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

Final Exam Algorithm Theory

Thursday, February 27 2014, 9:00 – 10:30

Name: .....

Matriculation Nr.: .....

Signature.: .....

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

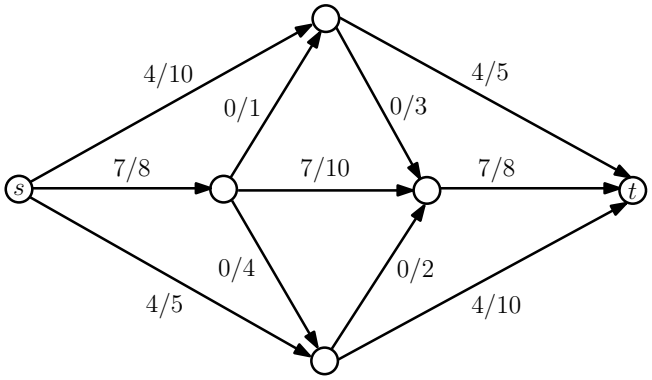
Instructions:

- 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 4 problems (with several questions per problem) and there is a total of 90 points.
- Use a separate sheet of paper for each of the 4 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.**

Question	Achieved Points	Max Points
1		29
2		19
3		21
4		21
Total		90

Problem 1: Short Questions (29 points)

(a) (9 points) Consider the following flow network, where for each edge, the capacity (second larger number) and a current flow value (first smaller number) are given. Draw the residual graph with all the residual capacities! In addition, give a best possible augmenting path (an augmenting path that improves the current flow by as much as possible).



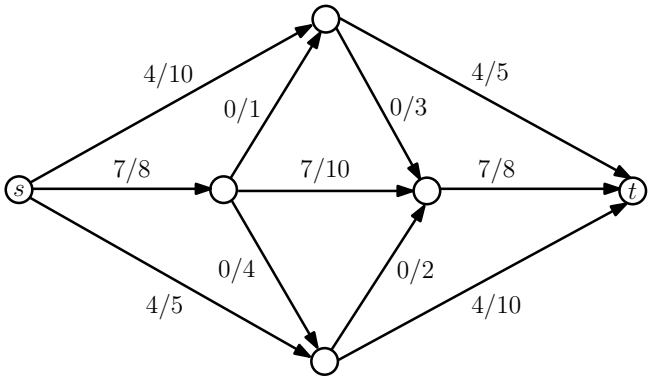
(b) (7 points) You are given a graph $G = (V, E)$ with n nodes and average degree $\Theta(\sqrt{\log n})$. Further, every edge of the graph has an integer weight between 1 and 10. Describe a minimum spanning tree algorithm that is as efficient as possible. What is the asymptotic running time of your algorithm?
(A short explanation for your stated running time can be helpful, but is not required.)

(c) (7 points) Consider a binomial heap data structure and assume that throughout a given execution, the number of elements in the binomial heap is never more than $n_{\max}$. Give a potential function which shows that the amortized time of *insert* is $O(\log n_{\max})$ and the amortized time of *delete-min* is $O(1)$.

(d) (6 points) Given a graph $G = (V, E)$, a 3-way cut is a partition of V into 3 sets V_1, V_2 and V_3. The size of a 3-way cut (V_1, V_2, V_3) is the number of edges between nodes in two different sets. In the lecture, we discussed the random contraction algorithm to compute a minimum cut. Describe how you would adapt the basic contraction algorithm such that it returns a three-way cut. What has to happen so that your algorithm returns a *specific* minimum three-way cut (V_1, V_2, V_3)?

Frage 1: Kurzfragen (29 Punkte)

(a) (9 Punkte) Wir betrachten folgendes Flussnetzwerk. Für jede Kante ist die Kapazität (zweite, grössere Zahl) und ein aktueller Fluss (erste, kleinere Zahl) angegeben. Zeichnen Sie den Residualgraphen, inklusive aller Residualkapazitäten. Geben Sie zudem einen bestmöglichen augmentierenden Pfad an (einen augmentierenden Pfad, welcher den aktuellen Fluss so gut wie möglich verbessert).



(b) (7 Punkte) Gegeben ist ein Graph $G = (V, E)$ mit n Knoten und Durchschnitts-Knotengrad $\Theta(\sqrt{\log n})$. Jede Kante des Graphen hat ein ganzzahliges Gewicht zwischen 1 und 10. Beschreiben Sie einen Algorithmus, der einen minimalen Spannbaum möglichst effizient berechnet. Was ist die asymptotische Laufzeit Ihres Algorithmus?
(Eine kurze Begründung fuer die von Ihnen angegebene Laufzeit kann hilfreich sein, ist aber nicht erforderlich.)

