Algorithmen und Datenstrukturen 1

Ausgearbeitetes Übungsblatt 2

Datum: 15. April 2005
Erstellt mit LaTeX

Aufgabe 2.1

Aufgabenstellung:

(a) Schreiben Sie einen Algorithmus in Pseudocode, der für ein gegebenes Array A feststellt, ob dieses einen gültigen *Maximum-Heap* enthält (d.h. das größte Element soll in $A[1]$ stehen usw.). Der Inhalt des Arrays darf dabei natürlich nicht verändert werden. Analysieren Sie die Laufzeit Ihres Algorithmus in Θ-Notation.

(b) Welche der nachstehenden Zahlenfolgen erfüllt die Heapeigenschaft? Zeichnen Sie zu jeder der Zahlenfolgen den entsprechenden binären Baum, um Ihre Antwort zu begründen.

- $A = \langle 23, 19, 19, 13, 13, 13, 17, 17, 11, 11, 11 \rangle$
- $B = \langle 22, 20, 21, 16, 17, 18, 19, 10, 11, 12, 13 \rangle$
- $C = \langle 5, 3, 4, 4, -1, 3, -2, 1, -1, 0, -2, -3 \rangle$

Lösung:

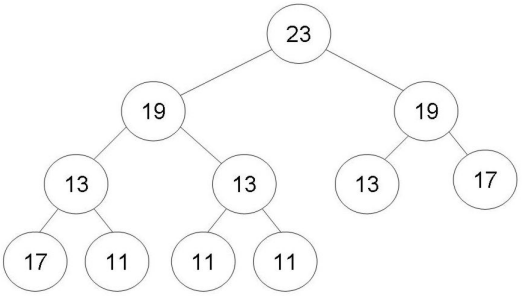
(a) Ein Maximum-Heap liegt dann vor, wenn für alle $i \in 2, 3, ..., n$ gilt: $k_i \leq k_{\lfloor \frac{i}{2} \rfloor}$. Diese Eigenschaft kann 1:1 in einen Algorithmus übernommen werden:

```
IstHeap(A,n) {
    if(n < 2) return true;
    for(i=2; i<=n; i++) {
        if(A[i] > A[floor(i/2)] return false;
    }
    return true;
}
```

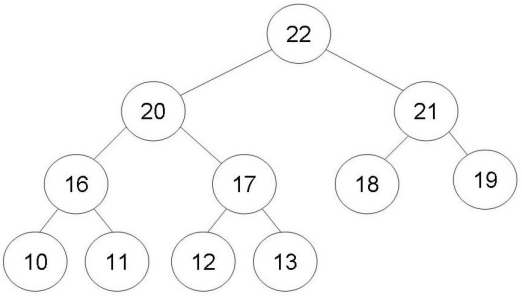
In diesem Algorithmus ist A das Feld, das auf seine Maximum-Heap-Eigenschaft hin überprüft werden soll, und n ist die Anzahl der Elemente dieses Feldes. Im Falle des Zutreffens der Maximum-Heap-Eigenschaft wird logisch wahr (true) zurückgegeben, ansonsten logisch falsch (false). Die Methode **floor** gibt die größte ganze Zahl kleiner oder gleich dem angegebenen Argument zurück; A[floor(i/2)] entspricht also $\lfloor \frac{i}{2} \rfloor$.

Die Laufzeit dieses Algorithmus ist auf Grund der for-Schleife linear, also in $\Theta(n)$.

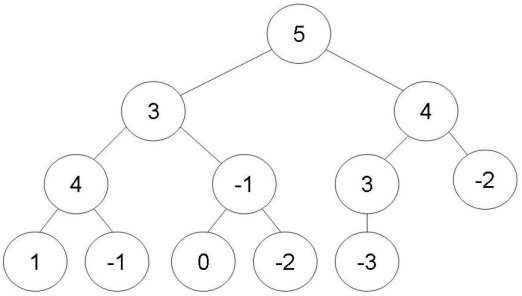
(b)
- A ist kein Maximum-Heap, weil $a_8 > a_4$ ist.



- B ist ein Maximum-Heap.



- C ist kein Maximum-Heap, weil $c_4 > c2$ ist.



Aufgabe 2.2

Aufgabenstellung:

(a) Gegeben ist die Zahlenfolge

$$A = \langle 16, 33, 18, 9, 17, 12, 43, 20, 21, 15, 6 \rangle .$$

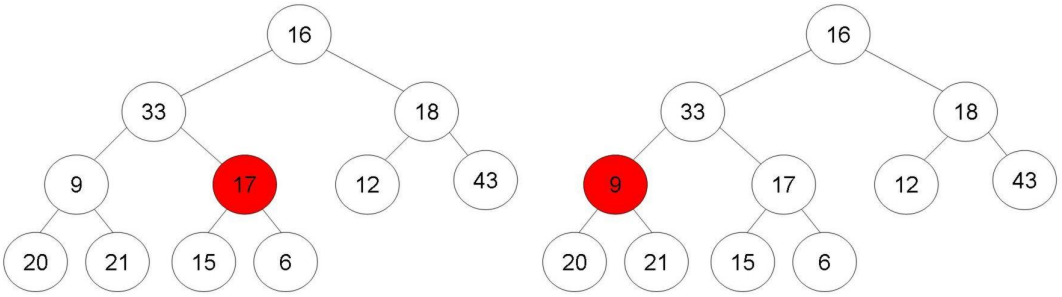
Erstellen Sie mittels des in der Vorlesung vorgestellten Verfahrens aus dieser Folge einen Heap. Zeichnen Sie dabei den Baum nach jeder Operation vom Typ *Versickere* auf.

