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

Informatik II: Algorithmen & Datenstrukturen

Montag, 29. August, 2014, 14:00 – 17:00

Name: .....

Matrikelnummer: .....

Unterschrift: .....

Blättern Sie nicht um bevor Sie dazu aufgefordert werden!

- Schreiben Sie **auf alle Blätter** (inklusive Deckblatt und etwaiger zusätzlicher Blätter) Ihren **Vornamen, Nachnamen** und Ihre **Matrikelnummer**.
- **Unterschreiben Sie das Deckblatt!** Ihre Unterschrift bestätigt, dass Sie alle Fragen ohne nicht erlaubte Hilfsmittel beantwortet haben.
- Schreiben Sie **lesbar** und nur mit **dokumentenechten** Stiften. Schreiben Sie nicht in **rot** oder **grün** und benutzen Sie keinen Bleistift!
- Alle schriftlichen Hilfsmittel (Bücher, Vorlesungsunterlagen, handschriftliche Notizen, etc.) sind erlaubt. **Elektronische Hilfsmittel sind nicht erlaubt.**
- Die Klausur besteht aus 8 Aufgaben und 120 Punkten. Zum Bestehen reichen 50 Punkte.
- Benutzen Sie für jede Aufgabe eine eigene Seite.
- Es wird nur eine Lösung pro Aufgabe gewertet. Vergewissern Sie sich, dass Sie zusätzliche Lösungen selbst entwerten. Falls mehrere Lösungen zu einer Aufgabe existieren, so wird die schlechtere Lösung gewertet.
- Die folgenden Regeln gelten überall, außer sie werden explizit außer Kraft gesetzt. Bei Laufzeitfragen ist wie üblich nur die asymptotische Laufzeit notwendig. Wenn Sie einen Algorithmus angeben sollen, so können Sie Pseudocode angeben, eine Beschreibung der Funktionsweise Ihres Algorithmus ist allerdings ausreichend. Algorithmen sind immer effizient zu konstruieren, d. h., mindestens polynomiell und i. d. R. schneller als eine naive Lösungsmethode. Algorithmen aus der Vorlesung können grundsätzlich als Blackbox verwendet werden.
- **Falls Sie Algorithmen entwerfen, welche Hashtabellen (als Blackbox) verwenden, dürfen Sie annehmen, dass alle Operationen der Hashtabelle $\mathcal{O}(1)$ Zeit benötigen.**
- **Erklären Sie Ihre Lösungen, außer es wird explizit darauf hingewiesen, dass dies nicht nötig ist! Nur das Endresultat aufzuschreiben ist nicht ausreichend.**

Frage	1	2	3	4	5	6	7	8	**Total**
Punkte									
Maximum	29	14	16	17	9	10	10	15	120

1 Kurze Fragen (29 Punkte)

a) (4 Punkte) Welche Eigenschaften müssen die Elemente eines Arrays $H[1, \ldots, n]$ besitzen, damit sie bei der Array-Implementierung aus der Vorlesung einen gültigen Heap (Prioritätswarteschlange) mit n Elementen bilden?

b) (3 Punkte) Ist es möglich, vergleichsbasiert, einen binären Suchbaum so zu implementieren, dass man mit $o(n \log n)$ Vergleichen n Schlüssel einfügen kann? Begründen Sie ihre Antwort.

c) (5 Punkte) Es sei ein DAG (gerichteter azyklischer Graph) gegeben. Nehmen Sie an, der Graph hat einen Knoten s, so dass jeder andere Knoten des Graphen über einen gerichteten Pfad von s erreichbar ist. Ergibt die Besuchsreihenfolge der Knoten einer bei s gestarteten BFS Traversierung eine topologische Sortierung des Graphen? Erkären Sie weshalb oder geben Sie ein Gegenbeispiel an.

d) (5 Punkte) Gegeben sei die folgende Häufigkeitsverteilung:

$$A : 10\%, B : 15\%, C : 8\%, D : 23\%, E : 11\%, F : 33\%.$$

Geben sie einen optimalen präfixfreien binären Code an.

e) (4 Punkte) Bestimmen Sie die Editierdistanz der Wörter *SONNE* und *MOND*. Geben Sie auch eine minimale Sequenz von Editieroperationen an, um von *SONNE* zu *MOND* zu kommen.

f) (3 Punkte) Sie haben eine Anwendung bei der zunächst n Elemente in eine Hashtable eingefügt werden und danach sehr viele Anfragen ($\gg n$) ausgeführt werden. Welches der in der Vorlesung besprochenen Hashing-Verfahren eignet sich am besten? Begründen Sie Ihre Wahl.

g) (5 Punkte) Zeichnen Sie einen gültigen Rot-Schwarz-Baum, welcher Tiefe 5 hat und welcher möglichst wenige Knoten enthält (NIL-Knoten sind hier nicht mitgezählt). Die Tiefe eines Rot-Schwarz-Baumes ist der größte Abstand eines Nicht-NIL-Knoten von der Wurzel.

