Fachhochschule Aachen Fachbereich Elektrotechnik und Informationstechnik	**Klausur - Bachelor** Algorithmen und Datenstrukturen (52106) Prof. Ingrid Scholl	WS 07/08 06.02.2008 Seite 1 / 26

Name (Blockschrift) Matrikelnummer

Hinweise:

Bearbeitungszeit: 180 Minuten

Hilfsmittel: Ein farbiges zweiseitig selbstgeschriebenes Hilfsblatt (DinA4), bitte deutlich mit Namen und Matrikelnummer versehen.
Ein nichtprogrammierbarer Taschenrechner.

- Bearbeiten Sie die Aufgaben direkt und nur auf den Aufgabenblättern oder Leerbögen
- Lösungen ohne erkennbaren und leserlichen Lösungsweg sind wertlos.
- Klausur zusammenlassen!!!
- Deckblatt mit Namen und Matrikelnummer beschriften und unterschreiben!
- Die letzte Seite der Klausur beinhaltet eine STL-Hilfe.
- Geben Sie bitte das selbstverfasste Hilfsblatt mit ab!

Hiermit bestätige ich, dass ich die Hinweise zur Kenntnis genommen habe:

(Unterschrift)

Aufgabe	**Max. Pkt.**	**Punkte**	
1	**10**		
2	**20**		
3	**13**		
4	**8**		
5	**11**		
6	**7**		
7	**9**		
8	**8**		
9	**7**		
10	**20**		
Gesamt	**113**		**Note:**

Aufgabe 1 (10 Punkte)

(a) Geben Sie die Bedingung für die O-Notation an:

$\exists K > 0 \; \exists n_0 \; \forall n > n_0 : |f(n)| \leq K \, g(n)$

(b) Zu welcher Komplexitätsklasse gehören die folgenden Funktionen?

$f_1(n) = 2n^2(5\log n + n)$ $= 10n^2 \log n + 2n^3 \rightarrow O(n^3)$

$f_2(n) = 8n \cdot \log n + 4\sqrt{n}$ $\rightarrow O(n \log n)$ // $\sqrt{n} = n^{1/2}$

$f_3(n) = 7n^2 \cdot \sqrt{n}$ $= 7n^{2+\frac{1}{2}} = 7n^{\frac{5}{2}} \rightarrow O(n^{5/2})$

(c) Nennen Sie drei Sortierverfahren, die mit quadratischem Aufwand sortieren.

Selection-
Insertion-
Bubble

(d) Ab welcher Komplexität wird eine Aussage über die Berechenbarkeit schwierig? Geben Sie ein Beispiel für die Größe von n und k an!

(e) Wie kann man die totale Korrektheit von Algorithmen zeigen?

(f) Welche Komplexität hat das Einfügen von einem Element im **worst case** bei einem binären Suchbaum mit n Elementen?

$O(n)$ // Der Baum ist nicht unbedingt ausgeglichen

(g) Welche Datenstrukturen kennen Sie, um möglichst ausgeglichene Bäume zu erhalten?

2 3 4 - Baum
AVL Baum
B Baum ---

(h) Wann ist ein Sortierverfahren stabil?

Wenn Duplikaten in der selben Reihenfolge stehen bleiben

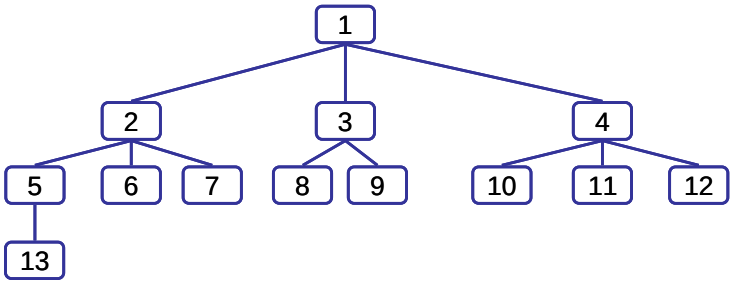
(i) Was ist der Nachteil von einem Greedy Algorithmus?

// testet nicht alle Möglichkeiten

Aufgabe 2 (20 Punkte)

Entwicklung einer Datenstruktur zu einem Ternärbaum, Implementierung eines rekursiven Preorder-Durchlaufs und eines iterativen Algorithmus zum Preorder-Durchlauf

Ein ternärer Baum ist ein Baum, bei dem jeder Knoten 0 bis 3 Nachfolgerknoten haben kann. Der folgende Baum ist zum Beispiel ein ternärer Baum:



Preorder: 1, 2, 5, 13, 6, 7, 3, 8, 9, 4, 10, 11, 12

(a) Entwickeln Sie eine Klasse, die eine Datenstruktur **für den Knoten** eines ternären Baumes definiert. Eine Ordnung der Knoten sei nicht bekannt.

(b) Entwickeln Sie eine Klasse, die eine Datenstruktur **für den ternären Baum** selbst definiert.