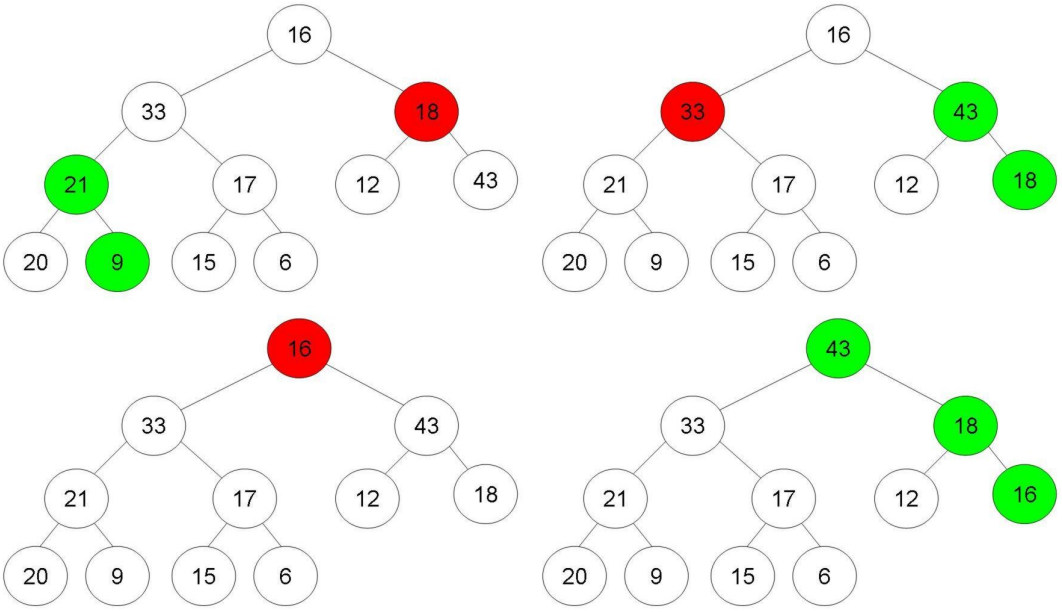
(b) Benutzen Sie das Verfahren Heapsort, um die Folge A zu sortieren, wobei Sie mit dem Heap aus (a) beginnen. Zeichnen Sie wiederum den Baum nach jeder Operation vom Typ *Versickere*.

Lösung:

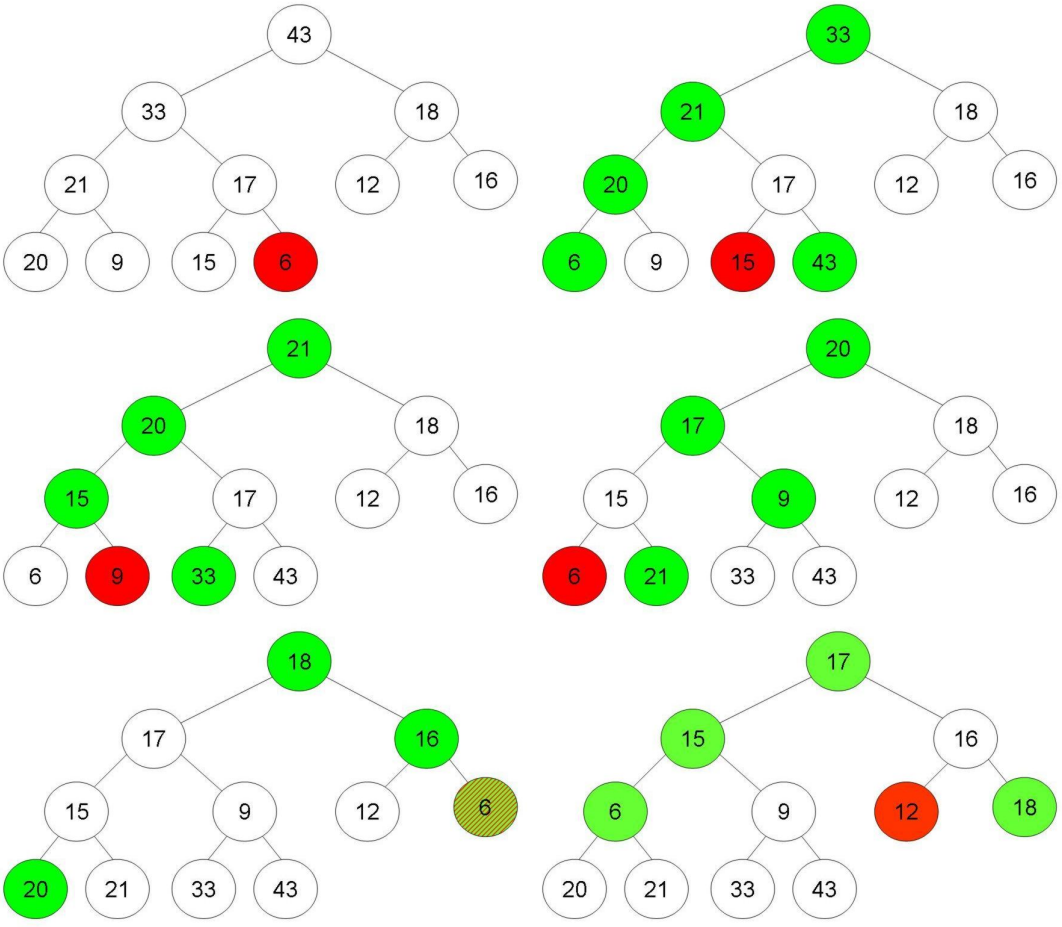
Grün dargestellt sind immer jene Knoten, deren Position sich verändert hat; rot dargestellt ist immer jener Knoten, dessen Position sich im Rahmen der folgenden Operation verändern kann.

