UNIVERSITY OF BRISTOL

Not a Real Examination Period

Department of Computer Science

2nd Year Practice Paper 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

Answers

Other Instructions

1. **You may bring up to four A4 sheets of pre-prepared notes with you into the exam, but no other written materials.**
2. **You may use a calculator with the Faculty seal of approval if you wish.**

TURN OVER ONLY WHEN TOLD TO START WRITING

Section 1 — Short-answer questions (75 marks total)

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)

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)

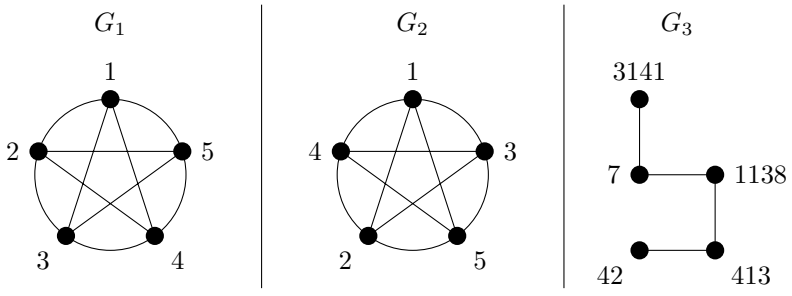
Solution: False.

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

Solution: False.

Question 2 (5 marks)

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)

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.

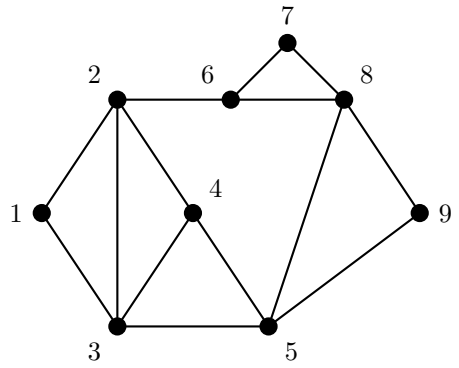
E. None of the above.

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)

Consider a depth-first search in the following graph starting from vertex 1.



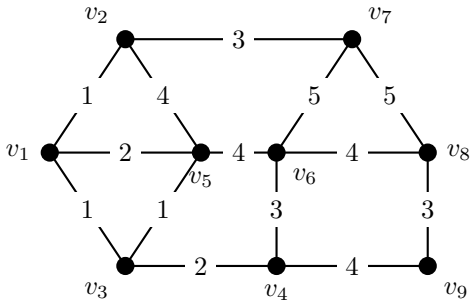
In the implementation of depth-first search given in lectures, we say vertex i is *explored* when `explored`$[i]$ is set to 1. Which vertex will be explored sixth? Assume that whenever the search has a choice of two or more vertices to visit next, it picks the vertex with the lowest number first.

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)

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. $\{v_8, v_9\}$.

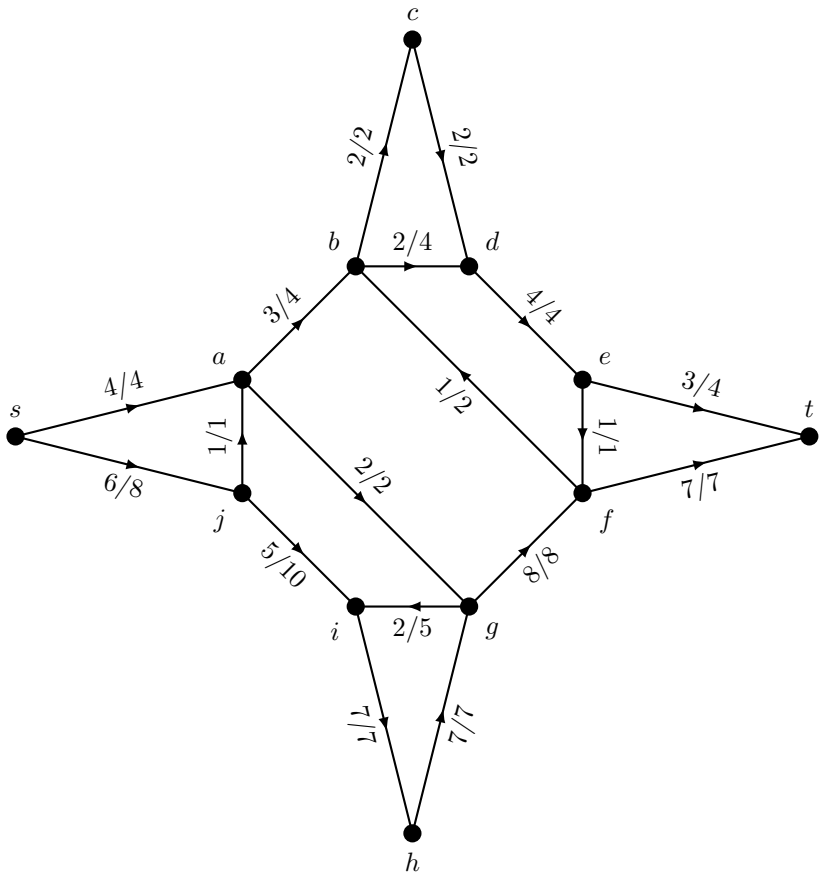
D. More than one of these edges will always be selected.