Hinweis: *Es gibt einen solchen Baum mit 14 Nicht-NIL-Knoten. Sie müssen nur einen gültigen Baum zeichnen. Eine Sequenz von Operationen, welche den Baum erzeugt, müssen Sie* ***nicht*** *angeben.*

2 Landau Notation (14 Punkte)

Begründen Sie alle Antworten zu dieser Aufgabe ausführlich.

a) (6 Punkte) Zeigen Sie mit der Definition von $\mathcal{O}(\cdot)$, dass $\sqrt[3]{3n + 10}$ in $\mathcal{O}(\sqrt[3]{n})$ enthalten ist.

b) (4 Punkte) Zeigen Sie, dass n^n in $\Omega(n!)$ liegt.

c) (4 Punkte) Gilt $\log_2(n^3) \in \Theta\big(\ln(3n^7)\big)$? Begründen Sie ihre Antwort.

3 Dynamische Programmierung (16 Punkte)

Wir suchen einen Algorithmus, der die minimale Münzenanzahl bestimmt, die man benötigt um eine vorgegeben Betrag B zu bezahlen (Man muss genau den Betrag B bezahlen, d.h. es gibt kein Rückgeld).

Genauer: Gegeben sei ein Array M mit k verschiedenen ganzzahligen Münzwerten $m_0, \ldots, m_{k-1} \in \mathbb{N}$, $k > 0$, $m_i \geq 1$ sowie ein ganzzahliger Betrag $B \in \mathbb{N}$. Wir wollen die Anzahl der Münzen $A := \sum_{i=0}^{k-1} a_i$ minimieren, wobei $a_0, \ldots, a_{k-1} \in \mathbb{N}_0$ ganze Zahlen sind, die $\sum_{i=0}^{k-1} a_i m_i = B$ erfüllen.

Ein fleißiger Student hat dazu den folgenden Algorithmus entworfen:

```
int minimum(int[] M, int B):
    sort(M)          # sorts M in decreasing order
    int i = 0
    int c = 0
    while B > 0:
        if B < M[i]:
            i += 1
            if i >= len(M):
                return -1      # no solution found
        else:
            B = B - M[i]
            c += 1

    return c
```

a) (3 Punkte) Welches Algorithmendesign-Prinzip benutzt der obige Algorithmus (Algorithmendesign-Prinzipien sind z. B. Dynamische Programmierung oder Divide and Conquer)?

b) (5 Punkte) Geben Sie Münzwerte $m_0, \ldots, m_{k-1}$, sowie einen Betrag B vor, so dass zwar eine Lösung existiert, der obige Algorithmus jedoch keine optimale Lösung findet.

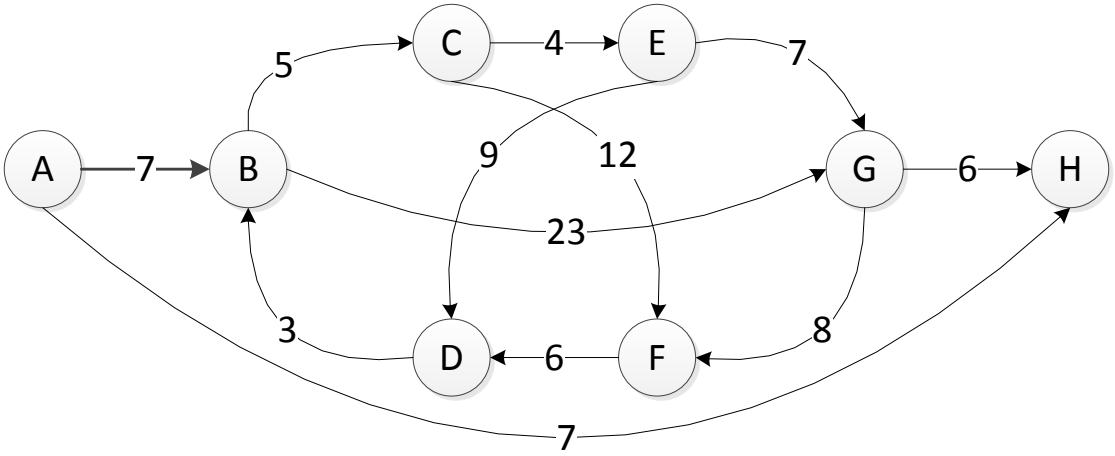
c) (8 Punkte) Beschreiben Sie einen auf dynamischer Programmierung basierenden Algorithmus, der die minimale Münzanzahl berechnet, die nötig ist um den Betrag B zu bezahlen.

   Was ist die asymptotische Laufzeit Ihres Algorithmus?

Hinweis: *Sie können bei Teilaufgabe c) davon ausgehen, dass es immer möglich ist den Betrag B mit den Münzen zu bezahlen.*

4 Kürzeste Wege (17 Punkte)

a) (8 Punkte) Führen Sie den Dijkstra Algorithmus auf dem gegebenen gerichteten Graph aus um kürzeste Wege von Knoten A zu allen anderen Knoten zu finden. Schreiben Sie für jeden der Knoten $\{B, \ldots, H\}$ die Länge des Pfades von A zu diesem Knoten aus und geben Sie an, in welcher Reihenfolge der Dijkstra Algorithmus die Knoten bearbeitet.



