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)

(Short question.) Which of the following options correctly finishes the following sentence? "We say $f(n) \in O(g(n))$ if there exist $C, n_0 > 0$ such that..."

A. $f(n_0) \leq Cg(n_0)$.

B. $f(n) \geq Cg(n)$ for all $n \geq n_0$.

C. $f(n) \leq Cg(n)$ for all $n \geq n_0$.

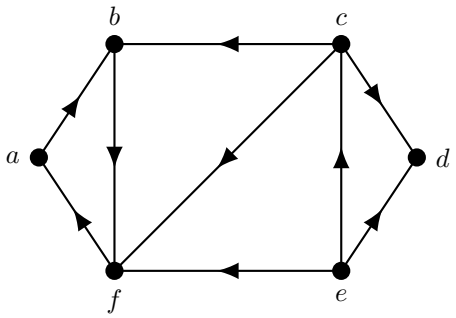
D. $f(n) \leq Cg(n)$ for some $n \geq n_0$.

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

Solution: C. $f(n) \in O(g(n)$ if there exist $C, n_0 > 0$ such that $f(n) \leq Cg(n)$ for all $n \geq n_0$. (Covered in lectures.)

Question 2 (5 marks)

(Short question.) Consider the directed graph G given below.



(a) What is the in-degree of f? (1 mark)

Solution: 3.

(b) What is the out-neighbourhood of d? (1 mark)

Solution: The empty set.

(c) Is G weakly connected? (1 mark)

Solution: Yes. For any two vertices u and v, G contains either a path from u to v or from v to u.

(d) Is G strongly connected? (1 mark)

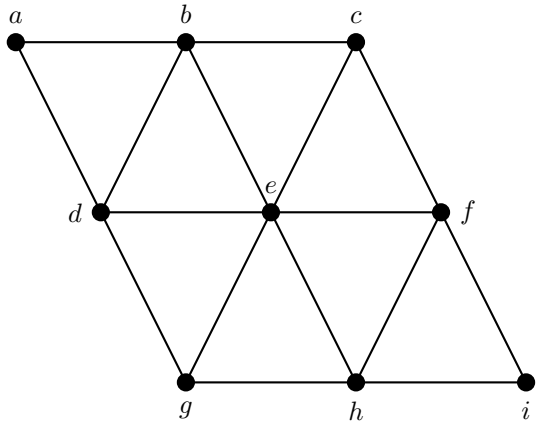
Solution: No. There is no path from any vertex in $\{a, b, f\}$ to any vertex in $\{c, d, e\}$.

(e) Does G contain an Euler walk? (1 mark)

Solution: No. Every vertex other than a has in-degree different from its out-degree.

Question 3 (5 marks)

(Short question.) Consider the graph G below.



Is each of the following subsets of $V(G)$ an independent set, a vertex cover, both, or neither?

(a) $\{e\}$. (1 mark)

Solution: Independent set only.

(b) $\{a, b, c, d, e, f, g, h, i\}$. (1 mark)

Solution: Vertex cover only.

(c) $\{a, e, i\}$. (1 mark)

Solution: Independent set only.

(d) $\{a, c, d, h\}$. (1 mark)

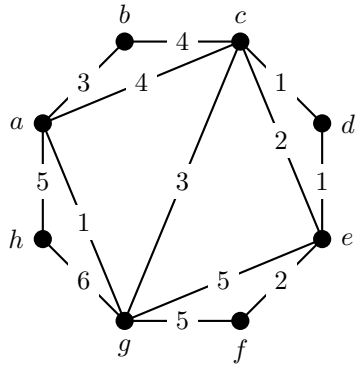
Solution: Neither a vertex cover nor an independent set.

(e) $\{a, c, d, e, h, i\}$. (1 mark)

Solution: Vertex cover only.

Question 4 (5 marks)

(Short question.) Suppose we run Prim's algorithm on the weighted graph below with initial vertex a. List the edges of the output minimum spanning tree in the order they are added by the algorithm. You may assume the algorithm breaks ties between edges any way you like.



Solution: There are two possible answers. One is $\{a, g\}$, $\{g, c\}$, $\{c, d\}$, $\{d, e\}$, $\{e, f\}$, $\{a, b\}$, $\{a, h\}$; the other is $\{a, g\}$, $\{a, b\}$, $\{g, c\}$, $\{c, d\}$, $\{d, e\}$, $\{e, f\}$, $\{a, h\}$.