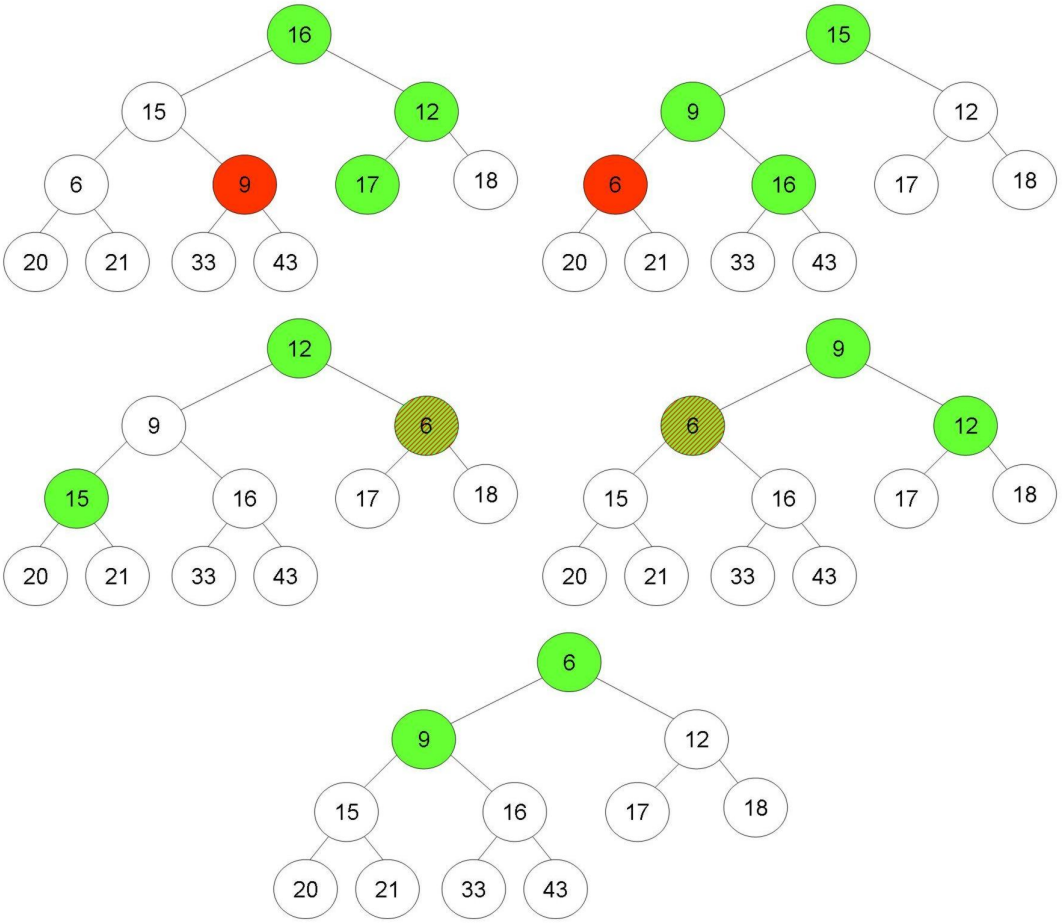
(a)




16 33 18 21 17 12 43 20 9 15 6
16 33 43 21 17 12 18 20 9 15 6
16 33 43 21 17 12 18 20 9 15 6
43 33 18 21 17 12 16 20 9 15 6

(b)


43 33 18 21 17 12 16 20 9 15 6
33 21 18 20 17 12 16 6 9 15 43
21 20 18 15 17 12 16 6 9 33 43
20 17 18 15 9 12 16 6 21 33 43
18 17 16 15 9 12 6 20 21 33 43
17 15 16 6 9 12 18 20 21 33 43



Aufgabe 2.3

Aufgabenstellung:

Gegeben sind folgende oktale Zahlen (d.h. Zahlen zur Basis 8):

$$\langle 2350, 1250, 2742, 2222, 1700, 1240, 56, 317 \rangle$$

Diese Zahlen sollen mit Hilfe des Verfahrens *Sortieren durch Fachverteilung* aufsteigend sortiert werden. Die Anzahl der Fächer entspricht der Basis 8, die Anzahl der Sammel- und Verteilungsphasen der maximalen Stellenanzahl. Geben Sie die Inhalte aller Fächer nach jeder Verteilungsphase und die Sortierung der Zahlen nach jeder Sammelphase an.

Lösung:

Zahlen: $\langle 2350, 1250, 2742, 2222, 1700, 1240, 56, 317 \rangle$

Erster Schritt:

Fach	**Datensätze**
0	2350, 1250, 1700, 1240
1	-
2	2742, 2222
3	-
4	-
5	-
6	56
7	317

Zahlen: $\langle 2350, 1250, 1700, 1240, 2742, 2222, 56, 317 \rangle$

Zweiter Schritt:

Fach	Datensätze
0	1700
1	217
2	2222
3	-
4	1240, 2742
5	2350, 1250, 56
6	-
7	-

Zahlen: $\langle 1700, 317, 2222, 1240, 2742, 2350, 1250, 56 \rangle$

Dritter Schritt:

Fach	Datensätze
0	56
1	-
2	2222, 1240, 1250
3	317, 2350
4	-
5	-
6	-
7	1700, 2742

Zahlen: $\langle 56, 2222, 1240, 1250, 317, 2350, 1700, 2742 \rangle$

Vierter Schritt:

Fach	Datensätze
0	56, 317
1	1240, 1250, 1700
2	2222, 2350, 2742
3	-
4	-
5	-
6	-
7	-

Sortierte Folge: $\langle 56, 317, 1240, 1250, 1700, 2222, 2350, 2742 \rangle$

Aufgabe 2.4

Aufgabenstellung:

Betrachten Sie den folgenden Algorithmus:

Eingabe: Array A von n positiven ganzen Zahlen
Ausgabe: Array B von n positiven ganzen Zahlen

```
1  k = 0;
2  für i = 1, . . . , n {
3    falls A[i] > k dann {
4      k = A[i];
5    }
6  }
7  Sei C ein neues Feld der Länge k;
8  für i = 1, . . . , k {
9    C[i] = 0;
```

```
10 }
11 für i = 1, . . . , n {
12   C[A[i]] = C[A[i]] + 1;
13 }
14 für i = 2, . . . , k {
15   C[i] = C[i] + C[i − 1];
16 }
17 für j = n, . . . , 1 {
18   B[C[A[j]]] = A[j];
19   C[A[j]] = C[A[j]] − 1;
20 }
```

Falls Sie den Algorithmus für fehlerhaft halten, schlagen Sie vor, durch welche Änderungen man ihn korrigieren könnte.

(a) Führen Sie den Algorithmus für das Array $A = \langle 3, 6, 4, 1, 3, 4, 1, 4 \rangle$ durch. Geben Sie dabei zumindest den Inhalt des Arrays C zu Beginn von Zeile 14 und zu Beginn von Zeile 17 an. Natürlich müssen Sie auch den Inhalt von B am Ende des Algorithmus angeben.

(b) Welche besondere Eigenschaft hat das Array B nach Beendigung des Algorithmus?

Lösung:

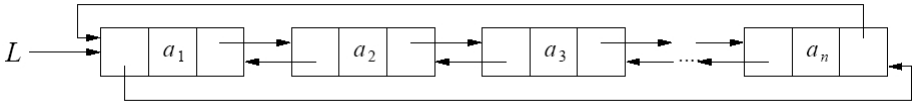
(a) Zeile 14: $C = \langle 2, 0, 2, 3, 0, 1 \rangle$
Zeile 17: $C = \langle 2, 2, 4, 7, 7, 8 \rangle$
Endergebnis: $B = \langle 1, 1, 3, 3, 4, 4, 4, 6 \rangle$

(b) Dieser Algorithmus erzeugt ordner die Elemente des Arrays A aufsteigend sortiert im Array B an. Dabei ist hervorzuheben, dass dieser Sortieralgorithmus sogar stabil funktioniert.

Aufgabe 2.5

Aufgabenstellung:

Schreiben Sie einen Algorithmus in Pseudocode, um in einer *absteigend* sortierten doppelt verketteten, zyklischen Liste L (siehe Abbildung) ein Element x sortiert einzufügen. Fertigen Sie auch eine Skizze hierfür an.



Geben Sie für diesen Algorithmus den Aufwand im Best-Case, Worst-Case und im Durchschnitt jeweils in Θ-Notation an.

Lösung:

Eingabe: Doppelt verkettete, absteigend sortierte Liste L, sortiert einzufügendes Element x.
Ausgabe: Doppelt verkettete Liste L, in die das Element x sortiert eingefügt ist.

```
Insert(L, x) {
    new(neu);
    neu.Inhalt = x;
    if(L == NULL) {
        neu.Vorgaenger = neu;
        neu.Nachfolger = neu;
        L = neu;
    }
    else {
        h = L;
        while(x < h.Inhalt && h.Nachfolger != L) {
            h = h.Nachfolger;
```

```
            }
            h.Vorgaenger.Nachfolger = neu;
            neu.Vorgaenger = h.Vorgaenger;
            neu.Nachfolger = h;
            h.Vorgaenger = neu;
            if(h == L) {
                L = neu;
            }
        }
        n++;
}
```

Dabei erzeugt die Methode **new** ein neues Element der verketteten Liste. Die Eigenschaften der einzelnen Elemente der verketteten Liste entsprechen jenen im Skriptum.

Aufgabe 2.6

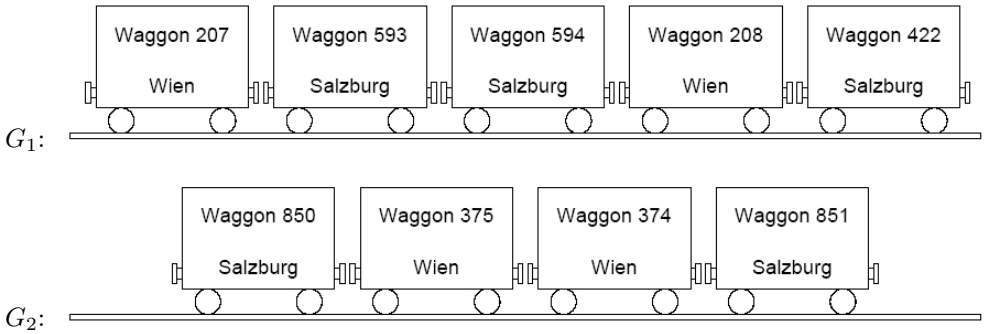
Aufgabenstellung:

In einem kleinen Bahnhof nahe Wien befinden sich auf den Gleisen G_1 und G_2 Eisenbahnwaggons, die für die Zielbahnhöfe Wien und Salzburg bestimmt sind, aber leider durcheinander stehen. Das Gleis G_3 ist leer.

