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

# Klausur Algorithmen und Datenstrukturen

8. August 2024, 14:00 -17:00

Name: .....

Matrikel Nr.: .....

Unterschrift: .....

## Erst öffnen wenn die Klausuraufsicht die Erlaubnis erteilt!

- Legen Sie Ihren **Studentenausweis** sichtbar vor sich.
- Schreiben Sie Ihren **Namen** und Ihre **Matrikelnummer** an die vorgesehenen Stellen.
- **Unterschreiben Sie** diese Seite um zu bestätigen, dass Sie die Fragen ohne unerlaubte Hilfsmittel beantwortet haben und die Klausuraufsicht über (gesundheitliche) Probleme informiert haben.
- Dies ist eine **Open Book** Klausur weshalb alle gedruckten und handgeschriebenen Materialien erlaubt sind. **Elektronische Hilfsmittel** sind **nicht** erlaubt.
- Schreiben Sie **lesbar** und nur mit **dokumentenechten** Stiften. Benutzen Sie **keine rote** Farbe und **keinen Bleistift**!
- Es wird **nur eine Lösung pro Aufgabe** gewertet. Vergewissern Sie sich, dass Sie zusätzliche Lösungen durchstreichen, andernfalls wird die schlechteste Lösung gewertet.
- Detaillierte Schritte können Ihnen zu **Teilpunkten** verhelfen falls das Endergebnis falsch ist.
- Die Schlüsselwörter **Zeigen Sie...**, **Beweisen Sie...**, **Begründen Sie...** oder **Leiten Sie ... her** zeigen an, dass Sie Ihre Antwort sorgfältig und gegebenenfalls formal begründen müssen.
- Die Schlüsselwörter **Geben Sie ... an** zeigen an, dass Sie lediglich die geforderte Antwort und keine Begründung liefern müssen.
- Die folgenden Regeln gelten **überall**, außer sie werden explizit außer Kraft gesetzt.
- Bei Laufzeitfragen ist nur die **asymptotische Laufzeit** notwendig.
- Wenn Sie einen Algorithmus angeben sollen, so können Sie Pseudocode angeben. Eine **ausreichend detaillierte** Beschreibung der Funktionsweise Ihres Algorithmus genügt jedoch.
- Algorithmen aus der Vorlesung **können als Blackbox** verwendet werden.
- Wenn es in der Aufgabenstellung nicht explizit anders steht, dürfen Sie annehmen das alle Operationen auf **Hashtabellen** konstante Laufzeit haben.
- **Lesen Sie jede Aufgabe sorgfältig** durch und stellen Sie sicher dass Sie diese verstanden haben!
- Falls Sie eine Frage zur Aufgabenstellung haben, geben Sie der Klausuraufsicht ein **Handzeichen**.
- Schreiben Sie Ihren Namen auf **alle Blätter**!
- Die beste Note erhalten Sie beim erreichen von 100 Punkten. Um die Klausur zu **bestehen** reichen 50 Punkte.

| Aufgabe | 1 | 2 | 3 | 4 | 5 | 6 | 7 | **Total** |
|---|---|---|---|---|---|---|---|---|
| Maximum | 28 | 15 | 12 | 15 | 20 | 15 | 15 | 120 |
| Punkte | | | | | | | | |

## Aufgabe 1: Kurze Fragen *(28 Punkte)*

(a) Sei $A = [n, 1, 2, \ldots, n-1]$ ein Array aus $n$ Elementen welcher bis auf die erste Stelle sortiert ist. Was ist die asymptotische Laufzeit die *Insertion Sort* benötigt um $A$ zu sortieren? Welche Zeit benötigt *Merge Sort* um $A$ zu sortieren? Begründen Sie Ihre Antworten. *(3 Punkte)*

(b) Fügen Sie folgende Sequenz von Schlüsseln in gegebener Reihenfolge in einen inital leeren binären Suchbaum ein: $13, 5, 2, 18, 10, 11, 3, 7, 6, 4$. Zeichnen Sie den resultierenden Suchbaum. Wie sieht der Suchbaum nach einem anschließenden *delete(5)* aus? Zeichnen Sie auch diesen Suchbaum. *(5 Punkte)*