Question 5 (5 marks)

(Short question.) Which of the following options correctly fills the blank in the following two sentences? "The greedy interval scheduling algorithm covered in lectures works by gradually building up a compatible set X. At each stage, the algorithm adds an interval to X which _____ subject to the new set X still being compatible."

A. "has a finish time as early as possible".

B. "has a start time as early as possible".

C. "intersects as few intervals in X as possible".

D. "is as short as possible".

E. None of the above.

Solution: A, has a finish time as early as possible.

Question 6 (5 marks)

(Short question.) Suppose that $G = ((V, E), w)$ is directed weighted graph, and let $s \in V$. Suppose that we are running Dijkstra's algorithm to find the distance $d(s, v)$ from s to v for all $v \in V$. Suppose that we have already found a set $X \subset V$ such that

for all $x \in X$, the distance $d(s, x)$ from s to x is known. In order to increase the size of X, Dijkstra's algorithm finds an edge from a vertex u inside X to a vertex v outside X with a certain property that makes the algorithm work. What is that property?

A. $d(x, u) \cdot w(u, v)$ is as small as possible.

B. $w(u, v)$ is as small as possible.

C. v's degree is as small as possible.

D. $d(x, u) + w(u, v)$ is as small as possible.

E. None of the above.

Solution: D, $d(x, u) + w(u, v)$ is as small as possible. (Covered in lectures.)

Question 7 (5 marks)

(Short question.) In lectures, we discussed two main versions of the union-find data structure, both of which could efficiently handle long sequences of operations. The optimised version of the data structure dynamically flattened tree components in `FindSet` and `Union` operations, while the unoptimised version did not do this.

(a) What is the worst-case running time of the **unoptimised** version on a sequence of n operations? (3 marks)

Solution: $\Theta(n \log n)$. (Note that I wouldn't take off marks for writing $O(\cdot)$ instead of $\Theta(\cdot)$ here.)

(b) What is the worst-case running time of the **optimised** version on a sequence of n operations? (2 marks)

Solution: $\Theta(n\alpha(n))$, where α is the inverse Ackermann function.

Question 8 (5 marks)

(Short question.) Which of the following algorithms would be most appropriate to find a shortest path between two given vertices in a **directed** graph with **both** negative-weight edges **and** negative-weight cycles?

A. Depth-first search.

B. Breadth-first search.

C. Dijkstra's algorithm.

D. The Bellman-Ford algorithm.

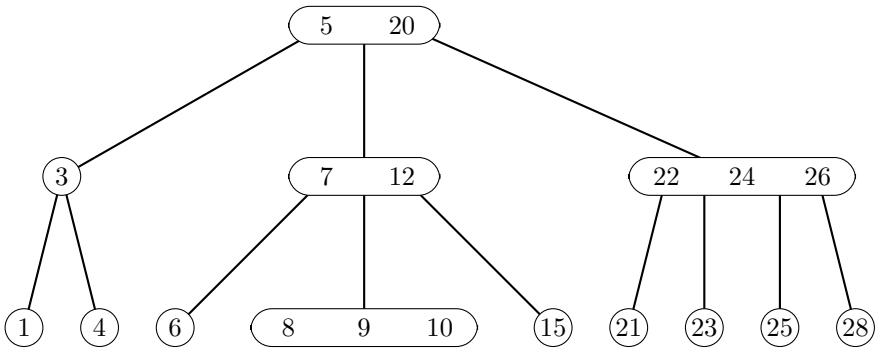
E. None of the above, or more than one of the above.

Solution: E, none of the above. Breadth-first search and depth-first search only work on unweighted graphs, Dijkstra's algorithm only works on graphs with non-negative edge weights, and Bellman-Ford only works on graphs without negative-weight cycles. (Covered in lectures.) This problem is actually NP-hard, so there is probably no polynomial-time algorithm for it.

Question 9 (5 marks)

(Short question.) If the below 2-3-4 tree were expressed as a red-black tree under the equivalence discussed in lectures, how many **red** nodes would it have in total?

(**Hint:** You don't need to fully convert the tree to answer this question.)



Solution: 6. None from the 2-nodes, 2 total from the two 3-nodes, and 4 total from the two 4-nodes.

Question 10 (5 marks)

(Short question.) Mark each of the following statements true or false.

