UNIVERSITY OF BRISTOL

August 2022 Reassessment 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.

You are strongly advised to attempt questions tagged as "short" before questions tagged as "medium", as they typically require less time for more marks. Be aware that Section 2 contains "short" questions as well.

Question 1 (5 marks)

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

(a) $n \in \Omega(\sqrt{n})$. (1 mark)

Solution: True.

(b) $n \in o(100n)$. (1 mark)

Solution: False.

(c) $n \in O(100n)$. (1 mark)

Solution: True.

(d) $\log n \in O(n^{1/100})$. (1 mark)

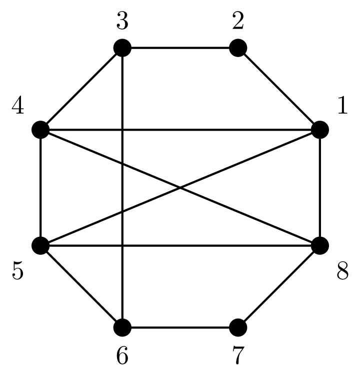
Solution: True.

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

Solution: False. We have $(\log n)^{\log n} = e^{\log n \cdot \log\log n} = n^{\log\log n} \in \omega(n)$.

Question 2 (5 marks)

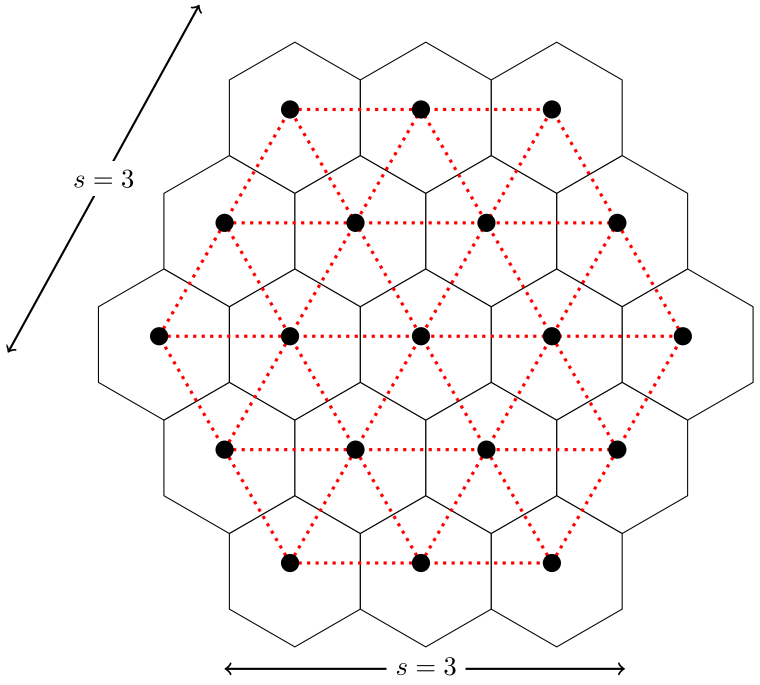
(Short question.) Write down an Euler walk for the following graph.



Solution: Any walk must start with 3 and end with 6 or vice versa. One example is 32187654158436.

Question 3 (5 marks)

(Medium question.) Let $s \geq 2$ be an integer. Consider a regular hexagonal arrangement of regular hexagonal cells, with each side consisting of s cells. Consider the graph G_s formed by taking each cell to be a vertex, and joining two cells by an edge if they share a side. An example is shown below for $s = 3$, where the black lines show the arrangement of cells and the red dotted lines show the edges of G_s.



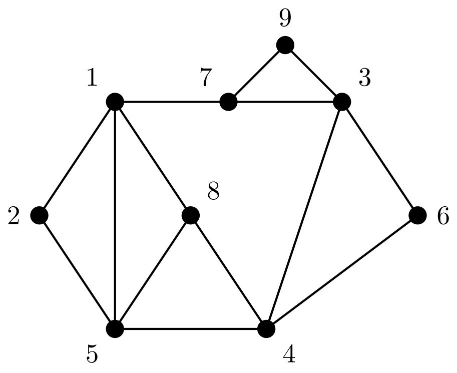
Give a formula for the number of edges in G_s in terms of s. You may use the fact that G_s has $3s^2 - 3s + 1$ vertices. (You do not need to show your working — only your final answer will be marked. You may wish to check that your answer is correct when $s = 2$.)

Solution: There are six cells at corners of the hexagon, which all have degree 3. There are $6(s-2)$ cells at edges of the hexagon, which all have degree 4. The remaining cells all have degree 6, and they form the vertices of a G_{s-1} so there are $3(s-1)^2 - 3(s-1) + 1$ of them. Putting this all together with the handshaking lemma, we see that

$$\begin{aligned}|E(G_s)| &= \frac{1}{2}\sum_{v \in V(G_s)} d(v) = \frac{1}{2}\Big(6 \cdot 3 + 6(s-2) \cdot 4 + \big(3(s-1)^2 - 3(s-1) + 1\big) \cdot 6\Big) \\ &= 9 + (12s - 24) + (9s^2 - 18s + 9 - 9s + 9 + 3) \\ &= 9s^2 - 15s + 6 = 3(3s^2 - 5s + 2).\end{aligned}$$

Question 4 (5 marks)

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



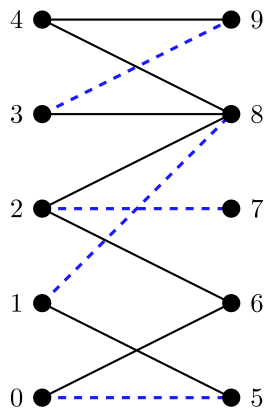
In the implementation of depth-first search given in lectures, we say vertex i is *explored* when `explored`$[i]$ is set to 1. (For example, the start vertex is explored first.) Which vertex will be explored sixth if the start vertex is 4? Assume that whenever the search has a choice of two or more vertices to visit next, it picks the vertex with the lowest number first.