(c) Für einen zusammenhängenden gewichteten Graph $G = (V, E, w)$ definieren wir einen Maximum Spanning Tree als ein Spannbaum $T$, so dass sein Gewicht $w(T) = \sum_{e \in T} w(e)$ maximal ist. Das heißt, für jeden anderen Spannbaum $T'$ gilt $w(T) \geq w(T')$.
Geben Sie einen effizienten Algorithmus an, der einen maximalen Spannbaum berechnet. Beweisen Sie, dass der Algorithmus korrekt ist und begründen Sie die Laufzeit. *(5 Punkte)*

(d) Für einen gewichteten Graphen $G = (V, E, w)$ mit ganzzahligen, positiven Kantengewichten ($w : E \to \mathbb{N}$) definieren wir zwei neue Gewichtsfunktionen.

$$w_1(e) := w(e) + 1$$
$$w_2(e) := w(e) + \frac{1}{|V|}$$

Beweisen oder widerlegen sie folgende Aussagen:

(i) Jeder kürzeste Pfad in $G$ ist auch ein kürzester Pfad in $G' = (V, E, w_1)$. *(2.5 Punkte)*

(ii) Jeder kürzeste Pfad in $G' = (V, E, w_1)$ ist auch ein kürzester Pfad in $G$. *(2.5 Punkte)*

(iii) Jeder kürzeste Pfad in $G$ ist auch ein kürzester Pfad in $G'' = (V, E, w_2)$. *(2.5 Punkte)*

(iv) Jeder kürzeste Pfad in $G'' = (V, E, w_2)$ ist auch ein kürzester Pfad in $G$. *(2.5 Punkte)*

(e) Um den Knuth-Morris-Pratt Algorithmus aus der Vorlesung auszuführen muss ein Array S vorberechnet werden. Geben Sie dieses Array S bezüglich dem Pattern P = "CCDCCCD" *(5 Punkte)* an.

## Lösungsblatt Aufgabe 1

## Aufgabe 2: Landau-Notation *(15 Punkte)*

(a) Geben Sie eine Sortierung der folgenden Funktionen bezüglich der Landau Notation an. Das heißt, für zwei aufeinanderfolgende Funktionen $f, g$ in dieser Sortierung gilt jeweils $f(n) \in O\big(g(n)\big)$. Die angegebenen Beziehungen müssen weder bewiesen noch begründet werden. *(5 Punkte)*

- $a(n) = 3^{(n^2)} - 3n^{13}$
- $b(n) = 5^{\log_5(\log_5(n^5))}$
- $c(n) = n + |\log_5(5^{-n})|$
- $d(n) = \sqrt[800]{n^4}$
- $e(n) = (\log_2(n))^{2000}$

(b) Beweisen oder widerlegen Sie anhand der Definition der Landau-Notation:

$$2^{\sqrt{\log_2 n}} \notin \Omega(\sqrt[100]{n})$$

*(5 Punkte)*

(c) Beweisen oder widerlegen Sie anhand der Definition der Landau-Notation:

$$\log_2(n!) \in \Theta(n \log_2 n)$$

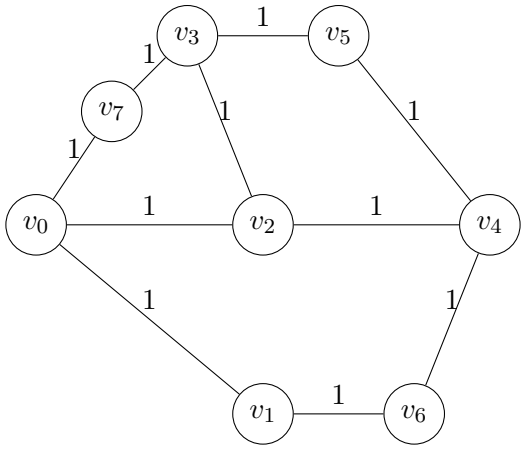
*(5 Punkte)*

## Lösungsblatt Aufgabe 2

## Aufgabe 3: Finde nahe Knoten *(12 Punkte)*

Der ländliche Raum soll besser vernetzt werden, dafür wird ein neues Verkehrsticket eingeführt. Mit diesem neuen Ticket können alle öffentlichen Verkehrsmittel im eigenen Dorf und in allen umliegenden Dörfern verwendet werden. Die Ticketinhaber hätten gerne ein Tool mit dem Sie herausfinden können, in welchen Dörfern ihr Ticket gültig ist. Entscheidend ist dafür das Heimatdorf $v$ und die Preisklasse $d$. Bei höheren Preisstufen gilt das Ticket auch noch in weiter entfernten Dörfern.
Wir modellieren das Problem als Graphen, ein Knoten pro Dorf und eine Kante zwischen zwei Dörfern wenn es eine direkte Straße gibt.
Ihnen fällt auf, dass keines der Dörfer eine direkte Straße zu mehr als 10 anderen Dörfern hat. Sie wollen diesen Fakt ausnutzen, um den Algorithmus, für kleine $d$, schneller zu machen.

