HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

MUSTERLÖSUNGEN ÜBUNGSSERIE 1

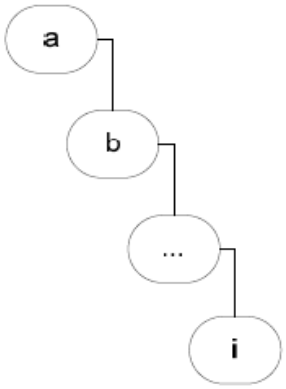
Algorithmen & Datenstrukturen AD2 / HS 2019

AD2 Team

Aufgabe 1 (Binary Search Trees - Verständnisfrage)
Gegeben sei folgende Liste von Elementen, die in einen binären Suchbaum eingetragen werden sollen: { a , b , c , d , e , f , g , h , i }
Zeichnen Sie den binären Suchbaum, falls diese Elementliste

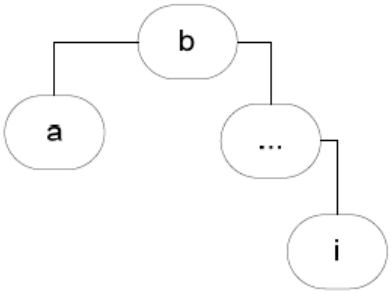
a) wie oben sortiert ist: entarteter Baum

Lösung:



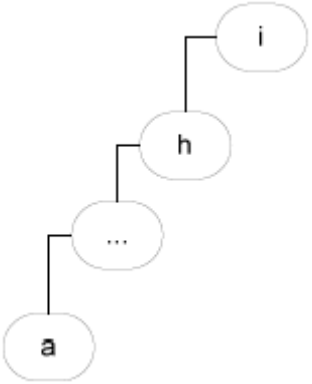
b) wie ändert sich der binäre Suchbaum, falls zyklisch rotiert wird
({ b , c , d , e , f , g , h , i , a } ...)

Lösung:
Drehung um die Wurzel



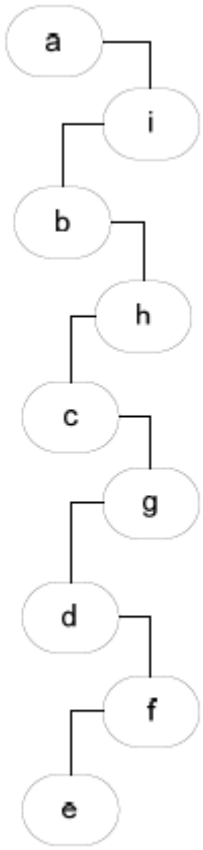
c) umgekehrt sortiert ist:

Lösung:



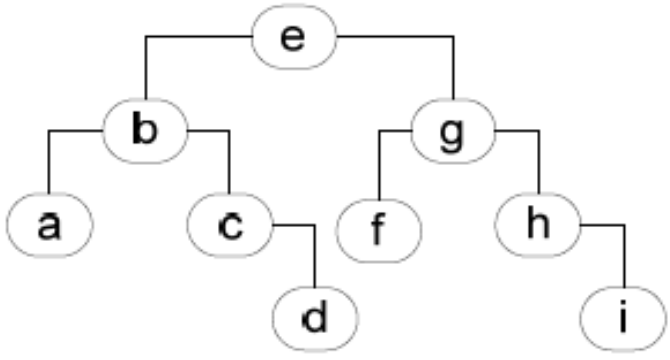
d) folgendermassen sortiert ist : { a , i , b , h , c , g , d , f , e }

Lösung:



e) Mit welcher Reihenfolge erhält man einen möglichst ausgeglichenen Baum?
Hinweis: Es gibt mehrere korrekte Lösungen.

Lösung:
{ e , b , a , c , d , g , f , h , i } ist eine mögliche Lösung



Animationen:

- *https://visualgo.net/en/bst*

Aufgabe 2
Eine Menge von n Zahlen kann sortiert werden, indem man diese zuerst in einen binären Suchbaum einfügt (sukzessive Anwendung von TREE-INSERT) und dann durch INORDER-TREE-WALK in sortierter Reihenfolge wieder ausgeben lässt. Bestimmen Sie die Worst-Case und Best-Case Laufzeiten für diesen Sortieralgorithmus.

Lösung:
Worst-Case:
Im Worst-Case werden die Elemente sortiert eingefügt, z.B. 1,2,3,4,5,6, …,n
Dies führt dazu, dass für das Element x auch x Schritte nötig sind um es einzufügen.
Für alle Elemente 1...n also:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Dies entspricht der Laufzeit $O(n^2)$

Best-Case:
Der Best-Case ist, wenn Elemente so eingefügt werden, dass ein vollständiger Binärbaum entsteht. Dies führt dazu, dass für das Element x nur $\log_2(x)$ Schritte nötig sind um es einzufügen. Für alle Elemente 1..n also:

$$\begin{aligned}\sum_{i=1}^{n} \lfloor \log_2(i) \rfloor &\leq \sum_{i=1}^{n} \log_2(i) \\ &= \log_2(1) + \log_2(2) + ... + \log_2(n) \\ &= \log_2(1 \cdot 2 \cdot ... \cdot n) \\ &= \log_2(n!) \\ &< \log_2(n^n) = n\log_2(n)\end{aligned}$$

die Laufzeit entspricht also $O(n\log_2(n))$.