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 $\{v_4, v_6\}$ and $\{v_2, v_7\}$ in some order. It will then add either $\{v_6, v_8\}$ or $\{v_4, v_9\}$, followed by $\{v_8, v_9\}$. 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 final tie is broken.

Question 6 (5 marks)

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.

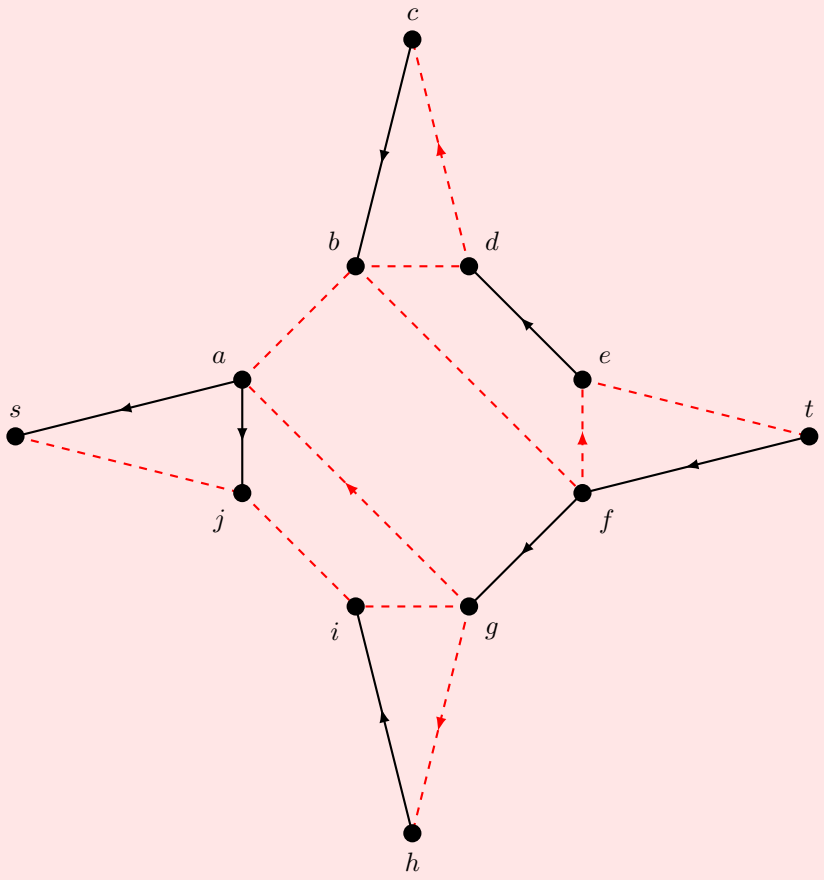
(c) Is $(\{s, b, e\}, \{a, c, d, f, g, h, i, j, t\})$ a valid cut? (1 mark)

(cont.)

Solution: Yes.

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

Solution: ***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)

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)

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)

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)

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 $\{1, \ldots, 7\}$, $\{8\}$, $\{9\}$ and $\{10\}$.

Question 10 (5 marks)

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.

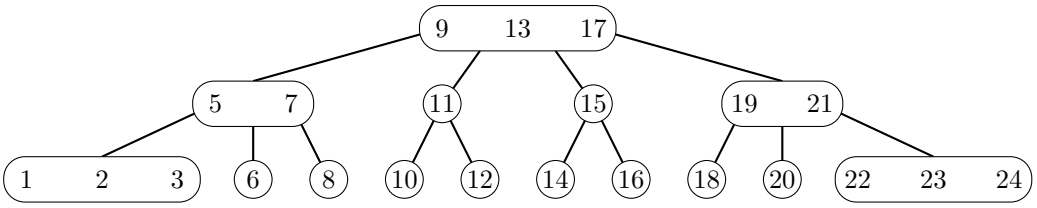
(d) The Bellman-Ford algorithm.

(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)

Let T be the 2-3-4 tree below.

