Teilklausur I in Informatik

Es sind insgesamt 80 Punkte zu erreichen
(Jeder Punkt entspricht einer Bearbeitungszeit von etwa 1 Minute)

Aufgabe 1: Quersumme (24 Punkte) – DF, PAD/PN, C/C++
Aufgabe 2: Teilbarkeit durch 3 (6 Punkte) – EBNF
Aufgabe 3: Bruchrechnung (26 Punkte) – DH, PA, C/C++
Aufgabe 4: Quadrat- und Kubikzahlen (24 Punkte) – Struktogramm, C/C++

Bitte kennzeichnen Sie jedes Blatt lesbar mit Ihrem Namen und Ihrer Matrikelnummer

Zugelassene Hilfsmittel: Keine

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

Aufgabe 1 — 24 Punkte

(2+14+8 = 24 Punkte) Als **Quersumme** einer Zahl `n` bezeichnet man die Summe der Ziffern der Zahl `n`.

Entwerfen Sie eine Funktion `quer`, die als Eingangsparameter eine Zahl `n` (Datentyp `unsigned int`) erhält und in der ebenfalls zu übergebenen Variablen `erg` (Datentyp auch hier `unsigned int`) die Quersumme von `n` berechnet.

Beispiele: Quersumme von 19092008 ist 29 (Hinweis: wir betrachten hier nicht die iterierte Quersumme, die oben zum Ergebnis 2 führen würde (2+9=11 und schließlich 1+1=2)). Quersumme von 102936 ist 21.

Geben Sie das Datenflussdiagramm (DF) der Ebene 0 an (Funktion als ein Kästchen) (2 Punkte), sowie einen Programmablaufplan mit Daten (PAD) (wahlweise ist statt dem PAD auch ein Programmnetz (PN) erlaubt) für die von Ihnen entworfene Funktion mit den nötigen Verfeinerungen (14 Punkte). Der Entwurf soll so detailliert sein, dass eine Umsetzung 1:1 in ein C/C++-Programm möglich ist, d.h., dass alle Fallunterscheidungen und Schleifen auch im Entwurf als solche vorkommen müssen. Beachten Sie: die Parameter sind an die Funktion zu übergeben und werden von dieser zurückgeliefert, Sie müssen sich *nicht* um die Ein- und Ausgabe der Werte kümmern.

(8 Punkte) Setzen Sie anschließend Ihren Entwurf in eine C/C++-Funktion um (nur die Funktion `quer`). Das Hauptprogramm `int main()` muss nicht geschrieben werden, ebenso brauchen Sie sich nicht um das Einlesen der Zahl `n` oder die Ausgabe des Ergebnisses zu kümmern.

Aufgabe 2 — 6 Punkte

(6 Punkte) Mit **EBNF** lassen sich auch Zahlen mit bestimmten Eigenschaften erzeugen: Eine Zahl ist genau dann durch 3 teilbar, wenn die Quersumme durch 3 teilbar ist. Ist die erste Ziffer z.B. eine 4 und soll die ganze Zahl durch 3 teilbar sein, so muss die Summe der restlichen Ziffern durch 3 geteilt den Rest 2 ergeben (z.B. 468).

Geben Sie EBNF-Regeln an, die genau die Dezimalzahlen erzeugen, die durch 3 teilbar sind. (Wer dabei beachtet, dass außer der Zahl 0 keine Zahl führende Nullen hat, erhält 2 Zusatzpunkte.)

Aufgabe 3 — 26 Punkte

(2+14+10 = 26 Punkte) **Bruchrechnung** ist immer wieder ein Problem. Gegeben sei der Datentyp

```
struct Bruch{
  int zaehler;
  unsigned int nenner;
};
```

Entwerfen Sie eine Funktion `kleiner`, die als Eingangsparameter zwei Brüche `b1` und `b2` erhält und in der ebenfalls zu übergebenen `bool`-Variablen den Wert `true` liefert, wenn der Wert des Bruchs `b1` kleiner oder gleich dem Wert des Bruchs `b2` ist, und `false` sonst. Ist der `nenner` eines Bruchs gleich 0, so soll der Wert des Bruchs hier stets als 0 betrachtet werden (ist der `zaehler` 0, gilt dies sowieso).

