Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn

Exam Algorithm Theory

Thursday, March 31, 2016, 09:00-10:30

Name: .....

Matriculation Nr.: .....

Signature: .....

Do not open or turn until told so by the supervisor!

Instructions:

- Write your name and matriculation number on the cover page of the exam and sign the document! Write your name on all sheets!
- Your signature confirms that you have answered all exam questions without any help, and that you have notified exam supervision of any interference.
- Write legibly and only use a pen (ink or ball point). Do **not** use **red**! Do **not** use a pencil!
- You are **not** allowed to use any material except for a dictionary and a hand-written summary of at most 5 A4 pages (corresponds to 5 single-sided A4 sheets!).
- There are 5 problems (with several questions per problem) and there is a total of 90 points. At most 40% are needed to pass the exam, and 80% will net you the best grade, i.e., 18 points are bonus points.
- Use a separate sheet of paper for each of the 5 problems.
- Only one solution per question is graded! Make sure to strike out any solutions that you do not want to be considered!
- **Explain your solutions! Just writing down the end result is not sufficient unless otherwise indicated.**

Question	Achieved Points	Max Points
1		27
2		14
3		14
4		17
5		18
Total		90

Problem 1: Short Questions (27 points)

(a) (8 points) Let $G = (V, E)$ be a flow network with source node s, sink node t, and a positive integer capacity function $c : E \to \mathbb{N}$. Propose an efficient algorithm that, given an edge $e \in E, c(e) > 0$, decides whether e belongs to all minimum s-t cuts in G.

(b) (7 points) Given a graph $G = (V, E)$. A set $S \subseteq V$ is called a *vertex cover* if and only if for every edge $\{u, v\} \in E$ at least one of its endpoints is in S. The minimum vertex cover problem is to find such a set S of minimum size.

We are considering the following online version of the minimum vertex cover problem. Initially, we are given the set of nodes V and an empty vertex cover $S = \emptyset$. Then, the edges appear one-by-one in an online fashion. When a new edge $\{u, v\}$ appears, the algorithm needs to guarantee that the edge is covered (i.e., if this is not already the case, at least one of the two nodes u and v needs to be added to S). Once a node is in S it cannot be removed from S.

Provide an online algorithm with competitive ratio at most 2. That is, your online algorithm needs to guarantee at all times that the vertex cover S is at most by a factor 2 larger than a current optimal vertex cover. Briefly discuss the correctness of your algorithm.

(c) (5 points) We are given a sequence $a_1, \cdots, a_n$ of n values. Recall from the lecture that computing all prefix sums can be done with $O(n)$ work and in $O(\log n)$ time in parallel with an infinite number of processors.

(1) (2 points) Argue why it is not possible to compute all prefix sums in $O(n^{1/3})$ time by using $\sqrt{n}$ processors.

(2) (3 points) What is the minimum number of processors (in Landau-Notation) with which you can guarantee that all prefix sums can be computed in $O(n^{1/3})$ time? Show that it cannot be done with fewer processors (asymptotically).

(d) (7 points) We are given a union-find data structure which is implemented as a disjoint-set forest with the "union by size" heuristic (without path compression). For any n show that there exists an $m \geq 3n$ and a sequence of m operations (MakeSet, Union, and FindSet) on n elements that takes $\Omega(m \log n)$ time to be performed on the data structure.

Aufgabe 1: Kurze Fragen (27 Punkte)

Lösen Sie die folgenden Aufgaben.

(a) (8 Punkte) Sei $G = (V, E)$ ein Flussnetzwerk mit Quelle s, Senke t, sowie einer positiven ganzzahligen Kapazitätsfunktion $c : E \to \mathbb{N}$ gegeben. Finden Sie einen effizienten Algorithmus, der zu einer Kante $e \in E, c(e) > 0$ als Eingabe entscheidet, ob e zu jedem minimalen s-t Schnitt in G gehört.