(c) Implementieren Sie eine **rekursive Funktion**, die diesen Baum in Preorder traversiert. Dabei wird bei einem Ternärbaum mit Wurzel p zunächst p ausgegeben, dann die Knoten des linken Teilbaumes in Preorder-Reihenfolge, anschliessend die Knoten des mittleren Teilbaumes in Preorder-Reihenfolge und zuletzt die Knoten des rechten Teilbaumes in Preorder-Reihenfolge.

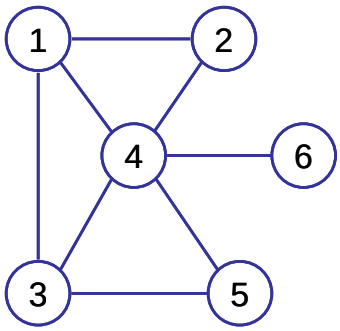
(d) Implementieren Sie eine **nicht-rekursive Funktion**, die mit Hilfe eines Stapels (Stack) die Knoten des Ternärbaumes in Preorder-Reihenfolge ausgibt. Verwende dazu die STL.

```
class Node3
{
public:
    int item;
    std::vector<Node3*> nachfolger;

    Node3(int item) : item(item) { this->nachfolger.resize(3, nullptr); };

};
```

```
class Baum3
{
public:

    Node3* wurzel;

    Baum3() :wurzel(nullptr) {}
    void preOrderRek(Node3* node) {
        if (!node)
            return;
        std::cout << node->item << " ";
        for (int i = 0; i < 3; i++)
        {
            preOrderRek(node->nachfolger[i]);
        }
    }

    void preOrderIte() {

        std::stack<Node3*> stack;

        if (wurzel)
            stack.push(wurzel);

        Node3* node;
        while (!stack.empty())
        {
            node = stack.top();
            stack.pop();

            std::cout << node->item << " ";

            for (int i = 2; i >= 0; i--)
            {
                if (node->nachfolger[i])
                    stack.push(node->nachfolger[i]);
            }
        }
    }

};
```

Aufgabe 3 (13 Punkte)

Gegeben ist der folgende Graph:



(a) Der Graph soll vom Knoten 1 ausgehend mit einer Breitensuche durchlaufen werden. Geben Sie eine mögliche Folge der Verarbeitungsreihenfolge an.

1 2 3 4 5 6 // Die Nachbarn werden hier aufsteigend zur PQ hinzugefügt

// oder 1 2 4 3 6 5

(b) Geben Sie zum Graphen die **Knotenliste** an.

[6, 8, 3, 2 3, 4, 2, 1, 4, 3, 1, 4, 5, 4, 1, 2, 3, 5
2, 3, 4, 1, 4]

(c) Geben Sie zum Graphen die **Adjazenzliste** an.

1 → 2 3 4
2 → 1 4
3 → 1 4 5
4 → 1 2 3 5 6
5 → 3 4
6 → 4

(d) Implementieren Sie unter Verwendung der STL eine Funktion **`getAdjazenzList`**, die aus einer Knotenliste eine Adjazenzliste berechnet und diese an das aufrufende Programm zurückgibt. (8 Punkte)

```
map < int, vector<int> >   getAdjazenzList(vector<int> nodelist)
```

```
std::map<int, std::vector<int>> result;
int resultIndex = 1;

for (int i = 2; i < nodelist.size() - 1; i += nodelist[i] + 1)
{
    for (int k = i + 1; k <= i + nodelist[i]; k++)
    {
        result[resultIndex].push_back(nodelist[k]);
    }
    resultIndex++;
}

return result;
```

Aufgabe 4 (8 Punkte)

Sortieren Sie die folgende Zahlenfolge in **absteigender** Reihenfolge

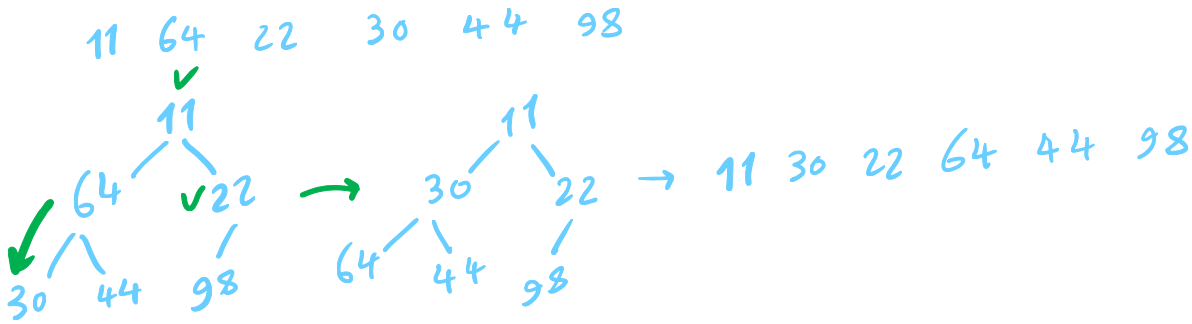
11 64 22 30 44 98

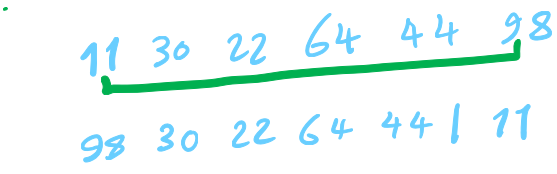
unter Verwendung von Heap Sort.