Solution: 2. DFS will traverse edges in the order:

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

Question 5 (5 marks)

(Short question.) Consider the graph G and the matching $M \subseteq E(G)$ shown below, where M is drawn with blue dashed lines.



Write down an augmenting path for M in G.

Solution: The available paths are 49381506, 481506, 60518394 and 605184.

Question 6 (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`$([8])$;
`Union`$(1, 2)$;
`Union`$(1, 3)$;
`Union`$(3, 4)$;
`Union`$(5, 6)$;
`Union`$(5, 1)$;
`Union`$(7, 8)$;
`Union`$(6, 7)$.

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

Solution: 1, spanning all of $\{1, \dots, 8\}$.

(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 five union commands, $\{1, 2, 3, 4\}$ and $\{5, 6\}$ both span depth-1 trees; they are then combined into a depth-2 tree by `Union`$(5, 1)$. `Union`$(7, 8)$ then forms another depth-1 tree, which is added as a child of the root by `Union`$(6, 7)$.

Question 7 (5 marks)

(Short question.) Let $\mathcal{G} = (G, c_E, c_V, s, t)$ be a vertex flow network with $|V(G)| = 100$. In lectures, we covered a way of finding a maximum flow in $\mathcal{G}$ by applying the Ford-Fulkerson algorithm to a new flow network (H, c, s', t'). How many vertices will H have, in this case? Choose **one** of the following options.

A. 98.

B. 100.

C. 102.

D. 198.

E. 200.

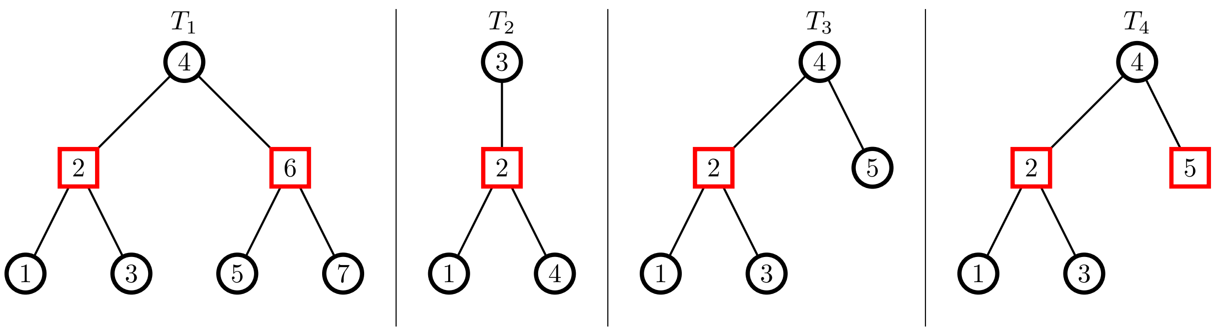
F. 202.

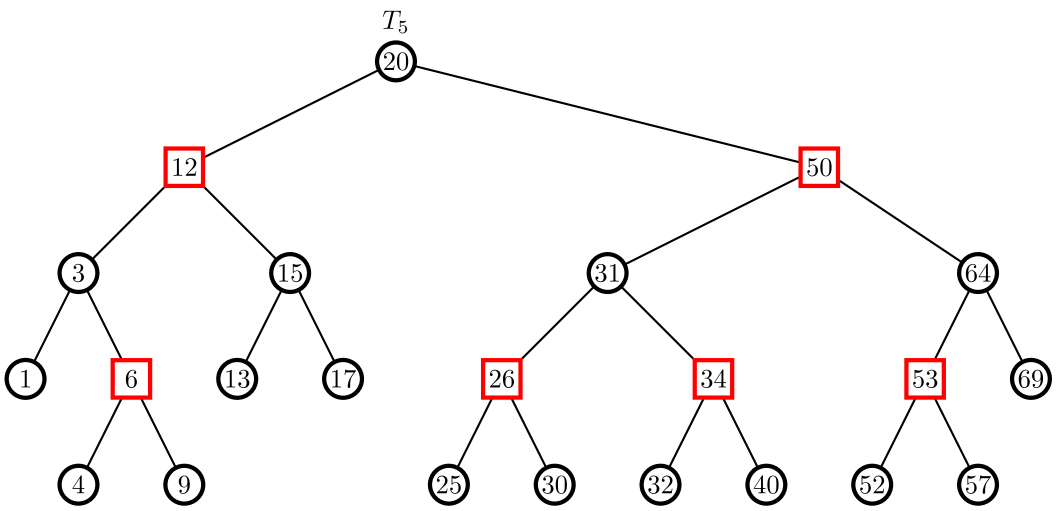
G. None of the above, or it's impossible to tell.

