UNIVERSITY OF BRISTOL

January 2021 Examination Period

Department of Computer Science

2nd Year Examination for the Degrees of
Bachelor in Computer Science
Master of Engineering in Computer Science
Master of Science in Computer Science

COMS20010
Algorithms II

TIME ALLOWED:
2 Hours
plus 30 minutes to allow for collation and uploading of answers.

Answers

Other Instructions

1. You may access any materials available on the COMS20010 unit page and any materials which you have created yourself, but no others.
2. You may use a calculator if you wish.

Section 1 — Short-answer questions (75 marks)

You do not need to justify your answers for any of the questions in this section, and you will not receive partial credit for showing your reasoning. Just write your answers down in the shortest form possible, e.g. "A" for multiple-choice questions, "True" for true/false questions, or "23" for numerical questions. If you do display working, circle or otherwise indicate your final answer, as if it cannot be identified then the question will not be marked.

Question 1 (5 marks)

(Short question.) For **each** of the following statements, identify whether it is true or false.

(a) $n \in O(n^2)$. (1 mark)

Solution: True.

(b) $n \in \omega(\log^2 n)$. (1 mark)

Solution: True.

(c) $n^2 + 50n - 12 \in O(\frac{1}{2}n^2 - n + 100)$. (1 mark)

Solution: True.

(d) $5^n \in \Theta(6^n)$. (1 mark)

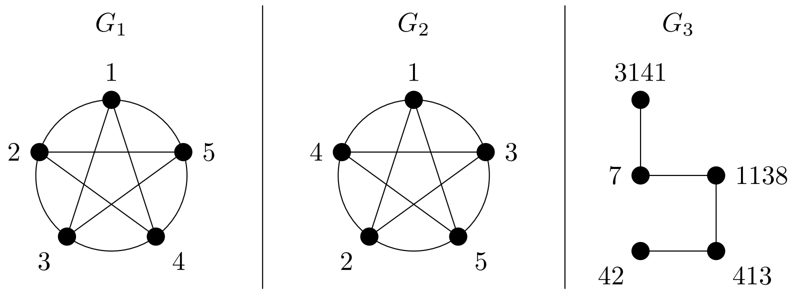
Solution: False.

(e) $mn^2 + m^2n \in O(m^{100}n)$. (1 mark)

Solution: False.

Question 2 (5 marks)

(Short question.) Consider the three graphs G_1, G_2 and G_3 shown below.



For **each** of the following statements, identify whether it is true or false.

(a) G_1 is equal to G_2. (1 mark)

Solution: True.

(b) G_2 is isomorphic to G_2. (1 mark)

Solution: True.

(c) G_1 contains a subgraph isomorphic to G_3. (1 mark)

Solution: True.

(d) G_1 contains an induced subgraph isomorphic to G_3. (1 mark)

Solution: False.

(e) G_1 contains an Euler walk from some vertex back to itself. (1 mark)

Solution: True.

Question 3 (5 marks)

(Short question.) You are working on a recently-established competitor to Google Maps, and you have been asked to estimate storage requirements for adding the small landlocked nation of Tropico to the service. Tropico's road network will be stored internally as a graph: Each junction and dead end is a vertex, and the roads joining them are edges. After doing some research, you discover that Tropico's road network has 100 dead ends, 350 3-way junctions, 50 4-way crossroads, and one terrifying central roundabout with 20 separate exits (which you should consider as a 20-way junction). How many edges will the resulting graph have? Choose **one** of the following options.

A. 645.

B. 685.

C. 1290.

D. 1370.

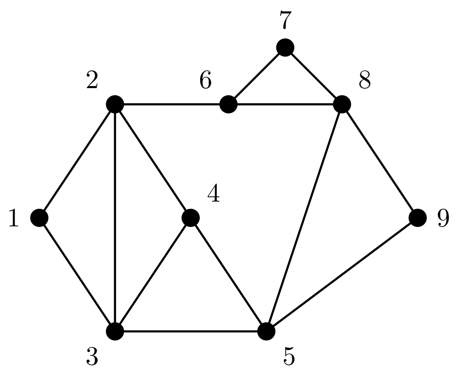
E. None of the above.

Solution: B — 685. By the handshaking lemma, for all graphs G we have $|E(G)| = \frac{1}{2}\sum_{v\in V(G)} d(v)$. Here, this expression becomes

$$|E(G)| = \frac{1}{2}\Big(100\cdot 1 + 350\cdot 3 + 50\cdot 4 + 1\cdot 20\Big) = 685.$$

Question 4 (5 marks)

(Short question.) Consider a depth-first search in the following graph starting from vertex 1.



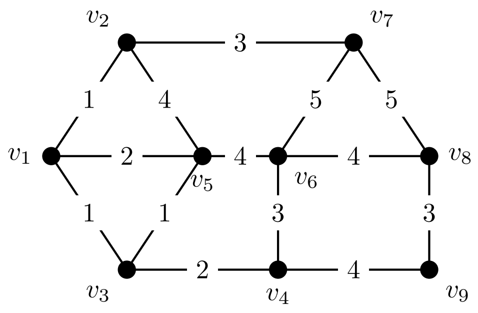
In the implementation of depth-first search given in lectures, we say vertex i is *explored* when `explored`$[i]$ is set to 1. Which vertex will be explored sixth? Assume that whenever the search has a choice of two or more vertices to visit next, it picks the vertex with the lowest number first.

Solution: 8. DFS will traverse edges in the order:

$$\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,8\},\{8,6\},\{6,7\},\{8,9\}.$$

Question 5 (5 marks)

(Short question.) Consider the following edge-weighted graph.



Which of the following edges will **always** be selected as part of a minimum spanning tree by Prim's algorithm starting from v_1, no matter how it breaks ties? Choose **one** of the following options.