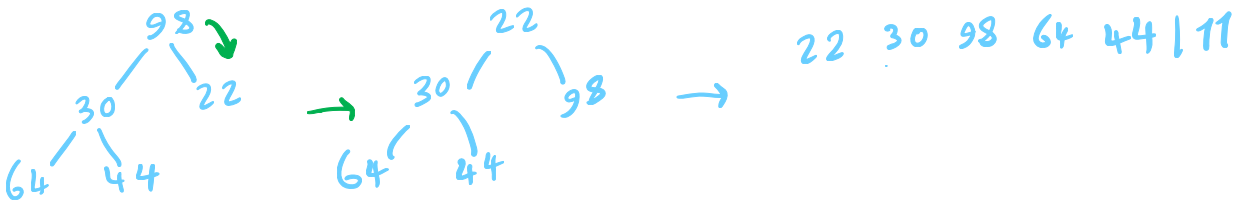
(a) Verwenden Sie dazu einen MinHeap oder einen MaxHeap?

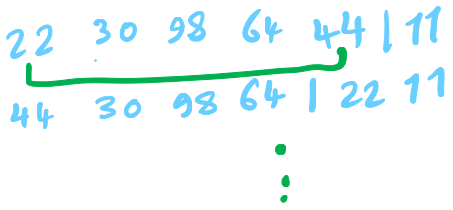
(b) Zeigen Sie den Aufbau des Heaps in Baumnotation und das Ergebnis des Heaps in Baum- und Arraynotation.

(c) Führen Sie die Sortierung mit dem Heap Sort Verfahren durch und zeigen Sie jeweils die notwendigen Zwischenschritte **in der Baum- und Arraydarstellung nach** jedem entfernten Element.





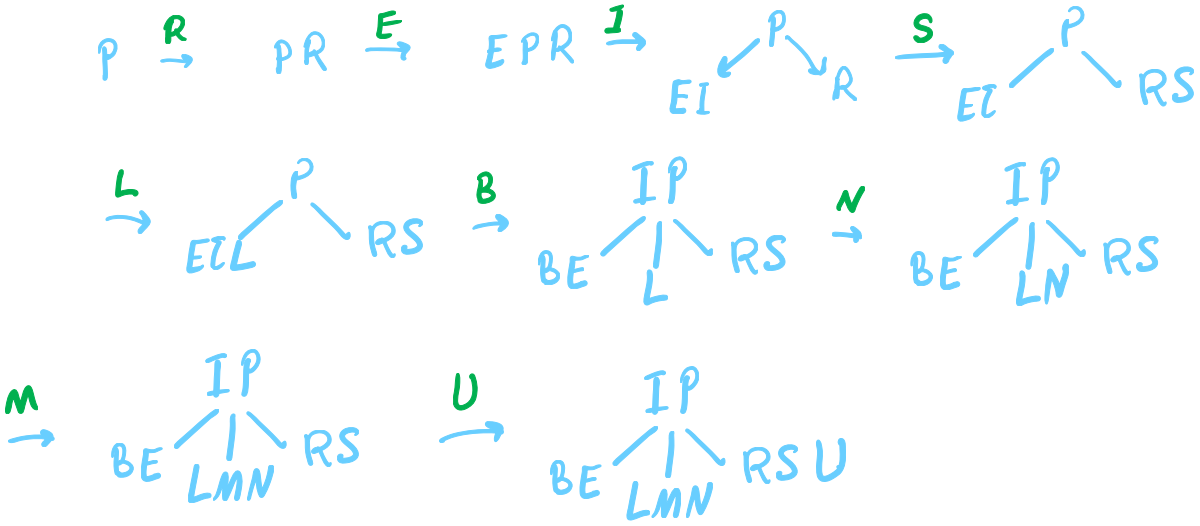




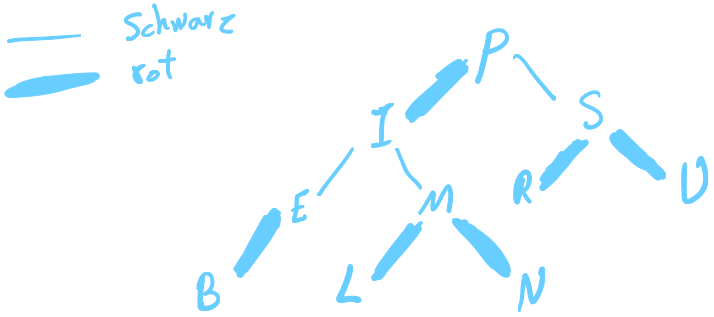
Aufgabe 5 (11 Punkte)

Gegeben sei die Buchstaben-Folge PREISELBEERENMUS.

