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) $n \in \Omega(\sqrt{n})$. (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)

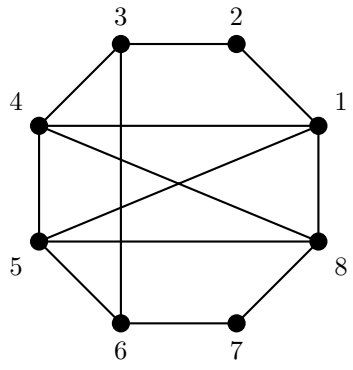
Solution: True.

(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)

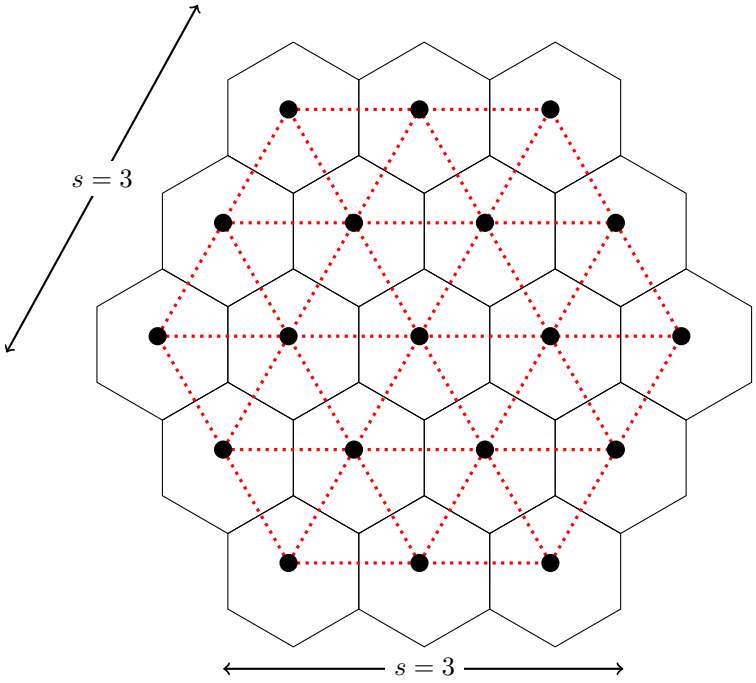
Write down an Euler walk for the following graph.



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

Question 3 (5 marks)

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.



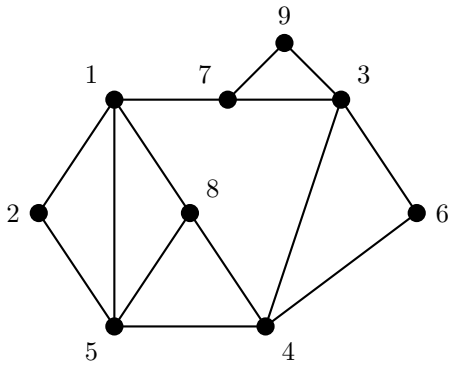
Give a formula for the number of edges in G_s in terms of s. You may use the fact that G_s has $3s^2 - 3s + 1$ vertices. (You do not need to show your working — only your final answer will be marked. You may wish to check that your answer is correct when $s = 2$.)

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)

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



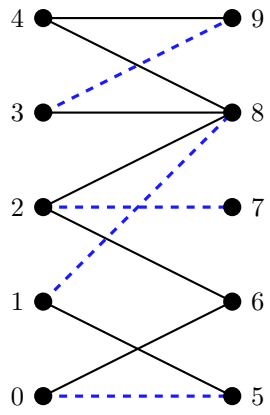
In the implementation of depth-first search given in lectures, we say vertex i is *explored* when `explored`$[i]$ is set to 1. (For example, the start vertex is explored first.) Which vertex will be explored sixth if the start vertex is 4? Assume that whenever the search has a choice of two or more vertices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

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)

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 $\{1, \ldots, 8\}$.

(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)

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.