(a) A linear programming problem with variables x and y could not have $x^2 + y^2$ as an objective function. (1 mark)

Solution: True. Every linear programming problem's objective function must be linear, i.e. of the form $\vec{a} \cdot \vec{x} \to \max$ or $\vec{a} \cdot \vec{x} \to \min$ for some vectors $\vec{a}$ of coefficients and $\vec{x}$ of variables. (Covered in lectures.)

(b) The linear programming problem $x + y \to \max$ subject to $y \leq 2$, $x \geq 1$ and $x, y \geq 0$ is unbounded. (1 mark)

Solution: True. For any $a \geq 1$, $(x, y) = (a, 0)$ is a solution with value a.

(c) The simplex algorithm always terminates in polynomial time. (1 mark)

Solution: False. There are examples where it takes exponential time. (Covered in lectures.)

(d) Every linear programming problem has at least one feasible solution. (1 mark)

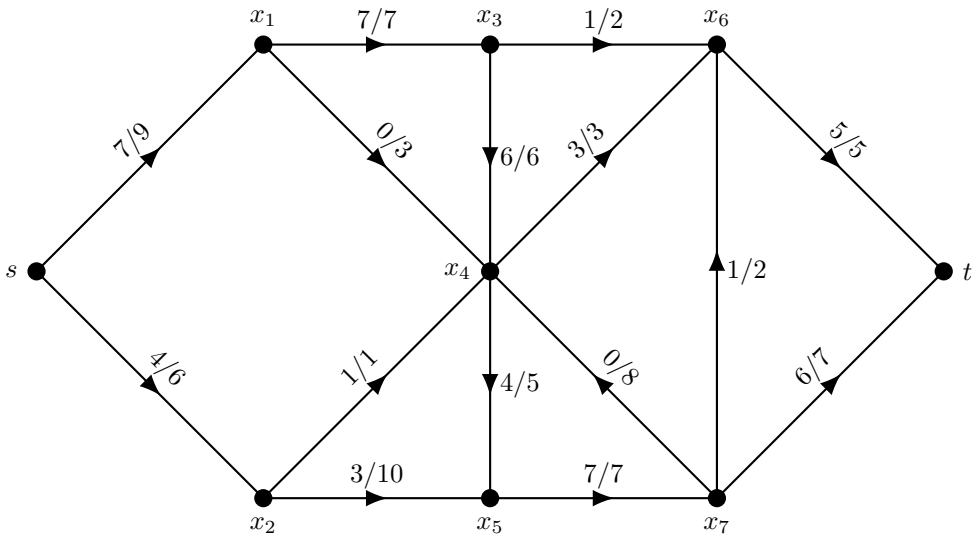
Solution: False. Many linear programming problems have no feasible solutions, e.g. $x \to \max$ subject to $x \leq 1$ and $x \geq 2$. (Covered in lectures.)

(e) Every linear programming problem can be expressed in standard form. (1 mark)

Solution: True. The process is explained in lectures.

Question 11 (10 marks)

(Parts (a)–(e) are short questions, parts (f) and (g) are medium questions.) Consider the flow network (G, c, s, t) and the flow f shown below.



(a) What is $c(x_7, x_4)$? (1 mark)

Solution: 8.

(b) What is $f^-(x_6)$? (1 mark)

Solution: 5. x_6 receives 1 flow from x_3, 3 flow from x_4 and 1 flow from x_7.

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

Solution: 11. This is $f(s, x_1) + f(s, x_2) = 7 + 4$.

(d) Is $(\{x_1, x_3, x_6\}, \{s, x_2, x_4, x_5, x_6, x_7, t\})$ a valid cut? (1 mark)

Solution: No. A valid cut would have the source vertex s in one part of the partition and the sink vertex t in the other.

(e) Is $(\{s, x_1, x_6\}, \{x_2, x_3, x_4, x_5, x_7, t\})$ a valid cut? (1 mark)

(cont.)

Solution: Yes. It's true that $G[\{s, x_1, x_6\}]$ is not a connected graph, but this is not a requirement for the cut to be valid (and this was highlighted on the quiz).

(f) Find an augmenting path P for f which the Edmonds-Karp algorithm might push flow down. (**Hint:** What's the difference between Edmonds-Karp and Ford-Fulkerson?)

(3 marks)

Solution: $\boldsymbol{sx_1x_4x_3x_6x_7t}$ is the only possible answer. (There is another augmenting path, namely $sx_2x_5x_4x_3x_6x_7t$, but Edmonds-Karp always chooses a *shortest* augmenting path.)

