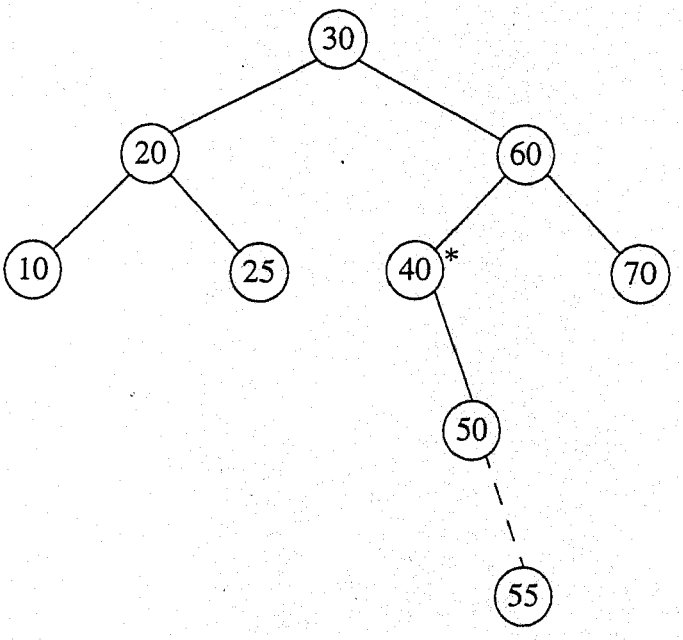
Musterlösungen zur Hauptklausur „1663 Datenstrukturen“

12. August 2000

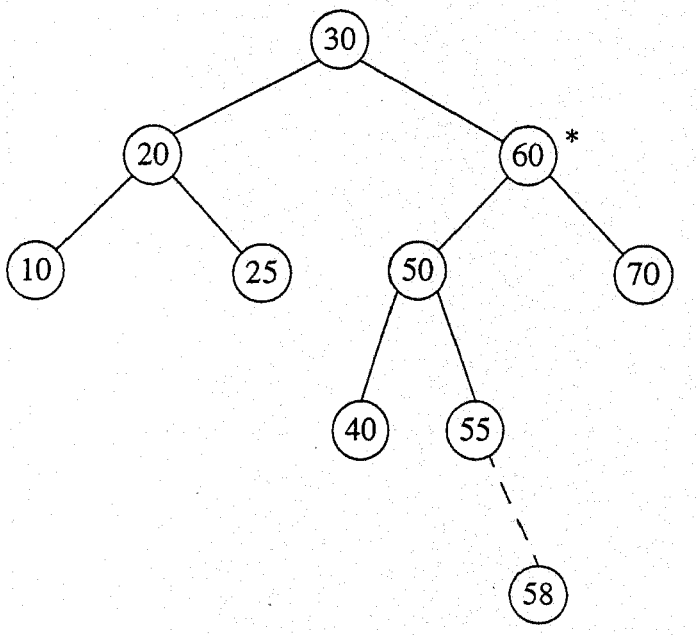
Aufgabe 1

(a)

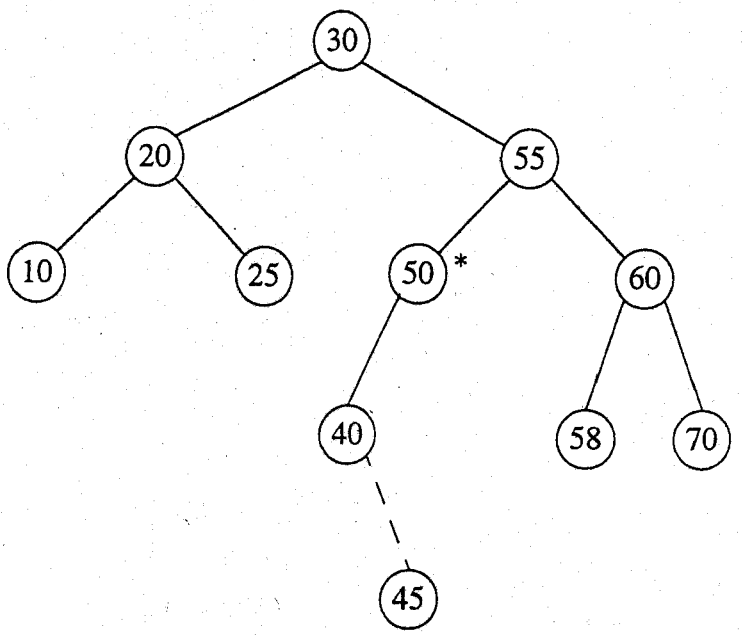
Einfügen von 55:



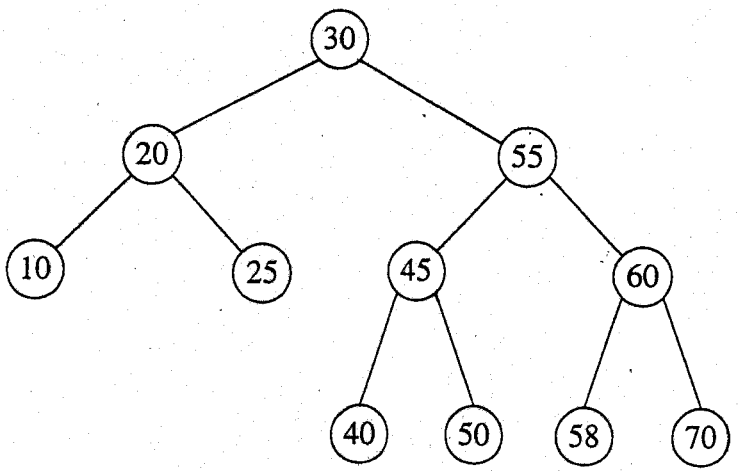
Rotation. Einfügen von 58:



Doppelrotation. Einfügen von 45:

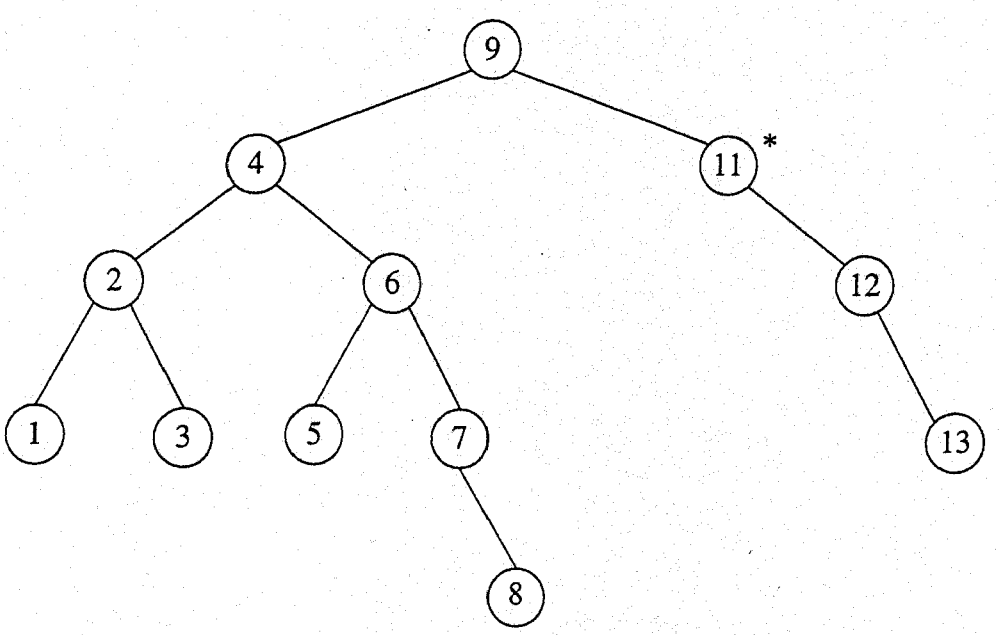


Doppelrotation:

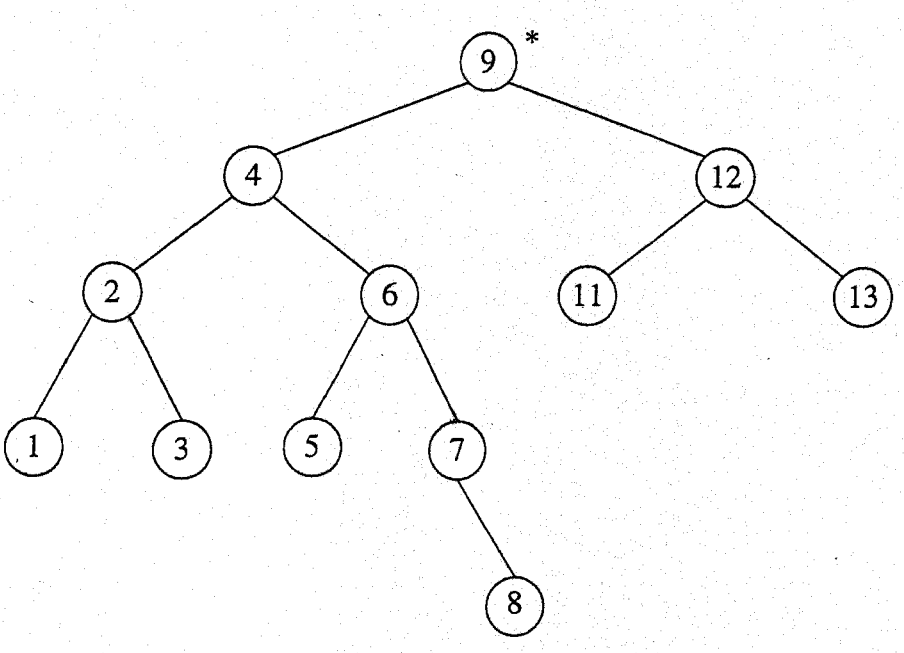


(b)

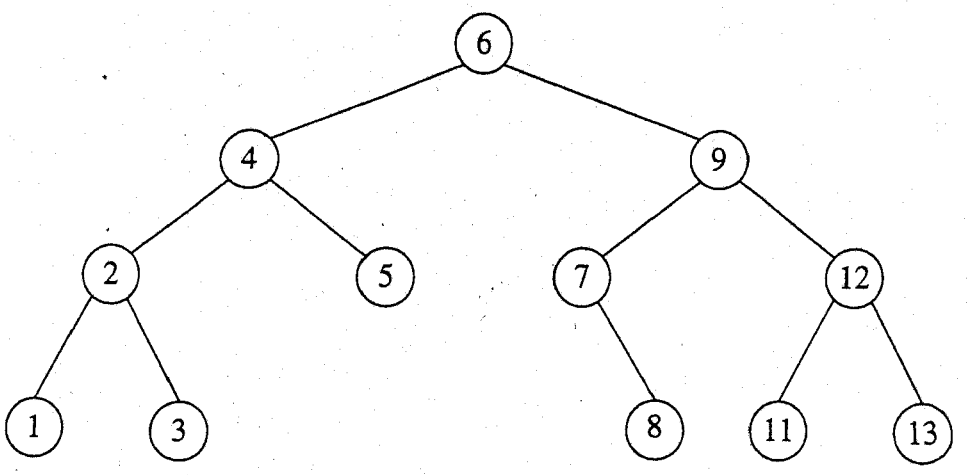
Wir erhalten zunächst:



Rotation:



Doppelrotation:



(c)

Der Zeitbedarf für jede der Operationen Suchen, Einfügen und Löschen ist O(log n). Beim *Suchen* muß im schlimmsten Fall der längstmögliche Pfad von der Wurzel zu einem Blatt verfolgt werden. Die Länge dieses Pfades entspricht der maximalen Höhe eines AVL-Baums, also etwa 1,44 log n. Beim *Einfügen* wird wieder schlimmstenfalls der längste Pfad durchlaufen, um dann maximal eine Doppelrotation durchzuführen. Die Kosten für eine Rotation oder Doppelrotation sind jedoch unabhängig von n. Beim *Löschen* schließlich werden maximal O(log n) Rotationen oder Doppelrotationen nötig, um alle aus der Balance geratenen Knoten auf dem Weg vom gelöschten Blatt zur Wurzel zu rebalancieren, nachdem das zu löschende Blatt zuvor in O(log n) Schritten gefunden wurde.

