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

Exam Algorithm Theory

Tuesday, September 06, 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		28
2		14
3		18
4		14
5		16
Total		90

Problem 1: Short Questions (28 points)

(a) (4 points) Consider the following divide and conquer algorithm to compute a minimum spanning tree of a graph $G(V, E)$ with distinct edge weights:

(I) Partition V into two sets V_1 and V_2 which are (approximately) equal in size. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be the subgraphs of G induced by V_1 and V_2 respectively.

(II) Recursively solve a minimum spanning tree problem on each of the two subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

(III) Add the minimum-weight edge across the cut (V_1, V_2) to connect the two sides.

Argue why this algorithm correctly computes an MST or show that the algorithm fails.

(b) (6 points) An independent set A of a **matroid** (E, I) is called *maximal* if for every element $x \in E \setminus A$, the set $A \cup x$ is not an independent set. Show that all maximal independent sets in a matroid have the same size.

(c) (4 points) Assume that you are given a parallel algorithm with $\Theta(n^{4/3})$ work (total number of operations) and $\Theta(n^{1/3})$ span (critical path length). What is the optimal asymptotic running time of this algorithm when you have p processors available (as a function of p and n)?

(d) (6 points) Recall the *weighted interval scheduling* problem of the lecture: We are given a set of intervals $[a_i, b_i]$ with weights w_i and the goal is to find a maximum weight subset of pairwise non-overlapping intervals.

A natural greedy algorithm for this problem would be as follows. Sort the intervals in decreasing order according to their weight by length ratio $w_i/(b_i - a_i)$ and try to add the intervals greedily in this order.

Show that this algorithm can be arbitrarily bad, i.e., show that for every constant $C > 1$, there is a problem instance for which the total weight of an optimal solution is more than C times the total weight of the solution of the described greedy algorithm.

(e) (8 points) For each of the following two statements, say whether it is true or false. In case a statement is true, explain why it is true, if it is false, give a (small) counterexample.

1) (4 points) In a graph where all edge weights are positive, if a subset S of the edges connects all vertices and has minimum total weight, then the edges in S form a tree.

2) (4 points) In a graph where edge weights may be positive or negative, if a subset S of the edges connects all vertices and has minimum total weight, then the edges in S form a tree.

Aufgabe 1: Kurze Fragen (28 Punkte)

(a) (4 Punkte) Wir betrachten den folgenden "Divide-and-Conquer"-Algorithmus, um einen Minimalen Spannbaum eines Graphen $G = (V, E)$ mit paarweise verschiedenen Kantengewichten zu berechnen:

(I) Partitioniere V in zwei (ungefähr) gleich grosse Mengen V_1 and V_2.
Seien $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ die Teilgraphen, welche von V_1 bzw. V_2 induziert werden.

(II) Löse das Minimale Spannbaum Problem rekursiv auf den Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$.

(III) Füge die leichteste Kante über den Schnitt (V_1, V_2) hinzu, um die zwei Teile zu verbinden.

Argumentieren Sie, wieso dieser Algorithmus einen Minimalen Spannbaum berechnet oder zeigen Sie, dass der Algorithmus nicht funktioniert.

(b) (6 Punkte) Ein Independent Set A eines **Matroids** (E, I) heisst *maximal* falls für jedes Element $x \in E \setminus A$ gilt, dass die Menge $A \cup x$ nicht ein Independent Set ist.

Zeigen Sie, dass alle maximalen Independent Sets eines Matroids die gleiche Grösse haben.

(c) (4 Punkte) Betrachten Sie einen parallelen Algorithmus mit Work $\Theta(n^{4/3})$ (Gesamtzahl der Operationen) und Span $\Theta(n^{1/3})$ (Länge eines längsten kritischen Pfades). Was ist die optimale asymptotische Laufzeit des Algorithmus, wenn er auf p Prozessoren ausgeführt wird. (Geben Sie die Lösung als Funktion von p und n an.)

(d) (6 Punkte) Wir betrachten das *Weighted Interval Scheduling* Problem aus der Vorlesung: Gegeben sei eine Menge mit Intervallen $[a_i, b_i]$ mit Gewichten w_i. Ziel ist es, eine Teilmenge von paarweise nicht-überlappenden Intervallen mit maximalem Gesamtgewicht zu finden.