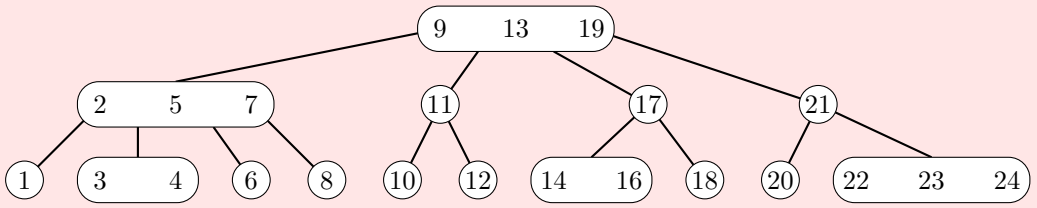


Let U be the 2-3-4 tree formed by first using the `Insert` operation to add 4 into T, then using the `Delete` operation to remove 15 from T.

(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)

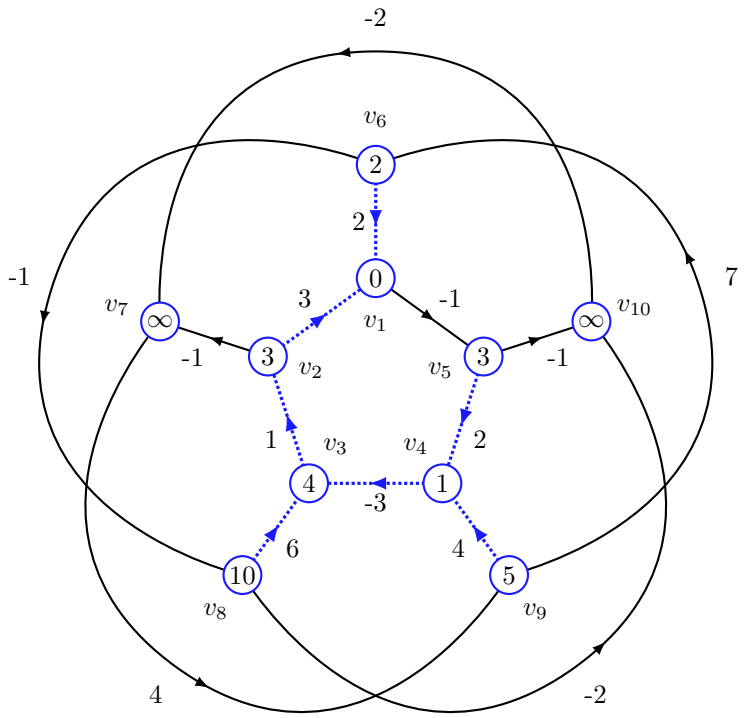
Solution: (9 13 19), (2 5 7), (3 4) for a), (9 13 19), (19), (14 16) for b). The insertion operation will split (9 13 17) and (1 2 3) on the way down before inserting 4 into the (3) node. The deletion operation will find the predecessor of 15, namely 14; on its way down it will then re-fuse (9), (13) 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)

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)

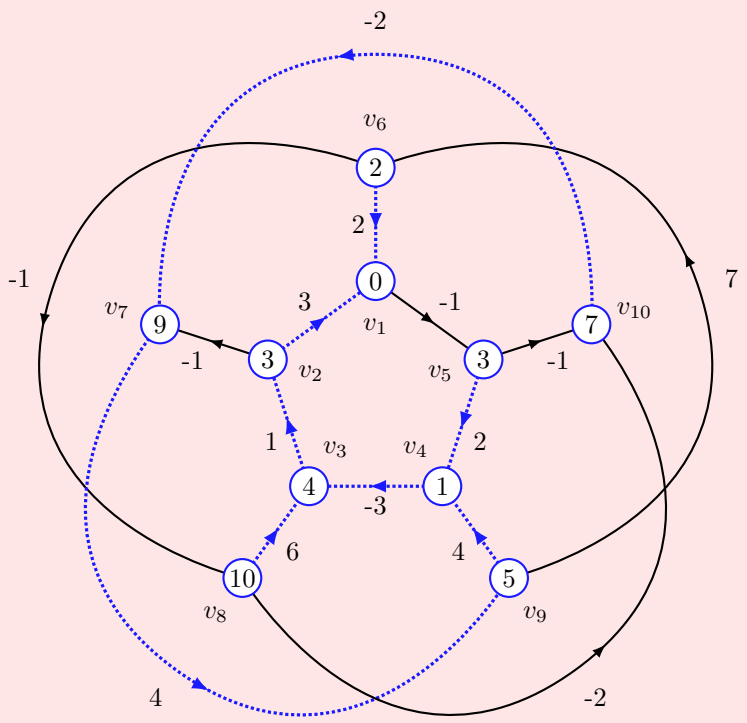
Solution: 7.