(c) (7 Punkte) Im Folgenden betrachten wir die "Binomial Heap" Datenstruktur und nehmen an, dass im Verlauf einer gegebenen Ausführung die Anzahl Elemente in der Datenstruktur $n_{\max}$ nicht übersteigt. Geben Sie eine Potentialfunktion an, welche für die *insert*-Operation eine amortisierte Zeit von $O(\log n_{\max})$ zeigt, sowie für die *delete-min*-Operation eine amortisierte Zeit von $O(1)$.

(d) (6 Punkte) Ein "3-Way"-Schnitt eines Graphen $G = (V, E)$ ist eine Partition von V in drei Mengen V_1, V_2 und V_3. Die Grösse eines "3-Way"-Schnitts (V_1, V_2, V_3) ist die Anzahl Kanten zwischen Knoten in verschiedenen Mengen. In der Vorlesung haben wir den "Random Contraction" Algorithmus gesehen, um einen minimalen Schnitt zu berechnen. Adaptieren Sie den Basisalgorithmus, so dass er einen "3-Way"-Schnitt zurückgibt. Beschreiben Sie zudem, was passieren muss, damit der Algorithmus einen *bestimmten* minimalen "3-Way"-Schnitt (V_1, V_2, V_3) zurückgibt.

Problem 2: Filling Two Knapsacks (19 points)

In this problem, we consider a variation of the knapsack problem, where we have two instead of only one knapsack. Formally, we have items $1, \ldots, n$ and each item i has a positive integer *weight* $w_i \in \mathbb{N}$ and a positive *value* $v_i > 0$. Further, we have two knapsacks of capacities W_1 and W_2. We need to pack the items into the knapsacks such that

- Each item is in at most one of the knapsacks
- For $j \in \{1, 2\}$, the *total weight* of the items in knapsack j is at most W_j.
- The *total value* of the items that are packed in either knapsack is maximized.

(a) (5 points) When first looking at the problem, one could think that it is equivalent to the standard knapsack problem with one knapsack of capacity $W' := W_1 + W_2$. Prove that this is not true by showing that in some cases, the total value that can be packed into one knapsack of capacity $W' = W_1 + W_2$ can be *arbitrarily* larger than the total value that can be packed into two knapsacks of capacities W_1 and W_2.

(b) (6 points) Assume that $W_1 \geq W_2$. A simple strategy would be to first compute an *optimal* solution for a knapsack of capacity W_1. Afterwards, with the remaining elements, an *optimal* solution for the knapsack of capacity W_2 is computed. Show that this algorithm always computes at least a 2-approximation for the problem.

(c) (8 points) Give a dynamic programming algorithm that *optimally* solves the problem (for integer weights). What is the running time of your algorithm?
(A short explanation for your stated running time can be helpful, but is not required.)

Frage 2: Zwei Rucksäcke packen (19 Punkte)

Wir betrachten nun eine Variante des Rucksack (Knapsack)-Problems, bei welchem wir zwei, statt nur einen Rucksack haben. Formal betrachten wir folgendes Problem. Die Eingabe besteht aus den Objekten $1, \ldots, n$, wobei jedes Objekt ein positives ganzzahliges *Gewicht* w_i, sowie einen positiven *Wert* $v_i > 0$ hat. Zudem haben wir zwei Rucksäcke mit Kapazitäten W_1 und W_2. Wir müssen die Objekte so in die zwei Rucksäcke packen, dass

- jedes Object in höchstens einen Rucksack kommt,
- für $j \in \{1, 2\}$, das *Gesamtgewicht* der Objekte in Rucksack j, W_j nicht übersteigt,
- der *Gesamtwert* der Objekte, welche in einen der beiden Rucksäcke gepackt werden, maximiert wird.

(a) (5 Punkte) Bei einem ersten Blick auf das Problem könnte man denken, dass es äquivalent zum Standard-Rucksackproblem mit einem Rucksack der Grösse $W' := W_1 + W_2$ ist. Beweisen Sie, dass dies nicht der Fall ist, indem Sie zeigen, dass in gewissen Fällen der Gesamtwert der Objekte, welche in einen grossen Rucksack der Grösse $W' = W_1 + W_2$ gepackt werden können, *beliebig* viel grösser ist, als der Gesamtwert der Objekte, welche in zwei Rucksäcke der Grössen W_1 und W_2 gepackt werden können.