Ein natürlicher Greedy-Algorithmus für dieses Problem ist folgender. Sortieren die Intervalle in absteigender Ordnung nach dem Gewicht pro Länge Verhältnis $w_i/(b_i - a_i)$ und versuche, die Intervalle in dieser Reihenfolge zur aktuellen Lösung hinzuzufügen.

Zeigen Sie, dass dieser Algorithmus beliebig schlecht sein kann. Das heisst, zeigen Sie, dass es für jede Konstante $C > 1$ eine Probleminstanz gibt, bei welcher das Gesamtgewicht einer optimalen Lösung mehr als C mal grösser ist, als das Gesamtgewicht der Lösung des beschriebenen Greedy-Algorithmus.

(e) (8 Punkte) Bestimmen Sie für die folgenden zwei Aussagen, ob sie wahr oder falsch sind. Falls eine Aussage wahr ist, erklären Sie warum, falls die Aussage falsch ist, geben Sie ein (kleines) Gegenbeispiel.

1) (4 Punkte) In einem Graph mit nur positiven Kantengewichten sei eine Teilmenge S der Kanten gegeben, welche alle Knoten verbindet und minimales Gesamtgewicht hat. Dann formen die Kanten von S einen Baum.

2) (4 Punkte) In einem Graph mit positiven und negativen Kantengewichten sei eine Teilmenge S der Kanten gegeben, welche alle Knoten verbindet und minimales Gesamtgewicht hat. Dann formen die Kanten von S einen Baum.

Problem 2: Amortization (14 points)

Suppose a sequence of n operations are performed on an (unknown) data structure in which the i-th operation costs i if i is an exact power of 2, and 1 otherwise.

Operation	1	2	3	4	5	6	7	8	9	...	15	16	17	...
Actual Cost	1	2	1	4	1	1	1	8	1	...	1	16	1	...

Table 1: Operations and their actual costs

Use the **potential function** method to show that each operation has constant amortized cost.

Hint: *The number of consecutive operations that are not an exact power of* 2 *and are performed immediately before operation* $(i+1)$ *is* $i - 2^{\ell(i)}$ *where* $\ell(i) := \lfloor \log_2 i \rfloor$.

Remark: If you did not find a potential function to show that each operation has constant amortized cost, you can use the **accounting** method and get up to 8 points instead.

Aufgabe 2: Amortisierung (14 Punkte)

Wir nehmen an, dass eine Folge von n Operationen auf einer (unbekannten) Datenstruktur ausgeführt werden, auf welcher die i-te Operation Kosten i hat, falls i eine Zweierpotenz ist und in allen anderen Fällen Kosten 1 hat.

Operation	1	2	3	4	5	6	7	8	9	...	15	16	17	...
Kosten	1	2	1	4	1	1	1	8	1	...	1	16	1	...

Table 2: Operations and ihre Kosten

Benutzen Sie die **Potentialfunktions**-Methode, um zu zeigen, dass alle Operationen konstante amortisierte Kosten haben.

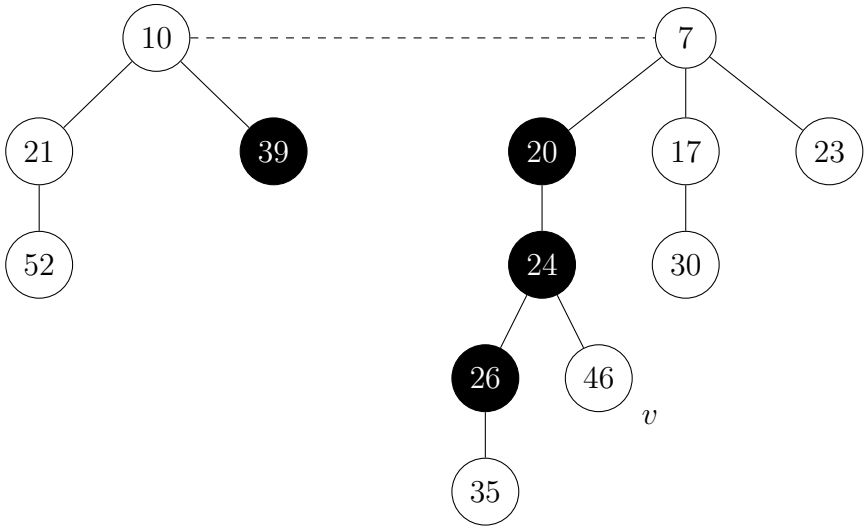
Hinweis: *Die Anzahl der Operationen, welche keine Zweierpotenz sind und welche direkt vor Operation* $(i+1)$ *ausgeführt werden, ist* $i - 2^{\ell(i)}$*, wobei* $\ell(i) := \lfloor \log_2 i \rfloor$.