Solution: D — 198. The construction in lectures forms H by replacing each of the hundred vertices in G with a two-vertex gadget except for the source s and the sink t, for a total of 98 extra vertices added.

Question 8 (5 marks)

(Short question.) Which of the following trees $T_1, \ldots, T_5$ are valid red-black trees? (In case you are unable to distinguish the colours, the red nodes are drawn as squares and the black nodes are drawn as circles.)





Solution: T_1, T_3 and T_5 are valid. T_2 is invalid because the root has degree 1, and T_4 is invalid because 45 and 423 are two different root-leaf paths containing different numbers of black nodes.

Question 9 (5 marks)

(Short question.) Consider an instance of interval scheduling with interval set

$$\mathcal{R} = \{(1,3),(2,6),(4,9),(5,6),(6,11),(7,8),(9,11),(10,12),(11,12),(11,13)\}.$$

Qu. continues . . .

(a) When the greedy interval scheduling algorithm discussed in lectures is run on this input, which interval will it choose third? (3 marks)

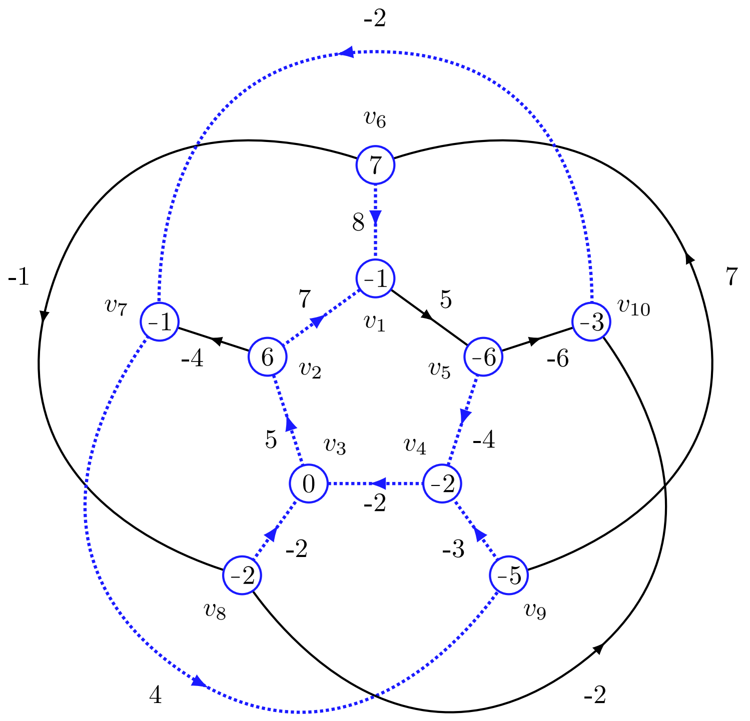
Solution: $(7, 8)$.

(b) What is the size of a maximum compatible set? (2 marks)

Solution: 5. The full set outputted by the algorithm is $\{(1, 3), (5, 6), (7, 8), (9, 11), (11, 12)\}$.

Question 10 (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 $d(v, v_3)$ for all vertices v, i.e. the single-**sink** version of the algorithm with sink v_3. 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.



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_2? (2 marks)

(cont.)

Solution: -5.

(b) What is the weight of v_6? (2 marks)

Solution: -3.

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

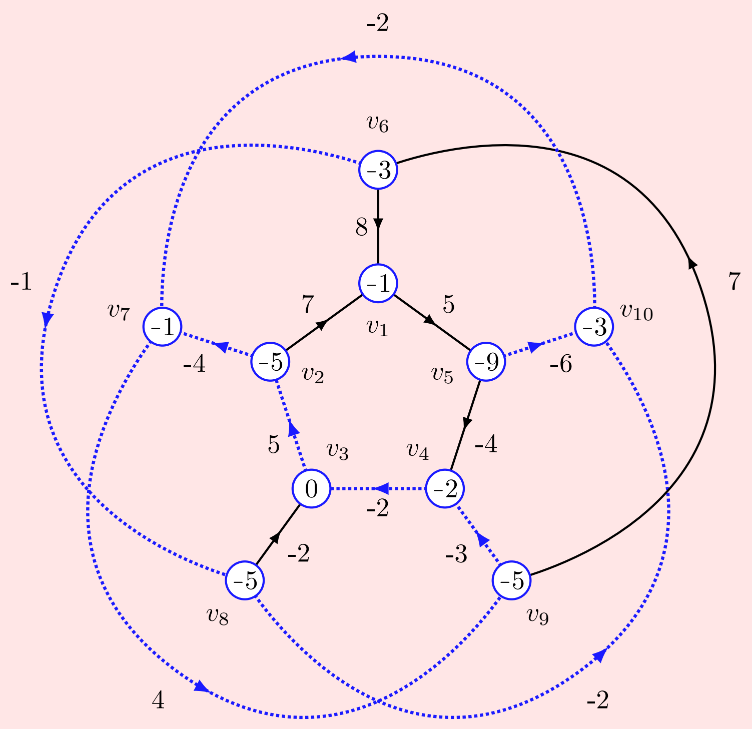
Solution: -3.