(a) Zeigen Sie, wie der 2-3-4-Baum als TopDown-Verfahren entsteht, wenn man nacheinander die 10 Buchstaben ohne Duplikate einfügt: PREISLBNMU



(b) Stellen Sie das Endergebnis als Rot-Schwarz-Baum dar.
Markieren Sie eindeutig die „roten“ Kanten! (Bitte nicht in Rot markieren).



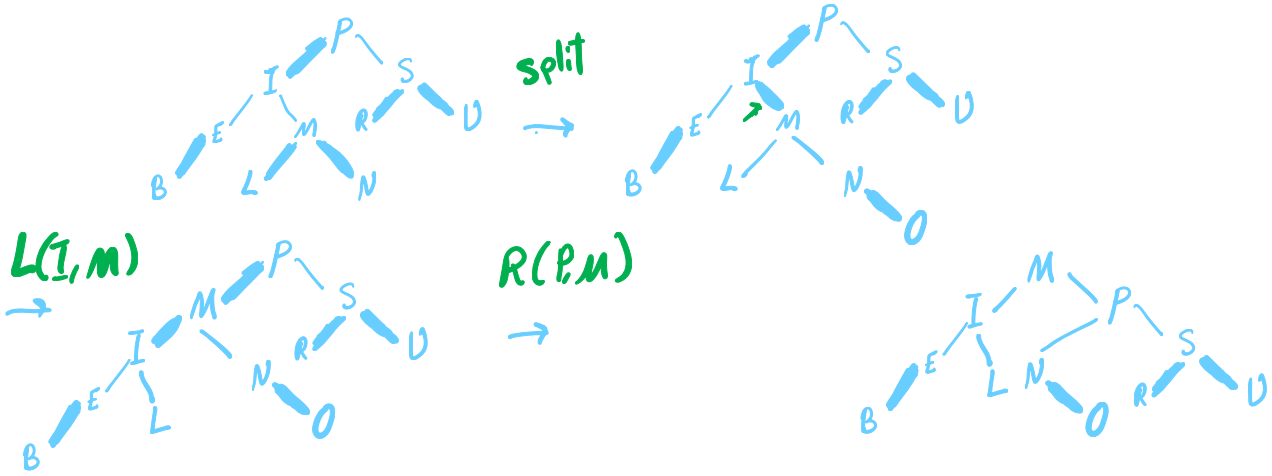
(c) Wann müssen in einem Rot-Schwarz-Baum beim Top-Down-Verfahren Kanten umgefärbt werden?

1) Einen 4 Knoten auf dem Suchpfad gefunden
2) Ein Knick gefunden :  bei der Rotation

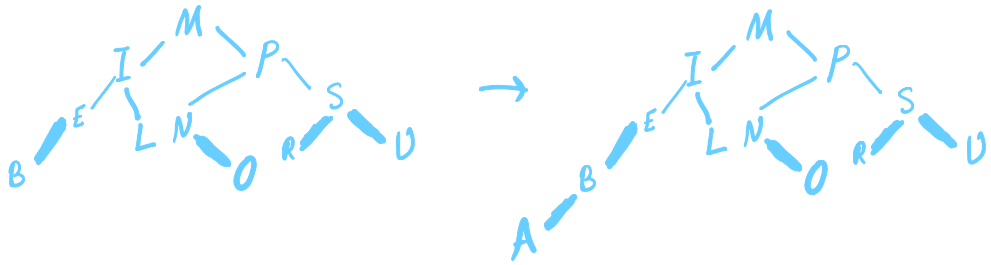
(d) Wann kommt es in einem Rot-Schwarz-Baum zu Rotationen?

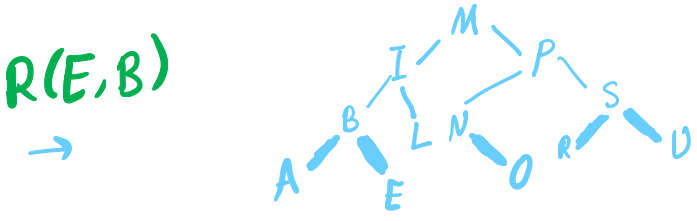
Wenn 2 rote Knoten aufeinanderkommen

(e) Fügen Sie nun **im Rot-Schwarz-Baum aus (b) Top Down** den **Schlüssel O** als roten Knoten ein. Zeigen Sie die Zwischenergebnisse und das Ergebnis des korrekten Rot-Schwarz-Baum.