Formal: Wir haben einen gewichteten Graphen $G = (V, E, w)$ in dem alle Kanten Gewicht 1 haben ($\forall e \in E : w(e) = 1$) und in dem jeder Knoten maximal 10 Nachbarn hat. Geben Sie einen effizienten Algorithmus, der, für Eingabeparameter $v \in V$ und eine Distanz $d \in \mathbb{N}$, alle Knoten findet deren Distanz zu $v$ maximal $d$ ist.




Zum Beispiel: Für den Startknoten $v_5$ und die Distanz $d = 2$ sollten die Knoten $\{v_7, v_3, v_2, v_4, v_6\}$ ausgegeben werden.
Begründen Sie warum der Algorithmus korrekt ist und analysieren Sie die Laufzeit in Abhängigkeit der beiden Parameter $n := |V|$ und $d$.
Denken Sie über folgende Fragen nach und geben Sie explizite Antworten:

- In welcher Form muss der Graph gegeben sein damit Ihr Algorithmus die behauptete Laufzeit erreicht?
- Wo kommt in Ihrer Analyse vor, dass jeder Knoten nur 10 Nachbarn hat?
- Für welche Werte von $d$ läuft Ihr Algorithmus in Zeit $o(n)$?
  *Hinweis: Daraus können Sie bereits schließen, dass Ihr Algorithmus nicht einmal ein Array der Größe $n$ initialisieren darf.*

## Lösungsblatt Aufgabe 3

## Aufgabe 4: Heaps mit kleinen Werten *(15 Punkte)*

Ihr neues Startup will nur noch "einfache Prozesse" und hat deswegen entschieden, dass jeder Aufgabe nur noch eine von $C$ verschiedenen Prioritäten zugewiesen werden darf.
Ihre Aufgabe ist es, eine speziell angepasste Priority Queue zu implementieren. Dabei sollen Sie die zusätzlichen Eigenschaft ausnutzen, dass alle Schlüssel zwischen 1 und $C$ liegen.

(a) Geben Sie eine Datenstruktur an, in der die Operationen *insert* und *delMin* (bzw. *getMin*) unterstützt werden (*decreaseKey* muss nicht unterstützt werden). *insert* muss Laufzeit $O(1)$ haben, *delMin* muss Laufzeit $O(C)$ haben und die Datenstruktur darf nicht mehr als $O(n + C)$ Speicher brauchen, wobei $n$ die Anzahl der Elemente in der Datenstruktur ist. Beweisen Sie, dass Ihre Datenstruktur diese Vorraussetzungen erfüllt. *(10 Punkte)*

(b) Geben Sie eine Datenstruktur an, in der die obigen Vorraussetzungen erfüllt werden, aber in der *delMin* eine Laufzeit $O(\sqrt{C})$ hat. Beweisen Sie, dass Ihre Datenstruktur diese Vorraussetzungen erfüllt. *(5 Punkte)*

*Hinweis: Wenn Sie Aufgabe* $(b)$ *korrekt beantworten, erhalten Sie automatisch auch alle Pukte von Aufgabe* $(a)$*.*

## Lösungsblatt Aufgabe 4

## Aufgabe 5: Schimpfwort Filter *(20 Punkte)*

Sie arbeiten an einer Chat Plattform für Schulen. Es gab wiederholt Beschwerden, weil die Schüler*innen Beleidigungen verwenden. Deshalb wurde ein Schimpfwort Filter eingebaut, der alle Wörter aus einer Schmipfwörterliste $S$ zensiert. Es hat natürlich nicht lange gedauert, bis die Schüler*innen eine Lösung gefunden haben. Sie haben angefangen einzelne Buchstaben durch Zahlen zu ersetzen, oder z.B. den Buchstabe 'd' mit 't' zu vertauschen. So bleiben die Wörter noch erkennbar, aber der Filter findet sie nicht mehr.
Wir versuchen also für ein gegebenes Wort $w$ zu entscheiden, ob es ein verstecktes Schimpfwort ist, oder nicht.