Bemerkung: Falls Sie keine Potentialfunktion finden, um zu zeigen, dass alle Operatioinen konstante amortisierte Kosten haben, können Sie auch die "Accounting"-Methode verwenden, um dies zu zeigen und erhalten dafür maximal 8 Punkte.

Problem 3: Fibonacci Heaps (18 points)

(a) (4 points) Explain why the operations **insert** and **decrease-key** on a Fibonacci heap are called *lazy* operations.

(b) (8 points) Consider the following Fibonacci heap (black nodes are marked, white nodes are unmarked). How does the given Fibonacci heap look after a **decrease-key**$(v, 15)$ operation and how does it look like after a subsequent **delete-min** operation?



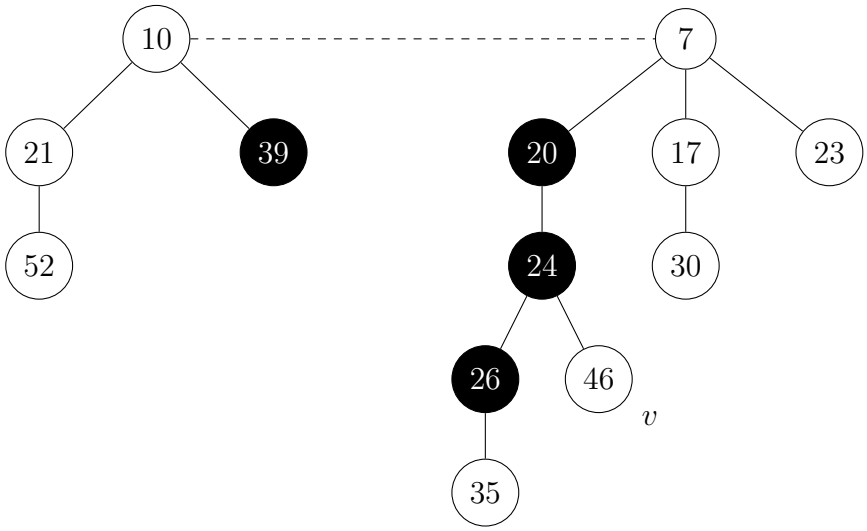
(c) (6 points) Fibonacci heaps are only efficient in an *amortized* sense. The time to execute a single, individual operation can be large. Show that in the worst case, the **delete-min** operation can require time $\Omega(n)$ (for any heap size n).

Hint: *For general n, describe an execution that starts with an empty heap and in which there is a **delete-min** operation that requires time linear in n.*

Aufgabe 3: Fibonacci Heaps (18 Punkte)

(a) (4 Punkte) Erklären Sie, wieso die Fibonacci Heap Operationen **insert** and **decrease-key** als *"lazy"* Operationen bezeichnet werden.

(b) (8 Punkte) Betrachten Sie den folgenden Fibonacci Heap (schwarze Knoten sind markiert, weisse Knoten sind nicht markiert). Wie sieht der Fibonacci Heap nach einer **decrease-key**$(v, 15)$-Operation aus und wie sieht er nach einer darauffolgenden **delete-min**-Operation aus?



(c) (6 Punkte) Fibonacci Heaps sind nur in einem *amortisierten* Sinn effizient. Die Laufzeit einer einzelnen Operation kann gross sein. Zeigen Sie, dass die **delete-min**-Operationim Worst Case $\Omega(n)$ Zeit benötigen kann (für jede Heap-Grösse n).