Beispiele: ist `b1=(3,5)` und `b2=(2,3)`, so hat nach Aufruf von `kleiner(b1,b2,erg)` die Variable `erg` den Wert `true`; ist `b1=(3,5)` und `b2=(0,1)`, so hat nach Aufruf von `kleiner(b1,b2,erg)` die Variable `erg` den Wert `false`; ist `b1=(6,3)` und `b2=(2,1)`, so hat nach Aufruf von `kleiner(b1,b2,erg)` die Variable `erg` den Wert `true` (da 2=2); ist `b1=(7,0)` und `b2=(0,3)`, so hat nach Aufruf von `kleiner(b1,b2,erg)` die Variable `erg` den Wert `true` (da `b1` und `b2` als 0 interpretiert werden).

Geben Sie das Datenhierarchiediagramm (DH) für den Datentyp `Bruch` an (2 Punkte), sowie einen Programmablaufplan (PA) für die von Ihnen entworfene Funktion mit den nötigen Verfeinerungen (14 Punkte). Der Entwurf soll so detailliert sein, dass eine Umsetzung 1:1 in ein C/C++-Programm möglich ist, d.h., dass alle Fallunterscheidungen und Schleifen auch im Entwurf als solche vorkommen müssen. Beachten Sie: die Parameter sind an die Funktion zu übergeben und werden von dieser zurückgeliefert, Sie müssen sich *nicht* um die Ein- und Ausgabe der Werte kümmern.

(10 Punkte) Setzen Sie anschließend Ihren Entwurf in eine C/C++-Funktion um (nur die Funktion `kleiner`). Das Hauptprogramm `int main()` muss nicht geschrieben werden, ebenso brauchen Sie sich nicht um das Einlesen der Brüche oder die Ausgabe des Ergebnisses zu kümmern.

Aufgabe 4 — 24 Punkte

(14+10 = 24 Punkte) **Quadrat- und Kubikzahlen:** Entwerfen Sie eine Funktion `aufvier`, die zu einer gegebenen Zahl `n` alle positiven ganzen Zahlen `<=n` in aufsteigender Reihenfolge ausgibt, die entweder Quadratzahl oder Kubikzahl sind. Ist eine Zahl sowohl Quadrat- als auch Kubikzahl (z.B. ist $64 = 8^2 = 4^3$), so soll diese nur einmal ausgegeben werden. Der Aufruf `aufvier(120)` soll so z.B. die Zahlen `1 4 8 9 16 25 27 36 49 64 81 100` in dieser Reihenfolge ausgeben.

(14 Punkte) Geben Sie ein Struktogramm für die von Ihnen entworfene Funktion an. Der Entwurf soll so detailliert sein, dass eine Umsetzung 1:1 in ein C/C++-Programm möglich ist, d.h., dass alle Fallunterscheidungen und Schleifen auch im Entwurf als solche vorkommen müssen. Beachten Sie: der Parameter `n` wird an die Funktion übergeben, Sie müssen sich *nicht* um die Eingabe des Wertes `n` kümmern.

(10 Punkte) Setzen Sie danach Ihren Entwurf in C/C++ um (nur die Funktion `aufvier`, das Hauptprogramm `int main()` muss nicht geschrieben werden).

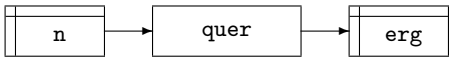
Hinweis: Es ist hier kein Sortieralgorithmus gefordert. Sie benötigen in dieser Aufgabe keine Arrays. Es reichen neben dem Eingangsparameter `n` zwei weitere Variablen aus (Sie dürfen aber auch mehr Variablen verwenden).