(d) What is the currently-stored path from v_5 to v_3 (again, **after** carrying out the iteration)? (2 marks)

Solution: $v_5v_{10}v_7v_9v_4v_3$.

(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 the iteration in the question is as shown below.



The next iteration of Bellman-Ford will update the distance of v_1 to -4 and the distance of v_6 to -6. (This will be the last iteration.)

Question 11 (10 marks)

(Medium question.) You are trying to get a high score in the popular video game Tambourine Hero. In this game, you score points by shaking your tambourine in time to a song playing in the background. After playing certain beats of the song, you are awarded “star power”; your stored star power is an integer between 0 and 100. At any time when you have 50 or more star power, you can “activate star power”. You will then lose star power at a rate of one point per beat until you run out, at which point it is deactivated; while star power is activated, you will earn double points. Any extra star power you are awarded over 100 points is wasted.

Formally, a *song* is a series of *beats* $b_1, \dots, b_t$. Each beat b_i is a pair (p_i, s_i) of p_i *points* and s_i *star power*, where p_i and s_i are non-negative integers. Let P_i be your total points after beat i, and let S_i be your total star power after beat i. Initially, $P_0 = S_0 = 0$. Subsequently, after beat $i \geq 1$,

$$P_i = \begin{cases} P_{i-1} + 2p_i & \text{if star power active,} \\ P_{i-1} + p_i & \text{otherwise.} \end{cases}$$

$$S_i = \begin{cases} \min(100, S_{i-1} + s_i) - 1 & \text{if star power active,} \\ \min(100, S_{i-1} + s_i) & \text{otherwise.} \end{cases}$$

After each beat, if $S_i \geq 50$ and star power is not active, you may choose to activate star power before the next beat. It then stays active until the end of the next beat i with $S_i = 0$. **Fill in the blanks** in the following dynamic programming algorithm, which outputs a list of the beats on which to activate star power in order to achieve the maximum possible score.

(**Don't copy the whole algorithm out**, just write what should go in each blank! Each blank should contain a single expression, e.g. $\max(b, c)$ or next$[b][s][a]$. Two marks will be awarded per blank correctly filled in.)

```
Algorithm: BeATambourineHero
Let next$[b][s][a]$, score$[b][s][a] \leftarrow 0$ for all $0 \leq b \leq t$, all $0 \leq s \leq 100$, and all $a \in \{0,1\}$.
for $b = t-1$ to $0$ do
    for $s = 1$ to $100$ do
        Let new_s_inactive $\leftarrow$ ______.
        Let inactive_score $\leftarrow p_b +$ score$[b+1]$[new_s_inactive]$[0]$.
        Let activating_score $\leftarrow p_b +$ score$[b+1]$[new_s_inactive]$[1]$ if $s \geq 50$, and activating_score $\leftarrow -1$
          otherwise.
        Let score$[b][s][0] \leftarrow \max($inactive_score, activating_score$)$.
        Let score$[b][s][1] \leftarrow 2p_b +$ score$[b+1]$[new_s_inactive $- 1$][______] if new_s_inactive $\geq 2$, and
          score$[b][s][1] \leftarrow 2p_b +$ score$[b+1][0][0]$ otherwise.
        If activating_score $>$ inactive_score, let next$[b][s][0] = 1$.
        If new_s_inactive $\geq 2$, let next$[b][s][1] = 1$.
Let active $\leftarrow 0$ and $s \leftarrow 0$.
for $b = 0$ to $t-1$ do
    If ______ $= 0$ and ______ $= 1$, print “Activate star power after beat $b$”.
    Let $s \leftarrow \min(100, s + s_{b+1}) - 1$ if active $= 1$ and $s \leftarrow \min(100, s + s_{b+1})$ otherwise.
    Let active $\leftarrow$ next$[b+1][s]$[______].
```

Solution: $\min(100, s + s_b)$, 1, active, next$[b][s][0]$, and active. Here is the complete algorithm.

```
Algorithm: BeATambourineHero
Let $\texttt{next}[b][s][a]$, $\texttt{score}[b][s][a] \leftarrow 0$ for all $0 \le b \le t$, all $0 \le s \le 100$, and all $a \in \{0,1\}$.
for $b = t-1$ to $0$ do
    for $s = 1$ to $100$ do
        Let $\texttt{new\_s\_inactive} \leftarrow \min(100, s + s_b)$.
        Let $\texttt{inactive\_score} \leftarrow p_i + \texttt{score}[b+1][\texttt{new\_s\_inactive}][0]$.
        Let $\texttt{activating\_score} \leftarrow p_i + \texttt{score}[b+1][\texttt{new\_s\_inactive}][1]$ if $s \ge 50$, and
          $\texttt{activating\_score} \leftarrow -1$ otherwise.
        Let $\texttt{score}[b][s][0] \leftarrow \max(\texttt{inactive\_score}, \texttt{activating\_score})$.
        Let $\texttt{score}[b][s][1] \leftarrow 2p_i + \texttt{score}[b+1][\texttt{new\_s\_inactive} - 1][1]$ if $\texttt{new\_s\_inactive} \ge 2$, and
          $\texttt{score}[b][s][1] \leftarrow 2p_i + \texttt{score}[b+1][\texttt{new\_s\_inactive} - 1][0]$ otherwise.
        If $\texttt{activating\_score} > \texttt{inactive\_score}$, let $\texttt{next}[b][s][0] = 1$.
        If $\texttt{new\_s\_inactive} \ge 2$, let $\texttt{next}[b][s][1] = 1$.
Let $\texttt{active} \leftarrow 0$.
for $b = 0$ to $t-1$ do
    If $\texttt{active} = 0$ and $\texttt{next}[b][s][0] = 1$, print "Activate star power after beat $b$".
    Let $s \leftarrow \min(100, s + s_{b+1}) - \texttt{active}$.
    Let $\texttt{active} \leftarrow \texttt{next}[b+1][s][\texttt{active}]$.
```

The idea is that $\texttt{score}[b][s][a]$ holds the maximum possible score from the end of beat b with available star power s and star power currently active if $a = 1$ and inactive if $a = 0$.