A. $\{v_3, v_5\}$.

B. $\{v_6, v_8\}$.

C. $\{v_8, v_9\}$.

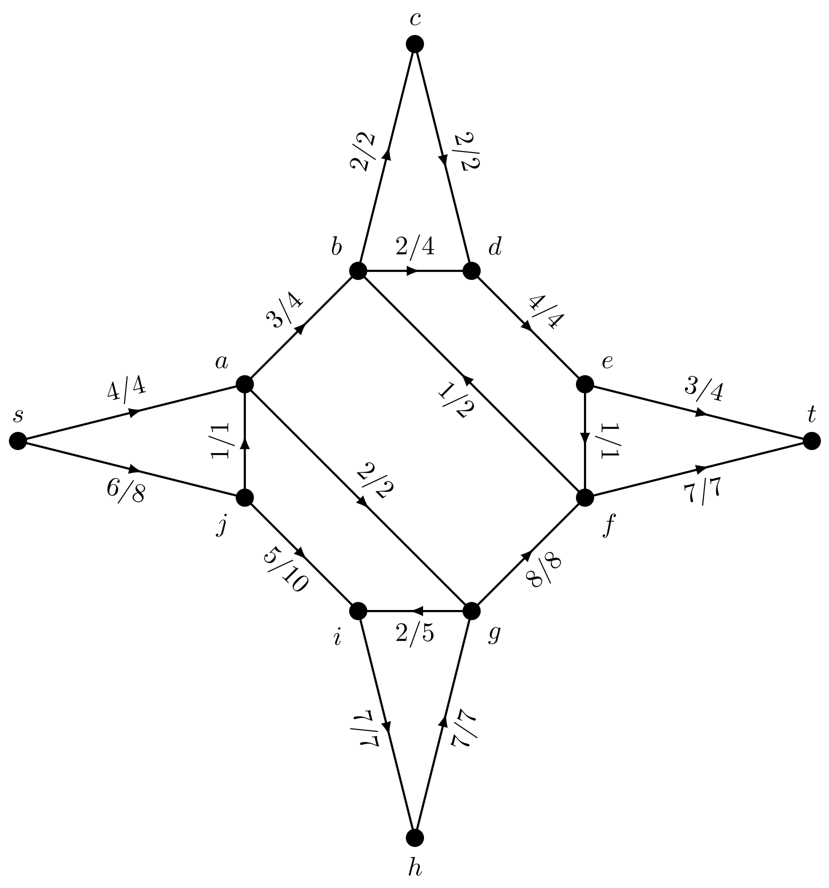
D. More than one of these edges will always be selected.

E. None of these edges will always be selected.

Solution: D — both $\{v_3, v_5\}$ and $\{v_8, v_9\}$ will always be selected. Prim's algorithm will first add the weight-1 edges $\{v_1, v_3\}$, $\{v_3, v_5\}$ and $\{v_1, v_2\}$ in some order. It will then add the weight-2 edge $\{v_3, v_4\}$. It will then add the weight-3 edges $\{v_4, v_6\}$, $\{v_2, v_7\}$ and $\{v_8, v_9\}$ in some order. It will then add either $\{v_6, v_8\}$ or $\{v_4, v_9\}$, followed by either $\{v_6, v_7\}$ or $\{v_7, v_8\}$. So $\{v_3, v_5\}$ and $\{v_8, v_9\}$ are always selected, and $\{v_6, v_8\}$ may or may not be selected depending on how the weight-4 tie is broken.

Question 6 (5 marks)

(Short question.) Consider the following vertex flow network with source s, sink t and flow f.



(a) What is the value of f? (1 mark)

Solution: 10.

(b) What is $f^-(a)$? (1 mark)

Solution: 5.

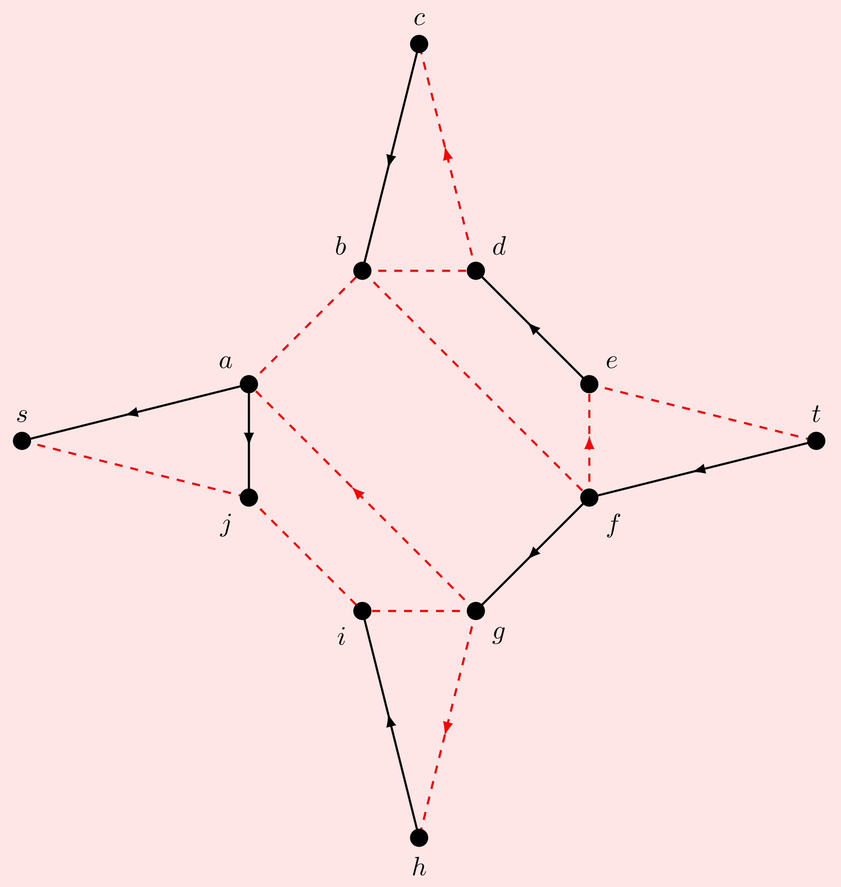
(c) Is $(\{s, b, e\}, \{a, c, d, f, g, h, i, j, t\})$ a valid cut? (1 mark)

(cont.)

Solution: Yes.

(d) Give an augmenting path for f. (2 marks)

Solution: $\boldsymbol{sjigabfet}$. This is the unique augmenting path — here is the residual graph, with a BFS tree from s to t highlighted in red. (Here bidirected edges are drawn as undirected.)



Question 7 (5 marks)

(Short question.) Which of the following CNF formulae are satisfiable?