Hinweis: *Beschreiben Sie, für allgemeines n, eine Ausführung, welche mit einem leeren Heap beginnt und in welcher eine **delete-min**-Operation Zeit linear in n benötigt.*

Problem 4: Network Flow (14 points)

Professor Adam has two children who, unfortunately, dislike each other. The problem is so severe that not only do they refuse to walk to school together, but in fact each one refuses to walk on any street that the other child has used. The children have no problem with their paths crossing at street intersections. Fortunately both the professor's house and *the only school* in town are on intersections, but beyond that he is not sure if it is going to be possible to send both of his children to the only school in town. The professor has a map of his town given as a graph $G = (V, E)$, where E are the streets and V are the intersections.

(a) (4 points) Show how to formulate the problem of determining whether both his children can go to the school in town as a maximum flow problem.

(b) (4 points) What algorithm from the lecture would you use to solve the problem? Assuming that for a street network $G = (V, E)$, we have $|E| \leq 3|V|$, what is the asymptotic running time of your algorithm as a function of the number of nodes $n = |V|$?

(c) (6 points) Now, assume that the two children start disliking each other even more and they now only accept to go to the school if their walks also avoid crossing some of the intersections. Assume that we are given a subset $U \subseteq V$ of intersections and we now need to find two paths from the professor's home to the school such that every intersection in U and every street is part of at most one of the two paths. You can of course assume that the professor's home and the school are not in U.

How can you now reduce the problem to a maximum flow problem?

Aufgabe 4: Netzwerkfluss (14 Punkte)

Professor Adam hat zwei Kinder, welche sich leider nicht mögen. Das Problem ist so schlimm, dass sich die Zwei nicht nur weigern, gemeinsam den selben Weg zur Schule zu nehmen, sie weigern sich beide sogar, auf irgendeiner Straße zu gehen, welche das andere Kind schon benutzt hat. Die Kinder haben kein Problem damit, falls sich Ihre Wege auf Straßenkreuzungen überschneiden. Zum Glück sind das Haus des Professors und die einzige Schule der Stadt beide an Straßenkreuzungen. Darüber hinaus ist sich der Professor aber nicht sicher, ob es möglich ist, beide Kinder zur einzigen Schule in der Stadt zu schicken. Der Professor hat eine Karte seiner Stadt als Graph $G = (V, E)$ gegeben, auf welcher E die Straßen und V die Straßenkreuzungen sind.

(a) (4 Punkte) Zeigen Sie, wie das Problem herauszufinden ob beide Kinder zur Schule in der Stadt gehen können, als Maximales Flussproblem formuliert werden kann.

(b) (4 Punkte) Welchen Algorithmus aus der Vorlesung würden Sie verwenden, um das Problem zu lösen und was ist die Laufzeit des Algorithmus als Funktion der Anzahl Knoten $n = |V|$, falls Sie annehmen, dass bei einem Straßennetzwerk $G = (V, E)$ immer $|E| \leq 3|V|$ gilt?

(c) (6 Punkte) Nehmen Sie jetzt an, dass sich die Kinder irgendwann noch weniger mögen und dass Sie nun nur bereit sind, zur selben Schule zu gehen, falls ihre Wege auch vermeiden, sich an gewissen Kreuzungen zu überschneiden. Wir nehmen an, dass eine Teilmenge $U \subseteq V$ der Straßenkreuzungen gegeben ist und dass wir nun zwei Wege vom Haus des Professors zur Schule finden müssen, so dass jede Kreuzung in U und jede Straße in höchstens einem der beiden Wege vorkommt. Sie können natürlich davon ausgehen, dass das Haus des Professors und die Schule nicht in U sind.

   Wie kann das Problem jetzt als Maximals Flussproblem formuliert werden?

Problem 5: Randomized Independent Set Algorithm (16 points)

Let $G = (V, E)$ be a graph with n vertices and m edges. An independent set in a graph G is a subset $S \subseteq V$ of the nodes such that no two nodes in S are connected by an edge. We consider the following randomized algorithm to compute an independent set S. We let $d := \frac{1}{n}\sum_{v \in V} \deg(v) = \frac{2m}{n}$ be the average node degree.

(I) Start with an empty set S. Then independently add each node of V with probability $1/d$ to S (you can assume that $d \geq 1$).

(II) The subgraph induced by S might still contain some edges and we therefore need to remove at least one of the nodes of each of the remaining edges. For this, we use the following deterministic strategy. As long as S is not an independent set, pick an arbitrary node $u \in S$ such that u has a neighbor $v \in S$ and remove u from S.