Question 12 (5 marks)

(Short question.) A *Hamilton path* in a graph G is a path containing every vertex, i.e. a Hamilton cycle minus an edge. If G is a graph and $x, y \in V(G)$, we define $\mathrm{HP}(G, x, y)$ to be the problem of deciding whether or not G contains a Hamilton path from x to y, so (G, x, y) is a `Yes` instance if such a path exists and a `No` instance otherwise. Likewise, we define $\mathrm{HC}(G)$ to be the problem of deciding whether or not G contains a Hamilton cycle.

(a) Is it true that both HP and HC are decision problems? (1 mark)

Solution: Yes.

(b) Is it true that both HP and HC are in NP? (2 marks)

Solution: Yes. If I tell you that a sequence of edges forms a Hamilton cycle in G, or a Hamilton path in G from x to y, then you can verify this in polynomial time.

(c) Is it true that if we require every vertex in the input graph G to have degree at least $|V(G)|/2$, then HC is in P? (You may assume P $\neq$ NP.) (2 marks)

Solution: Yes. In this case, the correct output of $\mathrm{HC}(G)$ is always `Yes` by Dirac's theorem.

Question 13 (5 marks)

(Short question.) Consider the following reduction between the problems HC and HP introduced in Question 12. Let G be an instance of HC. For every edge $\{x, y\} \in E(G)$, run an algorithm (or oracle) for HP with inputs $G - \{x, y\}$, x, and y, where $G - \{x, y\}$ is the graph formed from G by removing $\{x, y\}$ from $E(G)$ and leaving $V(G)$ unchanged. If any of those algorithms output `Yes`, then return `Yes`; otherwise, return `No`.

(a) Is this a reduction from HC to HP, or a reduction from HP to HC? (3 marks)

Solution: From HC to HP.

(b) Is this a Karp reduction, or a Cook reduction? (2 marks)

Solution: A Cook reduction.

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.

You are strongly advised to attempt questions tagged as "short" before questions tagged as "medium", and questions tagged as "medium" before questions tagged as "long", as they typically require less time for more marks.

Question 14 (5 marks)

(Medium question.) Explain briefly why the reduction between HC and HP of Question 13 is valid. (A good answer here will likely be no longer than one paragraph, and certainly no longer than two paragraphs.)

Solution: The algorithm for HP is called at most $|E(G)|$ times on instances with $|E(G)|-1$ edges and $|V(G)|$ vertices; these quantities are all polynomial in the size of G as required by the definition of a Cook reduction. Suppose G is a `Yes` instance of HC. Then G contains a Hamilton cycle C; take an edge $\{x, y\} \in E(C)$. Then by following C, we obtain a Hamilton path from x to y in $G - \{x, y\}$, so one of the HP instances must output `Yes` and we return `Yes` as required. Conversely, suppose we return `Yes`, so that there exists $\{x, y\} \in E(G)$ such that $G - \{x, y\}$ contains a Hamilton path P from x to y; then Pxy is a Hamilton cycle, and so G is a `Yes` instance. Thus we return `Yes` if and only if G is a `Yes` instance of HC, as required.

Question 15 (15 marks)

(Short question.) You are running a long-haul trucking company. You have a fixed number of vehicles which transport material between a set of cities $C_1, \dots, C_k$. For all $i \in [k]$, you currently have $t_i \geq 0$ trucks in city C_i. Over the course of the day, each truck in city C_i can haul a load to any other city C_j with total profit $p_{i,j}$ (after accounting for fuel costs and so on). Each city contains a SimplifyTheProblem Inc. depot at which your trucks all stop. These depots handle loading, unloading, and various administrative tasks, but the depot in city C_i is only under contract to receive and unload $T_i \geq 1$ trucks per day; thus you must avoid sending more than T_i trucks to city C_i. Assume that between travel, loading, and unloading, each trip between any pair of cities takes the full day.

Your goal is to choose destinations for your trucks to maximise your total profit for today without any regard for the future. (Coincidentally, you also invest heavily in fossil fuels and cryptocurrencies.) Formulate this as a linear programming problem and give a **brief** explanation of what your variables represent and why your constraints and objective function are appropriate.