Teilklausur I in Informatik – Lösungshinweise

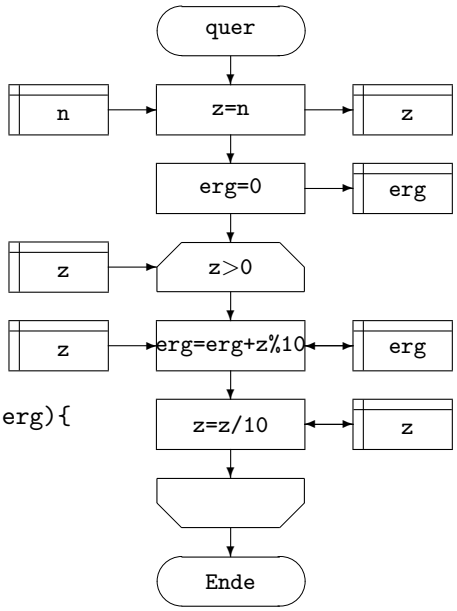
Aufgabe 1 — Quersumme

Der Zerlegen der Zahl in die Einzelstellen kann durch wiederholtes Modulo-10-Rechnen und Durch-10-Dividieren erledigt werden. Die Ziffern werden dann einfach aufaddiert.

DF:



PAD:




```
void quer(const unsigned int n, unsigned int & erg){
  unsigned int z=n;  erg=0;
  while (z>0) {
    erg=erg+z%10;
    z=z/10;
  }
}
```

Aufgabe 2 — Teilbarkeit durch 3

Man muss sich über die Variablen merken, ob die bisher erzeugten Ziffern durch 3 geteilt den Rest 0, 1 oder 2 ergeben. Bei Rest 0 kann auch abgebrochen werden.

```
<S> ::= 0 | 3 | 6 | 9 | 0<S> | 3<S> | 6<S> | 9<S> | 1<E> | 4<E> | 7<E> | 2<Z> | 5<Z> | 8<Z>
<E> ::= 2 | 5 | 8 | 0<E> | 3<E> | 6<E> | 9<E> | 1<Z> | 4<Z> | 7<Z> | 2<S> | 5<S> | 8<S>
<Z> ::= 1 | 4 | 7 | 0<Z> | 3<Z> | 6<Z> | 9<Z> | 1<S> | 4<S> | 7<S> | 2<E> | 5<E> | 8<E>
```

Um führende Nullen zu vermeiden, kann man z.B. die Regeln bei Rest 0 verdoppeln und die verbotene Regel `0<S>` streichen.

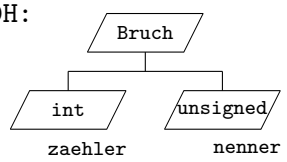
```
<S> ::= 0 | 3 | 6 | 9 | 3<N> | 6<N> | 9<N> | 1<E> | 4<E> | 7<E> | 2<Z> | 5<Z> | 8<Z>
<N> ::= 0 | 3 | 6 | 9 | 0<N> | 3<N> | 6<N> | 9<N> | 1<E> | 4<E> | 7<E> | 2<Z> | 5<Z> | 8<Z>
<E> ::= 2 | 5 | 8 | 0<E> | 3<E> | 6<E> | 9<E> | 1<Z> | 4<Z> | 7<Z> | 2<N> | 5<N> | 8<N>
<Z> ::= 1 | 4 | 7 | 0<Z> | 3<Z> | 6<Z> | 9<Z> | 1<N> | 4<N> | 7<N> | 2<E> | 5<E> | 8<E>
```

Aufgabe 3 — Bruchrechnung

Um die Sonderfälle einfach zu behandeln, wandelt man z.B. Zahlen der Form `(x,0)` in `(0,1)` um oder – wie hier angegeben – behandelt sie direkt als Zahl `(0,1)`. Es gilt dann `(a,b)<=(c,d)` genau dann, wenn `a*d<=b*c`. Dies umgeht gleichzeitig auch die Problematik der Ganzzahldivision.

Bei der Umsetzung würde man am Rechner dann noch merken, dass C sich (u.U. auch abhängig vom Compiler) bei der Multiplikation eines `int` mit einem `unsigned int` für die vorzeichenlose Multiplikation entscheidet (`(-3)*4` wird dann zu `4294967284`).