Bei Teilaufgaben b) und c) ist ein ungerichteter Graph $G = (V, E)$ mit Kantengewichtsfunktion $w : E \to \mathbb{N}$ gegeben. Das Ziel dieser Aufgabe ist es für einen Startknoten $s \in V$, kürzeste Pfade zu allen anderen Knoten zu finden.

b) (3 Punkte) Geben Sie einen möglichst effizienten Algorithmus an um das Problem zu lösen, wenn alle Kantengewichte identisch 1 sind, d.h., $w(e) = 1$ für alle $e \in E$.

Was ist die Laufzeit Ihres Algorithmus?

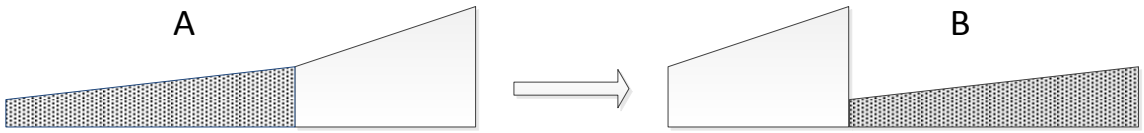
c) (6 Punkte) Geben Sie einen möglichst effizienten Algorithmus an um das Problem zu lösen, wenn alle Kantengewichte 1 oder 2 sind, d.h., $w(e) \in \{1, 2\}$ für alle $e \in E$.

Was ist die Laufzeit Ihres Algorithmus?

Hinweis: *Schnellere Algorithmen liefern mehr Punkte. Für (zu) langsame Algorithmen erhalten Sie nur wenige Punkte.*

5 Minimum Finden (9 Punke)

Sie sollen ein Array A der Länge n, welches aufsteigend sortiert mit n verschiedenen ganzen Zahlen gefüllt ist, erhalten. Leider geht bei der Übergabe etwas schief, so dass das Array nicht mehr korrekt sortiert ist. Statt des Arrays A erhalten Sie ein Array B, welches man wie folgt aus A erhält: Das Array A wird an einer Ihnen unbekannten Stelle r, $0 \leq r \leq n-1$ geteilt und dann vertauscht zusammen gesetzt, d.h. für $0 \leq j < r$ gilt $B[j] := A[r+j]$ und für $r \leq j < n$ gilt $B[j] := A[j-r]$.



Geben Sie einen Algorithmus an, welcher das kleinste Element im Array B findet und Laufzeit $\mathcal{O}(\log n)$ hat. Ihr Algorithmus bekommt lediglich das Array B als Eingabe und kennt das Array A nicht!

6 Mystische Suche (10 Punkte)

Betrachten Sie folgenden Pseudocode der Funktion *mystery(array)*, welche als Eingabe ein Array mit positiven ganzen Zahlen erhält.

```
int mystery(int[] array):
    result = []
    for i in range(0, len(array)):
        current = array[i]
        for j in range(i + 1, len(array)):
            if current == array[j]:
                result.append(current)
                break          # aborts inner for loop
    if len(result) == 0:
        return -1
    else:
        return result[0]
```

a) (5 Punkte) Welches Problem löst die Funktion *mystery*? Welche Laufzeit hat diese Funktion im *Worst Case*?

b) (5 Punkte) Wie kann die Funktion verbessert werden um das gleiche Problem in Linearzeit zu lösen?

7 Minimale Spannbäume (10 Punkte)

Bei den folgenden zwei Teilaufgaben müssen Sie jeweils einen zusammenhängenden, ungerichteten und gewichteten Graphen konstruieren, welcher aus einem Knoten v und noch mindestens 2 weiteren Knoten besteht. Die Kantengewichte des Graphen sollen alle verschieden sein.

a) (5 Punkte) Konstruieren Sie einen Graphen, so dass der minimale Spannbaum (MST) und der Shortest Path Tree von v **verschieden** sind (den Shortest Path Tree betrachten wir hierbei als ungerichteten Baum).

b) (5 Punkte) Konstruieren Sie einen Graphen, so dass der minimale Spannbaum (MST) und der Shortest Path Tree von v **gleich** sind (den Shortest Path Tree betrachten wir hierbei als ungerichteten Baum).

8 String Matching (15 Punkte)

Gegeben sei der folgende Pseudocode, welcher das String-Matching-Problem lösen soll.

```
int pattern_finder(String text, String pattern):
    int i = 0
    while i < len(text) - len(pattern) + 1:
        int counter = 0
        for j in range(0, len(pattern)):
            if text[i+j] == pattern[j]:
                counter +=1
            else:
                break
        if counter == 0:
            i += 1
        elif counter != len(pattern):
            i += counter
        else:
            return i
    return -1
```

a) (8 Punkte) Wieso ist der obige String-Matching Algorithmus nicht korrekt? Geben Sie ein Gegenbeispiel an. Welche Laufzeit hat der Algorithmus?

b) (7 Punkte) Der Algorithmus schlägt nicht für jedes Beispiel fehl. Für welche Eingaben funktioniert der Algorithmus trotzdem? Geben Sie eine (möglichst große) Klasse von Eingaben an.