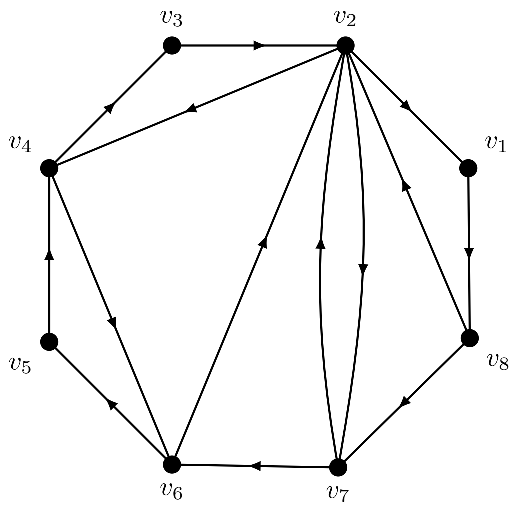
Solution: Let $x_{i,j}$ be the number of trucks we send from city i to city j. The LP is as follows:

$$\begin{aligned}\sum_{i,j=1}^{k} p_{i,j}x_{i,j} &\to \max, \text{ subject to}\\ \sum_{j=1}^{k} x_{i,j} &\le T_i \text{ for all } i \in [k],\\ \sum_{i=1}^{k} x_{i,j} &\le t_j \text{ for all } j \in [k],\\ x_{i,j} &\ge 0 \text{ for all } i,j \in [k].\end{aligned}$$

The objective function is our total profit. The first constraint says that we can't send more trucks out of any city C_i than we have in the city at the start of the day. The second constraint says that we can't send more trucks to any city C_j than its depot can process. The third constraint says we can't send a negative number of trucks from one city to another.

Question 16 (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

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

(cont.)

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

Solution: The lists should be:

$$v_1\colon [v_8], \qquad v_2\colon [v_1, v_4, v_7], \qquad v_3\colon [v_2], \qquad v_4\colon [v_3, v_6],$$

$$v_5\colon [v_4], \qquad v_6\colon [v_2, v_5] \qquad v_7\colon [v_2, v_6] \qquad v_8\colon [v_2, v_7].$$

Question 17 (5 marks)
(Short question.) Give an example of a graph which contains a length-5 cycle as a subgraph, but not as an induced subgraph.

Solution: One example (of many) would be a length-5 cycle with a chord, e.g.

$$V(G) = [5], \qquad E(G) = \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,1\},\{1,3\}\}.$$

Question 18 (15 marks)
(Medium question.) In this question, we work with a variant of SAT in which variables cannot be negated. Given literals a, b and c, which need not be distinct, an *even clause* $\text{EVEN}(x,y,z)$ evaluates to `True` if and only if either zero or two of x, y and z evaluate to `True`. A *width-3 positive OR clause* is an OR clause of three variables (i.e. **un-negated** literals). A *positive even formula* is a conjunction of even clauses and width-3 positive OR clauses. For example,

$$\text{EVEN}(a,\neg b,c) \wedge \text{EVEN}(a,a,d) \wedge (a \vee b \vee e) \wedge \text{EVEN}(c,d,e).$$

is a positive even formula, but $(\neg a \vee b)$ is not due to both the negated variable and the fact that the clause only contains two variables. The decision problem POS-EVEN-SAT asks whether a positive even formula (given as the input) is satisfiable, in which case the desired output is `Yes`.

(a) Give a Karp reduction from POS-EVEN-SAT to 3-SAT and briefly explain why it works. (5 marks)

Solution: Consider an instance of POS-EVEN-SAT. We convert it to CNF form in polynomial time. Any positive OR clause is already in CNF form, and we can write each even clause $\text{EVEN}(x,y,z)$ in CNF form as follows:

$$\begin{aligned}\text{EVEN}(x,y,z) &= \neg\Big((x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge \neg z)\Big) \\ &= (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee \neg z).\end{aligned}$$

Then the resulting CNF formula evaluates to `True` if and only if $\text{EVEN}(x,y,z)$ does, so the corresponding 3-SAT instance is satisfiable if and only if the original POS-EVEN-SAT instance is satisfiable, as required.

(b) Give a Karp reduction from 3-SAT to POS-EVEN-SAT and briefly explain why it works. (10 marks)

Solution: Consider an instance F of 3-SAT with variables $x_1, \ldots, x_n$; we will construct an instance F' of POS-EVEN-SAT in polynomial time. We first add a new variable t and a corresponding clause $\text{EVEN}(t, t, \neg t)$; observe that t must be `True` in any satisfying assignment. We further add clauses $\text{EVEN}(x_i, y_i, t)$ for each $i \in [n]$. Since t must be `True` in any satisfying assignment, it follows that $y_i = \neg x_i$ in any satisfying assignment. Finally, we copy the OR clauses of F into F', replacing each instance of a literal $\neg x_i$ with the corresponding variable y_i. If F' is satisfiable, then $y_i = \neg x_i$ for all i, so $x_1, \ldots, x_n$ form a satisfying assignment for F. Conversely, if F is satisfiable, then we obtain a satisfying assignment for F' on taking $y_i = \neg x_i$ and $t =$ `True`. Thus F is a `Yes` instance of 3-SAT if and only if F' is a `Yes` instance of POS-EVEN-SAT, as required.