(a) $x \vee \neg x$. (1 mark)

Solution: Satisfiable, e.g. $x =$ True.

(b) $\neg a \wedge (a \vee \neg c) \wedge (a \vee c)$. (1 mark)

Solution: Unsatisfiable. a must be False to satisfy the first clause, so c must be False to satisfy the second clause, and in that case the third clause can't be satisfied.

(c) $(x \vee y) \wedge (x \vee \neg y)$. (1 mark)

Qu. continues . . .

Solution: Satisfiable, e.g. $x = y =$ True.

(d) $(\neg a \vee \neg b \vee \neg c) \wedge (a \vee \neg b \vee \neg c) \wedge (\neg a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee c) \wedge (a \vee b \vee c)$.
(1 mark)

Solution: Satisfiable, e.g. $a =$ True, $b = c =$ False.

(e) $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3) \wedge \cdots \wedge (x_{99} \vee \neg x_{100}) \wedge (\neg x_{99} \vee x_{100})$.
(1 mark)

Solution: Satisfiable, e.g. $x_i =$ True for all i.

Question 8 (5 marks)
(Short question.)

Context: You are attempting to steal a valuable work of art from the Tate Modern. By using the ventilation ducts, you believe you can steal any artwork not directly in view of a camera. With great difficulty, you have determined the number of cameras in the building and a limited set of points where they could be mounted. You also know, for each mounting point, which artworks a camera mounted there would protect, and you have recorded this information in the form of a bipartite graph. Unfortunately, you haven't been able to narrow down the cameras' locations exactly. You don't want to go to the trouble of breaking in and then leave empty-handed, so you would like to know: Is it true that *however* the cameras are mounted, you will be able to steal at least one artwork?

Problem: The ArtThief problem is defined as follows. An instance of the problem consists of an integer k, a set A of artworks, a set P of mounting points, and a bipartite graph G with vertex classes A and P. The answer is `Yes` if for all sets of camera locations $C \subseteq P$ with $|C| = k$, there exists $a \in A$ which is not adjacent to any $p \in P$. Otherwise, the answer is `No`. Which of the following statements has a short and simple proof?

(a) ArtThief is in NP.

(b) ArtThief is not in NP.

(c) ArtThief is in Co-NP.

(d) ArtThief is not in Co-NP.

(e) More than one of the above, or none of the above.

Solution: C — ArtThief is in Co-NP. ArtThief is a decision problem, and if the answer is "no" then there must be an arrangement of k cameras that cover every vertex which can be verified in polynomial time, so it is in Co-NP. There is no obvious way of proving that ArtThief is or is not in NP, and in fact I as examiner can be confident that any obvious proof a student comes up with is wrong, since ArtThief is Co-NP-complete and the question reduces down to whether or not NP = Co-NP. (I wouldn't expect a student to prove that, though — just to notice that there is no proof as simple as the proof of Co-NP membership.)

Question 9 (5 marks)

(Short question.) Consider the "unoptimised" union-find data structure presented in lectures, in which a sequence of n operations has a worst-case running time of $\Theta(n \log n)$ rather than $\Theta(n\alpha(n))$. Let G be the graph of such a data structure initialised with the following commands:

`MakeUnionFind`([10]);
`Union`(3, 4);
`Union`(3, 5);
`Union`(3, 6);
`Union`(1, 2);
`Union`(1, 4);
`Union`(7, 1).

(a) How many components does G have? (2 marks)

Solution: 4.

(b) What is the maximum depth of any component of G? (Remember that depth is the greatest number of **edges** from the root to any leaf.) (3 marks)

Solution: 2. After the first four union commands, $\{3, 4, 5, 6\}$ and $\{1, 2\}$ both span depth-1 trees; they are then combined into a depth-2 tree by `Union`(1, 4), and `Union`(7, 1) just adds another child to the root. The final components have vertex sets $\{1, \dots, 7\}$, $\{8\}$, $\{9\}$ and $\{10\}$.

Question 10 (5 marks)

(Short question.) You are implementing rudimentary pathfinding in a video game, trying to find the shortest paths from a large number of enemies to the player in a maze. Which of the following algorithms would be the most efficient choice for this situation?

(a) Breadth-first search.

(b) Depth-first search.

(c) Dijkstra's algorithm.

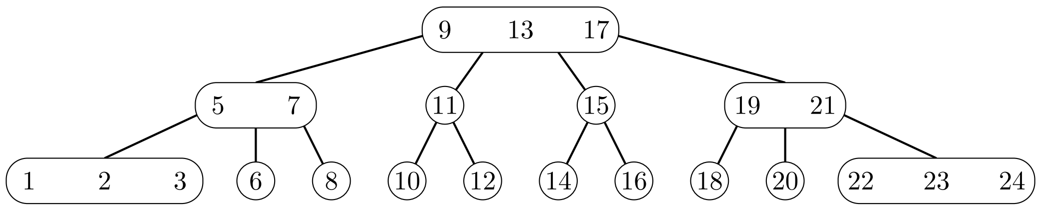
(d) The Bellman-Ford algorithm.

(e) None of the above algorithms would work.

Solution: C. If the pathfinding graph has m edges and n vertices, then Dijkstra's algorithm would run in $O(m \log n)$ time no matter how many enemies there are. Breadth-first or depth-first search would only find shortest paths from one enemy to the player at a time, potentially requiring $\Theta(m)$ time per enemy. Bellman-Ford requires $\Theta(mn)$ time; it is only efficient when dealing with negative-weight edges, and distances are positive so there are no negative-weight edges in this graph.

Question 11 (5 marks)

(Medium question.) Let T be the 2-3-4 tree below.

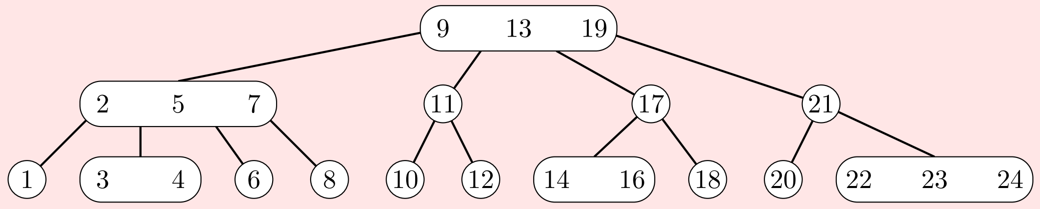


Let U be the 2-3-4 tree formed by first using the `Insert` operation to add 4 into T, then using the `Delete` operation to remove 19 from T.