Formal: Für eine gegebene Menge von Strings $S$ und ein *zu prüfendes* Wort $w$ soll entschieden werden, ob $w$ ein abgeändertes Schimpfwort ist. Es soll also überprüft werden, ob durch ersetzen von einem (oder keinem) Buchstaben ein Wort aus der Menge $S$ entsteht.

Beispiel: Sei $S = \{$Butterbirne, Ekelpaket, Gewitterziege, Halunke, Hanswurst, Hohlkopf, Schlawiner, Schuft$\}$ .
Hier ist "*Naseweis*" kein Schimpfwort, während "*Egelpaket*" und "*Butterbyrne*" als Schimpfwort eingestuft werden.

Für unsere Analyse definieren wir folgende Werte:

- $|S|$ die Anzahl an Wörtern in $S$.
- $W$ die Länge des längsten Wortes ($\text{len}(w) \leq W$ und $\forall s \in S : len(s) \leq W$).
- $|\Sigma|$, wobei das Alphabet $\Sigma$ die Menge aller möglicher Textzeichen ist (z.B. a, ..., z, A, ..., Z, 0, ..., 9, !, ?, ¿, &, ...).

Wir wollen für viele verschiedene Wörter $w$ entscheiden, ob diese zensiert werden müssen oder nicht. Deswegen wollen wir zuerst eine Datenstruktur anlegen und diese anschließend nutzen um für ein $w$ schnell die Antwort zu berechnen.

a) Geben Sie eine Datenstruktur an, die ihnen dabei hilft ein Wort $w$ schnell zu verarbeiten. Es muss möglich sein die Datenstruktur in Zeit $O(W \cdot |S|)$ zu initialisieren. Zeigen Sie wie man die Datenstruktur nutzen kann, um für ein Wort $w$ in $O(W^2 \cdot |\Sigma|)$ Zeit zu entscheiden, ob $w$ zensiert werden soll. *(7 Punkte)*

*Hinweis: Beachten Sie, dass viele Operationen auf $W$ langen strings $O(W)$ Zeit brauchen. Insbesondere benötigt das Hashen eines $W$ langen Worts $O(W)$ Zeit.*

Wir nehmen jetzt an, dass kein Wort länger als 20 Zeichen ist, also $W \leq 20$ und die Chat Nachrichten unicode Zeichen verwenden ($|\Sigma|$ ist ungefähr 149,813). Deshalb wollen wir ein Verfahren das nicht von der Größe des Alphabets $\Sigma$ abhängt.

b) Geben Sie eine Datenstruktur an die ihnen dabei hilft ein Wort $w$ noch schneller zu verarbeiten. Es muss möglich sein die Datenstruktur in Zeit $O(W^2 \cdot |S|)$ zu initialisieren. Zeigen Sie wie man die Datenstruktur nutzen kann, um für ein Wort $w$ in $O(W^2)$ Zeit zu entscheiden, ob $w$ zensiert werden soll. *(13 Punkte)*

*Hinweis: Versuchen Sie die Anzahl der verschiedenen Wörter, nach denen Sie in Ihrer Datenstruktur suchen, geschickt zu reduzieren.*

## Lösungsblatt Aufgabe 5

## Aufgabe 6: MST in Partitionierten Graphen *(15 Punkte)*

In der Vorlesung haben wir gesehen, dass wir einen Minimum Spanning Tree (MST) in Zeit $O(m \log n)$ berechnen können. Wir wollen für eine bestimmte Klasse von Graphen einen effizienteren Algorithmus entwickeln.
Unsere Graphen haben die folgende Form: Der Graph $G(V, E, w)$ ist partitioniert in $k$ Cluster $P_1, P_2, \ldots, P_k \subseteq V$, so dass $\bigcup_{i=1}^{k} P_i = V$ und $P_i \cap P_j = \emptyset$ für $i \neq j$. Jedes $P_i$ ist eine Clique (es gibt also von jedem Knoten in $P_i$ zu jedem anderen Knoten in $P_i$ eine Kante) und Kanten zwischen den $P_i$ existieren immer nur zwischen $P_i$ und $P_{i+1}$ (also immer nur zwischen aufeinanderfolgenden Clustern). Außerdem wissen wir, dass innerhalb eines Clusters alle Kantengewichte exakt 1 sind $(u, v \in P_i \rightarrow w(\{u, v\}) = 1)$ und Kantengewichte zwischen Clustern immer echt größer als 1 sind $(u \in P_i, v \in P_{i+1} \rightarrow w(\{u, v\}) > 1)$.