Question 19 (10 marks)

It is five minutes to midnight, and you are a secret agent trying to prevent a nuclear dead man's switch from destroying the world. In order to stop the bombs from going off, you have passcode audio files $C_1, \ldots, C_n$ which must be sent to a remote system exactly and without interruptions starting from midnight. For perfect fidelity, audio clip C_i must be sent at a rate of m_i megabits per second for t_i seconds. Unfortunately, you are working from the comic relief character's house, and their ISP implements throttling in an unpleasant way: if you send more than t megabits in the first t seconds of the day, your connection will go down for several minutes (thereby destroying the world). Fortunately, you can send the clips in any order, and not all of them have $m_i > 1$. Your goal is to save the world by sending all n audio files to the system at the specified bitrates, without gaps or interruptions from the ISP. You may assume this is always possible.

(a) (Medium question.) Prove that the greedy algorithm which sends a longest audio file first (i.e. a file with t_i maximum) **fails**, by giving a counter-example with a brief explanation. (5 marks)

Solution: There are many possible solutions to this, but here's one. Take $n = 5$, $m_1 = 2$, $t_1 = 2$, and $m_2 = m_3 = m_4 = m_5 = 0.5$ and $t_2 = t_3 = t_4 = t_5 = 1$. The greedy algorithm will try to broadcast C_1 first and blow up the world. However, broadcasting $C_2, \ldots, C_5$ first will result in transmitting 2 megabits in 4 seconds, leaving enough spare capacity to transmit C_1 last.

(b) (Long question.) State a greedy algorithm which works, and prove that it works using either an exchange argument or a "greedy stays ahead" argument. (5 marks)

Solution: One greedy algorithm which works is to always send a file with minimum bitrate first, i.e. a file with m_i as small as possible. Here's an exchange argument to prove this works. It's enough to show that if there is a valid ordering $\mathcal{S}$ with S_i streaming immediately before S_j such that $m_i > m_j$, then the solution $\mathcal{S}'$

with S_i and S_j exchanged is also valid — indeed, we can apply this fact repeatedly to move the lowest-bitrate clips to the start and end up with the same solution chosen by the greedy algorithm. Let b_0 be the total number of bits streamed at the start of S_i in $\mathcal{S}$, and let t_0 be the time at which S_i starts streaming. Then since $\mathcal{S}$ is valid, we must have

$$b_0 \le t_0, \qquad b_0 + m_i \le t_0 + 1, \qquad \cdots \qquad b_0 + t_i m_i \le t_0 + i$$

and

$$b_0 + t_i m_i + m_j \le t_0 + i + 1, \qquad \cdots \qquad b_0 + t_i m_i + t_j m_j \le t_0 + i + j.$$

In order for $\mathcal{S}'$ to be valid, we require

$$b_0 \le t_0, \qquad b_0 + m_j \le t_0 + 1, \qquad \cdots \qquad b_0 + t_j m_i \le t_0 + j$$

and

$$b_0 + t_j m_j + m_i \le t_0 + j + 1, \qquad \cdots \qquad b_0 + t_j m_j + t_i m_i \le t_0 + j + i.$$

Since $m_i > m_j$, this follows immediately from the first set of equations. Thus $\mathcal{S}'$ is a valid solution, and the result follows.

(A "greedy stays ahead" argument would also work, e.g. by arguing inductively that for all t, the solution returned by the greedy algorithm has sent at most as many bits at time t as any other solution.)

Question 20 (5 marks)

(Long question.) For any connected graph G, we say that a vertex $v \in V(G)$ is *outside the core* if $G - v$ is connected. Prove that any connected graph G with at least two vertices contains at least one vertex outside the core. (One possible approach uses induction.)

Solution: One approach is via induction on the number n of vertices in G. If $n = 2$, then G must be the graph consisting of a single edge; hence both vertices in G are outside the core. Suppose as an inductive step that any k-vertex connected graph contains at least one vertex outside the core for some $k \ge 2$, and let G be a $(k+1)$-vertex connected graph. If G is a tree, then G contains a leaf v, which is outside the core. Suppose instead that G contains a cycle C, and let $v \in V(G)$ be a vertex on that cycle. Let H be the component of $G - v$ containing $C - v$. Then by induction, H contains a vertex w outside the core, and moreover v sends at least two edges into H; hence $G - w$ is still connected.

Alternatively, pass to a spanning tree T of G. Any tree has at least one leaf w, and $G - w$ is still connected.

Question 21 (5 marks)

(Long question.) Consider a barter economy with goods $G_1, \dots, G_t$. For all i and j, you can directly trade x_i units of G_i for y_j units of G_j; this is expressed as the ratio $r_{i,j} = x_i/y_j$. Notice that you can trade one unit of G_i for $r_{ij}r_{jk}$ units of G_k, by first trading G_i for G_j and then trading G_j for G_k; the same holds for longer chains of trades. You are interested in becoming obscenely wealthy, and so you are looking for a *trading cycle* $C = x_{i_1} \dots x_{i_t} x_{i_1}$ such that the product $r_{i_1,i_2} \dots r_{i_{t-1},i_t} r_{i_t,i_1}$ of all the ratios is strictly greater than 1, leaving you with more goods G_1 than you started with. We call such a cycle an *arbitrage cycle*. Give a polynomial-time algorithm to decide whether an arbitrage cycle exists, and explain why it works.

(**Hint:** Recall that distances are not well-defined in a directed graph with cycles of negative total weight. In fact, you can check whether such cycles exist in polynomial time by running the Bellman-Ford algorithm for one extra iteration and seeing whether the weights change — if they do, then the graph contains a negative-weight cycle.)