(b) (6 Punkte) Nehmen wir nun an, dass $W_1 \geq W_2$ ist. Eine einfache Strategie wäre es, zuerst eine *optimale* Lösung für das Problem mit einem Rucksack der Grösse W_1 zu berechnen. Danach wird mit den restlichen Objekten eine *optimale* Lösung für einen Rucksack der Grösse W_2 berechnet. Zeigen Sie, dass dieser Algorithmus immer mindestens eine 2-Approximation berechnet.

(c) (8 Punkte) Geben Sie einen Algorithmus an, welcher dynamische Programmierung nutzt, um das Problem (für ganzzahlige Gewichte) *optimal* zu lösen. Was ist die Laufzeit des Algorithmus?
(Eine kurze Begründung fuer die von Ihnen angegebene Laufzeit kann hilfreich sein, ist aber nicht erforderlich.)

Problem 3: Discrete Fourier Transform (21 points)

a) (3 points) Let $N \geq 1$ be a positive integer, which is a power of 2. Given a real-valued vector $\boldsymbol{a} = (a_0, \ldots, a_{N-1})$ of length N, describe what the discrete Fourier transform $\mathrm{DFT}_N(\boldsymbol{a})$ of $\boldsymbol{a}$ is.
Hint: *You only need to specify what the discrete Fourier transform computes, not how to compute it efficiently.*

b) (6 points) The discrete Fourier transform (DFT) algorithm is based on the divide-and-conquer paradigm. Describe how a problem instance of size N is divided into smaller problem instances and how, after recursively computing solutions for the smaller inputs, the solution for the input of size N is computed. You can again assume that N is a power of 2.

c) (7 points) The goal now is to parallelize the DFT algorithm. First, let us consider the "divide" and the "combine" steps as discussed in (b). Give a *high-level* description of how to parallelize the two parts. You can assume that you have N processors available. What are the running times of your parallel "divide" and "combine" steps?

d) (5 points) Describe how to parallelize the whole DFT algorithm based on parallel versions of the "divide" and "combine" steps. What is the total work (T_1) and what is the span (T_∞) of the resulting parallel DFT algorithm (both asymptotically)? What is the asymptotic running time T_p when carrying out the algorithm on p processors?

Frage 3: Diskrete Fourier-Transformation (21 Punkte)

a) (3 Punkte) Sei N eine positive ganzzahlige Zahl und eine Zweierpotenz. Beschreiben Sie was die diskrete Fourier-Transformation $\mathrm{DFT}_N(\boldsymbol{a})$ für einen reellwertigen Vektor $\boldsymbol{a} = (a_0, \ldots, a_{N-1})$ der Länge N berechnet.
Tipp: *Sie müssen nur angeben, was die diskrete Fourier-Transformation berechnet, nicht wie man es effizient berechnet.*

b) (6 Punkte) Der diskrete Fourier-Transformation (DFT) Algorithmus basiert auf dem "Divide-and-Conquer" Paradigma. Beschreiben Sie, wie eine Probleminstanz der Grösse N in kleinere Teilprobleme unterteilt wird und wie, nachdem die Teilprobleme rekursiv gelöst wurden, die Lösung der Instanz der Grösse N berechnet wird. Sie können weiterhin annehmen, dass N eine Zweierpotenz ist.

c) (7 Punkte) Das Ziel ist es nun, den DFT Algorithmus zu parallelisieren. Betrachen wir zuerst den "Divide"- und den "Combine"-Schritt (die Schritte, welche Sie in Frage (b) beschrieben haben). Beschreiben Sie (ohne zu sehr ins Detail zu gehen), wie die zwei Schritte parallelisiert werden können. Sie können annehmen, dass N Prozessoren zur Verfügung stehen. Was sind die Laufzeiten Ihrer parallelen "Divide"- und "Combine"-Schritte?

d) (5 Punkte) Beschreiben Sie nun, wie man, basierend auf den parallelen "Divide"- und "Combine"-Schritten, den ganzen DFT-Algorithmus parallelisieren kann. Was sind T_1 (work) und T_∞ (span) des parallelen DFT-Algorithmus (beides asymptotisch). Was ist die asymptotische Laufzeit T_p, falls p Prozessoren zur Verfügung stehen?

