Name:	Matrikelnummer:

Prof. für Algorithmen
und Datenstrukturen
Prof. Dr. Hannah Bast
Niklas Schnelle

Algorithmen und Datenstrukturen
SS 2019

http://ad-wiki.informatik.uni-freiburg.de/teaching

UNI FREIBURG

Klausur mit Lösungsskizzen

Dienstag 27. August 2019, 13:00 - 15:00 Uhr, Audimax, KG II

Es gibt *fünf* Aufgaben. Für jede davon gibt es maximal 25 Punkte. Wir zählen nur die *vier besten* Aufgaben. Sie können also maximal 100 Punkte erreichen. Zum Bestehen reichen 50 Punkte.

Sie haben insgesamt 2 Stunden Zeit. Wenn Sie vier Aufgaben bearbeiten, haben Sie im Durchschnitt 30 Minuten pro Aufgabe Zeit.

Sie dürfen eine beliebige Menge an Papier, Büchern, etc. verwenden. Sie dürfen keinerlei elektronische Geräte wie Notebook, Mobiltelefon, etc. verwenden, insbesondere keine Geräte, mit denen Sie mit Dritten kommunizieren oder sich mit dem Internet verbinden können.

Wir werden die Klausur am selben und am nächsten Tag korrigieren. Die Klausureinsicht findet am Dienstag, den 3. September 2019 von 14:00 - 15:00 Uhr, in Gebäude 51, 2. OG, Raum 02-028 (Büro Prof. Bast) statt.

Bemerkung zu den Programmieraufgaben: Wann immer das Schreiben von Code gefragt ist, können Sie frei zwischen Python, Java und C++ wählen. Für kleinere, rein syntaktische Fehler, wie z.B. ein fehlendes Semikolon, gibt es keinen Punktabzug.

Wichtig: alle Programmieraufgaben dieser Klausur lassen sich mit wenig Code lösen. Keines Ihrer Programme sollte länger als 10 Zeilen sein, sonst gibt es Punktabzug oder gar keine Punkte. Dabei sollte in jeder Zeile nur eine Anweisung stehen, wie in der jeweiligen Sprache üblich. Einzige Ausnahme sind if *und* else *Anweisungen in Python: hier darf die Anweisung nach dem Doppelpunkt ausnahmsweise in derselben Zeile stehen.*

Aufgabe 1 (O-Notation, Sortieren und Laufzeit, 25 Punkte)

1.1 (10 Punkte) Sei $f(n) = n^2$. Finden Sie eine Funktion $g(n)$, so dass $g(n) < f(n)$ für alle $n < 10^9$, aber trotzdem nicht $g = O(f)$. Beweisen Sie dabei die Aussage, dass nicht $g = O(f)$ über die Definition von O und nicht über den Grenzwert.

1. Wir definieren $g(n) := n^3/10^9$
2. Damit gilt für $n < 10^9$, dass $g(n) < n^2 \cdot 10^9/10^9 = n^2 = f(n)$
3. Falls $g = O(f)$, gäbe es C und n_0, so dass für alle $n \geq n_0$, $g(n) \leq C \cdot f(n)$
4. Dann wäre $n^3/10^9 \leq C \cdot n^2$ und also $n \leq C \cdot 10^9$; das ist ein Widerspruch, also $g \neq O(f)$

1.2 (5 Punkte) Betrachten Sie den folgenden Sortieralgorithmus. Geben Sie für die Eingabe $[3, 2, 1]$ die I/E-Folge an, wobei ein I an Stelle i heißt, dass bei dem i-ten Durchlauf von Zeile 5 die *if*-Bedingung erfüllt war und ein E an der Stelle heißt, dass sie nicht erfüllt war. Die impliziten Bedingungen in den beiden *for*-Schleifen können für diese Aufgabe ignoriert werden.

```
1.  def slow_bubble_sort(A):
2.    n = len(A)
3.    for i in range(n - 1):
4.      for j in range(n - 1):
5.        if A[j] > A[j + 1]:
6.          A[j], A[j + 1] = A[j + 1], A[j]
```

