

Aufgabensammlung Algorithmen, Datenstrukturen und Programmierung

Aufgabensammlung für die Theoretische Übung zu
Algorithmen, Datenstrukturen und Programmierung

Lehrstuhl Management der Informationssicherheit

22. April 2008

Inhaltsverzeichnis

1	Algorithmen und logisches Denken	2
2	Elementare Datenstrukturen	3
3	Sortieren	5
4	Laufzeitanalyse	6
5	Suchen	8
6	Syntaxanalyse	10
7	Backtracking	10
8	Entrekursivierung	11
9	Komplexe Algorithmen	11
10	Graphen	13

1 Algorithmen und logisches Denken

Aufgabe 1.1 Matrizenmultiplikation

Diese Aufgabe bereitet die Implementierung einer Matrizenmultiplikation in der praktischen Übung vor.

- Frischen Sie Ihr Grundwissen über Matrizenrechnung auf.
- Welche Dimensionen (Zeilenzahl, Spaltenzahl) müssen zwei Matrizen A und B besitzen, damit sie miteinander multipliziert werden können ($A * B$)? Welche Dimension hat die resultierende Matrix C ?
- Wie viele Additionen und Multiplikationen von Zahlen benötigt die Multiplikation einer Matrix A (k Zeilen, l Spalten) mit einer Matrix B (m Zeilen, n Spalten).

Aufgabe 1.2 Analyse einer Funktion

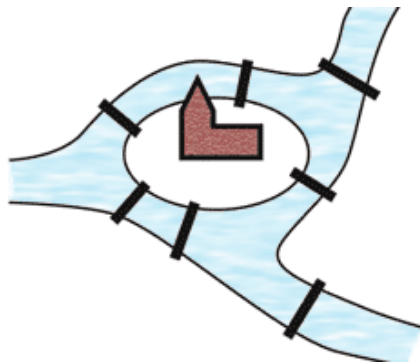
Gegeben ist folgende Funktion (Setzen Sie für den ersten Aufruf $x > y > 0$ voraus)

```
long methodeZ(long x, long y) {
    if (x >= y)
        return (x+methodeZ(x-1,y));
    else
        return 0;
}
```

- Was lässt sich mit dieser Funktion berechnen?
- Wie lässt sich das Ergebnis mit Hilfe einer Schleife ermitteln?
- Gibt es andere (einfache) Alternativen zur Ermittlung des Ergebnisses (v. a. für $y = 1$)?

Aufgabe 1.3 Königsberger Brückenproblem

Bei einem Rundgang durch Königsberg (heute Kaliningrad) stieß der Mathematiker Leonhard Euler (1707 - 1783) auf folgendes Problem: Bei einem Rundgang durch Königsberg sollen alle sieben Brücken über den Fluss Pregel jeweils genau einmal überschritten werden (vgl. Abbildung). Ist es möglich einen solchen Rundweg zu finden? Begründung.



Aufgabe 1.4 Informationsaustausch über das Telefon

n people each know a different piece of gossip. They can telephone each other and exchange all the information they know (so that after the call they both know anything that either of them knew before the call). What is the smallest number of calls needed so that everyone knows everything?

Aufgabe 1.5 Schachbrettknocheien

Das Schachspiel besitzt den wunderbaren Vorzug, durch Bannung der geistigen Energien auf ein eng begrenztes Feld selbst bei anstrengendster Denkleistung das Gehirn nicht zu erschöpfen, sondern eher seine Agilität und Spannkraft zu schärfen. (Stefan Zweig)

Hinweis: Ein Turm bedroht alle Felder seiner Spalte und alle Felder seiner Zeile. Ein Springer bedroht alle Felder, die von seiner aktuellen Position aus im Rösselsprung erreichbar sind („Zwei vor/zurück, eins zur Seite“ oder „Eins vor/zurück, zwei zur Seite“).

- Wie viele Türme, die sich gegenseitig nicht bedrohen, lassen sich auf einem $n * n$ Schachbrett platzieren.
- Wie viele Möglichkeiten gibt es, die oben ermittelte Maximalzahl von Türmen auf dem Brett anzuordnen?
- Können 32 Springer auf einem Schachbrett so angeordnet werden, dass sie sich gegenseitig nicht bedrohen? Falls ja, wie gehen Sie vor?
- Finden Sie für jede der Schachfiguren Dame, Läufer und König jeweils die maximale Anzahl N , für die man N derartige Figuren auf einem Schachbrett platzieren kann, ohne dass sie sich gegenseitig bedrohen. Begründen Sie, warum es sich dabei um die jeweilige Maximalzahl handelt!

2 Elementare Datenstrukturen

Aufgabe 2.1 Struktur, Liste, Array

Auf diversen Feiern der ersten Semesterwoche mussten Sie, wie schon oft, wieder einigen Ihrer Kommilitonen Geld leihen und stehen nun vor der Aufgabe dieses wieder einzutreiben. Sie beschließen daher ein Programm zur einfachen Verwaltung Ihrer Schuldner zu schreiben.