Problem 4: Computing Large Sparse Subgraphs (21 points)

Given a graph $G = (V, E)$ and an integer $k \geq 1$, we define a k-sparse edge set to be a subset $F \subseteq E$ of the edges such that the subgraph $H = (V, F)$ induced by the edges in F has maximum degree at most k. In the figure below, the thick edges form a 3-sparse edge set of the depicted graph. The size of a k-sparse edge set is the number of its edges. The maximum k-sparse subgraph problem (max-k-SSP) asks for a k-sparse edge set of largest possible size.

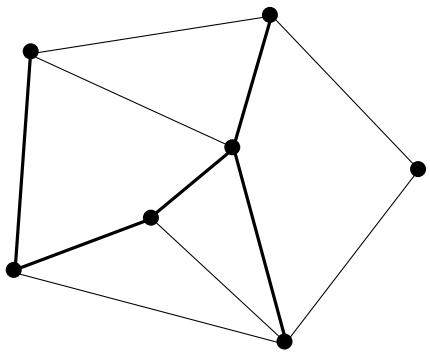
(a) (8 points) Assume that G is a bipartite graph. Show how to optimally solve the max-k-SSP by reducing it to a maximum flow problem. Also argue why the given maximum flow problem is equivalent to the max-k-SSP!

(b) (3 points) Assume that we compute a solution to the maximum flow problem of (a) by using the Ford Folkerson algorithm. Give the asymptotic running time as a function $n = |V|$, $m = |E|$, and k.

For $k = 1$, the max-k-SSP is equivalent to the maximum matching problem discussed in class. In the following, we consider an online variant of the maximum matching problem. We assume that the edges of G appear one at a time. For each edge, an online algorithm has to immediately decide whether to add it to the matching or not (without knowing the remaining edges).

(c) (5 points) Give a deterministic online algorithm with competitive ratio 2. Argue why your algorithm has competitive ratio 2.
Hint: *You do not need to reprove statements we already proved in the lecture.*

(d) (5 points) Show that no online algorithm can be strictly c-competitive for a constant $c < 2$.



Frage 4: Berechnen grosser "Sparse Subgraphs" (21 Punkte)

Gegeben ist ein Graph $G = (V, E)$ und eine natürliche Zahl $k \geq 1$. Wir definieren eine k-Sparse Kantenmenge als eine Teilmenge $F \subseteq E$ der Kanten, so dass der Teilgraph $H = (V, F)$, welcher durch die Kanten in F induziert wird, maximalen Knotengrad höchstens k hat. In der untenstehenden Figur beschreiben die fetten Kanten eine 3-Sparse Kantenmenge. Die Grösse einer k-Sparse Kantenmenge ist die Anzahl ihrer Kanten. Wir wollen das Maximum k-Sparse Subgraph Problem (Max-k-SSP) lösen, welches nach einer k-Sparse Teilmenge maximaler Grösse verlangt.

(a) (8 Punkte) Im Folgenden nehmen wir an, dass G ein bipartiter Graph ist. Zeigen Sie, wie das Max-k-SSP dann optimal gelöst werden kann, indem es auf ein maximales Flussproblem reduziert wird. Begründen Sie auch, wieso das maximale Flussproblem genau das gegebene bipartite Max-k-SSP löst!

(b) (3 Punkte) Angenommen, wir lösen das maximale Flussproblem aus (a) mit dem Ford Fulkerson Algorithmus. Berechnen Sie die asymptotische Laufzeit, in Abhängigkeit von $n = |V|$, $m = |E|$ und k.

Für den Fall $k = 1$ ist das Max-k-SSP äquivalent zum Maximum Matching Problem, welches wir in der Vorlesung angeschaut haben. Im Folgenden betrachten wir eine "Online"-Variante des Maximum Matching Problems. Wir nehmen an, dass die Kanten von G eine nach der anderen erscheinen. Ein Online-Algorithmus muss für jede Kante direkt entscheiden, ob er Sie zum Matching hinzufügt (ohne die restlichen Kanten zu kennen).

(c) (5 Punkte) Geben Sie einen deterministischen Online-Algorithmus mit "Competitive Ratio" 2. Zeigen Sie auch, wieso Ihr Algorithmus "Competitive Ratio" 2 hat.
Tipp: *Sie müssen Dinge, welche wir in der Vorlesung bewiesen haben, nicht neu beweisen.*

(d) (5 Punkte) Zeigen Sie, dass es keinen Online-Algorithmus geben kann, welcher "strictly c-competitive" für eine Konstante $c < 2$ ist.