(g) Show the result of pushing flow down P as in the Edmonds-Karp or Ford-Fulkerson algorithms. In other words, writing F for the new flow that results from pushing flow down P, give the value of $F(e)$ for all edges e such that $F(e) \neq f(e)$.

(2 marks)

Solution: We can push a total of 1 flow down the path. The changed values are:

$$F(s, x_1) = 8, \quad F(x_1, x_4) = 1, \quad F(x_4, x_3) = 5,$$
$$F(x_3, x_6) = 2, \quad F(x_6, x_7) = 0, \quad F(x_7, t) = 7.$$

If your answer to (a) was $sx_2x_5x_4x_3x_6x_7t$, then the corresponding answer to (b) would be

$$F(s, x_2) = 5, \quad F(x_2, x_5) = 4, \quad F(x_5, x_4) = 3, \quad F(x_4, x_3) = 5,$$
$$F(x_3, x_6) = 2, \quad F(x_6, x_7) = 0, \quad F(x_7, t) = 7.$$

Question 12 (5 marks)

(Short question.) What does it mean for a rooted tree T to be a DFS tree of a **connected** graph G?

A. $V(T) = V(G)$ and every edge in G is either between two adjacent layers of T or inside a layer of T.

B. $V(T) = V(G)$ and for all vertices $u, v \in V(G)$, the path uTv has length $d(u, v)$.

C. $V(T) = V(G)$ and T has $|V(G)| - 1$ edges.

D. $V(T) = V(G)$ and every edge in G that's not an edge in T is between a vertex and its ancestor in T.

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

Solution: D, every edge in G that's not an edge in T is between a vertex and its ancestor in T. A and B are properties of BFS trees, and C is a property of all trees.

Question 13 (5 marks)

(Medium question.) For each of the following statements, say whether it is certain to be true, conjectured to be true, conjectured to be false, or certain to be false. Here "Statement S is conjectured to be true/false" means that there is some conjecture discussed in the unit, such as P $\neq$ NP, which if correct would imply that S is true/false.

(a) 3-SAT is in Co-NP. (1 mark)

Solution: Conjectured to be false. If true, this would imply NP = Co-NP, and NP $\neq$ Co-NP is one of the conjectures covered in the unit.

(b) If X reduces to Y under Karp reductions, then X reduces to Y under Cook reductions. (1 mark)

Solution: Certain to be true. Any Karp reduction immediately gives a Cook reduction — just apply the Karp reduction to the instance, apply the oracle to the output, and return the result. (Covered in lectures.)

(c) Consider the decision problem with input (G, k), where G is a bipartite graph and k is a positive integer, where the desired output is `Yes` if G contains a matching of size at least k and `No` otherwise. This decision problem is in Co-NP. (1 mark)

Solution: Certain to be true. As covered in lectures, there is a polynomial time algorithm for the problem, and P $\subseteq$ Co $-$ NP.

(d) Any algorithm for an NP-complete problem (under Karp reductions) runs in exponential time or worse. (1 mark)

Solution: Certain to be false. Many NP-complete problems have sub-exponential time algorithms, and this fact is covered in lectures.

(e) If a problem is NP-hard under Cook reductions, then it is in NP. (1 mark)

Solution: Certain to be false. A problem doesn't have to be in NP (or even a decision problem) in order to be NP-hard, and this fact is covered in lectures.

Question 14 (5 marks)

(Medium question.) Draw a weighted undirected graph with **no** minimum spanning tree. You do not need to justify your answer.

Solution: Any disconnected graph is a valid answer.

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 15 (5 marks)

Let (G, c, s, t) be a flow network. List the conditions that must be fulfilled for a function $f\colon E(G) \to \mathbb{R}$ to be a valid flow according to the definition given in lectures. (No further information is required.)

Solution: There are two conditions:

- For every edge $e \in E(G)$, we have $f(e) \geq 0$ and $f(e) \leq c(e)$.
- For every vertex $v \in V(G) \setminus \{s, t\}$, we have $\sum_{x \in N^-(v)} f(x, v) = \sum_{x \in N^+(v)} f(v, x)$.

Question 16 (10 marks)

A *facet* of a polyhedron is the line along which two faces meet. For example, a cube has twelve facets, four along each square face. The *elongated square gyrobicupola* is a deeply horrible polyhedron with 26 faces, 8 of which are triangles and 18 of which are squares. How many facets does it have? (**Hint:** Apply the handshaking lemma to a suitable graph. Partial credit will be given if you show your working.)