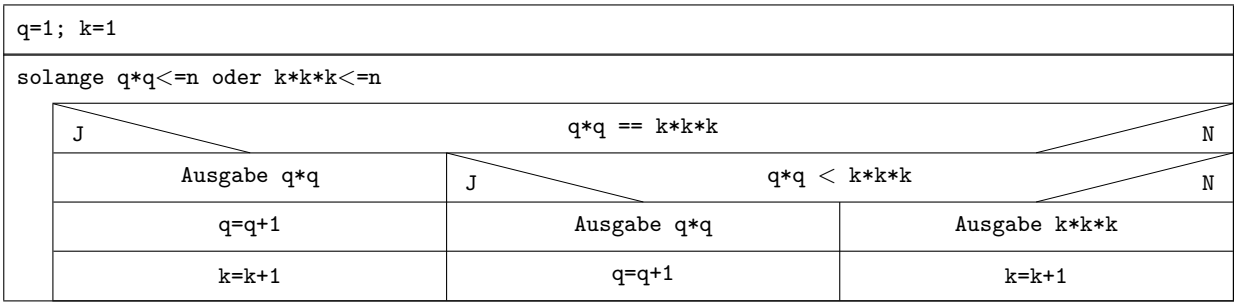
DH:



PA:


```
void kleiner(const Bruch b1, const Bruch b2, bool & erg){
  if (b1.nenner==0) {
    if (b2.nenner==0) erg=true;
    else { if (0<=b2.zaehler) erg=true; else erg=false; }
  } else {
    if (b2.nenner==0) { if (b1.zaehler<=0) erg=true; else erg=false; }
    else {
      if (b1.zaehler*(int)b2.nenner<=b2.zaehler*(int)b1.nenner)
        erg=true; else erg=false; }
  }
}
```

Aufgabe 4 — Quadrat- und Kubikzahlen



Da Quadrat- und Kubikzahlen streng monoton wachsen, reichen zwei Variablen `q` und `k`, um sich zu merken, welches die nächste auszugebene Quadratzahl (`q*q`) bzw. Kubikzahl (`k*k*k`) ist. Sind die Zahlen gleich, so erhöhen sich `q` und `k`, ansonsten nur eine der beiden. Dies wiederholt man solange wie die nächste Quadrat- oder Kubikzahl noch unterhalb der Obergrenze liegt. Wer etwas Rechenzeit sparen will, kann sich in weiteren Variablen `q2=q*q` und `k3=k*k*k` die betreffenden Werte ablegen, statt diese mehrfach zu berechnen.

```
void aufvier(const unsigned int n){
  unsigned int q=1, k=1;
  while (q*q<=n || k*k*k<=n) {
    if (q*q==k*k*k) {
      cout << q*q << ' '; q=q+1; k=k+1;
    } else if (q*q<k*k*k) {
      cout << q*q << ' '; q=q+1;
    } else {
      cout << k*k*k << ' '; k=k+1;
    }
  }
}
```

Teilklausur II in Informatik

Es sind insgesamt 80 Punkte zu erreichen
(Jeder Punkt entspricht einer Bearbeitungszeit von etwa 1 Minute)

Aufgabe 1: Binäre Suchbäume (12 Punkte) – Eigenschaften, Einfügen, Pre/In/Postorder
Aufgabe 2: Sortieren (18 Punkte) – Mergesort + Quicksort
Aufgabe 3: Dijkstra-Algorithmus (11 Punkte) – Durchführen, Alternativen
Aufgabe 4: Programm-Analyse (18 Punkte) – Durchführen, begründen, O-Notation
Aufgabe 5: Programmierung (21 Punkte) – Programmieren, O-Notation

Bitte kennzeichnen Sie jedes Blatt lesbar mit Ihrem Namen und Ihrer Matrikelnummer

Bearbeiten Sie die Aufgaben – soweit möglich – auf den Aufgabenblättern

Zugelassene Hilfsmittel: Keine

Lesen Sie die Aufgaben in Ruhe durch!
Viel Erfolg!

Aufgabe 1 — 12 Punkte

(4+4+4=12 Punkte) **Binäre Suchbäume:** Die Zahlen x, 7, 9, 5, 21, 42 und y sollen in dieser Reihenfolge in einen anfangs leeren binären Suchbaum eingefügt werden. Die Werte x und y sind dabei Platzhalter mit folgenden Eigenschaften: der resultierende binäre Suchbaum hat minimale Höhe und alle Werte im Suchbaum sind verschieden.