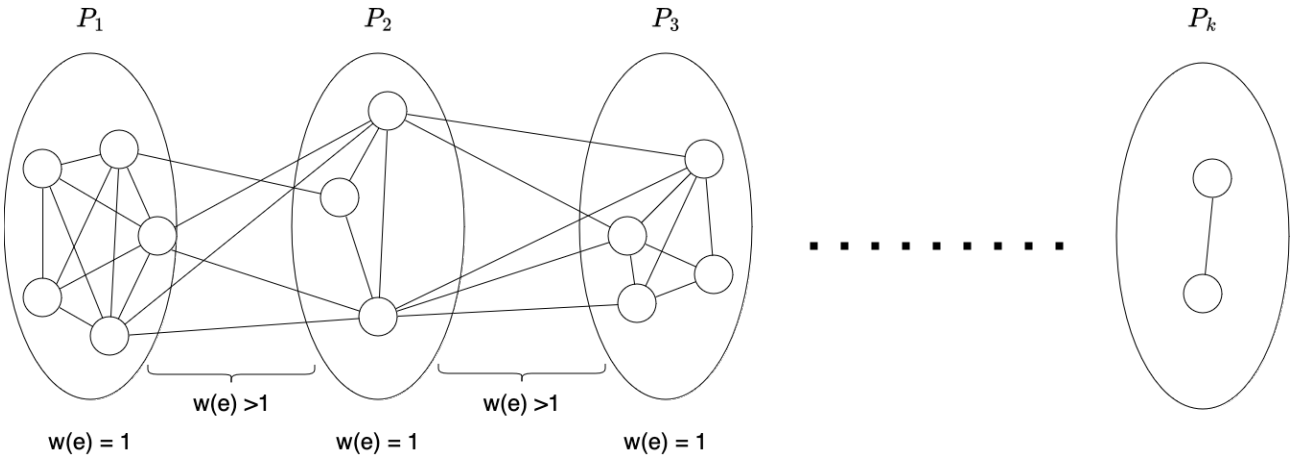



Figure 1: Ein Beispiel für einen partitionierten Graphen. Die Knoten in jedem $P_i$ formen eine Clique und alle Kantengewichte in der Clique sind genau 1. Die Kanten zwischen den $P_i$ sind immer zwischen $P_i$ und $P_{i+1}$ und alle Kantengewichte zwischen den $P_i$ sind echt größer 1.

Geben Sie einen Algorithmus der in Zeit $O(m)$ einen MST von einem solchen Graphen berechnet. Ihrem Algorithmus wird der Graph als Adjazenzliste gegeben, sowie die Partitionierung $P_1, \ldots, P_k$ (in korrekter Reihenfolge). Begründen Sie warum der Algorithmus korrekt ist und zeigen Sie außerdem, dass Ihr Algorithmus innerhalb der geforderten Laufzeit terminiert.

## Lösungsblatt Aufgabe 6

## Aufgabe 7: Dynamische Programmierung *(15 Punkte)*

Beim Speichern von deinem Textdokument ist ein Fehler aufgetreten! Alle Satzzeichen wie Punkt, Komma oder Leerzeichen sind verschwunden. Ein Satz aus dem Dokument würde also beipielsweise so aussehen:
"IchliebeesKlausurenzuschreiben"
Wir haben den Duden als Hashtabelle *dict* abgespeichert, in dieser können wir für jeden String $w$ checken ob $w$ ein gültiges Wort ist. Wenn $w$ im Duden ist, dann gibt $dict.contains(w)$ in konstanter Zeit *True* zurück.
Beschreiben Sie einen effizienten Algorithmus der entscheidet ob ein gegebener $n$ Buchstaben langer String $s[1, \ldots, n]$, wieder zu einem sinnvollen Text zusammengesetzt werden kann (also ob es eine Aneinanderreihung von Wörtern ist). Argumentieren Sie warum Ihr Algorithmus korrekt ist und analysieren Sie die Laufzeit.

## Lösungsblatt Aufgabe 7