Solution: Consider the graph G whose vertices are the faces of the elongated square gyrobicupola, joining two vertices if their faces meet at a facet. Then each facet of the polyhedron corresponds to exactly one edge, and each edge corresponds to exactly one facet, so the answer will be the number of edges of the graph. In this graph triangles have degree 3 and squares have degree 4, so by the handshaking lemma, the total number of edges is

$$\frac{1}{2} \sum_{v \in V(G)} d(v) = \frac{8 \cdot 3 + 18 \cdot 4}{2} = 48.$$

Question 17 (15 marks)

Recall that the Fibonacci sequence is defined by $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Prove that the number of independent sets in a length-k path is given by F_{k+3}. (**Hint:** Recall that the empty set is an independent set.)

Solution: We proceed by induction on k. Let $P = x_1 \ldots x_{k+1}$ be a length-k path. If $k = 0$, then there are $2 = F_{0+3}$ independent sets, namely $\{x_1\}$ and $\emptyset$. If $k = 1$, then there are $3 = F_{1+3}$ independent sets, namely $\{x_1\}$, $\{x_2\}$ and $\emptyset$. Suppose now that

$k \geq 2$ and the result holds up to $k-1$. Then the number of independent sets of P is equal to the number N_{in} of independent sets containing x_{k+1} plus the number N_{out} of independent sets not containing x_{k+1}. The former independent sets are precisely the independent sets of $x_1 \dots x_{k-1}$, while the latter are precisely the independent sets of $x_1 \dots x_k$; thus by induction we have $N_{\text{in}} = F_{k+1}$ and $N_{\text{out}} = F_{k+2}$. Thus the total number of independent sets in P is given by $F_{k+1} + F_{k+2} = F_{k+3}$, as required.

Question 18 (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 t be a natural number. Let G be an n-vertex graph with $2t$ vertices of odd degree and $n-2t$ vertices of even degree. Prove by induction or t or otherwise that we can write $E(G)$ as a **disjoint** union of edge sets

$$E(G) = E(P_1) \cup \dots \cup E(P_t) \cup E(H),$$

where $P_1, \dots, P_t$ are paths in G and H is a graph whose vertices all have even degree.

Solution: Base case: If $t = 0$, then we can simply take $H = G$.

Inductive step: Suppose the result holds for all $t \leq t_0$ for some $t_0 \geq 0$; then we must prove it holds for $t = t_0 + 1$.

We first form graphs G' and H^- by greedily removing cycles from G and adding them to H^- until no cycles remain. Since every vertex in a cycle has even degree, G' still has exactly k vertices of odd degree, and every vertex in H^- has even degree. Now let v be a vertex of odd degree in G', and let P be a path formed by starting from v and extending greedily until no further extensions are possible. (In other words, P is a maximal path with v as an endpoint.) Let x be the other endpoint of P, so that $N(x) \subseteq V(P)$. Since there are no cycles in G, x can only send one edge into $V(P)$, to its direct predecessor; thus $d(x) = 1$. Form G'' by removing P's edges from G; then G'' has $2t - 2 = 2(t-1)$ vertices of odd degree. By induction, we can write the edges of G'' as a disjoint union

$$E(G'') = E(P_1) \cup \dots \cup E(P_{t-1}) \cup E(H^+),$$

where $P_1, \dots, P_{t-1}$ are paths and H^+ is a graph whose vertices have even degree. Thus we have

$$E(G) = E(P_1) \cup \dots \cup E(P_{t-1}) \cup E(H^+) \cup E(P) \cup E(H^-),$$

and so we are done on taking $P_t = P$ and $H = H^+ \cup H^-$.

Alternative solution: Work by induction on t as before. This time, instead of forming G' and H^-, observe that by the handshaking lemma, each component of

G has an even number of vertices of odd degree. (Indeed, the handshaking lemma implies that for each component C of G, the sum $\sum_{v \in V(C)} d(v)$ is even; if there are an odd number of odd-degree vertices in C this cannot be true.) Thus we can take P to be any path between two vertices of odd degree in the same component.