(a) Betrachten Sie G_1, G_2 und G_3 als Stacks und schreiben Sie unter der Verwendung der unten angegebenen Operationen den Pseudocode für einen Rangieralgorithmus, der die Waggons so umordnet, dass anschließend auf G_1 alle Waggons nach Wien und auf G_2 alle Waggons nach Salzburg stehen. (Hinweis: Die Reihenfolge der Waggons für jede Zielrichtung ist unwichtig.)

- $push(G, W)$: stellt Waggon W auf Gleis G ab;
- $pop(G)$: holt den vordersten Waggon von Gleis G;
- $ziel(G)$: liefert den Zielbahnhof des ersten Waggons auf Gleis G, ohne diesen Waggon von G zu entfernen;
- $leer(G)$: überprüft, ob Gleis G leer ist.

(b) Auf den Gleisen G_1 bzw. G_2 stehen folgende Waggons:



Illustrieren Sie den Ablauf Ihres Algorithmus, indem Sie diese Waggons auf die gewünschten Gleise umordnen. Zeichnen Sie den Zustand der drei Rangiergleise bzw. der relevanten Datenstrukturen Ihres Rangieralgorithmus nach jeder Operation auf.

Lösung:

Naheliegend ist zunächst, alle Wagen von den Gleisen 1 und 2 auf Gleis 3 zu verschieben und die dort stehenden Wagen dann abhängig von ihrem Bestimmungsort zu verschieben:

```
Schlichte(G1, G2, G3, Ziel1, Ziel2) {
    while(!leer(G1)) {
        push(G3, pop(G1));
    }
    while(!leer(G2)) {
        push(G3, pop(G2));
    }
    while(!leer(G3)) {
```

```
        if(ziel(G3) == Ziel1) {
            push(G1, pop(G3));
        }
        else {
            push(G2, pop(G3));
        }
    }
}
```

Dieses Verfahren führt zu folgenden Verteilungen der Waggons auf den einzelnen Gleisen (hier sind nur die Zustände vor bzw. nach den while-Schleifen dargestellt):

G_1	207 - 593 - 594 - 208 - 422
G_2	850 - 375 - 374 - 851
G_3	-
G_1	-
G_2	850 - 375 - 374 - 851
G_3	422 - 208 - 594 - 593 - 207
G_1	-
G_2	-
G_3	851 - 374 - 375 - 850 - 422 - 208 - 594 - 593 - 207
G_1	207 - 208 - 375 - 374
G_2	593 - 594 - 422 - 850 - 851
G_3	-

Abgesehen von dieser Methode gibt es jedoch auch noch eine schneller sortierende Methode, bei der zunächst alle Wagen von Gleis 1 auf Gleis 3 gestellt werden. Dann werden die Wagen, welche auf Gleis 2 stehen, abhängig von ihrem Bestimmungsort entweder auf Gleis 1 oder auf Gleis 3 verschoben. Schließlich muss nur noch Gleis 3 sortiert werden:

```
Schlichte(G1, G2, G3, Ziel1, Ziel2) {
    while(!leer(G1)) {
        push(G3, pop(G1));
    }
    while(!leer(G2)) {
        if(ziel(G2) == Ziel1) {
            push(G1, pop(G2));
        }
        else {
            push(G3, pop(G2));
        }
    }
    while(!leer(G3)) {
        if(ziel(G3) == Ziel1) {
            push(G1, pop(G3));
        }
        else {
            push(G2, pop(G3));
        }
    }
}
```

Dies führt zu der folgenden Verteilung, wieder jeweils vor bzw. nach den while-Schleifen dargestellt

G_1	207 - 593 - 594 - 208 - 422
G_2	850 - 375 - 374 - 851
G_3	-
G_1	-
G_2	850 - 375 - 374 - 851
G_3	422 - 208 - 594 - 593 - 207
G_1	374 - 375
G_2	-
G_3	851 - 850 - 422 - 208 - 594 - 593 - 207
G_1	207 - 208 - 374 - 375
G_2	593 - 594 - 422 - 850 - 851
G_3	-

Aufgabe 2.7

Aufgabenstellung:

Gegeben sei eine Personendatenbank mit dem Ergebnis einer anonymisierten Umfrage, deren Einträge *person* in einem Array gespeichert werden. Ein Eintrag $person[i]$ enthält dabei folgende Daten:

- $person[i].id$: anonymisierte Codenummer (eindeutig)
- $person[i].land$: Land des Hauptwohnsitzes
- $person[i].groesse$: Körpergröße in cm
- $person[i].gewicht$: Körpergewicht in kg

Die Einträge im Feld seien nach Ländern aufsteigend sortiert.
Geben Sie den Pseudocode für eine Suche in diesem Array an, sodass für ein gegebenes Land *suchland* die Codenummern, Körpergrößen und Körpergewichte aller erfassten Personen, die in diesem Land ihren Hauptwohnsitz haben, ausgegeben werden.
Zusätzlich sollen am Ende die Mittelwerte für Körpergröße und Körpergewicht der im Rahmen dieser Suche gefundenen Personen ausgegeben werden.
Verwenden Sie zur Lösung eine modifizierte Variante der binären Suche.

Lösung:

Der folgende Algorithmus verwendet zunächst eine Variante der binären Suche zum Auffinden des kleinsten Elements mit dem gesuchten Schlüssel und dann eine ähnliche Variante zum Finden des größten Elements. Anschließend werden alle Elemente zwischen diesen beiden Elementen in ein neues Feld zusammengefasst und die Mittelwerte berechnet.