(a) What is the path traversed on applying the `Find` operation to search for 3 in U? (For example, in T, this would be $(9\ 13\ 17), (5\ 7), (1\ 2\ 3)$.) (2 marks)

(b) What is the path traversed on applying the `Find` operation to search for 16 in U? (3 marks)

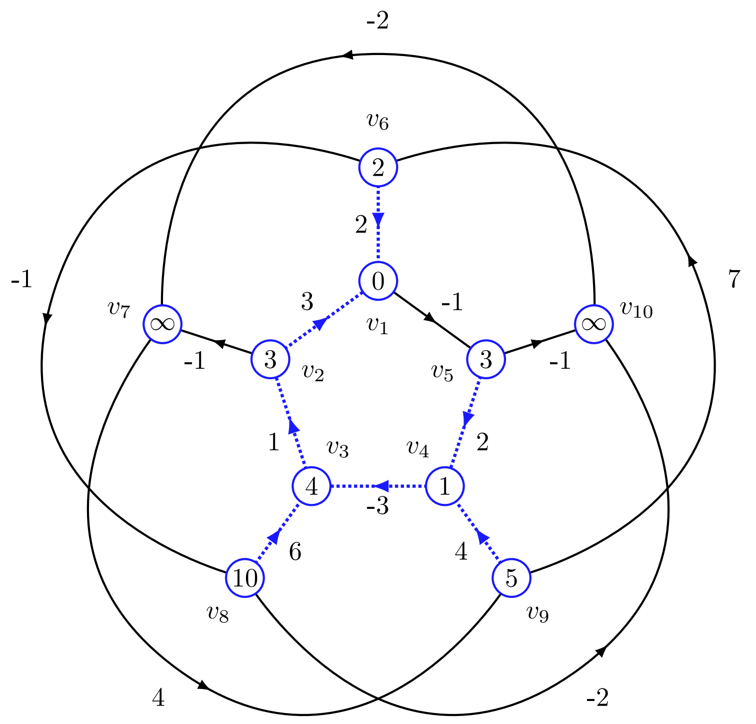
Solution: $(9\ 13\ 19), (2\ 5\ 7), (3\ 4)$ for a), $(9\ 13\ 19), (19), (14\ 16)$ for b). The insertion operation will split (9 13 17) and (1 2 3) on the way down before inserting 4 into the (3) node. The deletion operation will find the predecessor of 15, namely 14; on its way down it will then re-fuse (9), (13) and (17), transfer from (19 21) into (15), and fuse (14) and (16) to make (14 15 16); it will then delete 14 and overwrite 15 with 14. The final value of U is as shown below.



Question 12 (10 marks)

(Medium question.) Consider the edge-weighted directed graph below, pictured part of the way through executing the Bellman-Ford algorithm to find the distances $d(v, v_1)$ for all vertices v. The current bounds on distance recorded by the algorithm are written inside each vertex. The edges currently selected by the algorithm are drawn thicker, dotted, and in blue.

(cont.)



Carry out **one** further iteration of the Bellman-Ford algorithm — that is, updating each vertex exactly once — processing the vertices in the order $v_1, v_2, \ldots, v_{10}$. After carrying out this iteration:

(a) What is the weight of v_5? (2 marks)

Solution: 3.

(b) What is the weight of v_7? (2 marks)

Solution: 9.

(c) What is the weight of v_{10}? (2 marks)

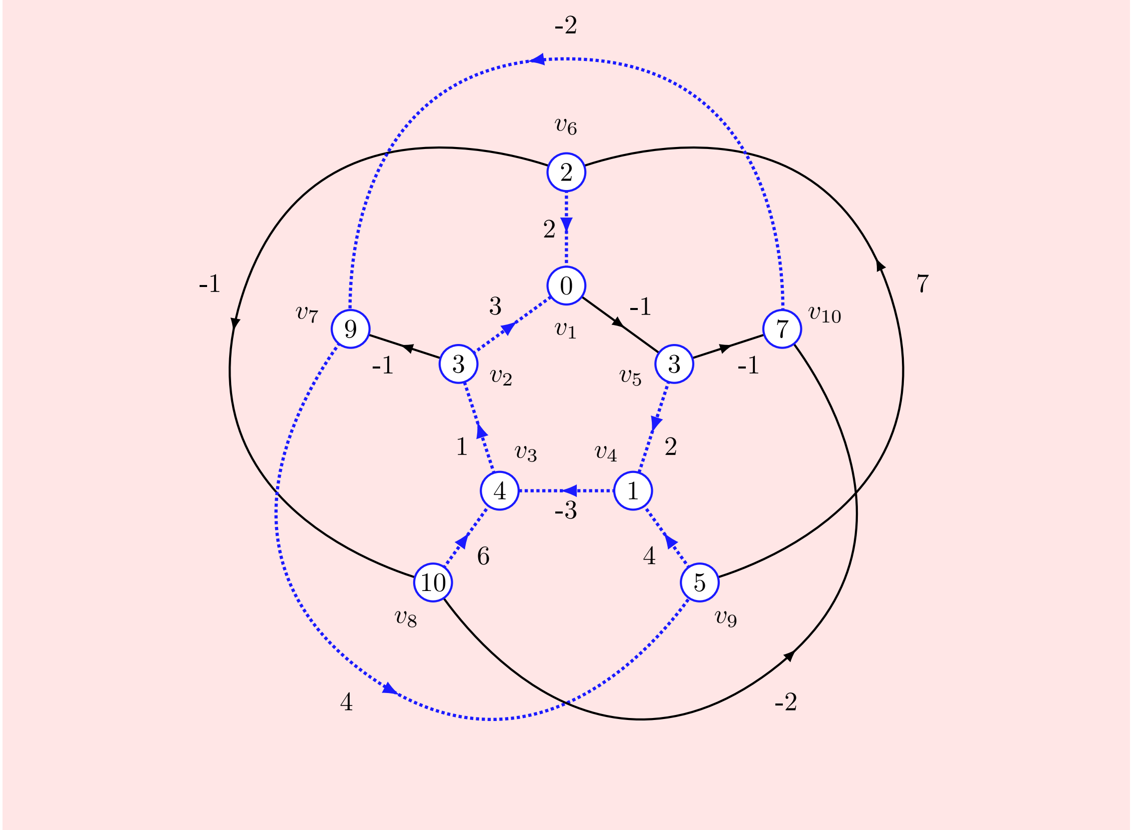
Solution: 7.