(b) (7 Punkte) Sei ein Graph $G = (V, E)$ gegeben. Eine Teilmenge der Knoten $S \subseteq V$ heißt *vertex cover* genau dann, wenn für jede Kante $\{u, v\} \in E$ mindestens einer der beiden Knoten der Kante zu S gehört. Beim *minimum Vertexcoverproblem* möchte man ein Vertexcover S mit minimaler Größe finden.

Wir betrachten die folgende Onlineversion des minimum Vertexcoverproblems. Zu Beginn erhalten wir die Knotenmenge V und ein leeres Vertexcover $S = \emptyset$. Die Kanten erhält man nun nacheinander (online fashion). Direkt nach dem Erhalt einer neuen Kante $\{u, v\}$ muss der Algorithmus garantieren, dass die Kante abgedeckt ist (d.h., falls es noch nicht der Fall ist, muss mindestens einer der Knoten u und v zu der Menge S hinzugefügt werden). Sobald ein Knoten zu S hinzugefügt wird, kann er nicht mehr aus S entfernt werden.

Finden Sie einen Onlinealgorithmus mit einer Kompetivitätsrate von höchstens 2. D.h., der Onlinealgorithm muss garantieren, dass das Vertexcover S zu jeder Zeit maximal doppelt so groß ist, wie ein derzeit optimales Vertexcover.

Zeigen Sie kurz die Korrektheit Ihres Algorithmus'.

(c) (5 Punkte) Sei eine Folge $a_1, \cdots, a_n$ mit n Werten gegeben. In der Vorlesung haben wir gezeigt, dass alle Prefixsummen mit Arbeit (work) $O(n)$ und in Zeit $O(\log n)$ parallel mit unendlich vielen Prozessoren berechnet werden können.

(1) (2 Punkte) Erklären Sie, wieso es nicht möglich ist alle Prefixsummen in Zeit $O(n^{1/3})$ mit $\sqrt{n}$ Prozessoren zu berechnen.

(2) (3 Punkte) Wie viele Prozessoren benötigt man im Minimum um alle Prefixsummen in Zeit $O(n^{1/3})$ zu berechnen (in Landau-Notation)? Zeigen Sie, dass es mit asymptotisch weniger Prozessoren nicht geht.

(d) (7 Punkte) Sei eine union-find Datenstruktur gegeben, die durch einen Disjunkte-Mengen Wald mit der "union by size" Heuristik und ohne Pfadkompression implementiert ist. Zeigen Sie für jedes n, dass es ein $m \geq 3n$ sowie eine Folge von m Operationen (MakeSet, Union, and FindSet) auf n Elementen gibt, die Zeit $\Omega(m \log n)$ benötigt um auf der Datenstruktur durchgeführt zu werden.

Problem 2: Amortized Analysis (14 points)

Consider a data structure with the operations *insert* and *deleteFive*. The operation insert, adds a single element to the data structure and has actual cost 3. The operation *deleteFive* removes 5 elements from the data structure if it contains at least 5 elements, otherwise it does not have any effect. The actual cost of *deleteFive* is 45 if it does delete elements and 1 otherwise.

Use the potential function method to show that each operation has amortized cost **at most** 10. [It is not sufficient to show that each operation has constant amortized cost.]

Aufgabe 2: Amortisierte Analyse (14 Punkte)

Betrachten Sie eine Datenstruktur mit den Operationen *insert* und *deleteFive*. Die Operation *insert*, fügt der Datenstruktur ein einzelnes Element hinzu und hat tatsächliche Kosten von 3 Einheiten. Die Operation *deleteFive* entfernt 5 Elemente aus der Datenstruktur, falls diese mindestens 5 Elemente enthält, im anderen Fall hat die Operation keine Auswirkung. Die tatsächlichen Kosten von *deleteFive* sind 45 falls sie 5 Elemente löscht und ansonsten 1.

Benutzen Sie die Potentialfunktionsmethode um zu zeigen, dass jede Operation **höchstens** amortisierte Kosten 10 hat.

[Hier ist es nicht ausreichend zu zeigen, dass jede Operation konstante amortisierte Kosten hat.]