1. Zeilen 5-6 werden $(n-1)^2$ mal durchlaufen, für Eingabe $[3, 2, 1]$ also 4 mal
2. Der Zustand des Feldes nach jedem der vier Durchläufe: $[2, 3, 1]$, $[2, 1, 3]$, $[1, 2, 3]$, $[1, 2, 3]$
2. Die entsprechende Sequenz bezüglich der Bedingung in Zeile 5 lautet: I, I, I, E

1.3 (10 Punkte) Bestimmen Sie für das folgende Programm die Ausgabe und die Laufzeit als $\Theta(...)$ in Abhängigkeit von n. Sie können dabei ohne Beweis benutzen, dass $\sum_{i=1}^{n} i^2 = 1/6 \cdot n \cdot (n+1) \cdot (2n+1)$. Hinweis: Bestimmen Sie zuerst, wie oft die beiden inneren Schleifen (Zeilen 4 und 5) insgesamt die Zeile 6 ausführen, in Abhängigkeit von i.

```
1.  def three_nested_loops(n):
2.    result = 0
3.    for i in range(n):
4.      for j in range(i):
5.        for k in range(j):
6.          result += 6
7.  return result
```

1. Zeile 6 wird $\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} j = \sum_{i=0}^{n-1} (i \cdot (i-1)/2)$ mal ausgeführt
2. Das ist gleich $\sum_{i=0}^{n-1} i^2/2 - \sum_{i=0}^{n-1} i/2 = 1/12 \cdot n \cdot (n-1) \cdot (2n-1) - 1/4 \cdot n \cdot (n-1)$
3. Vereinfacht ergibt sich $1/12 \cdot n \cdot (n-1) \cdot (2n-1-3) = 1/6 \cdot n \cdot (n-1) \cdot (n-2)$
4. Das Ergebnis ist also $n \cdot (n-1) \cdot (n-2)$ und die Laufzeit $\Theta(n^3)$

Aufgabe 2 (Hashing, 25 Punkte)

Wir betrachten in dieser Aufgabe für eine gegebene Größe m der Hashtabelle die folgende Klasse von Hashfunktionen: $H_m = \{\, h : a \cdot x^2 \bmod m \mid a \in \{0, \ldots, m-1\} \,\}$.

2.1 (10 Punkte) Zeichnen Sie die Hashtabelle für die Schlüsselmenge $S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ und die Hashfunktion $h(x) = 3 \cdot x^2 \bmod 5$. Benutzen Sie Hashing mit Verkettung.

1. Die Werte für $x^2 \bmod 5$ sind $0, 1, 4, 4, 1, 0, 1, 4, 4, 1$
2. Die Werte für $3 \cdot x^2 \bmod 5$ sind also $0, 3, 2, 2, 3, 0, 3, 2, 2, 3$
3. Die Hashtabelle sieht also wie folgt aus:

$0 \to [\,0, 5\,]$
$1 \to [\,]$
$2 \to [\,2, 3, 7, 8\,]$
$3 \to [\,1, 4, 6, 9\,]$
$4 \to [\,]$

2.2 (5 Punkte) Sei S eine zufällige Menge mit n verschiedenen ganzzahligen Schlüsseln und sei $h(x)$ die Hashfunktion aus Aufgabe 2.1. Berechnen Sie den Erwartungswert der Größe der Menge $S_1 = \{x \in S \mid h(x) = 1\}$.

1. Gemäß Aufgabe 2.1 ist für alle $x \in \{0, 1, 2, 3, 4\}$ der Hashwert $h(x) \neq 1$
2. Da $x \bmod 5 \in \{0, 1, 2, 3, 4\}$, gilt für jede beliebige ganze Zahl x, dass $h(x) \neq 1$
3. Damit ist für jede beliebige Schlüsselmenge S immer $S_1 = \emptyset$ und also $E|S_1| = 0$

2.3 (5 Punkte) Beweisen oder widerlegen Sie: Die Klasse H_5 ist 2-universell. Als Schlüsseluniversum können Sie die natürlichen Zahlen annehmen.