(b) You are a university librarian trying to maintain your service through budget cuts. You have a wide range of subject areas, and there are many good journals in each one. You want to maintain access to at least k good journals in each subject area while spending as little as possible. Unfortunately, journal companies are predatory and do not allow you to buy only the journals you want. Instead, they sell "bundles" of journals and require universities to buy the entire bundle, even if they only want a single journal in that bundle. Also, some journals are good in multiple subject areas.

Formally, the JournalBuying problem is defined as follows. You are given a set S of subject areas, a set J of journals, and a positive integer k. For each subject $s \in S$, you are given a set $J_s \subseteq J$ of interesting journals; these sets need not be disjoint, and there may be sets in J not included in any J_s. You are also given a collection of bundles $B_1, \ldots, B_t \subseteq J$, and a non-negative integer price p_i for each bundle B_i. These bundles also need not be disjoint. To solve this problem, you must output a collection X of bundles satisfying the following two properties:

(i) For all $s \in S$, $|J_s \cap \bigcup_{B \in X} B| \geq k$.

(ii) Subject to (i), $\sum_{B \in X} p_B$ is as small as possible.

Show that the JournalBuying problem is NP-hard.

Solution: Let (G, x) be an instance of vertex cover. We form an instance of JournalBuying as follows. Take $k = 1$, $J = V(G)$, $S = E(G)$, and $J_s = S$ for all $s \in S$. We number the vertices by $V(G) = \{v_1, \ldots, v_n\}$ and take $B_i = \{v_i\}$ and $p_i = 1$ for all $i \in [n]$. On applying our oracle to this JournalBuying instance, we obtain a collection X of vertices of G. Condition (i) holds if and only if X has at least one vertex from each edge of G, i.e. if and only if X is a vertex cover of G. Thus by condition (ii), X is a minimum vertex cover of G. We can therefore return `Yes` if the resulting bundle has total cost at most x and `No` otherwise to solve our original instance (G, x) of vertex cover. Note that this procedure takes polynomial time apart from the oracle call, as required.

(c) A *Hamilton path* in a graph G is a path containing every vertex in the graph exactly once — for example, a Hamilton cycle minus an edge is a Hamilton path. It is NP-hard to decide whether or not a graph has a Hamilton path. However, the naive algorithm of checking every possible path takes $\Omega(n!)$ time, and we can do better than this. Using dynamic programming, give an algorithm HamPath(G, v) with running time $O(n2^n)$ to decide whether a graph G has a Hamilton path starting at a given vertex $v \in V(G)$, and give a brief explanation of why it works. (You don't have to write out any code or even pseudocode — writing the recurrence relation that HamPath is based on is good enough.)

Solution: The key recurrence relation is:

$$\textsc{HamPath}(G, v) = \begin{cases} \texttt{True} & \text{if } |\mathrm{V(G)}| = 1, \\ \bigvee_{w \in N(v)} \textsc{HamPath}(G - v, w) & \text{otherwise.} \end{cases}$$

If $|V(G)| = 1$, this is obvious. Otherwise, any Hamilton path $P = vx_1 \dots x_t$ in G starting at v must have $x_1 \in N(v)$ and must have $x_1 \dots x_t$ be a Hamilton path in $G - v$; conversely, any Hamilton path $x_1 \dots x_t$ of $G - v$ with $x_1 \in N(v)$ becomes a Hamilton path for G on adding v to the start.

(d) Suppose we are given a weighted graph G. We say a spanning tree is an *EST* (Even-weight Spanning Tree) if the total weight of its edges is even, and an *MEST* (Minimum Even-weight Spanning Tree) if it is an EST whose total weight is as small as possible (among ESTs). Suppose that the subgraph H of G composed of even-weight edges is connected, so that we can find an EST by running Kruskal's algorithm on H — in this case, we say G is *evenly connected*. However, this EST might not be an MEST — perhaps all MESTs contain pairs of odd edges. Motivated by this, we decide to try the following variant EvenKruskal of Kruskal's algorithm.

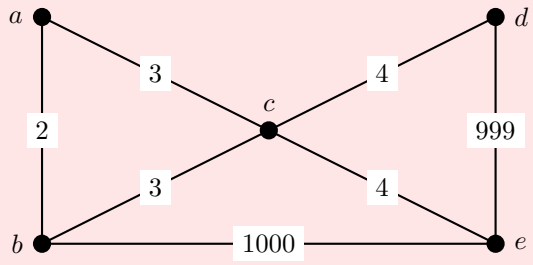
As short-hand, given a tree T, we say edge $e \in E(G)$ is *viable for* T if $e \notin E(T)$ and $T + e$ contains no cycles. Similarly, we say a pair of distinct edges $e, f \in E(G)$ is *viable for* T if $e, f \notin E(T)$ and $T + e + f$ contains no cycles.