Problem 3: Jumping in Arrays (14 points)

We are given an array of n positive integers ($\mathcal{A} = [a_1, a_2, \ldots, a_n]$) which is indexed from 1 to n. A small frog sits on the first entry of the array. The frog aims to reach the last entry of the array by one or several jumps, and it has to jump according to the following rule: when the frog is on the i^{th} entry of the array, it can only jump to the j^{th} entry if $0 < j - i \leq a_i$.

(a) (12 points) Using dynamic programming, design an algorithm to calculate the minimum number of jumps with which the frog can reach the last entry of the array from the first entry.

(b) (2 points) What is the worst case time complexity of your algorithm?

Aufgabe 3: Springen in Arrays (14 Punkte)

Sei ein Array ($\mathcal{A} = [a_1, a_2, \ldots, a_n]$) mit n positiven Integern gegeben. Das Array ist von 1 bis n indiziert.

Ein kleiner Frosch sitzt auf dem ersten Eintrag des Arrays. Der Frosch möchte den letzten Eintrag des Arrays mit einem oder mehreren Sprüngen erreichen. Wenn er springt, so muss er folgende Regel beachten: Falls er sich auf dem i-ten Eintrag des Arrays befindet, so kann er nur zum j-ten Eintrag springen, wenn $0 < j - i \leq a_i$ gilt.

(a) (12 Punkte) Benutzen Sie das Konzept der dynamischen Programmierung um einen effizienten Algorithmus zu entwerfen, welcher die minmale Anzahl an nötigen Sprüngen um den letzten Eintrag des Arrays zu erreichen ausgibt.

(b) (2 Punkte) Was ist die Worstcase Zeitkomplexität Ihres Algorithmus'?

Problem 4: Greedy Approximation for Knapsack (17 points)

In the lecture, we have considered the Knapsack problem: There are n items with positive weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$ and a knapsack (a bag) of capacity W such that $w_i \leq W$ for all $1 \leq i \leq n$. A feasible solution to the problem is a subset of the items such that their total weight does not exceed W. The objective is to find a feasible solution of maximum possible total value.

Consider the following greedy algorithm:

1. Sort the n items such that $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}$.
2. Fill the knapsack sequentially with items in the above sorted order starting with the item with largest value per weight. The algorithm stops either if there there are no more items left or it reaches an item $k \leq n$ which does not fit, i.e., $w_k > W - \sum_{i=1}^{k-1} w_i$.

(a) (5 points) Show that the solution of the greedy algorithm can be arbitrarily bad compared to an optimal solution.

(b) (12 points) Using a modification to the greedy algorithm, it is possible to get a 2-approximation for the problem. Present such a modified greedy algorithm and show that it provides approximation factor of 2.

Hint: *In the fractional knapsack problem one is allowed to fill the knapsack with fractions of items, rather than having to make a binary* 0-1 *choice for each item.*

You can use that the following variant of the above greedy algorithm is optimal for the fractional knapsack problem: The knapsack is sequentially filled with items in the above sorted order starting with the item with largest value per weight. When reaching an item $k \leq n$ which does not fit, we add a fraction of the item so that the knapsack is filled to its capacity W.

Aufgabe 4: Greedy Approximation für Knapsack (17 Punkte)

In der Vorlesung haben wir das Knapsackproblem angeschaut: Es seien n Gegenstände mit positiven Gewichten $w_1, \ldots, w_n$ und Nutzen $v_1, \ldots, v_n$ sowie ein Knapsack (ein Rucksack) mit Kapazität W gegeben, so dass für alle Gegenstände $w_i \leq W$ gilt. Eine zulässige Lösung für das Problem ist eine Teilmenge der Gegenstände (die eingepackten Gegenstände), so dass ihr Gesamtgewicht nicht die Knapsackkapazität W überschreitet. Das Ziel ist es eine zulässige Lösung zu finden, die den Gesamtnutzen der eingepackten Gegenstände maximiert.

