THE LONDON SCHOOL OF ECONOMICS AND POLITICAL SCIENCE

Summer 2019 Mock Exam

MA421

Advanced Algorithms

Suitable for all candidates

Instructions to candidates

This paper contains **4** questions. Your answers to **all 4** questions will count towards the final mark. All questions carry equal numbers of marks.

Answers should be justified by showing work.

Please write your answers in dark ink (black or blue) only.

Time Allowed	**Reading Time:**	*None*
	Writing Time:	*2 hours and 30 minutes*
You are supplied with:		*No additional materials*
You may also use:		*No additional materials*
Calculators:		*Calculators are not allowed in this examination*

Question 1

An independent set of a graph $G = (V, E)$ is a subset $V' \subseteq V$ of vertices such that each edge in E is incident to at most one vertex in V'. The independent set problem is to find a maximum-size independent set in G.

(a) Formulate a related decision problem for the independent set problem, and prove that it is in $\mathcal{NP}$.

(b) Show that the decision problem you defined in (a) is $\mathcal{NP}$-complete.
Hint: You can use a reduction from the clique problem.

(c) Suppose that you are given a "black-box" subroutine to solve the decision problem you defined in part (a). Give an algorithm to find an independent set of maximum size. The running time of your algorithm should be polynomial in $|V|$ and $|E|$, counting queries to the black box as a single step.

(d) Although the independent set decision problem is NP-complete, certain special cases of it are polynomial-time solvable. Give an efficient algorithm to solve the independent set problem when each vertex in G has degree at most 2. Analyse the running time, and prove that your algorithm works correctly.

Question 2

In the knapsack problem, we have n items with sizes $s_1, \ldots, s_n$ and values $v_1, \ldots, v_n$, and a threshold (total knapsack size) B, where $s_1, \ldots, s_n, v_1, \ldots, v_n$, and B are positive integers. We want to find a set $I \subseteq \{1, \ldots, n\}$, of total size $s(I) = \sum_{i \in I} s_i \leq B$, and for which the total value $v(I) = \sum_{i \in I} v_i$ is maximised. Suppose that the items are labeled in the order of non-increasing value-to-size ratio:

$$\frac{v_1}{s_1} \geq \frac{v_2}{s_2} \geq \cdots \geq \frac{v_n}{s_n}.$$

(a) Show that a decision version of the knapsack problem is $\mathcal{NP}$-complete.
Hint: You can use set partition or subset sum problem for a reduction.

(b) Consider the following greedy approximation algorithm for the knapsack problem: place items in the knapsack in the given order until the next item no longer fits. In other words, return the set $\{1, \ldots, k\}$ such that $\sum_{i=1}^{k} s_i \leq B$ but $\sum_{i=1}^{k+1} s_i > B$. Show that this greedy algorithm can return an arbitrarily bad solution: that is, the ratio of the total value obtained by an optimal solution to the total value of the greedy solution can be arbitrarily high.

Let $i_{\max}$ be an index for which $v_{i_{\max}}$ has the largest value. Modify the greedy algorithm in (b) so that it returns the set $\{1, \ldots, k\}$ as above or $\{i_{\max}\}$, whichever has greater value. You will be asked below to show that this is a $\frac{1}{2}$-approximation algorithm.

(c) Suppose that, given an instance of the knapsack problem, integer k is such that $\sum_{i=1}^{k} s_i \leq B$ but $\sum_{i=1}^{k+1} s_i > B$. Let OPT be the total value obtained by an optimal solution for that instance. Then, show that $\sum_{i=1}^{k+1} v_i \geq \mathrm{OPT}$.
Hint: You may want to consider the items picked by an optimal solution in the order of non-increasing value-to-size ratio as well.

(d) Given (c), prove that the modified greedy algorithm is a $\frac{1}{2}$-approximation algorithm for the knapsack problem.

Question 3

Consider Karger's randomised minimum-cut algorithm that works by contracting randomly chosen edges of the input graph. In this question, we will analyse a technique to reduce the error probability of that algorithm.

This new algorithm proceeds in phases. In each phase, the algorithm duplicates the current graph, and contracts randomly and independently chosen edges of each duplicate graph until the number of vertices in each copy decreases by a factor of $5/4$: for example, from n to $4n/5$. (You may assume that $4n/5$ is an integer.) Then, the algorithm recursively and independently runs on the two graphs obtained in this way, stopping recursion when $n \leq 3$ and outputting a minimum cut in the resulting graph. At the end, the algorithm returns the minimum of all cuts found by the recursion.

You can visualise a complete binary tree representing the execution of this algorithm. The root corresponds to the input graph. Each edge in the binary tree represents one phase of the edge contractions on one of duplicates of the output graph from the previous phase.

(a) Give a brief intuitive reasoning why one might use the approach described above to find a minimum cut, instead of contracting edges until only two vertices remain in a single phase.

(b) Fix a minimum cut C in a graph G with n vertices. Show that the probability that no edge of C is contracted in a single phase (when reducing the number of vertices of G from n to $4n/5$) is at least $1/2$.

(c) Let $P(n)$ denote the probability that the above algorithm succeeds to find a minimum cut on a given input graph with n vertices. Derive a recurrence inequality for $P(n)$, which should be in terms of $P(4n/5)$.

(d) Use induction and the recurrence inequality you obtained in (c), to show that

$$P(n) \geq 1/\log_2 n,$$

for $n \geq 3$. You can use the fact that $2.32 < \log_2 5 < 2.33$.

(e) Based on the conclusion from (d), describe and analyse an algorithm that will output a minimum cut with probability at least $1 - \frac{1}{n}$.

Question 4

(a) Describe what are data stream algorithms, by giving their main characteristics. Describe a simple scenario where a data stream algorithm is much more desirable and practical than a conventional algorithm that requires random access to its input.

The following algorithm processes the elements in a data stream $\langle a_1, a_2, \ldots, a_m \rangle$, where each $a_i \in \{1, \ldots, n\}$, one at a time and stores frequency estimates for a subset A of up to $k-1$ values in $\{1, \ldots, n\}$, which we call the "tracked" elements. In the algorithm description, A denotes both the current set of values that are tracked as well as an array that keeps a count for each tracked value.

Algorithm 1 Finding heavy hitters in a data stream

Initialise: $A \leftarrow \emptyset$

Process a_j:
if $a_j \in A$ **then**
    $A[a_j] \leftarrow A[a_j] + 1$
else if $|A| < k - 1$ **then**
    Put a_j in A
    $A[a_j] = 1$
else
    for all $a \in A$ **do**
        $A[a] \leftarrow A[a] - 1$
        if $A[a] = 0$ **then**
            Remove a from A

Output: For $a \in A$, estimate $\hat{f}_a = A[a]$, else $\hat{f}_a = 0$.

(b) Argue that, for any $a \in \{1, \ldots, n\}$, $\hat{f}_a \leq f_a$, where f_a is the number of times value a appears in the stream: that is, $f_a = |\{i \mid a_i = a\}|$.

(c) Show that, for any $a \in \{1, \ldots, n\}$, $\hat{f}_a \geq f_a - (m/k)$.

(d) Argue how the output provided by the algorithm above can then be used to identify the values that appear more than m/k times in the data stream.

END OF PAPER