It is clear that the above algorithm computes an independent set S of G and our goal will be to analyze the expected size of S as a function of n and d.

(a) (3 points) Let X be the number of nodes in S after Step (I) of the algorithm. What is $\mathbb{E}[X]$?

(b) (4 points) Let Y be the number of edges in the subgraph induced by S after Step (I) of the algorithm. What is $\mathbb{E}[Y]$?

(c) (4 points) Use the answers to (a) and (b) to find a (best possible) lower bound on the expected size of the independent set which the algorithm computes, i.e., lower bound the size of S at the end of the algorithm. Your lower bound should be expressed as a function of n and d. Recall that $d = \frac{2m}{n}$ and you can therefore express m as a function of n and d.

Remark: *If you could not solve a) and b), you can get some of the points by generically expressing the lower bound on the expected size of S as a function of X and Y.*

(d) (5 points) You can now assume that the above algorithm has running time $T(n)$ and that it computes an independent set of size $\frac{n}{5d}$ with probability at least $\frac{1}{2}$. Show how to compute an independent set of size at least $\frac{n}{5d}$ with probability $1 - \frac{1}{n}$. What is the running time of your algorithm?

Aufgabe 5: Randomisierter Independent Set Alg. (16 Punkte)

Gegeben sei ein Graph $G = (V, E)$ mit n Knoten und m Kanten. Ein Independent Set eines Graphen G ist eine Teilmenge $S \subseteq V$ der Knoten, so dass keine zwei Knoten in S durch eine Kante verbunden sind. Wir betrachen den folgenden randomisierten Algorithmus, um ein Independent Set S zu berechnen. Im Folgenden sei $d := \frac{1}{n}\sum_{v \in V} \deg(v) = \frac{2m}{n}$ der durchschnittliche Knotengrad.

(I) Beginne mit einer leeren Menge S. Dann wird jeder Knoten von V unabhängig mit Wahrscheinlichkeit $1/d$ zu S hinzugefügt (Sie können annehmen, dass $d \geq 1$).

(II) Der durch S induzierte Teilgraph kann jetzt immer noch Kanten enthalten und wir müssen daher mindestens einen der Knoten von jeder verbleibenden Kante entfernen. Dafür verwenden wir folgendes deterministisches Vorgehen. So lange S kein Independent Set ist, nehmen wir einen beliebigen Knoten $u \in S$, welcher einen Nachbqr $v \in S$ hat und wir entfernen u aus S.

Es ist klar, dass der beschriebene Algorithmus ein Independent Set S berechnet. Unser Ziel wird es sein, die erwartete Grösse von S als Funktion von n und d zu berechnen.

(a) (3 Punkte) Sei X die Anzahl der Knoten in S nach Schritt (I) des Algorithmus. Berechne $\mathbb{E}[X]$!

(b) (4 Punkte) Sei Y die Anzahl der Kanten im durch S induzierten Teilgraph nach Schritt (I) des Algorithmus. Berechne $\mathbb{E}[Y]$!

(c) (4 Punkte) Benutzen die Antworten in (a) und (b), um eine möglichst gute untere Schranke für das vom Algorithmus berechnete Independent Set herzuleiten, d.h. finden Sie eine möglichst gute untere Schranke für die erwartete Grösse von S am Ende des Algorithmus. Geben Sie Ihre untere Schranke als Funktion von n und d an. Betrachten Sie dabei, dass $d = \frac{2m}{n}$ und dass daher m als Funktion von n und d ausgedrückt werden kann.

Hinweis: *Falls Sie (a) und (b) nicht lösen konnten, können Sie auch einige der Punkte bekommen, indem Sie die untere Schranke für die erwartete Grösse von S als Funktion von X und Y ausdrücken.*

(d) (5 Punkte) Nehmen Sie nun an, dass der obige Algorithmus Laufzeit $T(n)$ hat und dass er mit Wahrscheinlichkeit $\frac{1}{2}$ ein Independent Set mit mindestens $\frac{n}{5d}$ Knoten berechnet.

Zeigen Sie wie man ein Independent Set mit mindestens $\frac{n}{5d}$ Knoten mit Wahrscheinlichkeit $1 - \frac{1}{n}$ berechnen kann. Was ist die Laufzeit Ihres Algorithmus'?