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

Solution: $v_7 v_9 v_4 v_3 v_2 v_1$.

(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)

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 $A_1, \ldots, A_k$, 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 $h_i[j] \in \{1, \ldots, 16\}$. 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

(cont.)

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

```
1  Let actions[i] ← "" for all i ∈ [k].
2  Let hours[i], temp[i] ← 0 for all i ∈ [k].
3  for j = 1 to d do
4      for i = 1 to k do
5          temp[i] ← hours[i].
6      for i = 1 to k do
7          Let foo ← _____ + _____.
8          Let bar ← _____{temp[x]: x ∈ [k]}.
9          Let ℓ be such that temp[ℓ] = _____.
10         if foo > bar then
11             hours[i] ← foo.
12             Add "Work on the sword. " to the _____ of actions[i].
13         else
14             hours[i] ← bar.
15             Add "Switch to anvil ℓ. " to the _____ of actions[i].
16 for i = 1 to k do
17     Add "Start at anvil i. " to the start of actions[i].
18 Let ℓ be such that hours[ℓ] = max{hours[i]: i ∈ [k]}.
19 Return actions[ℓ].
```

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

Algorithm: WORLDCONQUEST

```
1  Let actions[i] ← "" for all i ∈ [k].
2  Let hours[i], temp[i] ← 0 for all i ∈ [k].
3  for j = 1 to d do
4      for i = 1 to k do
5          temp[i] ← hours[i].
6      for i = 1 to k do
7          Let foo ← temp[i] + h_i[j].
8          Let bar ← max{temp[x]: x ∈ [k]}.
9          Let ℓ be such that temp[ℓ] = bar.
10         if foo > bar then
11             hours[i] ← foo.
12             Add "Work on the sword. " to the start of actions[i].
13         else
14             hours[i] ← bar.
15             Add "Switch to anvil ℓ. " to the start of actions[i].
16 for i = 1 to k do
17     Add "Start at anvil i. " to the start of actions[i].
18 Let ℓ be such that hours[ℓ] = max{hours[i]: i ∈ [k]}.
19 Return actions[ℓ].
```

It's based on the recurrence

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

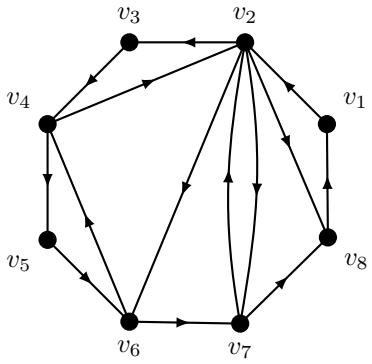
for the maximum possible number of hours spent on the sword starting at anvil i with j days to go until Jankon. However, in calculating the values of $T(i, j)$ for some fixed j, we only need the values of $T(i, j-1)$, so rather than storing the whole table we only store the $j-1$'st row (in `temp`) and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

Section 2 — Long-answer questions (75 marks total)

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)

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 (5 marks)

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

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

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

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

Question 16 (15 marks)

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 $j \in [m]$.

Question 17 (30 marks)

Solve **two** from the following four questions (15 marks each). **If you attempt more than two, you will not gain any extra credit, and only the first two questions attempted will be marked. If you do not wish one of your attempts to be marked, cross it out clearly along with all working.**

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

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

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

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

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

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

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

each distinct $i, j \in V$, let

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

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

(b) Consider the following greedy algorithm to find a large independent set in a graph.

Algorithm: GREEDYIS(G, k)

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

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

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

(c) Consider the following alternative greedy algorithm to

Algorithm: BETTERGREEDYIS(G, k)

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

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

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

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

Question 18 (15 marks)

Choose <u>one</u> of the three following problems to solve. **If you attempt more than one, you will not gain any extra credit, and only the first question attempted will be marked. If you do not wish one of your attempts to be marked, cross it out clearly along with all working.**

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

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

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

This can obviously be done in polynomial time. Now, F' is a `Yes` instance of Approx-SAT if and only if there is an assignment satisfying at least $2k$ of its $3k$

OR clauses. *Any* assignment satisfies exactly k of the new (trivial) OR clauses, and F only has k OR clauses to begin with, so this can only happen if F is satisfied. Hence F' is a `Yes` instance of Approx-SAT if and only if F is a `Yes` instance of SAT, as required.

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

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

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

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

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

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

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

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

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

(cont.)

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

END OF PAPER