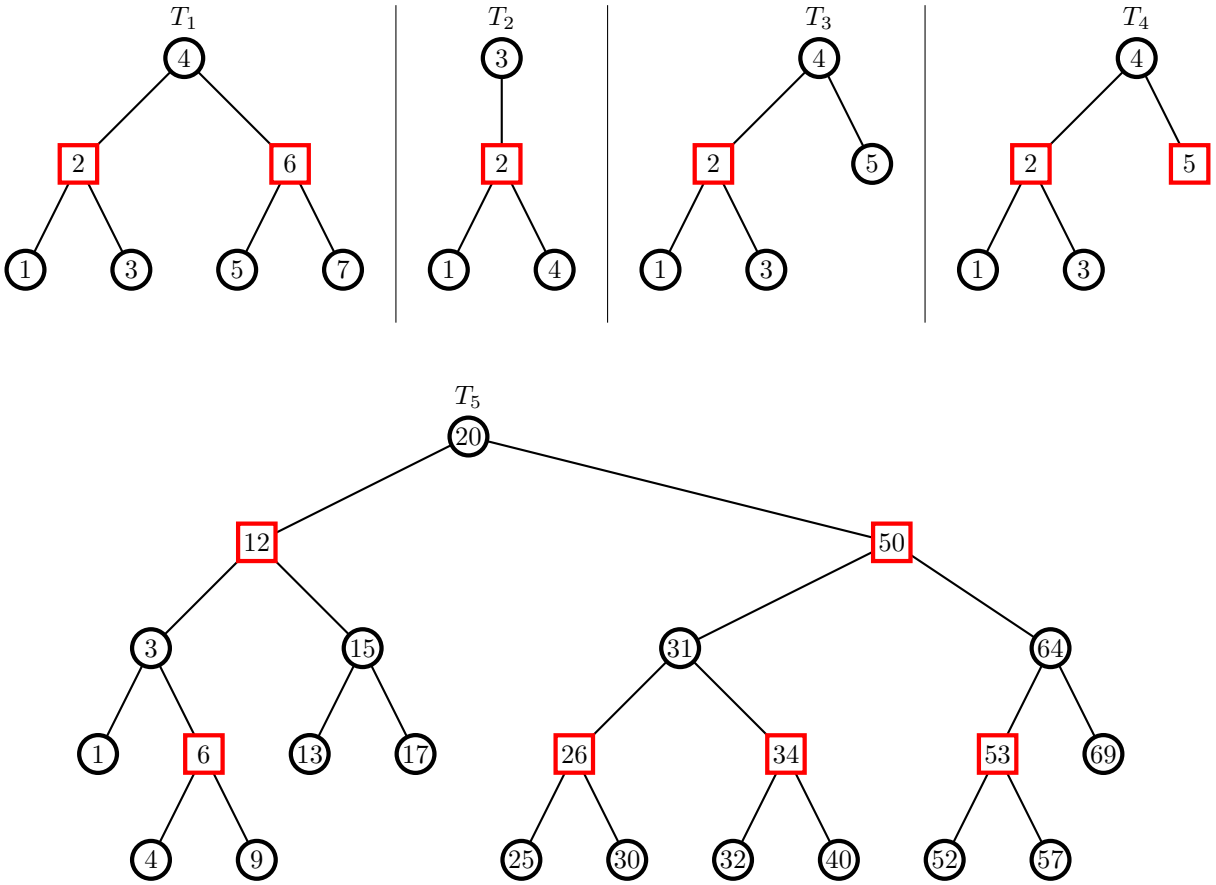
F. 202.

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)

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)

Consider an instance of interval scheduling with interval set

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

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

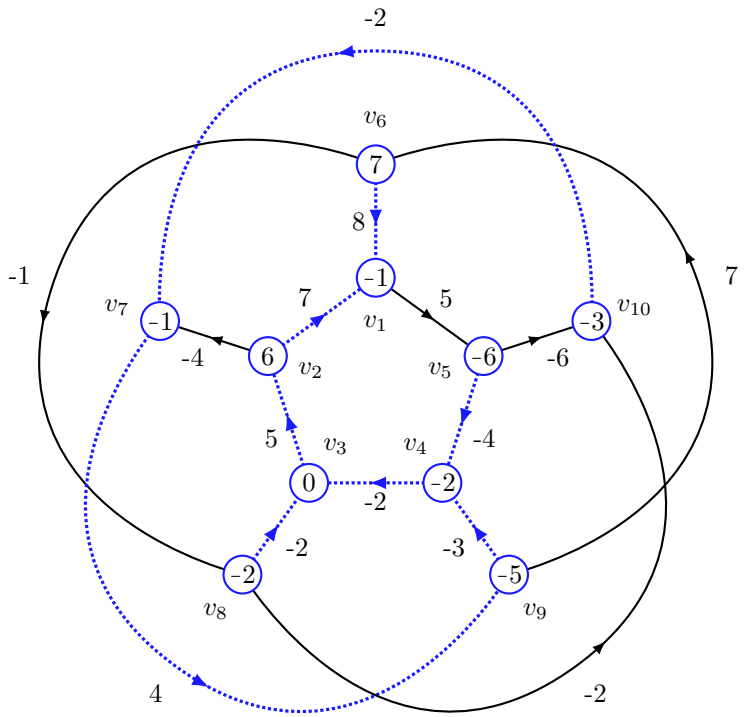
Solution: $(7, 8)$.