- Entwerfen Sie zunächst eine Klasse `Schuldner`, die zumindest folgende Angaben aufnehmen kann:
 - Name
 - geliehener Betrag
 - Fälligkeitsdatum
 - Telefonnummer
 - Wohnort bekannt (j/n)
- Entwerfen Sie eine weitere Datenstruktur `SchuldnerSammlung`, die mehrere `Schuldner` aufnehmen kann als Array/Feld. Wie würde eine Reihung der Schuldner als Liste aussehen? Welche Variante würden Sie vorziehen?
- Diskutieren sie allgemein die Vor- und Nachteile von Listen und Feldern.

Aufgabe 2.2 Doppelt verkettete Liste

Eine doppelt verkettete Liste mit Pseudoknoten dient dazu, verschiedene Aufträge zu speichern. Jeder Auftrag ist charakterisiert durch eine Auftragsnummer, den Namen eines Verantwortlichen und ein Fertigstellungsdatum.

- Definieren Sie eine entsprechende Klasse. Welche Befehle müssen zum Erzeugen eines neuen Knotens in einem Programm verwendet werden?
- Stellen Sie eine Liste mit drei Aufträgen graphisch dar. Visualisieren Sie außerdem die Operationen „Einfügen eines Auftrags“ und „Entfernen eines Auftrags“.

Aufgabe 2.3 Stapel

Führen Sie folgende Operationen auf einem Anfangs leeren Stapel aus (zeilenweise von links nach rechts; `push` legt ein Element auf dem Stapel ab, `pop` nimmt ein Element vom Stapel).

<code>push('A');</code>	<code>push('L');</code>	<code>push('F');</code>	<code>push('R');</code>	<code>push('E');</code>	<code>push('D');</code>
<code>push('E');</code>	<code>push('S');</code>	<code>push('I');</code>	<code>push('N');</code>	<code>pop();</code>	<code>push('B');</code>
<code>push('O');</code>	<code>pop();</code>	<code>pop();</code>	<code>push('P');</code>	<code>push('L');</code>	<code>push('E');</code>
<code>pop();</code>	<code>pop();</code>	<code>pop();</code>	<code>push('E');</code>	<code>push('R');</code>	<code>pop();</code>
<code>pop();</code>	<code>pop();</code>	<code>pop();</code>	<code>pop();</code>		

Stellen Sie den Inhalt des Stapels nach jeder Operation sowie die vom Stapel gelesenen Elemente dar.

Aufgabe 2.4 Warteschlange

Führen Sie folgende Operationen auf einer anfangs leeren Warteschlange aus (zeilenweise von links nach rechts; `put` fügt ein Element an die Warteschlange an, `get` entnimmt ein Element aus der Schlange).

<code>put('P');</code>	<code>put('O');</code>	<code>get();</code>	<code>put('L');</code>	<code>get();</code>	<code>get();</code>
<code>put('O');</code>	<code>put('N');</code>	<code>put('I');</code>	<code>get();</code>	<code>put('U');</code>	<code>get();</code>
<code>put('M');</code>	<code>get();</code>	<code>get();</code>	<code>get();</code>		

Stellen Sie den Inhalt der Schlange nach jeder Operation sowie die aus der Schlange gelesenen Elemente dar.

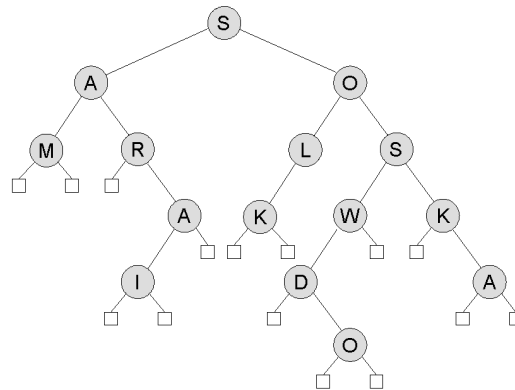
Aufgabe 2.5 Ausdrücke auswerten mit Stapel

Werten Sie folgenden arithmetischen Ausdruck mit Hilfe eines Stapels analog zu dem Vorgehen der Vorlesung (S. 29) aus.

$$(12 + ((1 + ((8 + 7) * 2)) * 61))$$

Aufgabe 2.6 Binärbäume

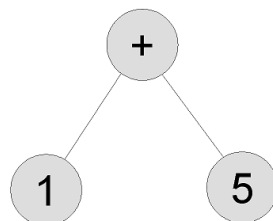
Gegeben ist unten stehender Binärbaum.



- Kennzeichnen Sie
 - die Wurzel
 - einen internen Knoten
 - einen externen Knoten
 - einen Teilbaum
- Wie groß ist die Tiefe des Baums?
- Wie viele interne Knoten enthält der Baum?
- Traversieren Sie den Baum in Preorder und Inorder Strategie
- Wie groß ist die Tiefe eines Binärbaums mit 15 internen Knoten im ungünstigsten und im günstigsten Fall?