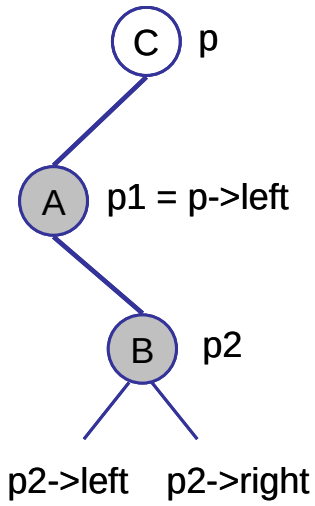
(f) Fügen Sie nun **im Rot-Schwarz-Baum aus (e) Top Down** den **Schlüssel A** als roten Knoten ein. Zeigen Sie die Zwischenergebnisse und das Ergebnis des korrekten Rot-Schwarz-Baum.





Aufgabe 6 (7 Punkte)

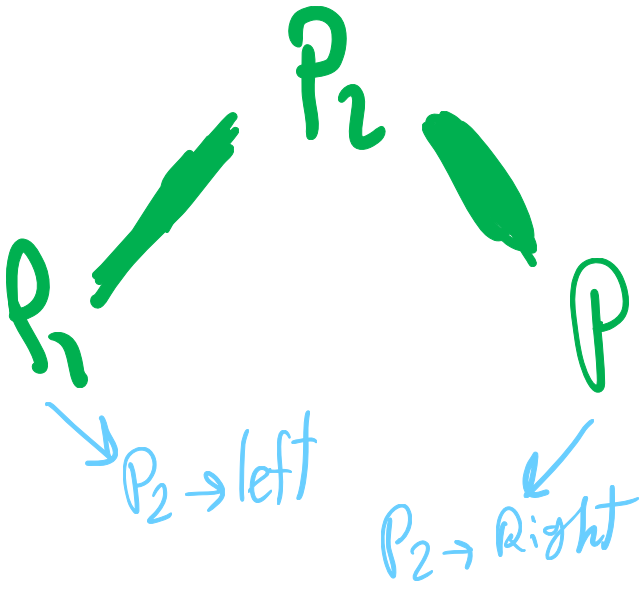
Implementieren Sie eine Links-Rechtsrotation in einem Rot-Schwarz-Baum.



Knoten C ist schwarz, die Knoten A und B sind rot. Die Funktion soll die Linksrotation zwischen A und B und dann anschließend die Rechtsrotation zwischen C und B durchführen. Überlegen Sie, ob und wie Sie die Farben der Knoten nach der Rotation anpassen müssen.

```
class treenode {
   public:
      int item;
      treenode *left;
      treenode *right;
      int color;
}
```

Skizzieren Sie zunächst das Ergebnis der Links-Rechtsrotation mit allen Knoten, Referenzen und Knotenfarben:



```
void redBlackLeftRightRotation (treenode * &p)
{   if(!p || !p->left || !p->left->right)
        return;

    treenode *p1 = p->left;
    treenode *p2 = p1->right;

    p1->right = p2->left;

    p->left = p2->right;

    p2->left = p1
    p2->right = p

    p2->color = false;
    p->color = true;

    p=p2;
}
```

Aufgabe 7 (9 Punkte)

(a) Was ist der Unterschied zwischen einem binären Suchbaum und einem AVL-Baum?

AVL Baum ist möglichst ausgeglichen, zusätzliche Attribute ist high

(b) Zeigen Sie den Aufbau eines AVL-Baumes, wenn Sie die folgenden Schlüssel **nacheinander** in einen anfangs leeren AVL-Baum einfügen:

30, 8, 70, 5, 28, 15, 3, 1

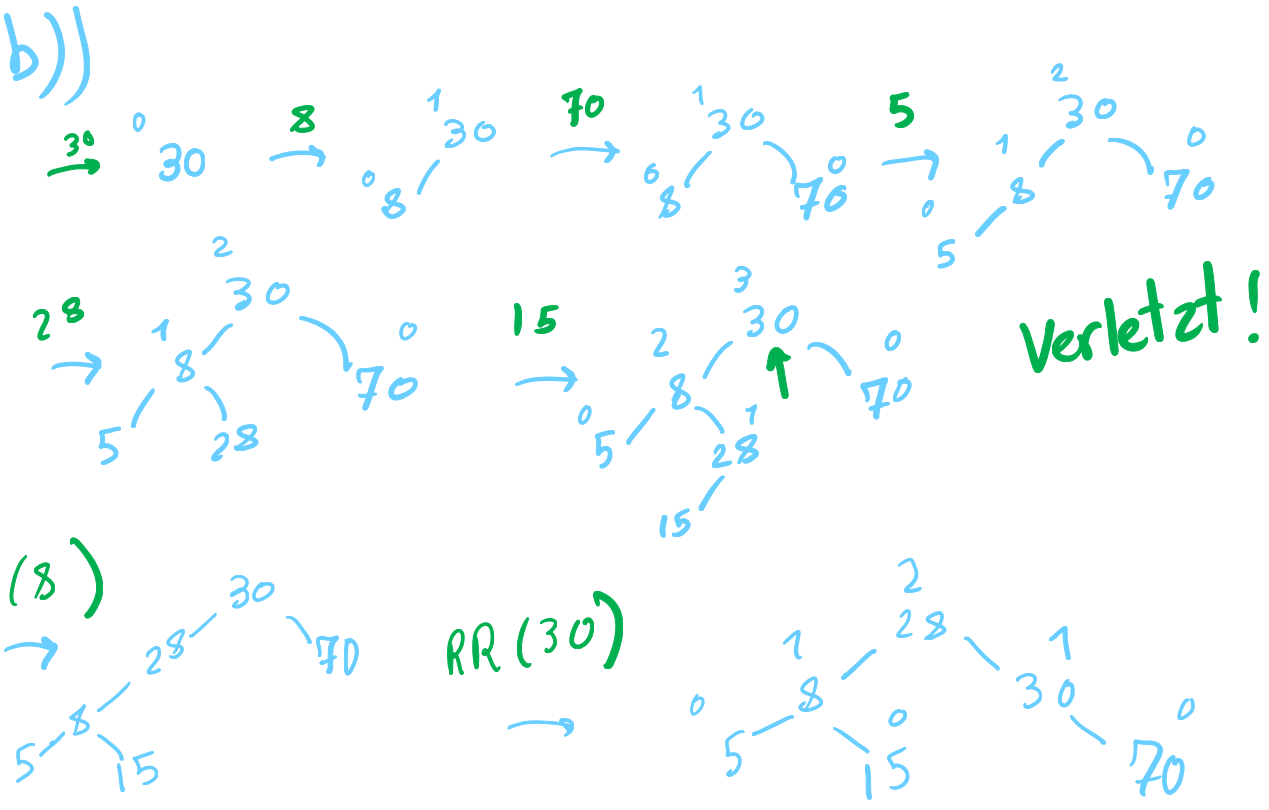
(c) Bei welchen einzufügenden Schlüsseln erhält man keinen AVL-Baum? 15, 1

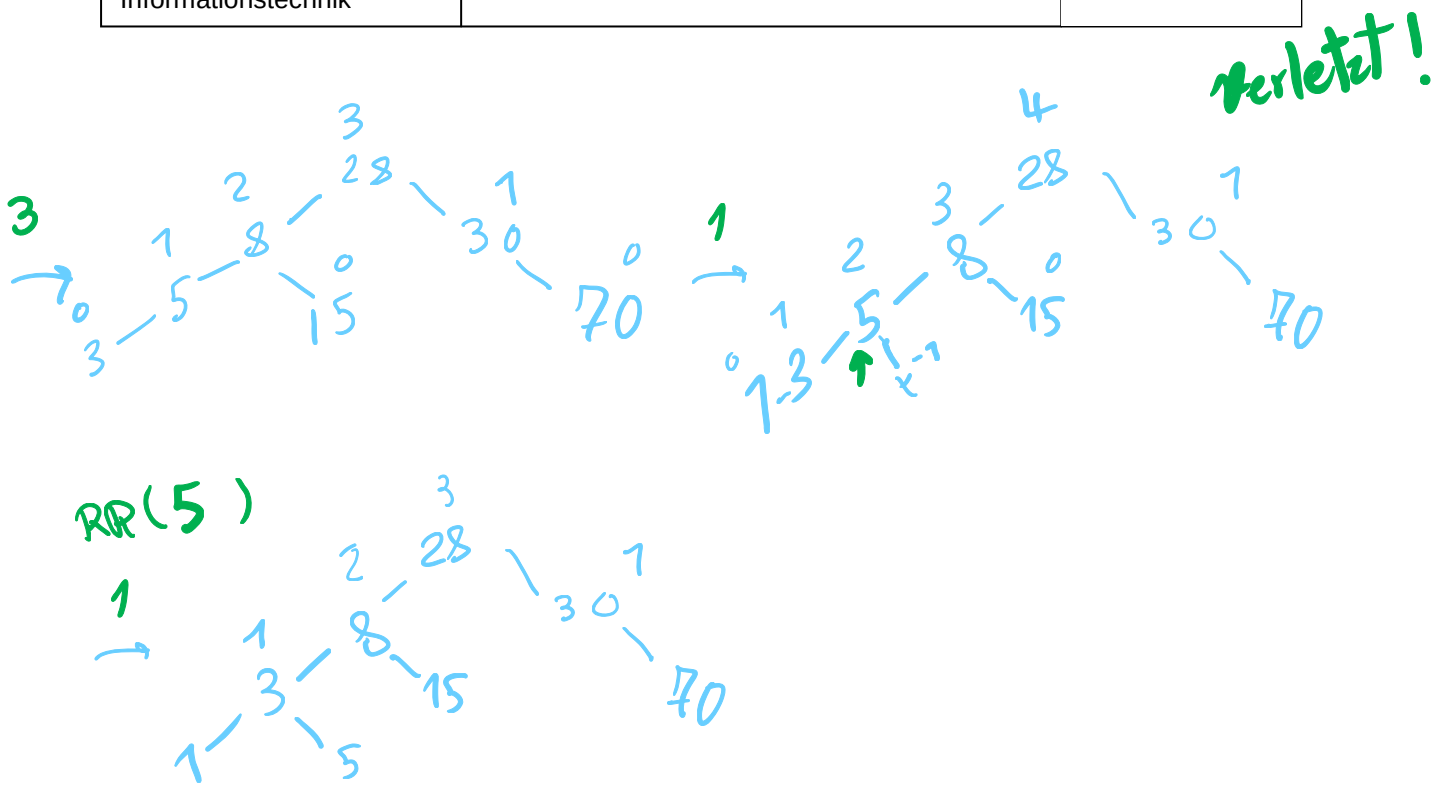
(d) Benennen Sie die Knoten, an denen es zu einer Balanceverletzung gekommen ist. Durch welche Rotationen konnte die Balance wieder hergestellt werden?