(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)

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)

Solution: -5.

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

(cont.)

Solution: -3.

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

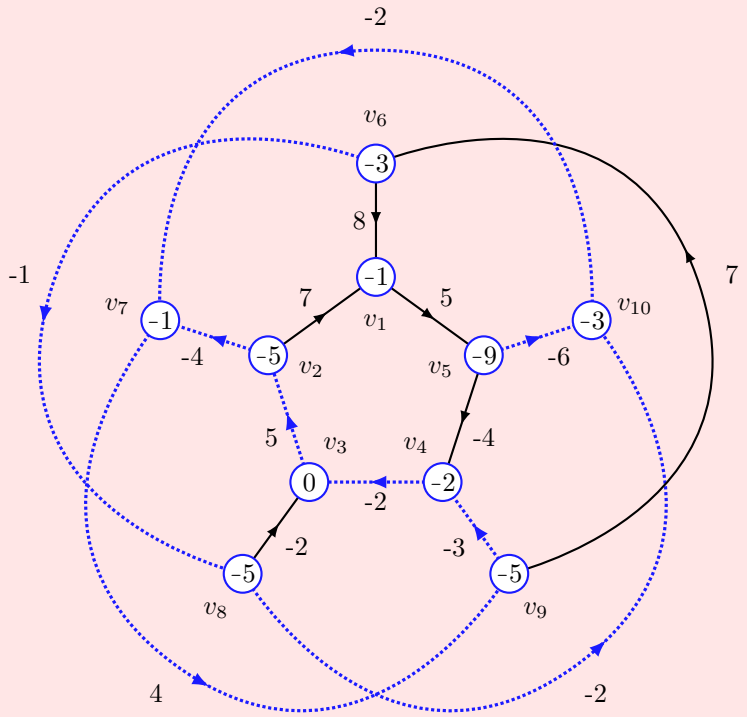
Solution: -3.

(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)

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* $b_1, \ldots, b_t$. 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 $\texttt{next}[b][s][a]$. Two marks will be awarded per blank correctly filled in.)

Algorithm: BeATambourineHero

Let $\texttt{next}[b][s][a]$, $\texttt{score}[b][s][a] \leftarrow 0$ 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 $\texttt{new\_s\_inactive} \leftarrow$ _____.
    Let $\texttt{inactive\_score} \leftarrow p_b + \texttt{score}[b+1][\texttt{new\_s\_inactive}][0]$.
    Let $\texttt{activating\_score} \leftarrow p_b + \texttt{score}[b+1][\texttt{new\_s\_inactive}][1]$ if $s \geq 50$, and $\texttt{activating\_score} \leftarrow -1$ otherwise.
    Let $\texttt{score}[b][s][0] \leftarrow \max(\texttt{inactive\_score}, \texttt{activating\_score})$.
    Let $\texttt{score}[b][s][1] \leftarrow 2p_b + \texttt{score}[b+1][\texttt{new_s_inactive} - 1][$_____$]$ if $\texttt{new\_s\_inactive} \geq 2$, and $\texttt{score}[b][s][1] \leftarrow 2p_b + \texttt{score}[b+1][0][0]$ otherwise.
    If $\texttt{activating\_score} > \texttt{inactive\_score}$, let $\texttt{next}[b][s][0] = 1$.
    If $\texttt{new\_s\_inactive} \geq 2$, let $\texttt{next}[b][s][1] = 1$.
Let $\texttt{active} \leftarrow 0$ 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 $\texttt{active} = 1$ and $s \leftarrow \min(100, s + s_{b+1})$ otherwise.
  Let $\texttt{active} \leftarrow \texttt{next}[b+1][s][$_____$]$.

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

Algorithm: BeATambourineHero

Let $\texttt{next}[b][s][a]$, $\texttt{score}[b][s][a] \leftarrow 0$ 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 $\texttt{new\_s\_inactive} \leftarrow \min(100, s + s_b)$.
    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 $s \geq 50$, 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 $\texttt{new\_s\_inactive} \geq 2$, 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 $\texttt{new\_s\_inactive} \geq 2$, 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)

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 $\text{HP}(G, x, y)$ 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 $\text{HC}(G)$ 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 $\text{P} \neq \text{NP}$.) (2 marks)

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

Question 13 (5 marks)

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 total)

Question 14 (5 marks)

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 15 (15 marks)

You are running a long-haul trucking company. You have a fixed number of vehicles which transport material between a set of cities $C_1, \ldots, C_k$. 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:

$$\sum_{i,j=1}^{k} p_{i,j} x_{i,j} \to \max, \text{ subject to}$$

$$\sum_{j=1}^{k} x_{i,j} \leq T_i \text{ for all } i \in [k],$$

$$\sum_{i=1}^{k} x_{i,j} \leq t_j \text{ for all } j \in [k],$$

$$x_{i,j} \geq 0 \text{ for all } i,j \in [k].$$

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)

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 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) 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 $\{50, 51 \ldots, 56\}$.