Aufgabe 2.7 Binärbäume

Binärbäume können zur Darstellung von arithmetischen Ausdrücken verwendet werden. Der Ausdruck $(1 + 5)$ lässt sich folgendermaßen repräsentieren:



Stellen Sie folgenden arithmetischen Ausdruck als binären Baum dar. Die Auswertung soll Inorder erfolgen. Beschreiben Sie Ihr Vorgehen zur Konstruktion des Baums.

$$7 + ((63/7) + 21) * ((5 - 3) + (6 * 10))$$

Aufgabe 2.8 Operatorbaum, Prefix- und Postfixnotation

Transformieren Sie die Terme

- $(a + b) * (c - d)$

- b) $(a * b - c) + d * (e + f)$
 c) $(a + b) * c - (d + e) * (f - g)$

in Prefix- und Postfixnotation, indem Sie jeweils den zugehörigen Operatorbaum in geeigneter Weise durchlaufen.

3 Sortieren

Aufgabe 3.1 Sortieren durch Einfügen

Besorgen Sie sich ein Kartenspiel. Entnehmen Sie zufällig acht Karten. Nehmen Sie diese Karten mit einer Hand auf und sortieren Sie sie mit der anderen Hand. Wählen Sie dabei eine für Ihr Lieblingskartenspiel geeignete Reihenfolge.

Analysieren Sie Ihre Sortierstrategie und versuchen Sie diese als Algorithmus formulieren.

„Wenn ein [Karten]spieler seine Handkarten sortiert, geht er in der Regel so vor, dass er sich jeweils eine Karte ansieht und sie an die richtige Position in den bereits betrachteten Karten steckt.“(Sedgewick) Vergleichen Sie dieses Vorgehen mit Ihrem eigenen und informieren Sie sich über den Algorithmus Insertion Sort. Von welcher Ordnung ist die Laufzeit dieses Algorithmus? Warum wird das von Sedgewick beobachtete Vorgehen Ihrer Ansicht nach von vielen Spielern angewandt?

Aufgabe 3.2 Veranschaulichung verschiedener Sortierverfahren

Besorgen Sie sich acht verschieden große Geldstücke. Legen Sie diese Münzen in einer Linie in zufälliger Ordnung vor sich und sortieren Sie diese anhand ihres Durchmessers mit Hilfe folgender Algorithmen aufsteigend von links nach rechts:

- Selection Sort
- Insertion Sort
- Bubble Sort

Aufgabe 3.3 Analyse von Quicksort

In der Vorlesung wurde gezeigt, dass die Laufzeit von Quicksort stark von der Wahl der Pivot-Elemente abhängt. Gehen Sie für die folgenden Überlegungen von zufälliger Wahl der Pivot-Elemente aus.

- a) Welche Pivot-Elemente werden im worst-case gewählt? Welche Konsequenz hat das für die Partitionierung des Arrays?
- b) Formulieren Sie eine Rekurrenz für die worst-case Laufzeit von Quicksort und lösen Sie diese auf. Von welcher Ordnung ist die Laufzeit von Quicksort im worst-case?

Aufgabe 3.4 Stabilität von Sortierverfahren

Ein Sortierverfahren ist stabil, wenn es die relative Reihenfolge der Elemente mit doppelten Schlüsseln in der Datei bewahrt.

Ist der Algorithmus Bubble Sort stabil? Ist der Algorithmus Selection Sort stabil?

Beispiel: Es sind Autor-Werk-Paare zu sortieren.

ursprüngliche Ordnung:

Rilke - „Das Buch der Bilder“
 Rilke - „Das Stundenbuch“
 Zweig - „Das Lamm der Armen“
 Boccaccio - „Decamerone“
 Zweig - „Schachnovelle“

Sortierung nach Autor (nichtstabiles Verfahren):

Boccaccio - „Decamerone“
 Rilke - „Das Stundenbuch“
 Rilke - „Das Buch der Bilder“
 Zweig - „Das Lamm der Armen“

Zweig - „Schachnovelle“

Sortierung nach Autor (stabiles Verfahren):

Boccaccio - „Decamerone“

Rilke - „Das Buch der Bilder“

Rilke - „Das Stundenbuch“

Zweig - „Das Lamm der Armen“

Zweig - „Schachnovelle“

Aufgabe 3.5 Quicksort Beispiel

Sortieren Sie folgende Buchstabenfolge mit Hilfe des in der Vorlesung vorgestellten Quicksort-Algorithmus. Stellen Sie die Teilarrays nach jedem Partitionierungsschritt dar.