- (4 Punkte) Geben Sie an, welche Werte die Variablen x und y unter Annahme dieser Eigenschaften haben können.
- (4 Punkte) Wählen Sie mögliche Werte für x und y, fügen Sie die Zahlen in obiger Reihenfolge in einen anfangs leeren binären Suchbaum ein. Es reicht die Angabe des Suchbaums, Sie müssen nicht das Verfahren des Einfügens beschreiben. (Falls Sie die erste Teilaufgabe nicht lösen konnten, wählen Sie x=8 und y=13; der resultierende Suchbaum muss in diesem Fall nicht minimale Höhe haben.)
- (4 Punkte) Geben Sie die Zahlen des so entstandenen binären Suchbaums in Preorder, Inorder und Postorder an.

Aufgabe 2 — 18 Punkte

(8+10 = 18 Punkte) **Sortieren:**

- (8 Punkte) Führen Sie den Mergesort-Algorithmus mit folgenden Zahlen durch.

  52 18 42 37 24 86 13 23

  Machen Sie in der Bearbeitung deutlich, in welcher Reihenfolge die rekursiven Aufrufe beim Sortieren stattfinden (z.B., indem Sie den jeweils nächsten rekursiven Aufruf und jeweils das Ergebnis des Mischens zweier sortierter Teilfelder in eine eigene Zeile schreiben – oder numerieren Sie die einzelnen Schritte soweit nötig durch).

- (10 Punkte) Führen Sie den Quicksort-Algorithmus mit folgenden Zahlen durch.

  52 18 42 37 24 86 13 23

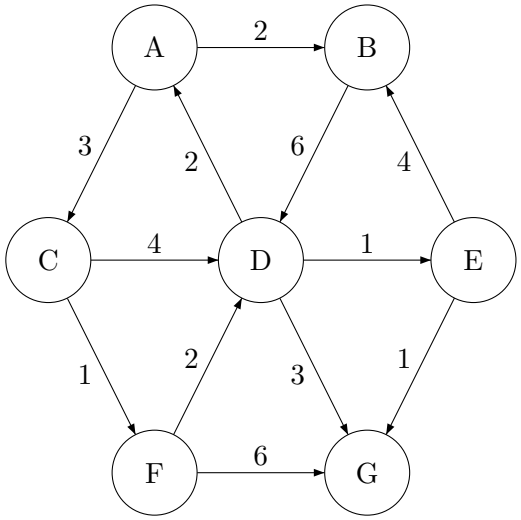
  Es reicht, wenn Sie dabei die Quicksortschritte (Aufteilen des (Teil-)Felds in Elemente kleiner gleich und größer gleich dem Pivot-Element) der ersten 4 rekursiven Aufrufe durchführen (der erste rekursive Aufruf ist dabei `Quicksort(0,7)`). Wählen Sie als Pivot-Element dabei stets das mittlere des zu sortierenden Teilfeldes.

  Aus Ihrer Bearbeitung muss hervorgehen, welche Zahlen bei dem Prozess vertauscht wurden und wie diese zu vertauschenden Zahlen gefunden wurden. Ebenso muss ersichtlich sein, wie die Parameter der rekursiven Aufrufe ermittelt wurden.

Aufgabe 3 — 11 Punkte

(2+9=11 Punkte) **Kürzeste-Wege-Berechnung mit dem Algorithmus von Dijkstra:**

(9 Punkte) Führen Sie den Dijkstra-Algorithmus auf folgendem Graphen aus:
Zu Beginn ist lediglich die Entfernung zum Startknoten F mit d(F)=0 bekannt. Im Dijkstra-Algorithmus wird in jeder Iteration ein Knoten aus der Randmenge R entfernt. Geben Sie den Zustand der Menge R jeweils vor diesem Entfernen eines Knotens an. Geben Sie die berechnete Entfernung d() für alle Knoten an (die Vorgänger-Knoten pred() müssen nicht mit berechnet werden).
(2 Punkte) Ist die Abarbeitungsreihenfolge der Knoten eindeutig? Falls ja, begründen Sie dies; falls nein, geben Sie an, an welchen Stellen Sie auch einen anderen Knoten aus der Menge R hätten wählen können.



