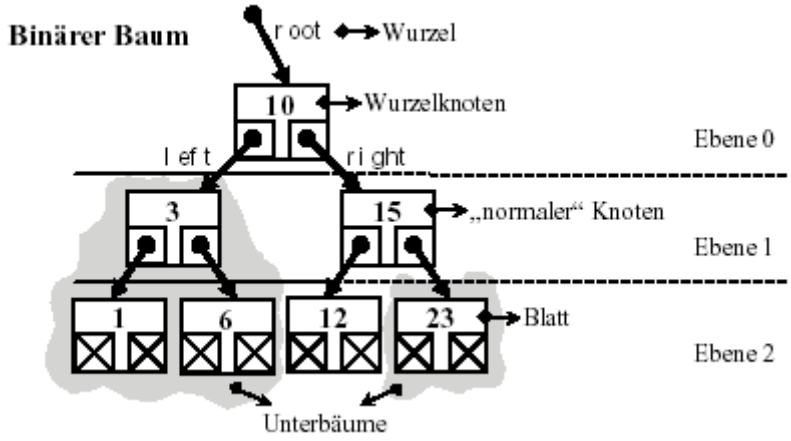
15.2 Der binäre Baum

15.2.1 Begriffsbestimmungen

Die linearense Listen sind einfache und relativ vielseitige Datenstrukturen. Doch bei einer gar nicht unwichtigen Funktion versagen sie kläglich: **beim Suchen** eines gespeicherten Datenelements. Das nun ist die Stärke des binären Baums.



Die Grafik zeigt den Aufbau eines binären Baumes und bereits einige damit verbundene, wichtige Begriffe.
Zunächst zur Struktur des Baumes. Wie auch bei den linearen Listen benötigt man eine Referenz, von der aus auf die Datenstruktur zugegriffen werden kann. Bei den Listen war dies listBegin (und zusätzlich noch listEnd), bei Bäumen (ganz allgemein) wird diese Referenz meist **root**, die **Wurzel**, genannt. Ist der Baum nicht leer, so verweist root auf den sogenannten Wurzelknoten.
Jeder Knoten besitzt Daten (data). Zusätzlich besitzt der binäre Baum zwei weitere Referenzen (daher der Name: binär/zwei), hier **left** und **right** genannt, die ihrerseits wieder auf weitere Knoten des Baumes verweisen können.
Vielleicht hat man schon erkannt, dass eine Sortierung der Daten durchgeführt wird (erst diese macht überhaupt eine effiziente Suche möglich). Ausgehend vom Wurzelknoten, dessen Datenelement in der Grafik die Größe 10 besitzt, liegen links davon alle Elemente, die kleiner sind, und rechts alle diejenigen, die größer sind. Da drängt sich gleich eine Frage auf: Was passiert mit einem Datenelement, das die gleiche Größe besitzt? Hier ist es Geschmacksache, wohin dieses gesteckt wird. Bei unserer Implementierung wird es nach rechts geschoben.

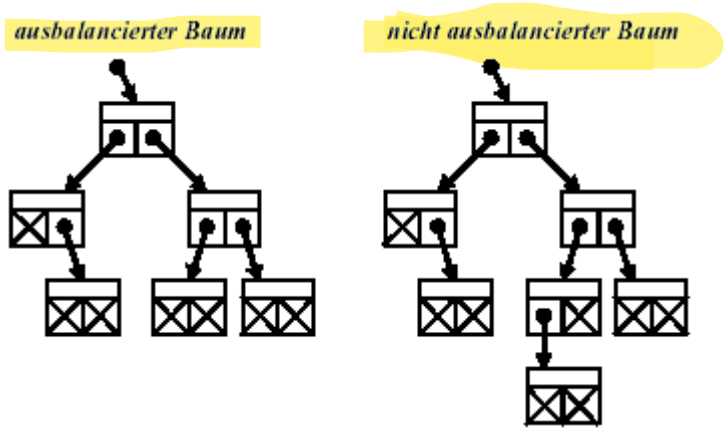
Im Zusammenhang mit Bäumen als Datenstruktur gibt es noch einige weitere Begriffe, die man kennen sollte.
Wohl einer der wichtigsten ist der des **Unterbaums** (in der Grafik sind einige Beispiele grau unterlegt). Betrachtet man die Referenz left des Wurzelknotens, so könnte man diese auch als root eines kleineren binären Baumes sehen, der dann den Wurzelknoten mit Größe 3 besitzt. Die Bedingungen für einen Baum sind erfüllt: alle kleineren Datenelemente liegen links, die größeren rechts. Auch der einzelne Knoten mit dem Datenelement der Größe 23 ist natürlich ein Unterbaum (des Baumes mit Wurzelelement 15, der wiederum Unterbaum des Gesamtbaumes ist), einer, der eben nur ein Element enthält.