WALTERELIASDISNEY

„When you're curious, you find lots of interesting things to do. And one thing it takes to accomplish something is courage.“

4 Laufzeitanalyse

Aufgabe 4.1 Größenordnungen für Asymptotisches Laufzeitverhalten

Bei Laufzeitfunktionen $f : \mathbb{N} \rightarrow \mathbb{R}$ ist man praktisch nur an deren asymptotischem Verhalten interessiert. Man betrachtet also nur Eigenschaften die für alle $n \in \mathbb{N}$ gelten, welche größer als ein bestimmtes $n_0 \in \mathbb{N}$ sind. In der Praxis treten häufig die folgenden Funktionen zur Beschreibung des asymptotischen Wachstums auf:

- $(n!)^2$
- $n \log n$
- $\log n$
- n^q
- n^c
- c^n
- n
- 1
- $n!$

Es gilt $0 < q < 1 < c$.

a) Ordnen Sie alle Funktionen nach der Relation \prec an.

b) Erstellen Sie für obige Funktionen eine Tabelle in der Sie die Funktionswerte für verschiedene n (z.B. $n = 1, 5, 10, 100$) darstellen.

c) Ordnen Sie die Funktionen den folgenden Begriffen für Laufzeiten zu:

- superexponentiell
- exponentiell
- polynomial
- fast linear
- linear
- sub linear
- logarithmisch
- sub logarithmisch
- konstant

Hinweis: Rechner für große Zahlen: <http://www.wisis.ufg.edu.sv/labvirtual/math/math2/21/BigCalculator.html>

Aufgabe 4.2 Worst-case Laufzeiten in O -Notation

Geben Sie die worst-case-Laufzeiten der folgenden Probleme in der O -Notation an. Geben Sie jeweils eine kurze Begründung Ihrer Antwort.

- Das maximale Element aus einem unsortierten Feld mit n Elementen bestimmen.
- Das (sortierte) Einfügen eines Elementes in eine sortierte Liste mit n Elementen.
- Das Hinzufügen (push) eines Elementes zu einem mit einer Liste implementierten Stapel, der bereits n Elemente enthält.
- Das Sortieren von n Elementen durch Bubble Sort.
- Das Sortieren von n Elementen durch Mischen (merge sort).
- Die (binäre) Suche eines Elementes in einem sortierten Feld (array) von n Elementen.
- Die Berechnung der Fakultät von n .
- Die Berechnung des Skalarprodukts zweier Vektoren mit je n Elementen.
- Die Matrixmultiplikation zweier (n,n) -Matrizen.

Aufgabe 4.3 getPowerOf

Führen Sie eine Laufzeitanalyse der Funktion `getPowerOf()` durch. Erklären Sie die Einzelschritte Ihrer Berechnung.

```
int getPowerOf(int a, int x) {
    // requires (x >= 0);
    int power = 1;
    while (x != 0) {
        power = power * a;
        x--;
    }
    return power;
}
```

Aufgabe 4.4 Schnelle Exponentiation

Zur Berechnung von x^8 ist es geschickter,

```
x *= x;
x *= x;
x *= x;
```

zu programmieren und somit mit drei Multiplikationen auszukommen als etwa mit

```
p = 1;
for (int i=0; i<8; ++i) {
    p *= x;
}
```

acht Multiplikationen zu benötigen. Diese Beobachtung führt zu einem Algorithmus für die schnelle Exponentiation. Geben Sie den Algorithmus in Pseudocode an.

Hinweis: Jede positive Ganzzahl (also auch jeder Exponent) kann als Summe von 2er Potenzen dargestellt werden.

Aufgabe 4.5 Stringvergleich

Analysieren Sie die worst-case-Laufzeiten der Funktionen `compareStrings1` und `compareString2`.

Geben Sie die Laufzeit in der O -Notation an.

```
boolean compareCharArrays1
(char[] str1, char[] str2){
    int len1 = 0;
    int len2 = 0;
    int x = 0;
    int i = 0;
    while (len1 < str1.length) {len1++;}
    while (len2 < str2.length) {len2++;}
    while (x < len1) {
        if (str1[x] != str2[x])
            break;
        for (; i <= x; i++)
            System.out.println(str1[i]);
        x++;
    }
    if (x < len1)
        return false;
    else
        return true;
}
```

```
boolean compareCharArrays2
(char[] str1, char[] str2){
    int len1 = 0;
    int len2 = 0;
    int x = 0;
    while (len1 < str1.length) {len1++;}
    while (len2 < str2.length) {len2++;}
    while (x < len1) {
        if (str1[x] != str2[x])
            break;
        for (int i = 0; i <= x; i++)
            System.out.println(str1[i]);
        x++;
    }
    if (x < len1)
        return false;
    else
        return true;
}
```

Aufgabe 4.6 Auflösen von Rekurrenzen

Die alten Römer gingen erstmals in der Auseinandersetzung mit den Latinern (340 - 338 v. Chr.) nach dem Grundsatz „divide et impera“ - trenne (die Gegner) und beherrsche (sie dadurch) - vor. Dabei verhandelten sie mit ihren Gegnern einzeln und räumten den Städten je nach ihrer Bedeutung unterschiedliche Vorrechte ein. Dieses Prinzip nutzen auch etliche rekursive Algorithmen (engl. divide an conquer, dt. teile und herrsche).