Algorithm: EvenKruskal

```
1  Let T be the graph with V(T) = V(G) and E(T) = ∅.
2  while T is not a spanning tree of G do
3      Let T_next = T.
4      if there is a viable even-weight edge for T then
5          Let e be a viable even-weight edge whose weight is as small as possible.
6          Update T_next ← T + e.
7          Let w_even ← w(e).
8      else
9          Let w_even → ∞.
10     if there is a viable pair of odd-weight edges for T then
11         Let f, g be a viable pair of odd-weight edges whose total weight is as small as possible.
12         Let w_odd ← (w(f) + w(g))/2.
13         if w_odd < w_even then
14             Update T_next ← T + f + g.
15     Update T ← T_next.
16 Return T.
```

Give a counterexample to show that EvenKruskal need not output an MEST when given an evenly-connected graph as input, and give a brief explanation of why your counterexample works.

Solution: EvenKruskal is a variant of Kruskal's algorithm that adds either an even edge or two odd edges at each stage, preferring the odd edges if they have lower average weight. There are many counterexamples, but here is one of them. Let G be the graph below.



Notice G is evenly-connected, as *abecd* is a Hamilton path. The first choice EvenKruskal makes will be to add the edge $\{a, b\}$. The pair $\{a, c\}$ and $\{b, c\}$ is no longer viable, so it will now add the edges $\{c, d\}$ and $\{c, e\}$, followed by $\{d, e\}$ and either $\{a, c\}$ or $\{b, c\}$ for a total weight of $2+4+4+3+999 = 1012$. However, the spanning tree formed by deleting $\{d, e\}$ and $\{b, e\}$ from the graph has weight only 20.

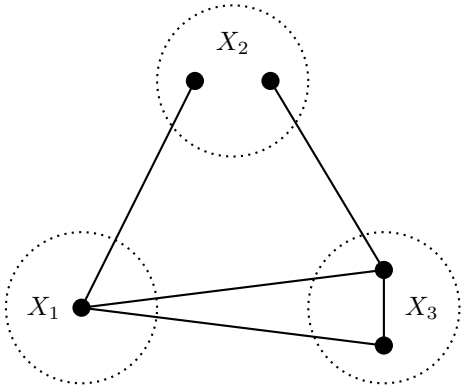
Question 19 (15 marks)

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) The decision problem of k-way min-cut is defined as follows. The input is an undirected graph $G = (V, E)$ and two positive integers k and a with $k \leq |V|$. A *k-way cut* is a list of sets $X_1, \ldots, X_k \subseteq V(G)$, and the *value* of a k-way cut is the total edge weight between sets, so that

$$\text{value}(X_1, \ldots, X_k) = \sum_{i \neq j} |\{\{u, v\} \in E \colon u \in X_i, v \in X_j\}|$$

For example, the 3-way cut shown for the graph below has value 4.



The desired output is `Yes` if G contains a k-way cut of value at most a, and `No` otherwise.

Meanwhile, the decision problem of integer linear programming takes as input an integer b and a linear programming problem which need not be in standard form. The desired output is `Yes` if the problem has a feasible solution which yields objective function value at most b **and** such that every variable in the solution has an

integer value. For example, a valid instance might take $b = 2$ and take the linear programming problem to be

$$\begin{aligned} x + y - z &\to \min, \text{ subject to} \\ x &\geq 1, \\ 3x - y + z &= 4, \\ 2x + y - z &\leq -3, \\ x, y, z &\in \mathbb{N}. \end{aligned}$$

Give a Karp reduction from k-way min-cut to integer linear programming and **briefly** explain why it works. (**Hint:** One approach has k variables per vertex and one variable per edge.) (15 marks)

Solution: Let (G, k, x) be an instance of k-way min-cut. We define an instance of integer linear programming as follows. Take $b = a$, and take our variables to be $\{x_{v,i} : v \in V, i \in [k]\}$ and $\{y_e : e \in E\}$. Then our linear program will be

$$\begin{aligned} \sum_{e \in E} y_e &\to \min, \\ x_{v,1} + \cdots + x_{v,k} &= 1 \text{ for all } v \in V \text{ and } i \in [k], \\ 0 \leq x_{v,i} &\leq 1 \text{ for all } v \in V \text{ and } i \in [k], \\ y_{\{u,v\}} &\geq -1 + x_{u,i} + \sum_{j \neq i} x_{v,j} \text{ for all } \{u, v\} \in E \text{ and } i \in [k], \\ 0 \leq y_{\{u,v\}} &\leq 1 \text{ for all } \{u, v\} \in E. \end{aligned}$$