Aufgabe 4 — 18 Punkte

(15+3 = 18 Punkte) **Analyse eines Listenalgorithmus:** Sie haben in einem Buch folgenden Listenalgorithmus gefunden:

```
struct Liste{
  int value;
  Liste* next;
};

Liste* miste(Liste *eins, Liste *zwei){
  if (eins==zwei) return eins;
  if (eins==0) return zwei; // Fall beide 0 oder eins 0 abgehakt
  if (zwei==0) return eins; // Fall zwei 0 auch abgehakt
  if (eins->value<zwei->value) {
    eins->next = miste(eins->next,zwei);
    return eins;
  } else {
    zwei->next = miste(eins,zwei->next);
    return zwei;
  }
}

void putliste(Liste* p){
  while (p) { cout << p->value << ' '; p=p->next; }
  cout << endl;
}
```

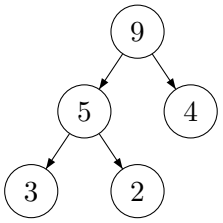
Das Buch verrät jedoch nicht, was der Algorithmus `miste` tut.

- (15 Punkte) Gegeben sind die Zeiger `p1` auf die Liste mit den Elementen (5,12,21,34) und `p2` auf die Liste mit den Elementen (17,42,47,53). Stellen Sie die Listen graphisch dar (jedes Element einer Liste als Kästchen mit dem `next`-Zeiger als Pfeil auf das entsprechende nächste Kästchen) und führen Sie den Algorithmus `miste` mit den Parametern `p1` und `p2` durch (Aufruf `p3=miste(p1,p2);`). Welche Zeiger werden verändert, welche Elemente neu erzeugt? Welche Ausgabe ergibt nach diesem Aufruf die Befehlsfolge `putliste(p1); putliste(p2); putliste(p3);`?
- (3 Punkte) Geben Sie die Laufzeit des Algorithmus in O-Notation abhängig von der Anzahl n_1 der Elemente der ersten Liste und der Anzahl n_2 der Elemente der zweiten Liste an.
- Zusatzaufgabe (knifflig, 3 Punkte): was kann passieren, wenn die Abfrage `eins==zwei` entfällt?

Aufgabe 5 — 21 Punkte

(3+18 = 21 Punkte) **Programmierung:** Wir schreiben in dieser Aufgabe eine Ausgaberoutine für Binärbäume, deren Knoten mit natürlichen Zahlen beschriftet sind.

Ziel ist, für einen Binärbaum eine Ausgabe der Form a+b+c+d+...+x+y zu erhalten. Die Ausgabe für nebenstehenden Baum soll also z.B. 3+5+2+9+4 lauten (eine Lösungsmöglichkeit führt auf diese Reihenfolge der Zahlen, Ihre Lösung muss die Zahlen nicht unbedingt in dieser Reihenfolge auflisten – eine weitere naheliegende Lösung führt auf die Ausgabe 9+5+3+2+4)



Die geforderte Ausgabe lässt sich leicht rekursiv erzeugen: Ist der Baum leer, so ist nichts auszugeben, besteht der Baum nur aus einem Knoten, so ist nur dieser auszugeben. Überlegen Sie sich, welche Ausgabe nötig ist,

- wenn beide Unterbäume nicht leer sind,
- wenn genau ein Unterbaum nicht leer ist.

- (3 Punkte) Geben Sie eine C/C++-Datenstruktur `BinBaum` für einen Binärbaum mit "`unsigned int`"-Werten an. (18 Punkte) Schreiben Sie eine (rekursive) Funktion `ausgabe`, die einen Zeiger auf die Wurzel eines Binärbaums übergeben bekommt und die Zahlen im Binärbaum durch Plus-Zeichen getrennt ausgibt. Nach der letzten Zahl darf kein Pluszeichen folgen!
- Zusatzaufgabe (3 Punkte): Schreiben Sie Ihre Funktion so, dass die Ausgabe die Form a+b+c+d+...+x+y=z hat. In obigen Beispiel soll die Ausgabe also 3+5+2+9+4=23 lauten. (Hinweis: die Berechnung der Summe der Zahlen lässt sich leicht in die Funktion integrieren. Die Ausgabe des Gleichheitszeichens und der Summe kann außerhalb der Funktion mit einem weiteren Befehl erfolgen.)
- Zusatzaufgabe (3 Punkte): Geben Sie die Laufzeit Ihres Programms in O-Notation in Abhängigkeit der Anzahl `n` der Elemente im Binärbaum an.