Bemerkung (Stirling Formel): $\lim_{n\to\infty} \log_2(n!) = n\log_2(n)$

Auslesen:
Das Auslesen aus dem Binärbaum erfolgt in beiden Fällen mit Inorder-Tree-Walk welcher $\Theta(n)$ benötigt.

Total:
Die totale Laufzeit ist nicht abhängig vom Auslesen, da sowohl $O(n^2)$ als auch $O(n\log_2(n))$ eine höhere Laufzeit haben als $\Theta(n)$.

Aufgabe 3
Nehmen wir an, wir haben Zahlen zwischen 1 und 1000 in einem binären Suchbaum gespeichert und wir suchen die Zahl 363.
Folgend sind mehrere Suchpfade aufgelistet (Suchpfad: Knoten welche traversiert werden):

a)	2	257	401	398	330	344	397	363
b)	924	220	911	244	898	258	362	363
c)	925	202	911	240	918	245	363	
d)	399	387	219	266	382	381	278	363
e)	935	278	347	681	299	392	358	363

Welche der obigen Suchpfade *können kein gültiger Suchpfad* sein?

Hinweis: Stellen Sie den Suchpfad mit einer Skizze dar.

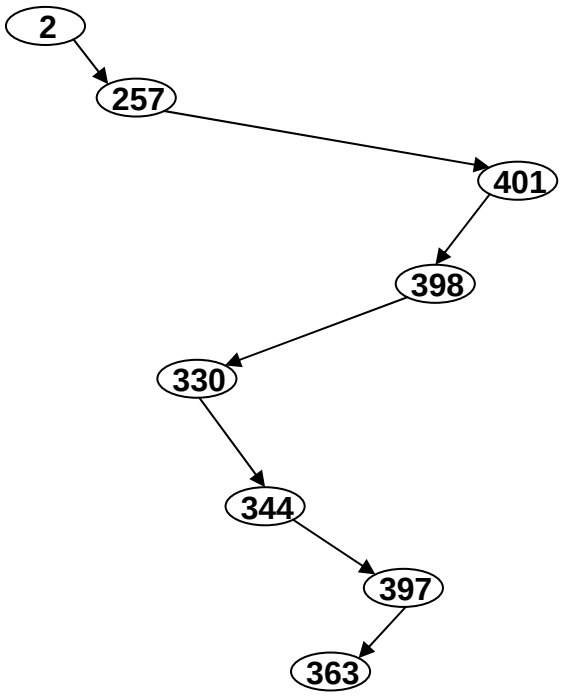
Lösung:
c) und e) sind keine möglichen Suchpfade.

Das Kriterium ist wie folgt: Nach jedem Element dürfen anschliessend entweder nur noch höhere (wenn nächste Zahl grösser ist) oder niedrigere Elemente (wenn nächste Zahl kleiner ist) folgen. Wenn z.B. also nach 421 noch ein Element 410 UND ein Element 430 folgt, ist der Suchpfad nicht gültig.

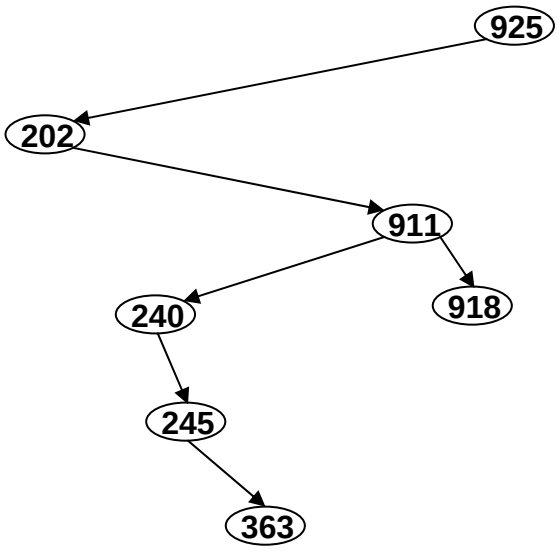
Weitere Variante:
Wenn man mit dem Suchpfad einen ‚binärern Suchbaum' aufbaut, dürfen keine ‚Verzweigungen' vorkommen.

Beispiel:

a)



c)



Aufgabe 4
Vergleichen Sie die Suche im binären Suchbaum mit der Suche durch Einschachtelung("binäre Suche") in einem Array.
Erstellen Sie hierfür eine ArrayList und fügen sie der Reihe nach Elemente ein. Die Einfügeposition soll mittels binärer Suche bestimmt werden (siehe *BinarySearchArrayTest.java* auf dem Skripte-Server).

Lösung:
Siehe „BinarySearchArrayTest.java“