Eingabe: Aufsteigend sortiertes Feld **person**, welche die entsprechenden Daten enthält; Schlüssel **suchland**, nach dem gesucht werden soll.
Ausgabe: Feld **Ergebnis**, welches die gesuchten Daten enthält; Mittelwerte für Körpergröße und Gewicht der gefundenen Personen (**Groesse**, **Gewicht**).

```
Suche(person, suchland) {
    Pos1 = 0; // Index des ersten Elements mit dem gesuchten Schlüssel
    Pos2 = 0; // Index des letzten Elements mit dem gesuchten Schlüssel

    // Anfang suchen
    repeat {
        m = floor((l+r)/2);
        if(suchland == person[m].land) {

            // Bereich mit gesuchten Schlüsseln beginnt vor m und endet nach m
            if(m > 0) {
                if(suchland > person[m - 1].land) {
                    Pos1 = m;
                }
                else {
                    r = m -1 ;
                }
            }
            else {
                r = m - 1;
            }
        }
        else {
            if(suchland > person[m].land) {
                r = m - 1;
            }
            else {
                l = m + 1;
            }
        }
    }
    until(Pos1 > 0 || l > r);

    // Anfang gefunden?
    if(Pos1 > 0) {
```

```
        // Ende suchen
        n = length(person);
        repeat {
            m = floor((l+r)/2);
            if(suchland == person[m].land) {

                // Bereich mit gesuchtem Schlüssel beginnt vor m und endet nach m
                if(m < n) {
                    if(suchland < person[m + 1].land) {
                        Pos2 = m;
                    }
                    else {
                        l = m + 1;
                    }
                }
                else {
                    Pos2 = m;
                }
            }
            else {
                if(suchland > person[m].land) {
                    r = m - 1;
                }
                else {
                    l = m + 1;
                }
            }
        }
        until(Pos2 > 0 || l > r);
    }

    Groesse = 0;
    Gewicht = 0;

    if(Pos1 > 0 && Pos2 > 0) {
        Ergebnis = new array(Pos2 - Pos1 + 1);
        for(a = Pos1; a <= Pos2; a++) {
            Ergebnis[a - Pos1 + 1].land = person[a].land;
            Ergebnis[a - Pos1 + 1].id = person[a].id;
            Ergebnis[a - Pos1 + 1].groesse = person[a].groesse;
            Ergebnis[a - Pos1 + 1].gewicht = person[a].gewicht;
            Groesse += person[a].groesse;
            Gewicht += person[a].gewicht;
        }
        Groesse /= (Pos2 - Pos1 + 1);
        Gewicht /= (Pos2 - Pos1 + 1);
    }
    else {
        Ergebnis = new array(0);
    }
}
```

Hier werden unter anderem folgende Methoden verwendet:

- **floor**(a): Liefert die größte ganze Zahl kleiner oder gleich a.
- **length**(A): Liefert die Anzahl der Elemente des Feldes A.
- **new array**(a): Erzeugt ein neues Feld mit a Elementen.

Abgesehen von der hier gezeigten Möglichkeit gäbe es noch weitere Möglichkeiten, die Aufgabenstellung zu bewältigen:

- Man sucht mit Binärer Suche das erste Element in der Liste, dessen Land mit dem gesuchten Land übereinstimmt. In weiterer Folge werden nacheinander alle dahinter folgenden Elemente, deren Land ebenfalls mit dem gesuchten Land übereinstimmt, ausgegeben.
- Man sucht mit Binärer Suche ein beliebiges Element, dessen Land mit dem gesuchten Land übereinstimmt. Von diesem Element aus geht man nacheinander alle Elemente in beiden Richtungen durch, um Anfang und Ende der Reihe der gewünschten Elemente zu finden. Anschließend werden alle Elemente innerhalb der gefundenen Grenzen ausgegeben.

Aufgabe 2.8

Aufgabenstellung:

Funktioniert der folgende Algorithmus, der als Eingabe ein Feld $A[0, ..., n-1]$ und einen Suchschlüssel x erwartet, als binäre Suche? Begründen Sie Ihre Antwort.

```
1  m = floor((n-1)/2);
2  wiederhole {
3    falls A[m] < x dann {
4      m = floor(m + m/2);
5    }
6    sonst {
7      falls A[m] > x dann {
8        m = floor(m/2);
9      }
10     sonst {
11       retourniere m;
12     }
13   }
14 } bis (m < 0) oder (m >= n);
15 retourniere -1;
```

Falls Sie den Algorithmus für fehlerhaft halten, schlagen Sie vor, durch welche Änderungen man ihn korrigieren könnte.

Lösung:

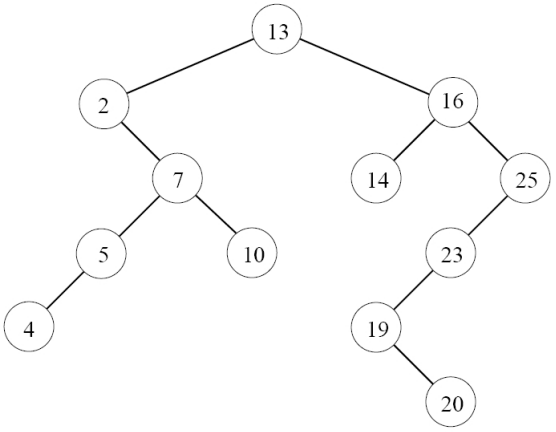
Der Algorithmus arbeitet nicht korrekt, weil die Variable m im ersten Schritt zwar jenen Teil des Arrays, der betrachtet wird, in die Hälfte teilt, in weiterer Folge allerdings nicht mehr. Die Einführung einer Variablen g, welche die Größe der Hälfte des betrachteten Ausschnitts des Arrays angibt, behebt dieses Problem:

```
m = floor((n-1)/2);
g = m;
repeat {
    if(A[m] < x) {
        m = floor(m + g/2);
    }
    else if(A[m] > x) {
        m = floor(m - g/2);
    }
    else {
        return m;
    }
    g = floor(g/2);
}
until(g == 0);
return -1;
```