(b) Let $G = (V, E)$ be a connected graph with edge weights given by $w \colon E \to \mathbb{R}$. You may assume that every edge gets a different weight. Let C be a cycle in G, and let e be the highest-weight edge in C. It can be shown that no minimum spanning tree of G contains e.

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

(cont.)

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

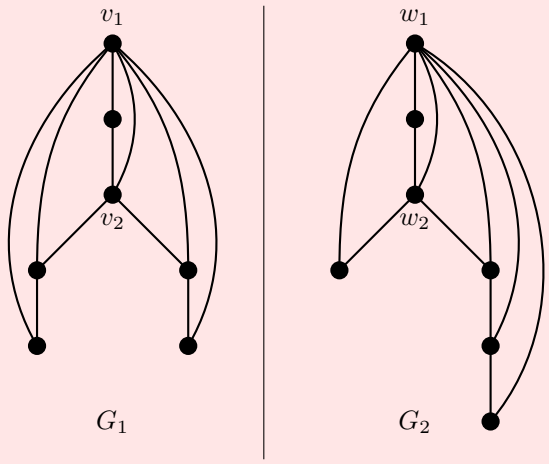
(c) John is trying to come up with an algorithm to test whether two **connected** graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic in polynomial time, and comes up with the following.

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 Δ be the maximum degree of G_1.
Let v be an arbitrarily-chosen vertex of degree Δ in G_1.
Using BFS, let L_i(v) = {x ∈ V_1 : d(v, x) = i in G_1} for all i ∈ [n].
for w ∈ V_2 of degree Δ in G_2 do
    Using BFS, let L_i(w) = {x ∈ V_2 : d(w, x) = i in G_2} for all i ∈ [n].
    if for all i, d ∈ [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.

(d) 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* 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`.

i. Give a Karp reduction from POS-EVEN-SAT to 3-SAT and briefly explain why it works.

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 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 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.

ii. Give a Karp reduction from 3-SAT to POS-EVEN-SAT and briefly explain why it works.

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 EVEN$(t, t, \neg t)$; observe that t must

be `True` in any satisfying assignment. We further add clauses 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 18 (15 marks)

Choose **one** 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) 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 $k \geq 2$, 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.

(b) Consider a barter economy with goods $G_1, \ldots, G_t$. 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* $C = x_{i_1} \ldots x_{i_t} x_{i_1}$ such that the product $r_{i_1,i_2} \ldots r_{i_{t-1},i_t} r_{i_t,i_1}$ 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 $\{G_1, \ldots, G_t\}$, 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.

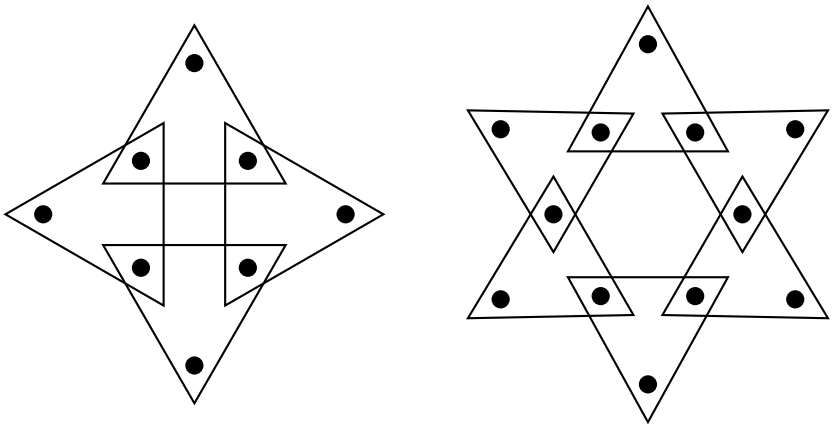
The total weight of a cycle $C = G_{i_1} \ldots G_{i_t} G_{i_1}$ in G is given by

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

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.

(c) 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" $f_1, \ldots, f_x$;

(cont.)

- 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 unnegated 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