Für viele Standardalgorithmen der Art „teile und herrsche“ (z. B. für Quicksort im günstigsten Fall) gilt folgende Rekurrenz:

$$T(n) = 2 * T(n/2) + n \text{ für } n \geq 2 \text{ mit } T(1) = 0$$

Lösen Sie die Rekurrenz auf.

Aufgabe 4.7 Rekurrenzen

Lösen Sie folgende Rekurrenz auf, für den Fall, dass n eine Potenz von α ist (also $n = \alpha^m$).

$$T(n) = T(n/\alpha) + 1 \text{ für } n \geq 2 \text{ mit } T(1) = 0$$

Aufgabe 4.8 O-Notation

Welche der folgenden Aussagen sind wahr? Begründen Sie Ihre Behauptungen.

- n^2 ist $O(n^3)$
- n^3 ist $O(n^2)$
- 2^{n+1} ist $O(n^2)$
- $(n + 1)!$ ist $O(n!)$
- $(f(n) \text{ ist } O(n)) \Rightarrow ((f(n))^2 \text{ ist } O(n^2))$
- $(f(n) \text{ ist } O(n)) \Rightarrow (2^{f(n)} \text{ ist } O(2n))$

5 Suchen

Aufgabe 5.1 Heap-Grundlagen

Ein Heap ist eine, meist auf Bäumen basierende, abstrakte Datenstruktur. Heaps werden z.B. bei Prioritätswarteschlangen eingesetzt, da mit Hilfe der Heap-Bedingung sicher gestellt werden kann, dass stets das Element mit der höchsten Priorität an der Wurzel des Baumes zu finden ist.

- Ist ein in absteigender Reihenfolge sortiertes Array ein Heap?
- Welche Zahl der Folge 50, 33, 44, 22, 77, 35, 43, 11 muss erniedrigt werden, damit die Heap-Eigenschaft erfüllt ist? Es sei dabei die übliche \geq Ordnung zugrunde gelegt. Begründen Sie Ihre Antwort!

Aufgabe 5.2 Aufbau eines Heap

Ein Heap kann aus den Arrayelementen $a[1], a[2], \dots, a[N]$ aufgebaut werden, durch N -maligen Aufruf der Funktion

- Gehen Sie aus von dem Array
 $a[1]=1, a[2]=6, a[3]=9, a[4]=2, a[5]=7, a[6]=5, a[7]=2, a[8]=7, a[9]=4, a[10]=10$
Zeichnen Sie den zugehörigen vollständigen binären Baum, wobei Sie die bei einem Heap übliche Verkettung verwenden.
- Machen Sie nun die Unterbäume der beiden untersten Niveaus durch Anwendung der Funktion `downheap` zu Heaps, dann

Aufgabe 5.3 Heap-Operationen

Es liege ein ursprünglich leerer Heap vor. Davon ausgehend werden die folgenden Operationen nacheinander ausgeführt:

`insert(10), insert(5), insert(2), replace(4), insert(6), insert(8),
remove(), insert(7), insert(3).`

Bedeutung der Operationen (Implementierung: siehe auch Vorlesung)

`insert(x)`: Einfügen eines neuen Elements x

`remove()`: Entfernen des größten Elements

`replace(x)`: Ersetzen des größten Elements durch ein neues Element x

Skizzieren Sie nach jedem Schritt den entstehenden Heap als Liste und als binären Baum.

Aufgabe 5.4 2-3-4 Baum, Rot-Schwarz Baum

- Skizzieren Sie den 2-3-4-Baum, der erzeugt wird, wenn die Schlüssel $E A S Y Q U T I O N$ in der angegebenen Reihenfolge in einen ursprünglich leeren Baum eingefügt werden.
- Zeichnen Sie einen entsprechenden Rot-Schwarz-Baum.

Aufgabe 5.5 Permutationen eines Wortes

Wie viele Permutationen eines Wortes aus n Buchstaben existieren allgemein?

Entwickeln Sie einen Algorithmus zur Berechnung aller Permutationen eines Wortes der Länge n . Geben Sie ihn entweder im Pseudo- oder im Programmcode an.

Führen Sie anschließend eine Laufzeitanalyse des Algorithmus durch und geben Sie dessen Komplexität in O -Notation an.

Beispiel: alle Permutationen der Zeichenfolge „ABCD“:

ABCD BACD CABD DABC
ABDC BADC CADB DACB
ACBD BCAD CBAD DBAC
ACDB BCDA CBDA DBCA
ADBC BDAC CDAB DCAB
ADCB BDCA CDBA DCBA

Aufgabe 5.6 Knuth-Morris-Pratt

- Ermitteln Sie die next-Tabelle des Knuth-Morris-Pratt-Algorithmus für das Suchmuster „ANANAS“.
- Denkaufgabe: Wie lässt sich der Algorithmus von Knuth, Morris und Pratt optimieren lässt, falls feststeht, dass das verwendete Alphabet binär ist, also z.B. $\Sigma = \{0, 1\}$.