Aufgabe 2.9

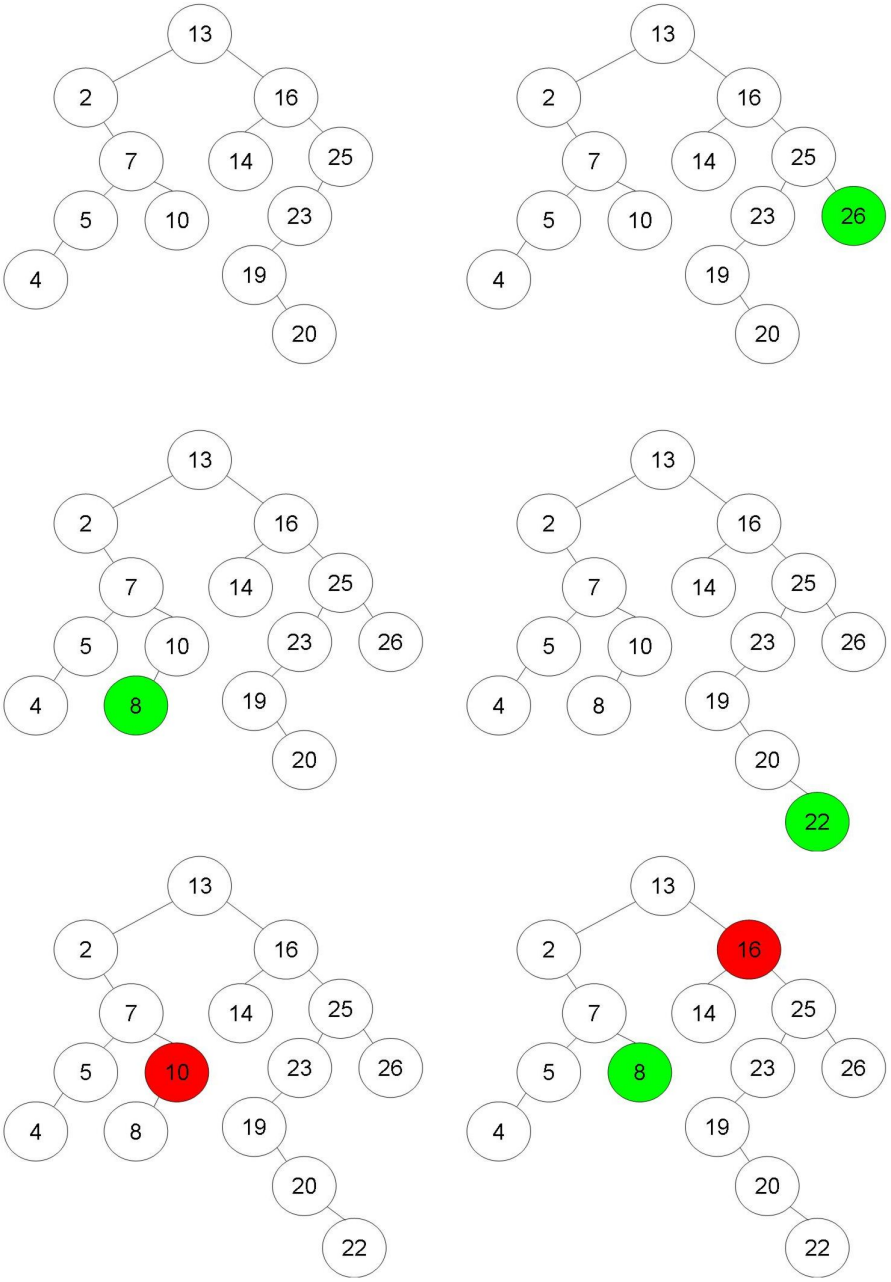
Aufgabenstellung:

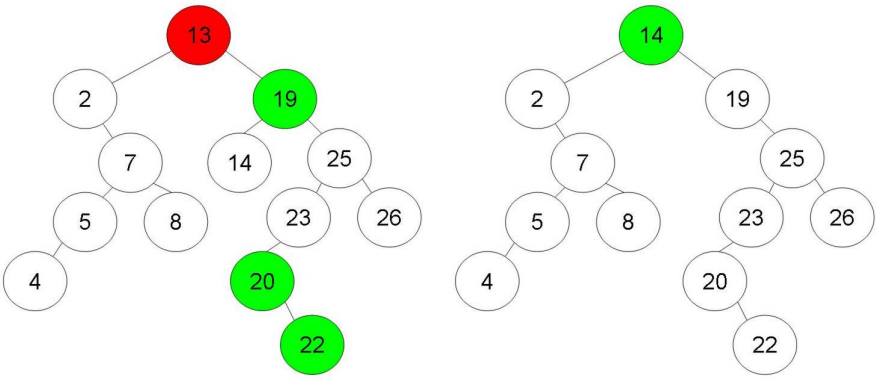
Fügen Sie in den abgebildeten binären Suchbaum in der angegebenen Reihenfolge die Knoten 26, 8 und 22 ein. Löschen Sie dann in der angegebenen Reihenfolge die Knoten 10, 16 und 13 mit dem Algorithmus aus der Vorlesung bzw. dem Skriptum. Zeichnen Sie den Zustand des Baumes nach jeder Operation.



Lösung:

In grün werden jene Knoten dargestellt, die sich im Rahmen der letzten Operation verändert haben; rot dargestellt sind jene Knoten, welche im Rahmen der folgenden Operation entfernt werden.