Aufgabe 2

Ausgangsfolge: 35 62 28 50 11 45

nach $i = 1$: 11 62 28 50 35 45

nach $i = 2$: 11 28 62 50 35 45

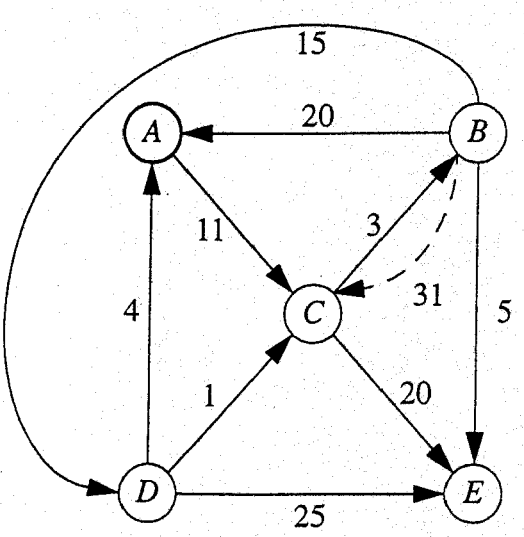
nach $i = 3$: 11 28 35 50 62 45

nach $i = 4$: 11 28 35 45 62 50

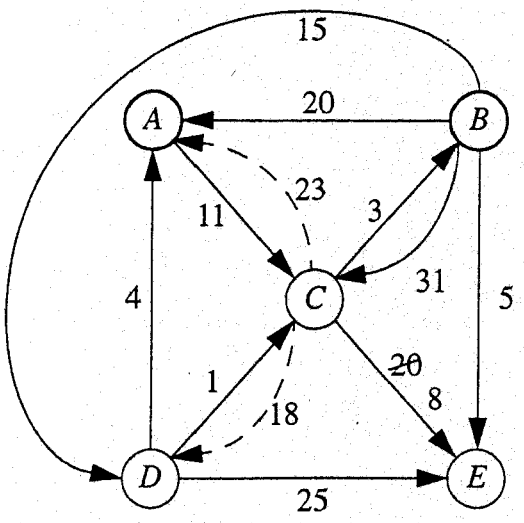
nach $i = 5$: 11 28 35 45 50 60

Aufgabe 3

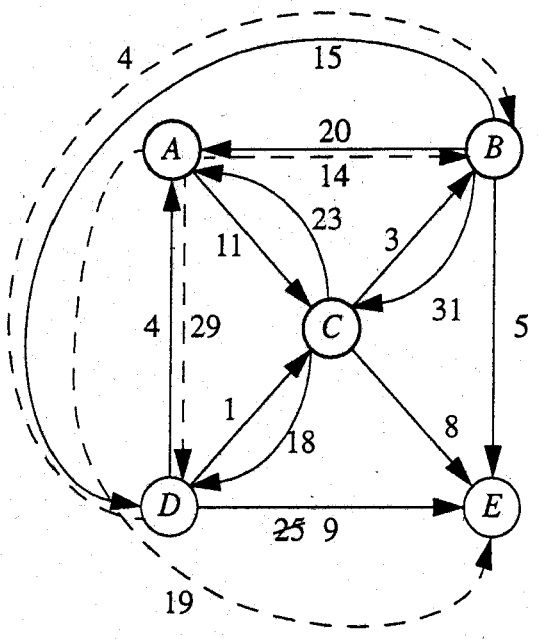
Die Knoten werden in lexikographischer Reihenfolge betrachtet. *G* nach Bearbeitung von *A*:



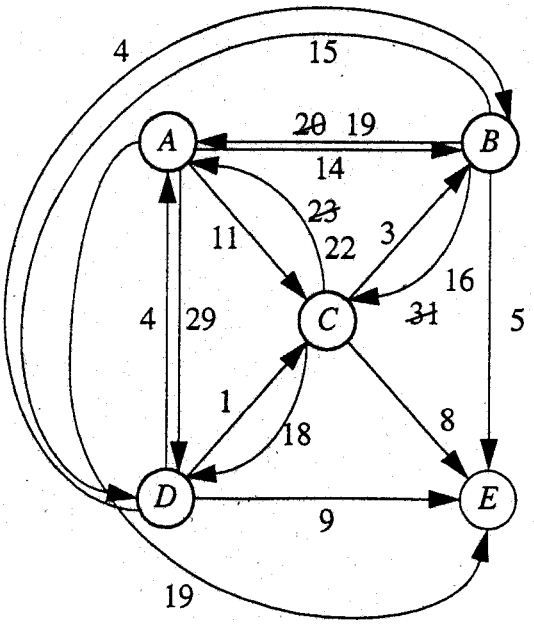
G nach Bearbeitung von *B*:



G nach Bearbeitung von *C*:



G nach Bearbeitung von *D*:

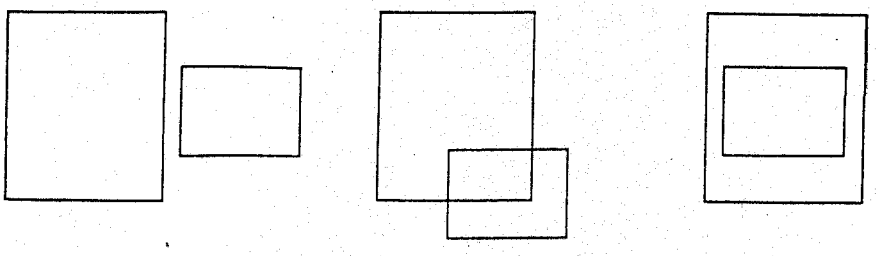


Die Bearbeitung von *E* führt zu keinen weiteren Veränderungen, da von diesem Knoten keine Kanten ausgehen.

Aufgabe 4

(a)

Es gibt im Wesentlichen drei Möglichkeiten, wie zwei Rechtecke zueinander liegen können:



Im ersten Fall muss keine Arbeit verrichtet werden, da keine Schnittpunkte existieren. Im zweiten Fall hilft der Segmentschnitt-Algorithmus. Mit diesem Algorithmus lassen sich Schnitte von Rechtecken finden, deren Kanten sich schneiden. Der dritte Fall wird mit dem Punkteinschluss-Algorithmus gelöst. Um Rechtecke zu finden, die einander ganz enthalten, wird in einem ersten Schritt der Reckteckmenge eine Menge von Punkten hinzugefügt, indem für jedes Rechteck ein beliebiger innerer Punkt als Repräsentant ausgewählt wird. Für zwei Rechtecke A, B mit ihren Repräsentanten P_A und P_B gilt nun: Falls A ganz in B enthalten ist, so ist auch P_A in B enthalten. Das Rechteckschnitt-Problem wird also durch Hintereinanderausführung des Segmentschnitt- und des Punkteinschluss-Algorithmus gelöst.

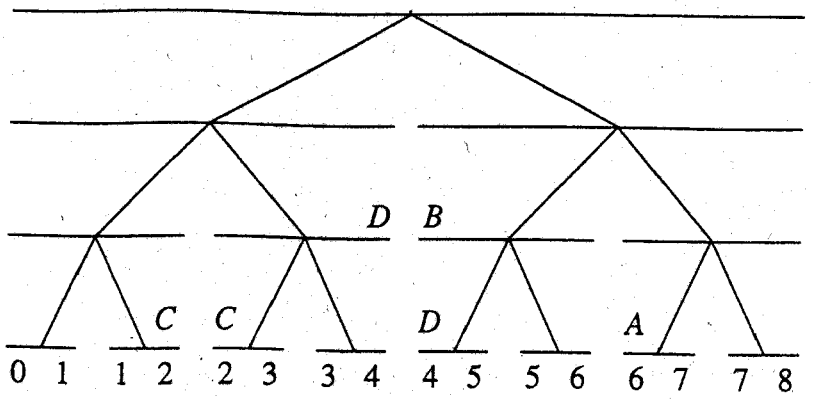
(b)