(d) What is the currently-stored path from v_7 to v_1? (2 marks)

Solution: $\boldsymbol{v_7 v_9 v_4 v_3 v_2 v_1}$.

(e) Is another iteration of the algorithm required to achieve accurate distances to v_1? (2 marks)

Solution: Yes. The state of the algorithm after one further iteration is as shown below.



The next iteration of Bellman-Ford will update the distance of v_8 from 10 to 5.

Question 13 (10 marks)

(Medium question.)

Context: You are a lowly apprentice to the master dwarven blacksmith Cheery Littlebottom, and you would like to use the facilities in your spare time to reforge the dread blade of A'Neem'Ay before the sacred festival of Jankon; this will allow you take over the world and avoid ever having to do Cheery's laundry again. Unfortunately, while Cheery's forge contains many anvils, most of them are in use most of the time, and as an apprentice you are unable to book any anvil time of your own — you must work in the gaps left by more important people. Worse, before you can make use of an anvil, you must spend a full day secretly dedicating it to A'Neem'Ay, and only one anvil can be dedicated this way at once. The one piece of good news is A'Neem'Ay has given you the ability to see the future, and you know in advance how much time you'll be able to get at each anvil on each day.

Problem: You are given a set of k anvils $A_1, \dots, A_k$, and a time limit of D days. For each anvil A_i, you are also given a length-D list h_i (indexed from 1) such that $h_i[j] \in \{1, \dots, 16\}$. For all $j \in [D]$, on day j, if you are currently at anvil i, then you can **either** spend $h_i[j]$ hours working at anvil A_i, **or** spend the full day switching to a different anvil. Your goal is to maximise your total time spent working, across all anvils. **Fill in the blanks** of the following dynamic programming algorithm, which solves the problem in $O(kD)$ time. **Hint:** This algorithm makes use of a space-saving technique that was also used in the Bellman-Ford algorithm.

(The first four blanks are worth two marks each, the last two are worth one mark each. **Don't copy the whole algorithm out**, just write what should go in each blank!)

```
Algorithm: WorldConquest
Let actions[$i$] ← "" for all $i \in [k]$.
Let hours[$i$], temp[$i$] ← 0 for all $i \in [k]$.
for $j = 1$ to $d$ do
    for $i = 1$ to $k$ do
        temp[$i$] ← hours[$i$].
    for $i = 1$ to $k$ do
        Let foo ← ______ + ______.
        Let bar ← ______{temp[$x$]: $x \in [k]$}.
        Let $\ell$ be such that temp[$\ell$] = ______.
        if foo > bar then
            hours[$i$] ← foo.
            Add "Work on the sword. " to the ______ of actions[$i$].
        else
            hours[$i$] ← bar.
            Add "Switch to anvil $\ell$. " to the ______ of actions[$i$].
for $i = 1$ to $k$ do
    Add "Start at anvil $i$. " to the start of actions[$i$].
Let $\ell$ be such that hours[$\ell$] = max{hours[$i$]: $i \in [k]$}.
Return actions[$\ell$].
```

Solution: temp[i] and $h_i[j]$ (in some order), max, bar, start, and start. The final algorithm is:

```
Algorithm: WorldConquest
Let actions[$i$] ← "" for all $i \in [k]$.
Let hours[$i$], temp[$i$] ← 0 for all $i \in [k]$.
for $j = 1$ to $d$ do
    for $i = 1$ to $k$ do
        temp[$i$] ← hours[$i$].
    for $i = 1$ to $k$ do
        Let foo ← temp[$i$] + $h_i[j]$.
        Let bar ← max{temp[$x$]: $x \in [k]$}.
        Let $\ell$ be such that temp[$\ell$] = bar.
        if foo > bar then
            hours[$i$] ← foo.
            Add "Work on the sword. " to the start of actions[$i$].
        else
            hours[$i$] ← bar.
            Add "Switch to anvil $\ell$. " to the start of actions[$i$].
for $i = 1$ to $k$ do
    Add "Start at anvil $i$. " to the start of actions[$i$].
Let $\ell$ be such that hours[$\ell$] = max{hours[$i$]: $i \in [k]$}.
Return actions[$\ell$].
```

Qu. continues ...

It's based on the recurrence

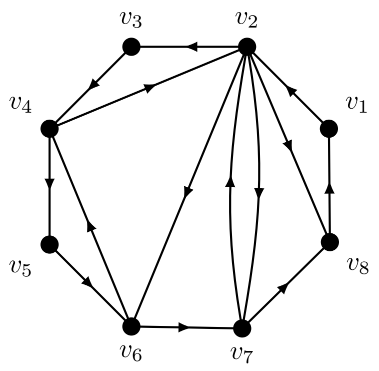
$$T(i,j) = \max\Big(\{T(x, j-1) \colon x \in [k]\} \cup \{h_j[i]\}\Big),$$
$$T(i,0) = 0$$

for the maximum possible number of hours spent on the sword starting at anvil i with j days to go until Jankon. However, in calculating the values of $T(i,j)$ for some fixed j, we only need the values of $T(i,j-1)$, so rather than storing the whole table we only store the $j-1$'st row (in `temp`) and the j'th row (in `hours`).

Section 2 — Long-answer questions (75 marks)

In this section, you should give the reasoning behind your answers unless otherwise specified — you **will** receive credit for partial answers or for incorrect answers with sensible reasoning.

Question 14 (10 marks)
(Short question.) Consider the directed graph G below.