1. Für $x = 2$ und $y = 3$ ist $3 \cdot x^2 \bmod 5 = 3 \cdot y^2 \bmod 5 = 4$
2. Damit ist für dieses Schlüsselpaar $|\{h \in H_5 : h(x) = h(y)\}| = 5$
3. Für 2-Universalität müsste die Größe der Menge ≤ 2 sein; H_5 ist also nicht universell

2.4 (5 Punkte) Schreiben Sie eine Funktion *check_universality(m, c, u)*, die berechnet, ob die Klasse H_m für das Schlüsseluniversum $U = \{0, \ldots, u-1\}$ c-universell ist. Die Funktion soll entsprechend *True* oder *False* zurückgeben.

```
 1.  def check_universality(m, c, u):
 2.    for x in range(u):
 3.      for y in range(u):
 4.        count = 0
 5.        for a in range(m):
 6.          if (a * x * x) % m == (a * y * y) % m:
 7.            count += 1
 8.      if count > c:
 9.        return False
10.    return True
```

Aufgabe 3 (Dynamische Felder und Potenzialfunktionen, 25 Punkte)

Für diese Aufgabe nehmen wir ein dynamisches Feld der anfänglichen Größe $s_0 = 0$ und Kapazität $c_0 = 1$ an. Wir nehmen außerdem an, dass es nur *append* Operationen gibt und dass das Feld immer genau dann vergrößert wird, wenn die Kapazität für das neue Element nicht mehr ausreicht. Seien s_i und c_i die Größe und Kapazität nach der i-ten Operation. Dann soll nach einer Operation, die zu einer Vergrößerung führt, $c_i = \lceil 3/2 \cdot s_i \rceil$ sein.

3.1 (10 Punkte) Schreiben Sie eine Funktion *predict_capacity(n)*, die die Kapazität des zu Beginn leeren Feldes nach n Operationen zurückgibt. Die Laufzeit der Funktion ist für diese Aufgabe irrelevant.

```
1. def predict_capacity(n):
2.   size = 0
3.   capacity = 1
4.   for i = range(n):
5.     if capacity == size:
6.       capacity = math.ceil(3/2 * (size + 1))
7.     size += 1
8.   return capacity
```

3.2 (10 Punkte) Sei T_i die Laufzeit der i-ten Operation. Definieren Sie eine geeignete Potenzialfunktion Φ und Konstanten A und B und beweisen Sie damit, dass $T_i \leq A + B \cdot (\Phi_i - \Phi_{i-1})$ für jedes i.

1. Definiere $\Phi_i = c_i - s_i$, also die Anzahl der noch "freien" Plätze nach der i-ten Operation
2. Für geeignete A', B' ist die Laufzeit einer Operation $T_i \leq A'$ bzw. $T_i \leq A' + B' \cdot s_i$
3. Für eine billige Operation ist $\Phi_i - \Phi_{i-1} = -1$
4. Für eine billige Operation ist also $T_i \leq A' + B' - B' = (A' + B') + B' \cdot (\Phi_i - \Phi_{i-1})$
5. Für eine teure Operation ist vorher $\Phi_{i-1} = 0$ und hinterher $\Phi_i = \lceil 3/2 \cdot s_i \rceil - s_i \geq s_i/2$
6. Für eine teure Operation ist also $T_i \leq A' + 2B' \cdot (\Phi_i - \Phi_{i-1})$
7. Für $A := A' + B'$ und $B := 2B'$ ist also in jedem Fall $T_i \leq A + B \cdot (\Phi_i - \Phi_{i-1})$

3.3 (5 Punkte) Nehmen wir an, es wurden bereits (auf dem zu Beginn leeren Feld) $2n^2$ Operationen ausgeführt und nun werden weitere n Operationen ausgeführt. Bestimmen Sie die Gesamtlaufzeit dieser n Operationen als $\Theta(...)$ in Abhängigkeit von n im besten und im schlechtesten Fall.