30 → LR(8) RR(30)
5 → RR(5)

(e) Gibt es zu diesem AVL-Baum eine Einfügereihenfolge der Schlüssel, die zu diesem Baum führt, ohne dass Rotationen stattfinden? Falls ja, so geben Sie diese Folge an. Ja

28, 8, 30, 3, 15, 70, 1, 5




3
28
8
15
5
3
30
70
verletzt!
4
28
8
15
5
3
30
70
RR(5)
3
28
8
15
3
5
1
30
70

Aufgabe 8 (8 Punkte)

(a) Was unterscheidet einen B-Baum von einem 2-3-4-Baum?

234-Node hat Maximal 3 Schlüssel
B-Node hat Maximal M schlüssel

(b) Was ist der Unterschied zwischen einem B- und einem B*-Baum?

B^+ speichert die Info in Blättern

(c) Wieviele Schlüssel kann ein **Blattknoten** im B*-Baum der Ordnung M=3 minimal und maximal aufnehmen?

M=L

minimal: 2 maximal: 3

(d) Wieviele Schlüssel kann ein **interner Knoten** im B*-Baum der Ordnung M=3 minimal und maximal aufnehmen?

minimal: 1 maximal: 2

(e) Erzeugen Sie einen **B*-Baum** mit M=3 durch Einfügen von folgenden Schlüsseln:
700, 132, 153, 601, 176, 706, 275, 750

Aufgabe 9 (7 Punkte)

Gegeben sei eine anfangs leere Hashtabelle *h* mit 11 Feldern, in die der Reihe nach die folgenden Schlüssel eingefügt werden sollen:

16, 21, 15, 10, 5, 19, 32

Fügen Sie die angegebene Schlüsselmenge mittels **doppelten Hashings** ein und geben Sie die Belegung der Tabelle nach der letzten Einfügeoperation an. Zeigen Sie jeweils die Berechnungen zur Kollisionsbeseitigung.

Schlüssel modulo der Tabellengröße:

X	16	21	15	10	5	19	32
X mod 11	5	10	4	10	5	8	10

Belegung der Hashtabelle h nach doppelten Hashing: R = 7, $(x + i(7 - x \% 7)) \% 11$

Index	0	1	2	3	4	5	6	7	8	9	10
H[Index]	-1	-1	32	10	15	16	-1	5	19	-1	21

Wie lautet die allgemeine Hashfunktion für das doppelte Hashing zu dieser Aufgabe an (bitte bekannte Werte einsetzen) ?

$h_i(x)$ = $(x + i(7 - x \% 7)) \% 11$

Aufgabe 10 (20 Punkte)

(a) Welche allgemeinen Traversierungsarten durch einen Graphen kennen Sie?

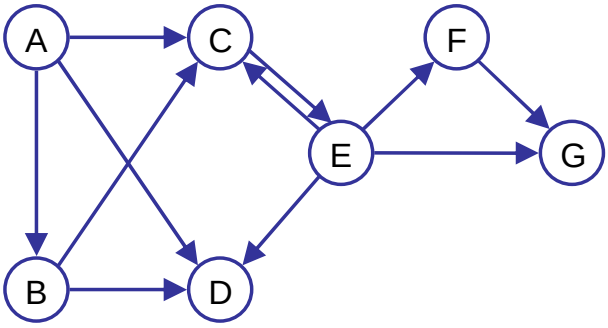
Tiefen- und Breitensuche

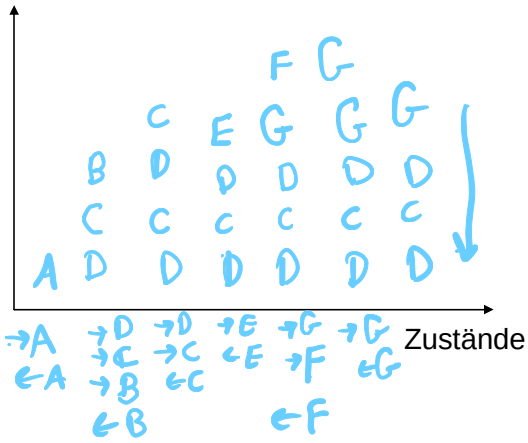
(b) Welche Datenstruktur wird für die iterative Tiefensuche verwendet?

