ALG_UE 2 Protokoll

Beschreibung der rekursiven Funktionen

Die beschriebenen rekursiven Funktionen sind ehrlich gesagt nicht wirklich optimiert, bei der AVL Berechnung kann zum Beispiel darauf verzichtet werden, dass für jeden Knoten die Höhe rechts und links berechnet wird, es ist so jedoch lesbarer und verständlicher. Auch bei der Durchschnittsberechnung des Baumes ist es eigentlich nicht notwendig jeden Knoten 2 mal zu besuchen (einmal für die Anzahl und einmal für die Summe) wenn man dem Baum 2 Member zuweist (Summe und Anzahl) und diese in der rekursiven Funktion setzt.

Hinzufügen von neuen Werten:

```
void tree::insert(node* leaf, int key) {
    if (key < leaf->key_value)
    {
        if (leaf->left != NULL)
            insert(leaf->left, key);
        else
        {
            leaf->left = new node;
            leaf->left->key_value = key;
            leaf->left->left = NULL;    /
            leaf->left->right = NULL;   /
        }
    }
    else if (key > leaf->key_value)
    {
        if (leaf->right != NULL)
            insert(leaf->right, key);
        else
        {
            leaf->right = new node;
            leaf->right->key_value = key;
            leaf->right->left = NULL;  //
            leaf->right->right = NULL; //
        }
    }
    //If key == key_value just ignore it
}
```

Beim Hinzufügen von neuen Werten wird zuerst überprüft ob der einzufügende Wert kleiner oder größer ist als der des aktuellen Blatts. Falls er gleich ist wird er verworfen.

Je nachdem ob der Wert kleiner oder größer ist wird das linke bzw. rechte Blatt darauf überprüft ob es vorhanden ist. Falls ja, wird das Insert nochmal ausgeführt, nur dass nun das nächste Blatt als Parameter mitgeliefert wird.

Wenn kein nächstes Blatt in Einfügerichtung vorhanden ist wird ein neues Blatt erstellt, der Wert zugewiesen und die nächsten Knoten des neuen Blattes auf NULL gesetzt. Sobald dies passiert ist beendet sich die rekursive Funktion automatisch, da weder Werte zurückgegeben werden noch nach dem rekursiven Insert-Aufruf weiterer Code steht.

Minimum/Maximum errechnen:

```
int tree::min(node* leaf) {
    if (leaf == NULL) {
        return 0;  // Only if th
    }

    if (leaf->left != NULL) {
        return min(leaf->left);
    }

    return leaf->key_value;
}
```

Die Funktionen für Minimum und Maximum sind sich sehr ähnlich. Beim Minimum wird so lange das linke Blatt besucht, bis keines mehr da ist und dann der Wert zurückgegeben. Das Maximum macht das Selbe mit den rechten Blättern.

```
int tree::max(node* leaf) {
    if (leaf == NULL) {
        return 0;  // Only if the
    }

    if (leaf->right != NULL) {
        return max(leaf->right);
    }

    return leaf->key_value;
}
```


Markus Vogl, Thiago Gumhold

Durchschnitt errechnen (Summe & Anzahl):

```
double tree::avg() {
    if (root != NULL) {
        int s = sum(root);
        int c = count(root);
        return (((double)s) / ((double)c));
    }
    return 0;
}

int tree::sum(node* leaf) {
    unsigned int s = leaf->key_value;
    if (leaf->left != NULL) {
        s += sum(leaf->left);
    }
    if (leaf->right != NULL) {
        s += sum(leaf->right);
    }
    return s;
}

int tree::count(node* leaf) {
    unsigned int c = 1;
    if (leaf->left != NULL) {
        c += count(leaf->left);
    }
    if (leaf->right != NULL) {
        c += count(leaf->right);
    }
    return c;
}
```

Die Funktion avg ruft nur die rekursiven Funktionen für die Summe und die Anzahl der Blätter auf.

Auch hier sind die Funktionen für Summe und Anzahl sehr Ähnlich.

Jedes Blatt, dass nicht NULL ist wird besucht und zum zurückgegebenen Wert wird 1 für die Anzahl, oder der Blattwert für die Summe hinzuaddiert.

AVL Funktion (und Höhenberechnung) :

```
bool tree::avl(node* leaf, bool cout) {
    bool r = false;
    if (leaf != NULL) {
        r |= avl(leaf->right, cout);
        r |= avl(leaf->left, cout);
        int balance = height(leaf->right) - height(leaf->left);
        if (cout) {
            std::cout << "bal (" << leaf->key_value << ") = " << balance << " ";
            if (balance > 1 || balance < -1) { std::cout << "(AVL violation!)"; r = true; }
            std::cout << std::endl;
        }
    }
    return r;
    //nothing to do!!!
}
```

In der AVL-Funktion musste darauf geachtet werden, dass die Knoten in der richtigen Reihenfolge besucht werden. Solange der nächste Knoten nicht NULL ist wird AVL rekursiv aufgerufen, dann die Balance durch die Berechnung der Höhe rechts – der Höhe links errechnet und falls erwünscht das Ergebnis ausgegeben. r wird hierbei true sobald einmal true von einer AVL Funktion zurückgegeben wurde (wenn r true ist, ist der Baum KEIN AVL-Baum).

Die Höhe wird rekursiv berechnet. Es wird immer die Höhe links und rechts berechnet, wobei die Höhe eines NULL-Knotens 0 ist. Je nachdem welche berechnete Höhe größer ist wird diese Zahl + 1 als neue Höhe zurückgegeben.

```
int tree::height(node* leaf) {
    unsigned int h = 0;
    if(leaf != NULL){
        int l = leaf->left ? height(leaf->left) : 0;
        int r = leaf->right ? height(leaf->right) : 0;
        h = (l > r) ? l + 1 : r + 1;
    }
    return h;
}
```


Markus Vogl, Thiago Gumhold

Aufwandabschätzung

Der Best-Case ist ein AVL-Baum, der Base-Case beschreibt einen durchschnittlichen Baum, während der Worst-Case die schlimmste Baumanordnung (=Linked List).
N = Anzahl der eingefügten Elemente
H = Höhe des Baumes

AKTION	BEST-CASE	BASE-CASE	WORST-CASE
EINFÜGEN	O(log(N)) = O(H)	O(log(N)) ~ O(H)	O(N)
MIN/MAX	O(log(N)) = O(H)	O(log(N)) ~ O(H)	O(N)
DURCHSCHNITT	O(N)	O(N)	O(N)
AVL (SIEHE OBEN)	$O(N^2)$	$O(N^2)$	$O(N^2)$
HÖHENBERECHNUNG	O(N)	O(N)	O(N)