6 Syntaxanalyse

Aufgabe 6.1 Pfadangaben als formale Grammatik

Versuchen Sie in EBNF die Syntax von Pfadangaben, wie sie üblicherweise in Unix- oder DOS-Kommandos (wählen Sie eine Alternative) verwendet werden, zu beschreiben.

Aufgabe 6.2 Stringdarstellung eines binären Baums

Gegeben ist folgende Syntax für die Stringdarstellung eines binären Baums in EBNF.

```
Node ::= '(' Char Node Node ')' | '*'
Char ::= 'A' | 'B' | 'C' | ... | 'Z'
```

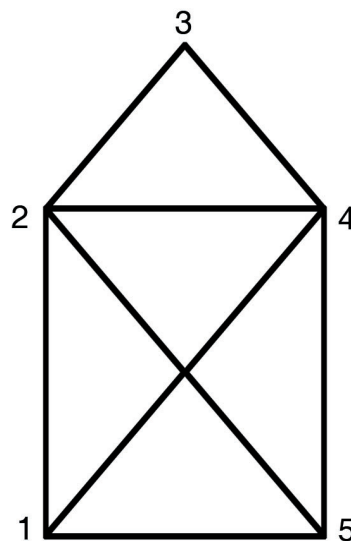
Überprüfen sie welcher der beiden folgenden Ausdrücke dieser Syntax entspricht und stellen Sie ihn als Baum dar. Korrigieren Sie den anderen Ausdruck so, dass er der Syntax entspricht.

- $R(I*(C(H**) (A**))*) (R(D**) (N*(I**) (X(O**) (N**))))$
- $(W(A(T**) (E**)) (R(G(A**) (T**)) (E**)))$

7 Backtracking

Aufgabe 7.1 Das Haus des Nikolaus

Das Haus des Nikolaus ist ein altes Zeichenspiel. Das Haus ist in unten stehender Abbildung dargestellt. Ziel des Spiels ist es das Haus in einem Zug aus acht Strecken zu zeichnen. Keine Strecke darf mehrmals durchlaufen werden.



- Versuchen Sie ausgehend von jeder der Ecken 1, 2 und 4 jeweils ein Haus zu zeichnen. Wenden Sie hierzu das Prinzip des Backtracking an und dokumentieren Sie Ihre einzelnen Schritte.
- Wie viele verschiedene Möglichkeiten gibt es das Haus zu zeichnen?

Aufgabe 7.2 Der Weg des Springers

Die folgenden Fragen beziehen sich auf das Problem *Der Weg des Springers* aus der Vorlesung und setzen ein normales, 8x8 Felder großes, Schachbrett voraus.

- Von welcher Ordnung ist die Laufzeit des Springerproblems?
- Gibt es eine Möglichkeit das Springerproblem effizienter als mit Hilfe von Backtracking zu lösen? Entwerfen Sie eine geeignete Heuristik.
- Versuchen Sie zu ermitteln wie viele verschiedene Wege des Springers existieren.

8 Entrekursivierung

Aufgabe 8.1 Berechnung der harmonischen Reihe

Entwerfen Sie eine linear rekursive Funktion (Java oder Pseudo-Code) zur Berechnung der harmonischen Reihe $\sum_{i=1}^n \frac{1}{i}$. Überführen Sie diese zunächst in eine endrekursive Funktion und beseitigen Sie anschließend in einem dritten Schritt die Rekursion völlig.

9 Komplexe Algorithmen

Aufgabe 9.1 Fibonacci-Zahlen

Die Fibonacci-Zahlen sind eine Folge natürlicher Zahlen, bei der jede Zahl die Summe der beiden vorangegangenen ist. Die Folge beginnt mit 1, 1, 2, 3, 5, 8, 13 und setzt sich so weiter fort. Entwerfen Sie einen *effizienten* Algorithmus zur Berechnung der Fibonacci-Zahlen und geben Sie ihn im Pseudocode an.

Aufgabe 9.2 Permutationen eines Wortes

Entwickeln Sie einen Algorithmus zur Berechnung aller Permutationen eines Wortes der Länge n . Geben Sie ihn entweder im Pseudo- oder Programmcode an.

Aufgabe 9.3 Ackermann-Funktion

Die Ackermann-Funktion ist folgendermaßen rekursiv definiert (m, n nichtnegativ, ganzzahlig):

$$A(m, n) = n + 1 \text{ für } m = 0$$

$$A(m, n) = A(m - 1, 1) \text{ für } m! = 0, n = 0$$

$$A(m, n) = A(m - 1, A(m, n - 1)) \text{ für } m! = 0, n! = 0$$

- Berechnen Sie $A(1, 5)$, $A(2, 3)$ und $A(3, 1)$. (Hinweis: Versuchen Sie es besser gar nicht erst per Hand ;-)
- Geben Sie eine rekursive Funktion zur Berechnung der Ackermann-Funktion an (z.B. im Pseudocode).

Aufgabe 9.4 Primzahltests

Entwerfen Sie eine Funktion `isPrimeNumber(int z)`, die testet, ob z eine Primzahl ist.