(a) Express G in adjacency matrix form. (5 marks)

Solution: Taking the i'th row/column of the matrix to represent v_i, we obtain

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

(b) Express G in adjacency list form. (5 marks)

Solution: The lists should be:

$$\begin{aligned}
v_1 &: [v_2]\\
v_2 &: [v_3, v_6, v_7, v_8]\\
v_3 &: [v_4]\\
v_4 &: [v_2, v_5]\\
v_5 &: [v_6]\\
v_6 &: [v_4, v_7]\\
v_7 &: [v_2, v_8]\\
v_8 &: [v_1].
\end{aligned}$$

Question 15 (15 marks)

(Short question.)

Context: You are mayor and administrator of the thriving dwarven settlement of Bronzemurder. A series of trade delegations have arrived, and you have goods to sell. In fact, you have a *lot* of goods to sell — more than the traders can carry back with them — and you would like to extract as much immediate profit from the situation as possible. (The fortress has recently struck adamantine, and you have heard rumours that this often ends poorly in the long run.)

Problem: You are given the following information:

- A set $\{T_1, \ldots, T_k\}$ of traders;
- A set $\{G_1, \ldots, G_n\}$ of types of goods available;
- The total weight c_i of goods that each trader T_i is able to carry;
- The weight w_j of one unit of each goods type G_j;
- The number n_j of units of each goods type G_j you have available to sell;
- The price $p_{i,j}$ that trader T_i is willing to pay for one unit of goods of type G_j.

You wish to maximise your total profit by selling your goods to the appropriate traders. Express this as a linear programming problem, under the assumption that a single unit of goods can be split into fractional parts and sold to different traders. **Briefly** explain what your variables represent, and the purpose of each line of your answer.

Solution: One possible solution is the following.

$$\sum_{i=1}^{k}\sum_{j=1}^{n} p_{i,j}x_{i,j} \to \max\text{, subject to}$$
$$\sum_{i=1}^{k} x_{i,j} \le n_j \text{ for all } j \in [n],$$
$$\sum_{j=1}^{n} x_{i,j}w_j \le c_i \text{ for all } i \in [k],$$
$$x_{i,j} \ge 0 \text{ for all } i \in [k],\ j \in [n].$$

The quantity being maximised is the total profit when we sell $x_{i,j}$ units of goods type G_j to trader T_i. The first set of constraints ensures that we don't sell more of any given goods type than we actually own. The second set of constraints ensures that we don't sell more of any given goods type to any given trader than the trader can actually carry. The third set of constraints ensures we are selling goods rather than buying them.

Question 16 (15 marks)

Consider the following problem, called FAULTYSAT. You are given a formula F in CNF form with $3m$ clauses. You wish to know whether it is possible to satisfy not F itself, but at least $2m$ of its clauses. For example, if $F = (\neg x) \wedge (\neg y) \wedge (x \vee y)$, then F itself is unsatisfiable, but still a Yes instance of FAULTYSAT (on setting x and y to False).

(a) (Short question.) Prove that FAULTYSAT is in NP. (5 marks)

Solution: FaultySAT is a decision problem as required. Moreover, for any instance of FAULTYSAT to which the answer is Yes, there exists an assignment of variables to F which satisfy at least $2m$ of F's clauses. Hence we can define `Verify`(F, w) to check whether w represents an assignment which satisfies at least $2m$ of F's clauses — it is easy to do this in $O(\text{length}(F))$ time, and the length of a suitable assignment in bits is $O(\text{length}(F))$.

(b) (Medium question.) Prove that FAULTYSAT is NP-complete. (10 marks)

Solution: By part (a), it suffices to prove that SAT reduces to FAULTYSAT. Let F be a CNF formula with m clauses, and let x be a new variable not appearing in F. Let $P = x \wedge (\neg x)$. Let F' be the formula $F \wedge P \wedge P \wedge \cdots \wedge P$, with m total copies of P, so that F' has $3m$ total clauses.

For any satisfying assignment of truth values to variables in F, by setting x to True we obtain an assignment for F' which satisfies exactly $2m$ clauses. Conversely, any assignment for F' will satisfy exactly m of the "P clauses", so if it satisfies $2m$ clauses in total then it must satisfy all m of the clauses in F. Hence F is a Yes instance of SAT if and only if F' is a Yes instance of FAULTYSAT, as required.

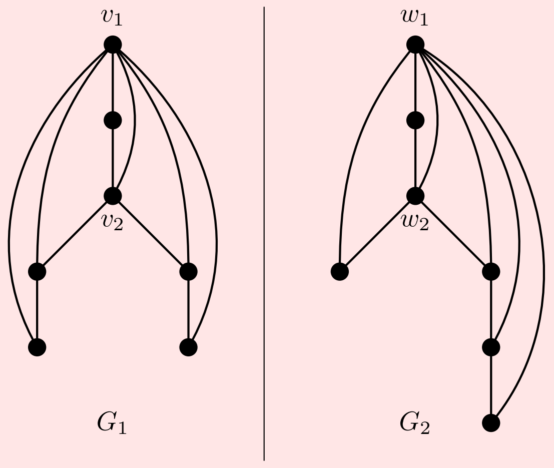
Question 17 (10 marks)

(Medium question.) John is trying to come up with an algorithm to test whether two **connected** graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic in polynomial time, and comes up with the following.

```
Algorithm: BadIsoTest
for $d = 0$ to $n$ do
    if $G_1$ doesn't contain the same number of degree-$d$ vertices as $G_2$ then
        Return No.
Let $\Delta$ be the maximum degree of $G_1$.
Let $v$ be an arbitrarily-chosen vertex of degree $\Delta$ in $G_1$.
Using BFS, let $L_i(v) = \{x \in V_1 : d(v, x) = i$ in $G_1\}$ for all $i \in [n]$.
for $w \in V_2$ of degree $\Delta$ in $G_2$ do
    Using BFS, let $L_i(w) = \{x \in V_2 : d(w, x) = i$ in $G_2\}$ for all $i \in [n]$.
    if for all $i, d \in [n]$, $L_i(v)$ contains the same number of degree-$d$ vertices in $G_1$ as $L_i(w)$ contains in $G_2$ then
        Return Yes.
Return No.
```

Give two connected graphs G_1 and G_2 which are not isomorphic, but which have the property that BadIsoTest(G_1, G_2) is guaranteed to incorrectly return Yes, no matter how v is chosen. **Briefly** explain why BadIsoTest(G_1, G_2) always returns Yes. (You do not have to explain why G_1 and G_2 are not isomorphic.)