Hinweis: es ist nur C/C++-Code für die Datenstruktur und die `ausgabe`-Funktion gefordert. Sie müssen sich nicht um die Eingabe und das Einfügen der Werte im Binärbaum kümmern und auch kein Hauptprogramm (`int main()`) schreiben.

Teilklausur II in Informatik – Lösungshinweise

Aufgabe 1 — Binäre Suchbäume

Damit die Höhe 3 erreichen kann, müssen 3 Werte kleiner als x werden. Wären dies 7, 5 und y, so erhält man mit dem rechten Unterbaum mit 9, 21 und 42 die Höhe 4. Somit gilt $5 < 7 < 9 < x$ und – um die Höhe 3 zu erreichen – $x < y < 21 < 42$. Die Werte x und y müssen somit der Ungleichung $9 < x < y < 21$ genügen. Es wird dann stets x die Wurzel, 7 die Wurzel des linken Unterbaums (mit linkem Kind 5 und rechtem Kind 9) und 21 die Wurzel des rechten Unterbaums (mit linkem Kind y und rechtem Kind 42) – mit z.B. $x = 12$ und $y = 18$.

Preorder ist stets x 7 5 9 21 y 42, Inorder stets 5 7 9 x y 21 42, Postorder stets 5 9 7 y 42 21 x.

Mit $x = 8$ und $y = 13$ ergibt sich 8 als Wurzel, 7 als Wurzel des linken Unterbaums (mit 5 als linkem und einzigem Kind), 9 als Wurzel des rechten Unterbaums (mit 21 als rechtem und einzigem Kind, das 13 wiederum als linkes und 42 als rechtes Kind hat). Preorder ist dann 8 7 5 9 21 13 42, Inorder 5 7 8 9 13 21 42, Postorder 5 7 13 42 21 9 8.

Aufgabe 2 — Sortieren

Mergesort: jeder rekursive Aufruf und jedes Mischen steht in einer separaten Zeile:

```
52 18 42 37 24 86 13 23
52 18 42 37
52 18
52
   18
18 52
      42 37
      42
         37
      37 42
18 37 42 52
               24 86 13 23
               24 86
               24
                  86
               24 86
                     13 23
                     13
                        23
                     13 23
               13 23 24 86
13 18 23 24 37 42 52 86
```

Quicksort: Angabe des Aufrufs, Pivot-Elements, der Tauschvorgänge und rekursiven Folgeaufrufe (Parameter sind jeweils linker Rand bis rechter Index und linker Index bis rechter Rand):

`Quicksort(0,7): Pivot=37, Tausche(52,23), Tausche(42,13), Tausche(37,24), Folgeaufrufe Quicksort(0,3) (und später Quicksort(4,7))`

`Quicksort(0,3): (23,18,13,24), Pivot=18, Tausche(23,13), Tausche(18,18), Folgeaufrufe Quicksort(0,0) (und später Quicksort(2,3))`

`Quicksort(0,0): tut nichts`

`Quicksort(2,3): (23,24), Pivot=23, Tausche(23,23), Folgeaufrufe (hier nicht mehr gefordert) wären Quicksort(2,1) und Quicksort(3,3)`

Aufgabe 3 — Dijkstra-Algorithmus

Wahlmöglichkeit besteht prinzipiell genau dann, wenn in R mehrere Knoten denselben Wert d() haben.