Aufgabe 9.5 Teilabschnittssummen

Wir betrachten endliche Folgen ganzer Zahlen. Ein Teilabschnitt einer solchen endliche Folge besteht aus einem zusammenhängenden Teil der Folge. Die Summe der Elemente eines solchen Teilabschnitts heißt Teilabschnittssumme. Ein Teilabschnitt, welcher bis zum rechten Rand einer vorgegebenen Folge reicht, wird rechter Randabschnitt genannt. Die größte Summe der rechten Randabschnitte wird rechtes Randmaximum genannt. Entsprechend werden linker Randabschnitt und linkes Randmaximum definiert. Beispiel:

$(+5, -8, +3, +3, -5, +7, -2, -7, +3, +5)$ ist eine endliche Folge.

Ein Teilabschnitt dieser Folge ist etwa $(+3, -5, +7, -2, -7)$.

Die entsprechende Teilabschnittssumme ist -4 .

Ein rechter Randabschnitt ist etwa $(-7, +3, +5)$.

Das rechte Randmaximum ist $+8$ und wird von dem rechten Randabschnitt $(+3, +5)$ angenommen.

- Entwickeln Sie unter Verwendung dieser Begriffe einen Divide-and-Conquer-Algorithmus zur Berechnung der maximalen Teilabschnittssumme einer endlichen Folge.
- Gibt es einen Algorithmus der effizienter Arbeitet als der Divide-and-Conquer-Algorithmus?
- Finden Sie einen effiziente Algorithmus zur Bestimmung der maximalen rechteckigen Teilabschnitte in einem 2-dimensionalen Feld.

Aufgabe 9.6 Analyse eines Algorithmus

Es seien a und b positive ganze Zahlen und Q eine wie folgt definierte rekursive Funktion:

$$Q(a, b) = 0 \text{ für } a < b$$

$$Q(a, b) = Q(a - b, b) + 1 \text{ für } a \geq b$$

- Berechnen Sie $Q(2, 3)$ und $Q(14, 3)$.

b) Was berechnet die Funktion Q ? Was ergibt folglich $Q(4712, 7)$?

Aufgabe 9.7 Inversionen eines Feldes

Es sei $a[]$ ein Feld von verschiedenen int-Zahlen. Falls $i < j$ und gleichzeitig $a[i] > a[j]$, so nennt man das Paar (i, j) eine Inversion von $a[]$.

- Wieviele Inversionen hat das Feld $2, 3, 8, 6, 1$? Wie lauten diese?
- Welches aus den Elementen der Menge $1, 2, 3, \dots, n$ gebildete Feld hat die meisten Inversionen? Wieviele sind das?

Aufgabe 9.8 Stable Marriage Problem

Entwerfen Sie einen Algorithmus der das stable marriage Problem löst.

Imagine you are a matchmaker, with one hundred female clients, and one hundred male clients. Each of the women has given you a complete list of the hundred men, ordered by her preference: her first choice, second choice, and so on. Each of the men has given you a list of the women, ranked similarly. It is your job to arrange one hundred happy marriages.

It should be immediately apparent that everyone is not guaranteed to get their first choice: if a particular man is the first choice of more than one woman, only one can be matched with him, and the other women will have to make do with less. Rather than guarantee the purest of happiness to everyone — a promise that almost surely would subject you to eventual litigation — your challenge is to make the marriages stable.

By this, we mean that once the matchmaker has arranged the marriages, there should be no man who says to another woman, “You know, I love you more than the woman I was matched with – let’s run away together!” where the woman agrees, because she loves the man more than her husband. In the spirit of equality, no woman should make such a successful proposal to a man: should she so propose, we want the man to respond, “Madam, I am flattered by your attention, but I am married to someone I love more than you, so I am not interested.”

Hinweis: Ein Algorithmus, der alle mögliche Paarbildungen berechnet, ist nicht zulässig, weil ineffizient.

Aufgabe 9.9 Türme von Hanoi

Das Spiel „Türme von Hanoi“ wurde vom französischen Mathematiker Edouard Lucas (1842 - 1891) erfunden. Gegeben sind drei Stäbe und n Scheiben unterschiedlicher Größe, welche sich auf die Stäbe stecken lassen. Anfangs stecken alle Scheiben der Größe nach geordnet (unten die größte, oben die kleinste) auf einem Pfahl. Ziel des Spiels ist es diesen Stapel, auf den rechten Pfahl zu übertragen. Dabei sind folgende Regeln zu beachten:

- Es darf jeweils nur eine Scheibe gezogen werden.
 - Eine Scheibe darf nur auf einem der drei Pfähle abgelegt werden.
 - Keine Scheibe darf (auch nicht zeitweise) auf einer kleineren liegen.
- Geben Sie Lösungen für 1, 2 und 3 Scheiben an, welche nur die jeweils minimale Anzahl an Zügen benötigen.
 - Wie sieht der Ablauf für allgemeine n aus?
 - Formulieren Sie eine Rekurrenzbeziehung für die rekursive Lösung des Problems. Lösen Sie die Beziehung auf.