Betrachten Sie den folgenden Greedyalgorithmus:

1. Sortiere die n Gegenstände so dass $\frac{v_1}{w_1} \geq \frac{v_2}{w_2} \geq \ldots \geq \frac{v_n}{w_n}$.
2. Fülle den Rucksack sequentiell mit den Gegenständen nach der obigen Ordnung beginnend mit dem Gegenstand, der das beste Nutzen pro Gewicht Verhältnis hat. Der Algorithmus stoppt, wenn entweder keine Gegenstände übrig sind oder ein Gegenstand $k \leq n$ erreicht wird, der nicht mehr herein passt, d.h., $w_k > W - \sum_{i=1}^{k-1} w_i$.

(a) (5 Punkte) Zeigen Sie dass die Lösung des Greedyalgorithmus im Vergleich zu einer optimalen Lösung beliebig schlecht sein kann.

(b) (12 Punkte) Mit einer Modifikation des Greedyalgorithmus' ist es möglich eine 2-Approximation für das Problem zu finden. Finden Sie solch einen Algorithmus und zeigen Sie, dass er eine 2-Approximation liefert.

Hinweis: *Beim Fractional Knapsack Problem darf man den Knapsack mit Teilen von Gegenständen füllen und muss nicht unbedingt ganze Gegenstände hineinlegen.*

Sie können benutzen, dass die folgende Variante des obigen Greedy Algorithmus optimal für das Fractional Knapsack Problem ist: Der Knapsack wird sequentiell mit den Gegenständen nach der obigen sortierten Reihenfolge gefüllt, beginnend mit dem Gegenstand mit dem besten Nutzen pro Gewicht Verhältnis. Sobald ein Gegenstand $k \leq n$ erreicht wird, der nicht mehr komplett in den Knapsack passt, wird soviel von ihm mitgenommen, dass der Knapsack bis zu seiner Kapazität W gefüllt ist.

Problem 5: Randomized Majority Evaluation (18 points)

Assume that we are given a rooted tree with n leaves. All leaves have the same distance ℓ from the root and all nodes except for the leaves have three children. A problem instance consists of such a tree and a boolean value for each leaf.

The value of each inner node is defined as the majority value of its children. Initially, the values of inner nodes are not given. The objective is to compute the value of the root.

The performance of an algorithm to solve this problem is measured by the number of leaves whose values are read by the algorithm.

(a) (5 points) Is there a deterministic algorithm that can solve the problem such that for every input, the algorithm does not need to read the values of all leaves?

(b) (3 points) Design a recursive randomized algorithm to determine the value of the root with certainty but without necessarily reading all the leaf values.

(c) (10 points) What is the expected number of leaves that your randomized algorithm needs to read? Give an upper bound for this expected number of leaves.

Aufgabe 5: Randomisierte Mehrheitsevaluation(18 Punkte)

Nehmen Sie an, dass ein gewurzelter Baum mit n Blättern gegeben ist. Alle Blätter haben den gleichen Abstand ℓ von der Wurzel und alle Knoten außer die Blätter haben drei Kinder. Eine Probleminstanz besteht aus solch einem Baum und einem boolean Wert für jedes Blatt.

Der Wert jedes inneren Knotens ist definiert als der Mehrheitswert seiner Kinder. Zu Beginn sind die Werte der inneren Knoten nicht gegeben. Ziel ist es den Wert der Wurzel zu berechnen.

Die Performance eines Algorithmus um dieses Problem zu lösen wird durch die Anzahl der Blätter dessen Werte gelesen werden bestimmt.

(a) (5 Punkte) Gibt es einen deterministischen Algorithmus um das Problem so zu lösen, dass der Algorithmus in jedem Fall nicht die Werte aller Kinder lesen muss?

(b) (3 Punkte) Designen Sie einen rekursiven randomisierten Algorithmus, der sicher den korrekten Wert der Wurzel ausgibt, jedoch nicht unbedingt die Werte aller Blätter abruft.

(c) (10 Punkte) Was ist die erwartete Anzahl an Blattwerten, die Ihr randomisierter Algorithmus lesen muss? Geben Sie eine obere Schranke für diesen Erwartungswert an.