Stack

(c) Führen Sie zu dem folgenden Graphen eine iterative Tiefensuche vom Knoten A ausgehend durch. Zeigen Sie alle Zwischenschritte und die Zustände der verwendeten Datenstruktur. Geben Sie zusätzlich die Verarbeitungsreihenfolge der Knoten an.

[A] entspricht „verarbeite Knoten A“,
A → entspricht „lege Knoten A auf die Datenstruktur“,
A ← entspricht „hole Knoten von der Datenstruktur und erhalte den Knoten A





→A ←A →D→C→B [A] ←B →D→C [B]
←C →E [C] ←E →G→F [E] ←F→G [F]
←G [G] ←G ←D [D] ←C ←D #

Verarbeitungsreihenfolge:

A B C E F G D

(d) Implementieren Sie eine Funktion, die für einen ungewichteten Graphen bestimmt, ob dieser zusammenhängend ist. Ein Graph ist zusammenhängend, wenn die Knoten über einen Pfad miteinander verbunden sind. Der Graph sei gegeben als Adjazenzliste.

(Tip: Traversieren Sie den Graphen mit einer iterativen Tiefensuche und merken Sie sich in einem mit 0 initialisiertem Feld, welche Knoten besucht wurden. Falls ein Knoten besucht wurde, wird in dem Feld ein positiver Wert für diesen Knoten gesetzt. Falls ein Knoten schon in der Datenstruktur der iterativen Tiefensuche gespeichert wurde, wird in dem Feld für den Knoten ein negativer Wert gesetzt.) (14 Punkte)

```
bool proofConnection(map< int, vector<int> > adjlist)
{
```

```
vector<bool> visited(adjlist.size(), false);
stack<int> st;
if adjlist.size() >0
    st.push(0);
int node;
while(!st.empty())
{
    node = st.top();
    st.pop();
    visited[node] =true;
    for(int i=0; i<adjlist[node].size(); i++)
    {  if(!visited[adjlist[node][i])
            st.push(adjlist[node][i]);
    }
}
for(int i=0; i<visited.size(); i++)
    if(!visited[i])
        return false;

return true;
}
```

STL-Hilfe:

Container, einige wichtige Funktionen und deren Verwendung:

```
vector<type> Feld

             vector<int> numbers;
             numbers.push_back(3); // Einfügen am Ende des Feldes
             numbers.pop_back();   // entfernt das letzte Element
             int last = numbers.back();   // liefert das letzte Element
             int first = numbers.front(); // liefert das erste Element
             int item = numbers[i]; // liefert das i-te Element
             int size = numbers.size();   // liefert die Größe des Feldes
             bool empty = numbers.empty(); // true, wenn das Feld leer ist,
                                           // andernfalls false

list<type>   doppelt verkettete Liste (s.Funktionen wie bei vector)
             list<int> numbers;
             numbers.push_back(5);   // fügt 5 ans Ende der Liste
             numbers.remove(5);      // löscht das Element 5


queue<type>  Queue
             queue<int> q;            // int – Warteschlange allokieren
             q.push(5);               // Element 5 in die Warteschlange
             bool leer = q.empty();   // Abfrage, ob Warteschlange leer ist
             int element = q.front(); // liefert erstes Element
             q.pop() ;                // löscht erstes Element

stack<type>  Stapel
             stack<int> nodes;            // int - Stapel allokieren
             bool ok = nodes.empty();     // überprüfe, ob Stapel leer ist
             int element = nodes.top();   // hole Element vom Stapel
              nodes.pop();                // entfernt Element vom Stapel
             nodes.push(5);               // 5 wird auf den Stapel gelegt
             int i = (int) nodes.size();  // Anzahl Elemente im Stapel


map<keytype, valuetype>  (Schlüssel, Referenz)-Container

             map<int, vector<int> > adjlist;    // Adjazenzliste als Map
             int size_adjlist = adjlist.size(); // Anzahl Knoten in der
                                                // Adjazenzliste
             vector<int> liste = adjlist[1]; // liefert den Vektor zum
                                             // Schlüssel 1
             liste[1].push_back(5); // Einfügen von Schlüssel 5 zur Liste
                                    // von Schlüssel 1
             int size = liste[1].size(); // liefert die Länge der Liste
```