) or otherwise, give an algorithm which, given an n-vertex connected graph $G = (V, E)$ in adjacency list format with $|E| = n + 50$, outputs a minimum spanning tree in $\boldsymbol{O(n)}$ time. Briefly explain why your algorithm works and why it runs in $O(n)$ time.

Solution: The algorithm repeatedly removes highest-weight edges from cycles until no cycles remain in the graph. By part (a), the edges we remove are not contained in any minimum spanning tree of G. Moreover, we can never disconnect a graph by removing an edge from a cycle, so by the Fundamental Lemma of Trees, after we have applied this procedure 51 times we will be left with an $(n-1)$-edge connected graph on n vertices, which is a tree (and therefore has no more cycles to remove). The minimum spanning tree of a tree is itself, so the algorithm will indeed return a minimum spanning tree of G. We can find cycles using depth-first or breadth-first search in $O(|E|)$ time, we can find the highest-weight edge on a given cycle in $O(n)$ time, and we repeat the process $51 \in O(1)$ times, so the total running time is $O(n + |E|) = O(n)$.

21. (10 marks) (Long question.) As in a previous question, consider a maze. This time, however, every junction in the maze contains a button, and some corridors are blocked by gates. On reaching a button, you have the option of pressing it as many times as you like; this will close some gates, open others, and toggle others (i.e. they open if they are closed and close if they are opened). See the picture below for an example.



Briefly describe an algorithm to find the shortest route from the entrance to the exit in $O(2^g n(g+\log n))$ time, pressing as many buttons as you like, where n is the number of junctions in the maze and g is the number of gates. You may assume the input is given as:

- A list J of all junctions in the maze plus the entrance and exit and the behaviour of their buttons;
- A list C of all corridors directly connecting two junctions, along with their lengths;
- A list G of all gates, along with their initial states (open or closed) and which corridors they are contained in.

You may assume $|C| = O(n)$; this is true for any valid 2D maze. You may use any algorithm covered in the course without explaining its implementation, and you do not need to prove your algorithm works. (**Hint:** Try reducing the problem to finding a single shortest path between two vertices in a graph.)

Solution: Let $c_1, \ldots, c_{2^g}$ be the list of all 2^g possible configurations of gate states, so that c_i labels elements of G with either "open" or "closed". For any given configuration c_i of the gates, there is a natural edge-weighted graph G_i whose vertices are the junctions of the maze, whose edges are the corridors not blocked by gates in configuration c_i, and whose edge weights are given by the corridor lengths. A shortest path between two points in this graph corresponds to a shortest path between those two points in the maze without pressing any buttons.

We now form the maze graph G by joining the graphs $G_1, \ldots, G_{2^g}$ as follows. We start with disjoint copies of all 2^g graphs. Then, for each junction v of the maze in each graph G_i, let c_j be the gate configuration resulting by pressing the button at v; we add a zero-weight edge from v in G_i to v in G_j. Traversing this edge in a path corresponds to pressing the button in the maze. We then add weight-zero edges from the exit vertex in each graph G_i to a new exit vertex z. Thus if the initial gate state is c_1 (say), then any path from the entrance to the exit in the maze corresponds to a unique path from the entrance vertex in G_1 to the exit vertex z. We can therefore find the shortest path through the maze by applying Dijkstra's algorithm to G. Since G has $2^g n$ vertices, since there are $O(n)$ corridors in the maze, and since we added only one edge to each corridor, Dijkstra's algorithm runs in time

$$O((2^g n + |E(G)|)\log(2^g n)) = O(2^g n \log(2^g n)) = O(2^g n(g + \log n)),$$

as required.

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) (Medium question.) You are attempting to form a committee of people to represent Computer Science in the 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.)

    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.

(b) (5 marks) (Long question.) Let $G = (V, E)$ be an n-vertex connected graph. For each $v \in V$, let T_v be a BFS tree of G rooted at v. Let $d_G(x, y)$ denote the distance between x and y in G, and let $d_v(x, y)$ denote the distance between x and y in T_v. Let

$$\sigma(G) = \sum_{(x,y)\in V} d_G(x, y), \qquad \sigma_v = \sum_{(x,y)\in V} d_v(x, y),$$

where both sums are over all **ordered** pairs x, y (so (x, y) and (y, x) are two different terms). Prove that $\sum_{v\in V} \sigma(T_v) \leq 2(n-1)\sigma(G)$.

Solution: Since T_v is a BFS tree rooted at v, we have $d_v(v, x) = d_G(v, x)$ for all $x \in V$. Moreover, since graph distances satisfy the triangle equality, for all $x, y \in V$ we have

$$d_v(x, y) \leq d_v(x, v) + d_v(x, y).$$

Breaking the definition of σ_v into terms with $v \in \{x, y\}$ and terms with $v \notin \{x, y\}$, and remembering that the sum is over pairs, it follows that

$$\begin{aligned}\sigma_v &= 2\sum_{x\in V} d_G(v, x) + \sum_{\substack{(x,y)\in V \\ v\notin\{x,y\}}} d_v(x, y) \\ &\leq 2\sum_{x\in V} d_G(v, x) + \sum_{\substack{(x,y)\in V \\ v\notin\{x,y\}}} \big(d_G(v, x) + d_G(v, y)\big).\end{aligned}$$

Each term $d_G(v, x)$ appears in the right-hand sum $2(n-2)$ times, so

$$\sigma_v \leq 2\sum_{x\in V} d_G(v, x) + 2(n-2) \sum_{x\in V\setminus\{x\}} d_G(v, x) = 2(n-1) \sum_{x\in V} d_G(v, x).$$

Summing both sides over all $v \in V$, we obtain

$$\sum_{v\in V} \sigma_v \leq 2(n-1)\sigma(G),$$

as required.

Optional note: At first glance this question looks like arbitrary algebra, but it's actually the first step of a pretty nice result. Dividing through by n, we see that the average value $\frac{1}{n}\sum_v \sigma_v$ of σ_v, taken over all the BFS trees, is at most $2(1-\frac{1}{n})\sigma(G)$. It follows that there must be some specific v with $\sigma_v \leq 2\sigma(G)$. In other words, despite T_v only containing a tiny proportion of G's edges, distances in T_v are on average only a factor of 2 worse than they were in G. This is called a "low-stretch spanning tree", and it's a pretty useful construction for when you want to approximate a large dense graph by a large sparse graph that you can work with faster.