Sei I eine Datenstruktur, die eine Menge von Intervallen speichert und als Operationen das Einfügen und Löschen von Intervallen und das Auffinden aller Intervalle, die eine Query-Koordinate enthalten, erlaubt. Während des Sweeps werden dann folgende Aktionen durchgeführt:

1. Falls eine linke Rechteckkante angetroffen wird, füge das Intervall-Objekt $([y_1, y_2], r)$ in I ein. Hierbei bezeichnen y_1, y_2 die y-Koordinaten der linken Rechteckkante und r den Namen des Rechtecks.
2. Falls eine rechte Rechteckkante angetroffen wird, entferne das Intervall $([y_1,y_2], r)$ aus I.
3. Falls ein Punkt $([x, y], r, \textit{Punkt})$ angetroffen wird, bestimme $A := \pi_2(\{([y_1, y_2], r') \in I \mid y \in [y_1, y_2]\})$; gib alle Paare in $A \times \{ r \}$ aus.

Der Segmentbaum ist gerade eine solche Datenstruktur, wie sie in 1. beschrieben ist.

(c)



Aufgabe 5

(a)

(i) $f(n) = 2n, n > 0$
$f(n) = O(n)$

(ii) $f(n) = 2^n - 1, n > 0$
$f(n) = O(2^n)$

(iii) $f(n) = n + 1, n > 1$
$f(n) = O(n)$

(b)

$M(1) = 0$

$M(n) = 2\,M(n/2) + 1, n > 1, n$ Zweierpotenz

(c)

$M(n) = n - 1.$

Die Aussage ist laut (b) korrekt für $n = 1$. Unter der Annahme der Korrektheit auch für alle $n > 1$, n Zweierpotenz, ergibt sich dann der Induktionsschritt von n nach $2n$ durch

$$M(2n) = 2\,M(2n/2) + 1 = 2\,M(n) + 1 = 2\,(n - 1) + 1 = 2n - 1.$$

Aufgabe 6

(a)

Wir benutzen den temporären Speicherplatz der Länge k für ein Array B mit den Indizes 1, ..., k. Zweck von B ist es, die Häufigkeit des Auftretens eines Elements aus $\{1, ..., k\}$ festzuhalten. B repräsentiert also eine Multimenge. Anschließend wird dann jedes Element i aus $\{1, ..., k\}$ $B[i]$-mal nach A geschrieben.

```
algorithm CountingSort(A, B, n, k);
  {Initialisierung.}
  for i := 1 to k do B[i] := 0 od;
  {Die Anzahl der Elemente, die gleich i sind, wird in B[i] festgehalten.}
  for j := 1 to n do B[A[j]] := B[A[j]] + 1 od;
  {Zurückschreiben der Elemente nach A.}
  l := 1;
  for i := 1 to k do
    for j := 1 to B[i] do
      A[l] := i;
      l := l + 1
    od
  od
end CountingSort.
```

(b)

Die erste *for*-Schleife benötigt $O(k)$ Zeit, die zweite *for*-Schleife $O(n)$ Zeit. Obwohl zuletzt zwei ineinander verschachtelte *for*-Schleifen auftreten, ist deren Laufzeit $O(n+k)$, da die Summe aller $B[i]$ für i aus $\{1, ..., k\}$ gleich n ist. Da gemäß Aufgabenstellung $k < n$ angenommen wird, ist $k = O(n)$. Insgesamt ergibt sich somit eine Laufzeit von $O(n)$, die die Forderung, $\Omega(n \log n)$ zu unterschreiten, erfüllt.