Hinweis: Der Ablauf lässt sich sehr gut mit unterschiedlich großen Münzen simulieren.

Lucas verkaufte das Spiel erstmals im Jahre 1883 als Spielzeug. Um den Verkauf ein wenig anzukurbeln erfand er folgende Geschichte: Im Großen Tempel von Benares, unter dem Dom, der die Mitte der Welt markiert, ruht eine Messingplatte, in der drei Diamantnadeln befestigt sind, jede eine Elle hoch und so stark wie der Körper einer Biene. Bei der Erschaffung der Welt hat Gott vierundsechzig Scheiben aus purem Gold auf eine der Nadeln gesteckt, wobei die größte Scheibe auf der Messingplatte ruht, und die übrigen, immer kleiner werdend, eine auf der anderen. Das ist der Turm von Brahma. Tag und Nacht sind die Mönche unablässig damit beschäftigt, den festgeschriebenen und unveränderlichen Gesetzen von Brahma folgend, die Scheiben von einer Diamantnadel auf eine andere zu setzen, wobei sie nur jeweils eine Scheibe auf einmal umsetzen dürfen, und zwar so, dass sich nie eine kleinere Scheibe unter einer größeren befindet. Sobald dereinst alle vierundsechzig Scheiben von der Nadel, auf die Gott sie bei der Erschaffung der Welt gesetzt hat, auf eine der anderen Nadeln gebracht sein werden, werden der Turm samt dem Tempel und allen Brahmanen zu Staub zerfallen, und die Welt wird mit einem Donnerschlag untergehen. (nach Bachmann B., Müller R.)

- d) Wie viele Züge benötigen die Mönche mindestens zur Erfüllung Ihrer Aufgabe?
 e) Besteht die Gefahr eines baldigen Weltuntergangs, wenn die Mönche pro Sekunde 1.000 Züge bewältigen?

Aufgabe 9.10 Matrix-Multiplikation nach Strassen

Matrixmultiplikation nach Strassen (1969): Die übliche Methode, zwei 2×2 -Matrizen zu multiplizieren, benötigt 8 Multiplikationen und 4 Additionen. Man kann jedoch auch mit 7 Multiplikationen und 18 Additionen auskommen, wenn man beachtet, dass

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} s_1 + s_2 - s_4 + s_6 & s_4 + s_5 \\ s_6 + s_7 & s_2 - s_3 + s_5 - s_7 \end{bmatrix},$$

wobei

$$\begin{aligned} s_1 &= (b - d) \cdot (g + h), \\ s_2 &= (a + d) \cdot (e + h), \\ s_3 &= (a - c) \cdot (e + f), \\ s_4 &= h \cdot (a + b), \quad s_5 = a \cdot (f - h), \\ s_6 &= d \cdot (g - e), \quad s_7 = e \cdot (c + d). \end{aligned}$$

Es sei n eine Potenz von 2.

- a) Entwickeln Sie einen *Divide-and-Conquer*-Algorithmus zur Multiplikation zweier $n \times n$ -Matrizen. *Hinweis*: Wenden Sie den 2×2 -Algorithmus rekursiv an auf das Paar von $n \times n$ -Matrizen, indem beide partitioniert werden in jeweils 4 Untermatrizen der Dimension $\frac{n}{2} \times \frac{n}{2}$.
 b) Geben Sie die Ordnung der Anzahl der erforderlichen Multiplikationen und Additionen beim obigen Algorithmus an und vergleichen Sie dies mit dem klassischen Verfahren. *Hinweis*: Verwenden Sie das *Master-Theorem*.

10 Graphen

Aufgabe 10.1 Inzidenzmatrix

Die *Inzidenzmatrix* eines gerichteten Graphen G mit n Knoten und m Kanten ist eine $n \times m$ -Matrix C mit den Einträgen

$$c_{i,j} = \begin{cases} +1, & \text{falls } i \text{ Anfangsknoten der Kante } j \text{ ist;} \\ -1, & \text{falls } i \text{ Endknoten der Kante } j \text{ ist;} \\ 0, & \text{sonst.} \end{cases}$$

Hierfür sind die Kanten mit $1, \dots, m$ nummeriert. Für ungerichtete Graphen definiert man

$$c_{i,j} = \begin{cases} 1, & \text{falls } i \text{ auf der Kante } j \text{ liegt;} \\ 0, & \text{sonst.} \end{cases}$$

- a) Geben Sie die Inzidenzmatrizen für den folgenden Graphen.
 b) Schreiben Sie C++-Funktionen, um festzustellen, ob zwei gegebene Knoten benachbart sind. Unterscheiden Sie dabei die Fälle des gerichteten und des ungerichteten Graphen.

Aufgabe 10.2 Gerichteter Graph und Adjazenzmatrix

Erstellen Sie für untenstehenden gerichteten Graphen eine Adjazenzmatrix.