Aufgabe 2.10

Aufgabenstellung:

Gehen Sie in der Folge davon aus, dass die Knoten des jeweils betrachteten Baumes alle paarweise verschieden sind.

(a) Schreiben Sie einen Algorithmus in Pseudocode, der aus der gegebenen „In Order"- und „Pre Order"- Durchmusterungsreihenfolge eines binären Baums zuverlässig den diesen Durchmusterungsreihenfolgen zugrunde liegenden binären Baum rekonstruiert.

(b) Bei der „In Order"-Traversierung eines gegebenen binären Baumes ergibt sich folgende Knotenliste:

A, L, G, O, R, I, T, H, M, U, S

Bei einer „Pre Order"-Traversierung des gleichen Baumes ergibt sich:

R, O, L, A, G, H, I, T, U, M, S

Zeichnen Sie diesen Baum mit Hilfe Ihres Algorithmus und beschriften Sie die Knoten.

(c) Gibt es einen ähnlichen Algorithmus auch für den Fall, dass „Pre Order" und „Post Order" gegeben ist? — Warum (nicht)?

Lösung:

(a) Der hier gezeigte Algorithmus arbeitet rekursiv. Der Algorithmus arbeitet mit zwei Feldern, dem Feld I, welches die Knotenreihenfolge als Ergebnis der „In-Order"- Traversierung abgibt, und ein analoges Feld P für die „Post-Order"- Traversierung. Enthalten ein Teil des Feldes I und ein Teil des Feldes P (jeweils unmittelbar aufeinander folgende Elemente) dieselben Elemente, so beginnt der Teilbereich des Feldes P mit dem Schlüssel der Wurzel des betrachteten Teilbaumes. Ausgehend davon gehört alles, was im betrachteten Teil des Feldes I vor dem Element mit dem Schlüssel der Wurzel steht, zum Teilbaum des linken Kindknotens des betrachteten Teilbaumes, und alles, was dahinter steht, zum Teilbaum des rechten Kindknotens des betrachteten Teilbaumes. Diese Ausdrücke können durch rekursives Aufrufen des Algorithmus analysiert und der Baum damit rekonstruiert werden.

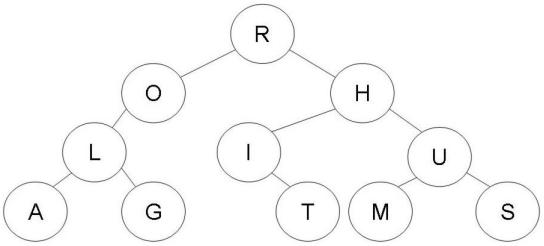
Eingabe: Feld P, welches das Ergebnis der „Pre-Order"- Traversierung enthält; Feld I, welches das Ergebnis der „In-Order"- Traversierung enthält; Grenzen **imin** und **imax** der zu betrachtenden Elemente des Feldes I; Grenzen **pmin** und **pmax** der zu betrachtenden Elemente des Feldes P; eventuell vorhandener Knoten **root**, an welchen der im Rahmen des Algorithmus erstellte Baum angehängt werden soll.

Ausgabe: Baum mit **root** als Wurzel

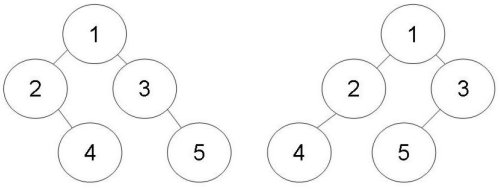
```
Reconstruct(P, I, pmin, pmax, imin, imax, var root) {
    if(root == NULL) {

        // Baum wird komplett neu erstellt (1. Schritt)
        r = new knot;
        r.key = P[pmin].key;
        root = r;

        if(pmin != pmax) {
            pos = Binaere-Suche(I, r.key, imin, imax);
            Reconstruct(P, I, pmin+1, pmin+pos, imin, pos-1, r);
            Reconstruct(P, I, pmin+pos+1, pmax, pos+1, imax, r);
        }
     }
     else {
        q = new knot;
        q.key = P[pmin].key;

        if(root.leftson == NULL) {
            root.leftson = q;
        }
        else {
            root.rightson = q;
        }

        if(pmin != pmax) {
            pos = Binaere-Suche(I, r.key, imin, imax);
            Reconstruct(P, I, pmin+1, pmin+pos, imin, pos-1, r);
            Reconstruct(P, I, pmin+pos+1, pmax, pos+1, imax, r);
        }
    }
}
```

Die Methode **new knot** erstellt einen neuen Knoten; der Algorithmus **Binaere-Suche** sollte aus dem Skriptum bekannt sein.

(b)



(c) Einen solchen Algorithmus kann es nicht geben, da eine Kombination aus Knotenreihenfolge einer „Pre-Order"- Traversierung und Knotenreihenfolge einer „Post-Order"- Traversierung nicht eindeutig ist, was am einfachen Beispiel zweier Bäume illustiert werden kann:



Beide Bäume haben als „Pre-Order"- Traversierung die Zahlenfolge $\langle 1, 2, 4, 3, 5\rangle$ sowie als „Post-Order"- Traversierung die Zahlenfolge $\langle 4, 2, 5, 3, 1\rangle$. Lediglich auf Grund der „In-Order"- Traversierung wäre eine Unterscheidung möglich (linker Baum: $\langle 2, 4, 1, 3, 5\rangle$, rechter Baum: $\langle 4, 2, 1, 5, 3\rangle$). Eine eindeutige Rekonstruktion des Baumes ist daher nicht möglich.