1. Best case: nach den $2n^2$ Operationen ist $c_i - s_i \leq n$, dann ist die Gesamtlaufzeit $\Theta(n)$, weil nicht realloziert werden muss und jede Operation in Zeit $\Theta(1)$ läuft
2. Worst case: nach den $2n^2$ Operationen ist $c_i - s_i = 0$, dann ist die Gesamtlaufzeit $\Theta(n^2)$, weil gleich zu Beginn $2n^2$ Elemente umkopiert werden müssen, danach ist Platz für $n^2 \geq n$ Elemente

Aufgabe 4 (Verschiedenes, 25 Punkte)

4.1 (5 Punkte) Bei einem (a, b)-Baum hat man für einen inneren Knoten mit k Kindern $k-1$ Wegweiser, wobei der i-te Wegweiser der größte Wert im Unterbaum des i-ten Kindes ist. Warum benötigt man nicht auch den größten Wert des Unterbaums des k-ten Kindes? Und was würde schiefgehen, wenn jeder innere Knoten auch diesen Wert speichern würde, also k Wegweiser hätte anstatt $k-1$? Beantworten Sie beide Fragen jeweils mit einem prägnanten Satz.

1. Für die Suche muss man an jedem inneren Knoten lediglich wissen, auf welchen Unterbaum man die weitere Suche beschränken kann; dafür sind $k-1$ Wegweiser ausreichend
3. Der Wert eines Blattes könnte dann an mehreren Stellen im Baum als Wegweiser vorkommen und dann wäre die amortisierte Laufzeit von insert und remove nicht mehr unbedingt $O(1)$

4.2 (5 Punkte) Beweisen Sie, dass $\mathrm{ED}(x, y) \leq \mathrm{ED}(x', y') + 1$, wobei $x' = x[1..n-1]$ mit $n = |x|$ und $y' = y[1..m-1]$ mit $m = |y|$ ist.

1. Sei $k = \mathrm{ED}(x', y')$ und sei $O_1, ..., O_k$ die Folge von Operationen, die x' in y' überführt
2. Falls das letzte Zeichen von x und y gleich ist, überführt die Folge auch x in y und $\mathrm{ED}(x, y) = k$
3. Falls die letzten Zeichen ungleich sind, nehme als Operation O_{k+1} das entsprechende replace.

4.3 (5 Punkte) Wahr oder falsch: Bevor Dijkstras Algorithmus sicher sein kann, dass er den kürzesten Weg zum Zielknoten berechnet hat, muss er die kürzesten Wege zu allen anderen Knoten im Graphen berechnet haben? Mit Begründung!

1. Sei s der Startknoten und t der Zielknoten
2. Bevor t gelöst werden kann, müssen alle Knoten v mit $\mu(v, s) < \mu(t, s)$ gelöst werden
3. Das sind aber nicht notwendigerweise alle Knoten im Graphen
4. Die Aussage ist also falsch

4.4 (10 Punkte) Gegeben sei ein binärer Heap mit n Elementen. Bestimmen Sie die Anzahl der I/O Operationen als $\Theta(...)$ in Abhängigkeit von n und der Blockgröße B für die Operation *repair_heap_upwards* im schlechtesten Fall, d.h. wenn entlang eines Pfades von einem Blatt bis zur Wurzel getauscht werden muss. Sie können annehmen, dass M/B größer als eine Konstante Ihrer Wahl ist und dass B und $n+1$ Zweierpotenzen sind. Hinweis: die Elemente in den obersten Schichten des Heaps (eine Schicht = Knoten gleicher Tiefe) stehen alle in einem Block.