The first and second constraints ensure that for all vertices $v \in V$, exactly one variable $x_{v,i}$ is equal to 1; thus k-way cuts map exactly to settings of the variables $x_{v,i}$. Moreover, under this map, the third and fourth constraints ensure that $y_{\{u,v\}} = 1$ if u and v lie in different sets X_i, and otherwise ensure that $y_{\{u,v\}} \in \{0, 1\}$. Clearly, in the optimal solution we will have $y_{\{u,v\}} = 0$ for all edges $\{u, v\}$ with endpoints in the same set X_i; thus in the optimal solution, the value of the objective function will equal the value of the k-way cut $(X_1, \ldots, X_k)$. It follows that the linear programming problem has a solution with value at most a if and only if G has a k-way cut with value at most a, as required. Finally, we observe that the linear programming problem is computable in time $O(k(|E| + |V|)) = O(|V||E|)$ since $k \leq n$, which is polynomial in $|(G, k, a)|$ as required.

(b) The *diameter* of a graph is the largest distance between any pair of vertices. Recall that if G is a graph then the *complement* of G, written G^c, is the graph with vertex set $V(G)$ and edge set $\{\{x, y\} : x, y \in V(G), x \neq y\} \setminus E(G)$ — that is, the graph formed by replacing all edges of G with non-edges and all non-edges with edges. Show that if G is a graph with diameter at least 4, then G^c has diameter at most 2.

(cont.)

Solution: Let $x, y \in V(G)$; we must show that $d(x,y) \leq 2$ in G^c. Suppose not, for a contradiction. Then there exist $x, y \in V(G)$ with $d(x,y) \geq 3$ in G^c.

Since $d(x,y) \geq 3$ in G^c, x must be adjacent to y in G. Since G has diameter at least 4, there must be two vertices a and b with $d(a,b) \geq 4$ in G. Neither x nor y can be adjacent to both a and b in G, or we would have $d(a,b) \leq 2$ in G. Suppose without loss of generality that x is not adjacent to a in G (or we can swap a and b). Then y must be adjacent to a in G, or xay would be a length-2 path in G^c from x to y. y must therefore not be adjacent to b in G. Thus x must be adjacent to b in G, or xby would be a length-2 path in G^c from x to y. But now $ayxb$ is a length-3 path from a to b, contradicting $d(a,b) \geq 4$.

(c) Suppose that a connected n-vertex, m-edge graph G (given in adjacency-list form) contains two vertices x and y which are distance $d(x,y) \geq n - c$ apart, for some integer $c \geq 1$. Give an algorithm which, given G, x, and an integer k as input, decides whether or not G contains a k-vertex clique in time $O(m + n + c^k)$. Briefly explain why it works and why it runs in the claimed time. (You do not need to give full pseudocode — an informal description is fine.)

Hint: You may find it helpful to use a BFS tree.

Solution: If $k \in \{1, 2\}$, then the algorithm simply returns `Yes`, as any connected graph on at least two vertices contains both a vertex and an edge.

Otherwise, run BFS starting at x to obtain a BFS tree T rooted at x in time $O(m+n)$. Since $d(x,y) \geq n - c$, T must contain at least $n - c + 1$ layers. It follows that all but at most c layers $L_1, \ldots, L_t$ contain only a single vertex, and that $L_1, \ldots, L_t$ between them contain at most c vertices. Since a vertex can only send edges to adjacent layers, it follows that every vertex in G has degree at most $c+1$. Moreover, since $k \geq 3$, every k-clique in G must contain at least one vertex from $L_1, \ldots, L_t$.

The algorithm therefore simply takes each vertex $v \in L_1, \ldots, L_t$, and checks whether each size-$(k-1)$ subset of $N(v)$ forms a k-clique together with v. There are $O(c)$ total vertices in $L_1, \ldots, L_t$, and there are $\binom{c+1}{k-1} \leq (c+1)^{k-1}$ subsets to check, so overall this takes $O(c^k)$ time.

END OF PAPER