Solution: Here is one possible answer:



BadIsoTest is guaranteed to take $v = v_1$, so the only candidate for w is w_1. Then $L_1(v)$ and $L_1(w)$ each contain one vertex of degree 4 and five vertices of degree 3, and $L_k(v)$ and $L_k(w)$ are all empty for all $k \geq 2$, so BadIsoTest(G_1, G_2) returns Yes. Any isomorphism from G_1 to G_2 would have to map v_1 to w_1 (as the only vertices of degree 6); but $G_1 - v_1$ and $G_2 - v_2$ are not isomorphic, e.g. there is a length-3 path starting from w_2 in $G_2 - w_1$ but no length-3 path starting from v_2 in $G_1 - v_1$.

More generally, any solution along the lines of "find two non-isomorphic graphs H_1 and H_2 with the same degree sequence, then form G_1 by adding a vertex joined to everything in H_1 and form G_2 by adding a vertex joined to everything in H_2" will work, along with many other possibilities.

Question 18 (5 marks)

(Medium question.) Let G be a graph, let M be a matching in G of size 50, and let M^* be a **maximum** matching in G. Suppose that the symmetric difference of M and M^* contains exactly exactly 6 components. What are the possible sizes of M^*? **Briefly** explain your answer. (**Hint:** You may find it helpful to use ideas from the proof of Berge's lemma.)

Solution: As in the proof of Berge's lemma, each component of $M \triangle M^*$ has either the same number of M-edges and M^*-edges, one more M-edge than M^*-edges, or one more M^*-edge than M-edges. So if there are six components, we have $|M| - 6 \leq |M^*| \leq |M| + 6$. Moreover, since M^* is a maximum matching, we have $|M| \leq |M^*|$. Combining these two facts, we obtain $|M| \leq |M^*| \leq |M| + 6$, i.e. $|M^*|$ could take any value in $\{50, 51 \dots, 56\}$.

Question 19 (5 marks)

(Long question.)

Context: The war is over, the forces of good have fallen, and Dread Emperor Triumphant stands, well, triumphant. The opposition was led by the Rogue Sorcerer, and the Dread Emperor has decreed that there will be a grand victory parade from the site of her final defeat to the newly-rechristened Imperial Palace. As the Dread Emperor's most senior viceroy, you have been put in charge of planning the route.

Unbeknownst to Triumphant, you are a double agent for the forces of good plotting his assassination. Rebels have set up traps in several isolated streets in the city, and you would like to lead the parade through as many of them as possible. To minimise suspicion, however, you plan to ensure that every street the parade passes through will take it closer to the Palace rather than further away (or the same distance).

Problem: You are given a set of points P, and a set of streets S joining pairs of points. You are given two points $D, IP \in P$, and a set of streets $T \subseteq S$. For each point $p \in P$, you are given the distance d_p of p from IP (in a straight line). You wish to find a path $x_0 \dots x_t$ along the streets in S such that:

- $x_0 = D$;
- $x_t = IP$;
- for all $i \in \{1, \dots, t\}$ we have $d_{x_i} < d_{x_i - 1}$;
- subject to the above, $x_0 \dots x_t$ passes through as many streets in T as possible.

Give an algorithm to solve this problem in polynomial time, and **briefly** explain why it works. (**Hint:** Try reducing this problem to one you already have an algorithm for.)

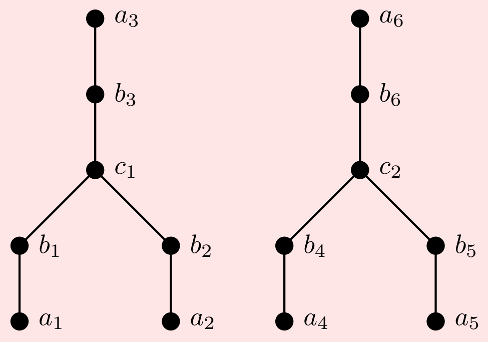
Solution: Form a graph G from the streets of the city in which junctions, the site of the Rogue Sorcerer's defeat, and the Imperial Palace are nodes, and two nodes are joined by an edge if there is a street directly joining them. Direct each edge from the endpoint further from the Palace to the endpoint closer to the Palace. Give each untrapped street a weight of 0, and each trapped street a weight of -1. Then a path from the final defeat to the Imperial Palace of length $-k$ corresponds exactly to a valid parade route passing through k trapped streets. Moreover, the graph contains no cycles at all and hence no negative-weight cycles. We may therefore solve the problem in polynomial-time by applying the Bellman-Ford algorithm to find a shortest path from the site of the final defeat to the Imperial Palace in G.

Question 20 (5 marks)

(Long question.) We say a *covering set* for a graph $G = (V, E)$ is a set $X \subseteq V$ such that every vertex in G is either in X or adjacent to some vertex in X, i.e. such that $V = X \cup N(X)$. (Recall that $N(X) = \bigcup_{v \in X} N(v)$.) You are interested in finding as **small** a covering set as possible in an n-vertex graph G, and come up with the following greedy algorithm: Initially, take $X = \emptyset$. Then, repeatedly add vertices of maximum degree in $G[V \setminus (X \cup N(X))]$ to X until $X \cup N(X) = V$, i.e. until X is covering. Give an example to show that the covering set this algorithm finds might be $\Omega(n)$ vertices larger than a minimum covering set, and **briefly** explain why your example works.

(You may choose how the algorithm breaks ties — that is, whenever there are multiple vertices of maximum degree in $G[V \setminus (X \cup N(X))]$, you may choose which of them the algorithm adds to X.)

Solution: One possible answer is the following. Let $A = \{a_1, \ldots, a_{3t}\}$, $B = \{b_1, \ldots, b_{3t}\}$, and $C = \{c_1, \ldots, c_t\}$ be distinct sets of vertices. For each $i \in [3t]$, join a_i to b_i by an edge; $i \in [t]$, join c_i to b_{3i-2}, b_{3i-1} and b_{3i} by an edge. The resulting graph is shown below for $t = 2$.



The greedy algorithm will first add every vertex from C to X, then every vertex from A, for a total of $4t$ vertices. However, the vertices of B by themselves form a covering set of size $3t$, for a difference of t. We have $n = 7t$, so $t = n/7 \in \Omega(n)$ and we're done.