1. Die $B-1$ Knoten der Tiefen $1, \ldots, \log_2 B$ stehen in einem Block
2. Ab da bestehen die Schichten (Knoten mit derselben Tiefe) aus $B, 2B, 4B, 8B, \ldots$ Blöcken
3. Ingesamt greift *repair_heap_upwards* auf jeweils ein Element aus $\log_2 n$ Schichten zu
4. Die oberen $\log_2 B$ Schichten benötigen also eine Blockop., die unteren $\log_2 n - \log_2 B$ Blockop.
5. Die Anzahl der Blockoperationen ist also $\log_2(n/B) = \Theta(\log(n/B))$

Aufgabe 5 (String Matching, 25 Punkte)

Für die ersten beiden Aufgaben nehmen wir an, dass die Zeichenketten Binärstrings sind, also jedes Zeichen entweder die Zahl 0 oder die Zahl 1 ist.

5.1 (10 Punkte) Führen Sie einen Durchlauf des Karp-Rabin Algorithmus mit Text 11001010001 und Muster 0101 durch. Das Muster und die Textstücke sollen dabei als Binärzahlen zur Basis 2 aufgefasst werden. Als Hashfunktion soll $h(x) = x \bmod 3$ verwendet werden. Das Muster hat also den Hashwert $h(5) = 2$ und das Textfenster der Größe 4 ab Stelle 0 hat den Hashwert $h(12) = 0$. Sie müssen für diese Aufgabe nicht beschreiben, wie Sie die Hashwerte berechnen, es reicht der jeweilige Wert.

Text: 11001010001, Pattern: $0101 = 5 \rightarrow 2$

1. $1100 = 12 \rightarrow 0$ ... NO
2. $1001 = 9 \rightarrow 0$ ... NO
3. $0010 = 2 \rightarrow 2$ ... CHECK $\rightarrow$ NO
4. $0101 = 5 \rightarrow 2$ ... CHECK $\rightarrow$ YES
5. $1010 = 10 \rightarrow 1$ ... NO
6. $0100 = 4 \rightarrow 1$ ... NO
7. $1000 = 8 \rightarrow 2$ ... CHECK $\rightarrow$ NO
8. $0001 = 1 \rightarrow 1$ ... NO

5.2 (5 Punkte) Schreiben Sie eine Funktion *next_hash(T, i, j, hx)*, die gegeben den Text T, eine Startposition i und eine Endposition j und den Hashwert hx von $T[i-1 \,..\, j-1]$, den Hashwert von $T[i \,..\, j]$ berechnet. Für $i = 0$ soll die Funktion in Zeit $O(j)$ laufen und der Wert von hx ignoriert werden. Für $i > 0$ soll die Funktion in Zeit $O(1)$ laufen. Sie können dabei annehmen, dass 2^k mit der Operation $1 << k$ in $O(1)$ Zeit berechnet werden kann.

```
1.  def next_hash(T, i, j, hx):
2.    if i > 0:
3.      return 2 * (hx - T[i - 1] * (1 << (j - i))) + T[j]
4.    else:
5.      result = 0
6.      for k = range(j + 1):
7.        result += T[k] * (1 << (j - k))
8.      return result % 3
```

5.3 (5 Punkte) Sei *shift* das vom KMP-Algorithmus vorberechnete Feld für ein Muster. Geben Sie ein Beispiel für ein Muster und das zugehörige *shift* Feld, so dass an einer Stelle $0 < \mathit{shift}[i+1] < \mathit{shift}[i]$ gilt.

```
Muster: DUDADUDU
Shift:  00101232
```

5.4 (5 Punkte) Beweisen Sie, dass immer $\mathit{shift}[i+1] \leq \mathit{shift}[i] + 1$ gilt.

1. Sei $k = \mathit{shift}[i+1]$, dann ist $\text{pattern}[i+2-k \,..\, i+1] = \text{pattern}[1 \,..\, k]$
2. Dann ist mit Sicherheit auch $\text{pattern}[i+2-k \,..\, i] = \text{pattern}[1 \,..\, k-1]$
3. Damit ist $\mathit{shift}[i] \geq k - 1 = \mathit{shift}[i+1] - 1$ und daraus folgt die Behauptung