Solution: First generate the graph with vertex set $\{G_1, \dots, G_t\}$, edge set $\{\{G_i, G_j\} : i, j \in [t], i \neq j\}$, and edge weights $w(\{G_i, G_j\}) = \log(1/r_{i,j})$. Run Bellman-Ford to check whether G has a negative-weight cycle, and return `Yes` if it does and `No` otherwise.

The total weight of a cycle $C = G_{i_1} \dots G_{i_t} G_{i_1}$ in G is given by

$$\log(1/r_{i_1,i_2}) + \cdots + \log(1/r_{i_{t-1},i_t}) + \log(1/r_{i_t,i_1}) = -\log\Big(r_{i_1,i_2} \cdot \ldots \cdot r_{i_{t-1},i_t} \cdot r_{i_t,i_1}\Big).$$

Thus the total weight of C in G is negative if and only if the product $r_{i_1,i_2} \cdot \ldots \cdot r_{i_{t-1},i_t} \cdot r_{i_t,i_1}$ is greater than 1, i.e. if and only if C is an arbitrage cycle, and our algorithm outputs the correct answer.

Question 22 (5 marks)

(Long question.) In the problem ZERO-SUM-SET, you are given a positive integer k and a set X of integers and asked whether there exists a size-k subset of X which sums to zero. For example, $k = 3$ and $X = \{2, -3, 1, 4\}$ is a `Yes` instance of ZERO-SUM-SET since $2 - 3 + 1 = 0$, but it would be a `No` instance for $k = 2$.

The problem ZERO-SUM-LISTS is similar, but the input is k lists $X_1, \dots, X_k$ of integers rather than a single set, and you are asked whether you can choose one integer from each list such that their sum is zero — that is, choose $x_1, \dots, x_k$ with $x_i \in X_i$ for all i and $\sum_i x_i = 0$. For example, $X_1 = \{2, 4, 5\}$, $X_2 = \{-4, -1, 5\}$, $X_3 = \{1, 2\}$ is a `Yes` instance of ZERO-SUM-LISTS because $2 - 4 + 2 = 0$, $2 \in X_1$, $-4 \in X_2$ and $2 \in X_3$.

Give a Karp reduction from ZERO-SUM-LISTS to ZERO-SUM-SET. (You may assume all arithmetic operations take constant time.)

Solution: There are many valid approaches, but probably the easiest ones to spot involve bit shifting through multiplication by powers of two. Let $(X_1, \dots, X_k)$ be an instance of ZERO-SUM-LISTS. If $k = 1$ then we take our instance of ZERO-SUM-SET to be $(1, X_1)$; it is immediate that this has the same answer and can be computed in

(cont.)

polynomial time. Otherwise, we have $k \geq 2$. For all $i \in [k]$, in polynomial time we can calculate

$$Y_i = \begin{cases} \{2^{k^2}x + 2^{(i-1)k} \colon x \in X_i\} & \text{if } i \in [k-1], \\ \{2^{k^2}x - \sum_{i=1}^{k-1} 2^{(i-1)k}\} & \text{if } i = k, \end{cases}$$

and let $Y = Y_1 \cup \cdots \cup Y_k$. Then we claim (k, Y) is a `Yes` instance of ZERO-SUM-SET if and only if $(X_1, \dots, X_k)$ is a `Yes` instance of ZERO-SUM-LISTS.

First suppose $(X_1, \dots, X_k)$ is a `Yes` instance; then there exist $x_1, \dots, x_k$ such that $x_i \in X_i$ for all i and such that $x_1 + \cdots + x_k = 0$. It follows that

$$(x_1 + 1) + (x_2 + 2^k) + \cdots + (x_{k-1} + 2^{(k-2)k}) + (x_k - 1 - 2^k - \cdots - 2^{(k-2)k}) = 0.$$

For all $i \in [k-1]$ we have $x_i + 2^{(i-1)k} \in Y_i$, and we have $x_k - \sum_{i=1}^{k-1} 2^{(i-1)k} \in Y_k$; hence these terms are all in Y, and (k, Y) is a `Yes` instance as required.

Now suppose (k, Y) is a `Yes` instance; then there exist $y_1, \dots, y_k \in Y$ such that $y_1 + \cdots + y_k = 0$. In particular, the k least significant bits of $y_1 + \cdots + y_k$ must be zero; hence $\{y_1, \dots, y_k\}$ must contain either at least one element of Y_1 and one of Y_k, or no elements of either. Similarly, bits $k+1$ through $2k$ of $y_1 + \cdots + y_k$ must be zero, and so $\{y_1, \dots, y_k\}$ must contain either at least one element of Y_2 and one of Y_k, or no elements of either. (Note in particular that bits $k+1$ through $2k$ can't be affected by terms $y_i \in Y_1$, since $k < 2^k$.) Continuing in this way, we see that for all $i \in [k-1]$, $\{y_1, \dots, y_k\}$ must contain either at least one element of Y_i and one of Y_k, or no elements of either. Since $\{y_1, \dots, y_k\}$ definitely contains an element of at least one set Y_i, we conclude that it contains at least one element of each of $Y_1, \dots, Y_k$, and since there are only k elements in total it must contain exactly one from each. The corresponding elements $x_i \in X_i$ also sum to zero, so $(X_1, \dots, X_k)$ is a `Yes` instance as required.

END OF PAPER