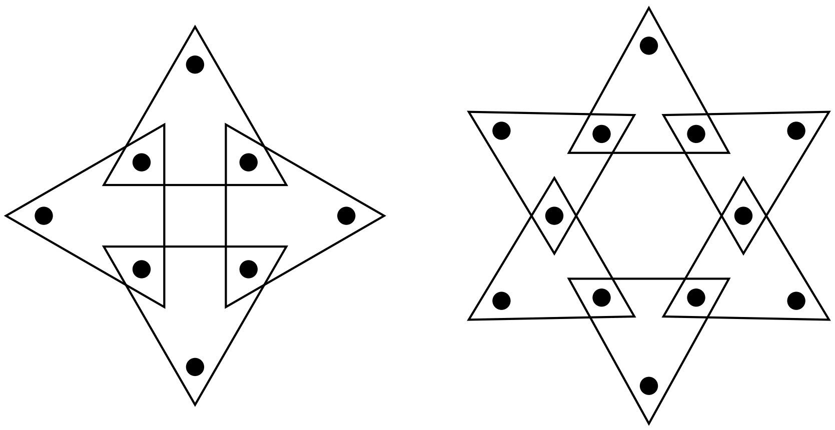
Question 21 (5 marks)
(Long question.) Let G_n be the graph whose vertex set is the set of all trees with vertex set $[n]$. (So the vertices of G_n are trees.) Two n-vertex trees T_1 and T_2 are joined by an edge in G_n if T_2 can be formed from T_1 by removing one edge, then adding one edge. Prove that G_n is connected.

Solution: Let T_1 and T_2 be n-vertex trees with $T_1 \neq T_2$. By the fundamental lemma of trees, both T_1 and T_2 have $n-1$ edges. We will prove that there are edges $e \in E(T_1)$ and $f \in E(T_2)$ such that removing e from T_1 and adding f to T_1 yields another tree; since this is equivalent to traversing an edge from T_1 in G_n, by repeatedly applying this result we obtain a path from T_1 to T_2 as required.

Order $[n]$ as $v_1, \dots, v_n$ according to a breadth-first search on T_1, so that v_i is the i'th vertex uncovered, and so that $T_1[v_1, \dots, v_i]$ is a tree for all i. Observe that $T_1[v_1] = T_2[v_1]$, and $T_1 \neq T_2$, so there exists a minimum j such that $T_1[v_1, \dots, v_j] \neq T_2[v_1, \dots, v_j]$. Moreover, since $T_1[v_1, \dots, v_{j-1}]$ and $T_2[v_1, \dots, v_{j-1}]$ are trees, v_j can send at most one edge $\{x, v_j\}$ into $\{v_1, \dots, v_{j-1}\}$ in T_1 and at most edge $\{y, v_j\}$ in T_2, or in both cases a cycle would be formed. We therefore take $e = \{x, v_j\}$ and $f = \{y, v_j\}$, and observe that $T_1 - e + f$ is still a tree as required. Indeed, it still has the same number of edges as T_1, and it is still connected since $T_1[v_1, \dots, v_{j-1}]$ is a tree — any path using the edge $\{x, v_j\}$ can be replaced by a path using $\{y, v_j\}$ and a path in $T_1[v_1, \dots, v_{j-1}]$ — so $T_1 - e + f$ is a tree by the fundamental lemma of trees.

Question 22 (5 marks)
(Long question.) You are running a polyamorous dating site, and are finding that the problem of finding compatible groups seems to be harder to solve than finding large matchings in a bipartite graph. Consider the following highly simplified version of the problem, in which each person is only interested in a closed relationship with exactly two others. You are given a set S of people, and a set T of compatible triples of people. We say $D \subseteq T$ is a *dating arrangement* if $t_1 \cap t_2 = \emptyset$ for all distinct $t_1, t_2 \in D$; you seek a dating arrangement which is as large as possible. Prove that even this simplified version of the problem is still NP-hard to solve exactly. You don't need to spell out every detail of the argument, but it should be clear how and why your reduction works.

Hint: Try reducing from 3-SAT. You will find gadgets similar to the ones shown below very useful, where the dots represent people and the triangles represent triples in T.



Solution: The general form of the above gadget is, for some integer $x \geq 1$:

- a collection of x "true terminals" $t_1, \ldots, t_x$;
- a collection of "false terminals" $f_1, \ldots, f_x$;
- a collection of "scaffolds" $p_1, \ldots, p_{2x}$.

The triples of the gadget are then given by

$$\{p_1, t_1, p_2\}, \{p_2, f_1, p_3\}, \{p_3, t_2, p_4\}, \{p_4, f_2, p_5\}, \ldots, \{p_{2x-1}, f_x, p_{2x}\}.$$

The key observation is that any dating arrangement can contain at most x triples from the gadget, and that it contains exactly x if and only if either every person in $\{t_1, \ldots, t_x\}$ is covered by a triple from the arrangement or every person in $\{f_1, \ldots, f_x\}$ is covered by a triple from the arrangement. We can use this to represent the state of a variable. Call this gadget an x-pointed star.

Let F be an arbitrary instance of 3-SAT, with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$. If x_i appears in k_i clauses, we represent x_i by a k_i-pointed star S_i. We represent each clause C_i by a pair of people c_i^-, c_i^+. If x_i appears as an un-negated literal in C_j, we add a triple joining c_j^-, c_j^+ to an unused true terminal of S_i; likewise, if x_i appears as a negated literal in C_j, we add a triple joining c_j^-, c_j^+ to an unused false terminal of S_i.

We claim that satisfying assignments of F then correspond to dating assignments of size $\sum_i k_i + m$. In one direction, any satisfying assignment corresponds to a dating assignment in which if x_i is set to True then we choose an arrangement from S_i which covers every false terminal, if x_i is set to False then we choose an arrangement from S_i which covers every true terminal, and every pair of clause people c_j^-, c_j^+ is covered by a triple including a terminal from a literal which is true in C_j. Conversely, it is not too hard to show that any maximum dating assignment must be of this form. Details of the reduction can be found in chapter 8.6 of Kleinberg and Tardos.

END OF PAPER