So wie den Begriff Wurzel gibt es noch weitere Bezeichnungen, die aus der biologischen Analogie entlehnt wurden, z.B. das **Blatt**. So wird ein Knoten genannt, der keinen Nachfolger mehr besitzt, d.h. im Falle des binären Baumes, dass die Referenzen left und right den Wert NULL besitzen müssen.

So wie die Darstellungen der linearen Listen kann auch die des binären Baumes als gerichteter Graph gelesen werden. **Ast** nennt man dann einen Weg vom Wurzelknoten bis zu einem Blatt. Zu beachten ist dabei, dass beim binären Baum der Weg von der Wurzel bis hin zu einem bestimmten Blatt immer eindeutig ist. Die Länge eines Astes wird definiert durch die Anzahl der Knoten, die dieser Weg beinhaltet.
Als **Tiefe** eines Baumes wird das Maximum aller Astlängen bezeichnet. Mit anderen Worten, die Tiefe ist die **Anzahl der Ebenen**, die ein Baum in seiner momentanen Gestalt belegt.
Weiters ist der Begriff des **ausbalancierten** Baumes wichtig. So bezeichnet man einen Baum, der nicht mehr Ebenen als unbedingt notwendig besitzt für eine bestimmte Anzahl (N) an gespeicherten Datenelementen:

Minimum der benötigten Ebenen = [ld(N)+1], bzw. 0 für N = 0

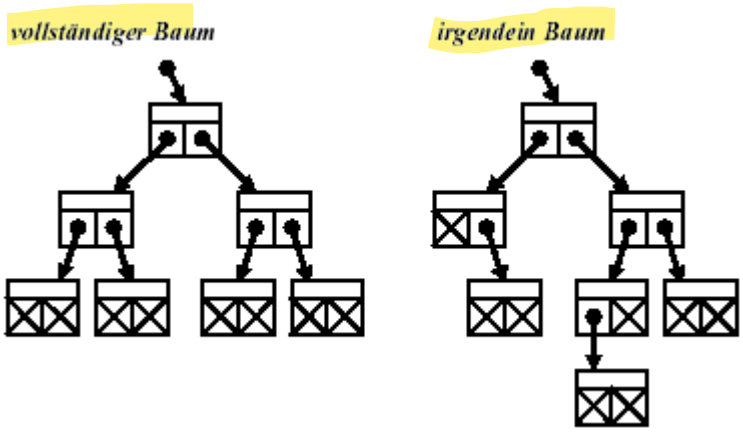
In der Grafik gilt der linke Baum als ausbalanciert, da für 6 Knoten beim binären Baum nun mal mindestens drei Ebenen benötigt werden. Beim rechten hingegen ist die Verwendung der vierten Ebene nicht unbedingt notwendig, da 7 Knoten auch in drei Ebenen Platz hätten. In dem hier gezeigten Fall wäre eine Umsortierung der Knoten notwendig, um dieses Ziel zu erreichen, da nicht einfach der Knoten in der vierten Ebene in den freien Platz verschoben werden kann (das darin gespeicherte Datenelement ist größer bzw. gleich groß als das im Wurzelknoten, darf daher nicht links davon zu liegen kommen!).



Ein Baum wird als vollständig bezeichnet, wenn es sich um einen ausbalancierten Baum handelt, dessen letzte Ebene voll besetzt ist. Vollständige Bäume müssen daher eine bestimmte Anzahl an Datenelementen enthalten (hier der Fall des binären Baumes):

notwendige (nicht hinreichende) Bedingung: $N = 2^x - 1$,

d.h. die Anzahl der Knoten muss eine Zweierpotenz minus 1 sein. In der folgenden Grafik erfüllen beide Bäume die angegebene Formel, trotzdem ist nur der linke als vollständig zu bezeichnen, da dieser zusätzlich noch ausbalanciert ist:



Und zu guter letzt noch eine Formel für binärer Baume, deren Herleitung wohl niemandem Kopfzerbrechen bereiten würde:

Maximale Anzahl d. Knoten einer Ebene = 2^{Ebene}

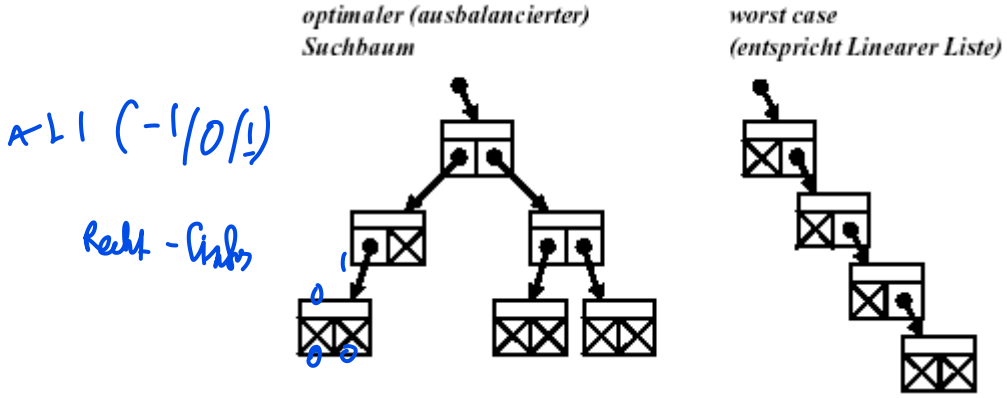
15.2.2 Anwendungsgebiete

Es wurde bereits angesprochen, Bäume sind, ganz allgemein, ideale Datenstrukturen zum Suchen gespeicherter Datenelemente bzw. zur Klärung der Frage, ob das Element der Begierde überhaupt vorhanden ist.
Doch bleiben wir beim binären Baum. Hier ist der Aufbau und die damit verbundene Sortierung der Daten für diese Fähigkeit verantwortlich. Denn bei einem bestehenden Baum ist der Weg von der Wurzel zu dem gesuchten Datenelement (egal, ob vorhanden oder nicht) immer eindeutig. Das bedeutet wiederum, dass die Anzahl der Suchschritte (Anzahl der besuchten Knoten) durch die Größe des längsten Astes (d.h. der Tiefe des Baumes) nach oben hin beschränkt ist.
Die folgende Tabelle listet auf, wie **viele Suchschritte** maximal (im optimalen Fall) notwendig sind, um ein gespeichertes Datenelement zu finden oder festzustellen, dass dieses nicht vorhanden ist. Diese Zahlen errechnen sich aus der bereits vorgestellten Formel für die minimale Anzahl an benötigten Ebenen für eine gegebene Menge an Datenelementen:

Anzahl der Datenelemente (N)	*Anzahl notwendiger Suchschritte (optimal)*
1	1
10	4
100	7
1.000	10
10.000	14
100.000	17
1.000.000	20

Das alles gilt aber nur für den Idealfall, dass nämlich der binäre Baum ausbalanciert ist. Denn nur in diesem Fall ist die Tiefe des Baumes für gegebene Anzahl an gespeicherten Datenelementen minimal. Ist der Baum nicht ausbalanciert, so existiert mindestens ein Knoten, der erst mit mehr Suchschritten als notwendig erreichbar ist.
Normalerweise hat man aber nicht das Glück, dass ein Baum nach Einfügen der verschiedenen Datenelemente ausbalanciert vorliegt. Der schlimmste mögliche Fall („worst case“) kommt dann zustande, wenn die Daten, die in den Binären Baum eingefügt werden

sollen, bereits sortiert vorliegen und dies nicht besonders behandelt wird. Denn was passiert in diesem Fall?
Man nimmt das erste Datenelement und speichert es im neu angelegten Wurzelknoten. Geht man davon aus, dass die Daten aufsteigend sortiert sind (für absteigend sortierte Daten gilt analoges), so ist das zweite Element größer als dass erste und muss demnach rechts vom Wurzelknoten gespeichert werden. Das nun folgende dritte Element ist größer als beide Vorgänger und wird daher im rechten Unterbaum des zweiten Knoten abgelegt und so geht es weiter. Was bei diesem Vorgang herauskommt, ist eine sortierte lineare Liste, bei der nur die Namen verschiedener Referenzen ausgetauscht wurden (root statt listBegin und right statt next):



Ist der Baum wirklich in einer linearen Liste ausgeartet, dann müssen im ungünstigsten Fall alle Datenelemente überprüft werden, bis das gesuchte Element gefunden wird (bzw. es feststeht, dass es nicht gespeichert ist). Und auch wenn man nicht so pessimistisch ist, im Mittel muss immer noch die Hälfte der Liste durchforstet werden, um diese Entscheidungen treffen zu können. Schon bei hundert Datensätzen ein unglaublicher Mehraufwand gegenüber einem ausbalancierten Baum (siehe Tabelle).