Ablauf mit Startknoten F: Initialisiert wird mit d(F)=0 und R={F}.
F wird als kleinster Wert wieder entfernt. Die Nachfolger von F sind D und G, die Werte werden auf d(D)=2 und d(G)=6 gesetzt, R={D,G}.
D hat den kleinsten Wert, von dort sind mit einer Kante A, E und G erreichbar, der Wert d(G) verringert sich, A und E kommen neu zu R hinzu (d(A)=4, d(E)=3, d(G)=5, R={A,E,G}).
E hat den kleinsten Wert, von dort sind mit einer Kante B und G erreichbar, der Wert d(G) verringert sich nochmals, B kommt neu zu R hinzu (d(A)=4, d(B)=7, d(G)=4, R={A,B,G}).
A und G haben den kleinsten Wert (hier besteht freie Wahl, wir nehmen zunächst G), von dort sind

keine weiteren Knoten erreichbar (d(A)=4, d(B)=7, R={A,B}).
A hat den kleinsten Wert, von dort sind mit einer Kante B und C erreichbar, der Wert d(B) verringert sich, C kommt neu zu R hinzu (d(B)=6, d(C)=7, R={B,C}).
Hätte man A zuerst gewählt, so wäre R jetzt R={B,C,G} und G würde nun gewählt werden müssen.
B hat den kleinsten Wert, von dort ist mit einer Kante nur D erreichbar, D hat aber bereits einen kleineren Wert (d(C)=7, R={C}).
C hat den kleinsten Wert, von dort sind mit einer Kante D und F erreichbar, deren Werte aber bereits kleiner sind.
R ist leer, der Algorithmus terminiert. Die kürzesten Entfernungen sind d=(4,6,7,2,3,0,4).

Aufgabe 4 — Programm-Analyse

Der Algorithmus biegt lediglich Zeiger um, erzeugt jedoch keine neuen Elemente. Der Next-Zeiger eines Listenelements wird jeweils auf das nächstkleinste Listenelement der beiden Listen umgebogen. Die ursprünglichen Listen bleiben nicht erhalten, es entsteht eine größere sortierte Liste (5,12,17,21,34,42,47,53), wobei der Zeiger `p1` auf die 5 zeigt, der Zeiger `p2` (ebenfalls wie vorher) auf die 17, der Zeiger `p3` zeigt auf das kleinste Element (die 5). Da die alten Listen nicht erhalten bleiben führt der Aufruf `putliste(p1)` zur Ausgabe 5 12 17 21 34 42 47 53, der Aufruf `putliste(p2)` zur Ausgabe 17 21 34 42 47 53 und der Aufruf `putliste(p3)` zur Ausgabe 5 12 17 21 34 42 47 53.

Jedes Element der Liste wird konstant oft berührt, die Laufzeit beträgt somit $O(n_1 + n_2)$.

Die Abfrage `eins==zwei` stellt sicher, dass hier zwei verschiedene Listen vorliegen. Der Aufruf `p3=miste(p1,p1)` würde sonst zu Zeigeroperationen führen, die zumindest am Ende der Liste den Null-Zeiger auf ein anderes Listen-Element umbiegen (je nach Belegung der Liste kann auch noch größerer Wirrwarr entstehen).

Aufgabe 5 — Programmierung

Sind beide Unterbäume nicht leer, so erzeuge erst die Ausgabe des linken Unterbaums (hat ebenfalls die Form a+b+...+x+y), drucke dann ein +, den Wert der Wurzel, ein weiteres + und dann die Ausgabe des rechten Unterbaums. Ist ein Unterbaum leer, so entfällt das entsprechende + und die Ausgabe für den fehlenden Unterbaum.

```
struct BinBaum {
  unsigned int value;
  BinBaum * left, * right;
};

int sum(BinBaum * root) {
  if (!root) { return 0; }
  else {
    int erg=root->value;
    if (root->left) {
      erg+=sum(root->left); cout << '+';
    }
    cout << root->value;
    if (root->right) {
      cout << '+'; erg+=sum(root->right);
    }
    return erg;
  }
}
```

Um die Summe noch korrekt auszugeben, ist nach dem Aufruf `erg=sum(root);` lediglich noch mit `cout << '=' << erg << endl;` das Gleichheitszeichen und die berechnete Summe auszugeben.
Ohne Berechnung der Summe entfallen lediglich alle `return` und `erg+=`.
Auch hier wird jedes Element im Binärbaum genau einmal bearbeitet, die Laufzeit ist somit $O(\mathtt{n})$.