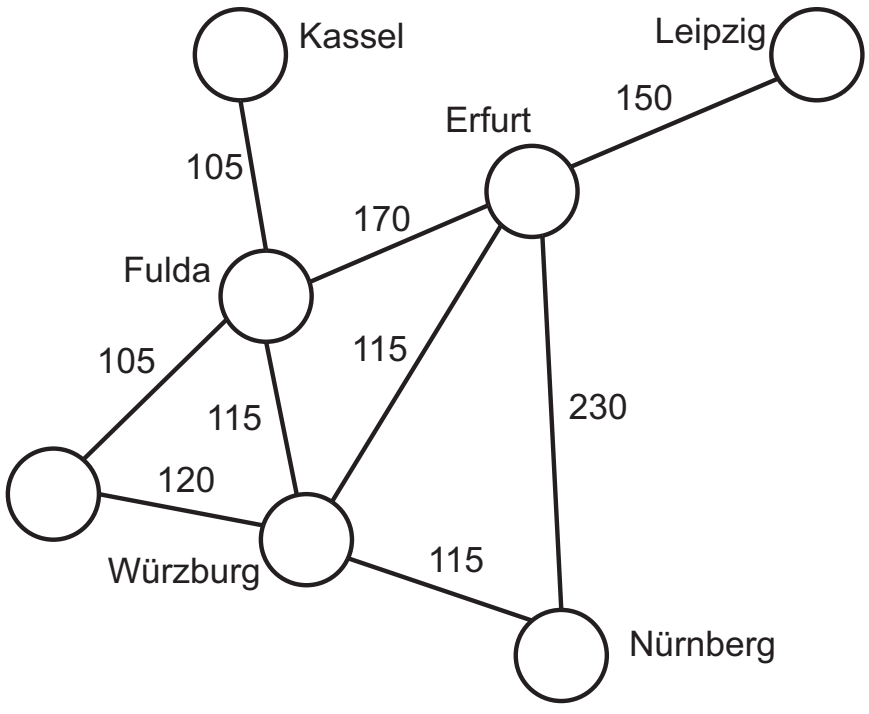
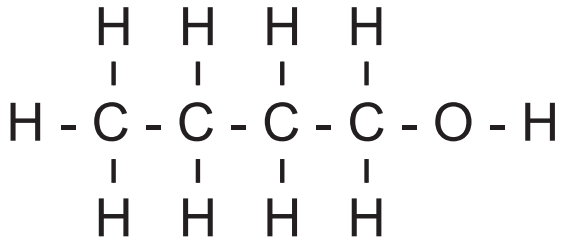
Einführung in die Objektorientierte Programmierung
Vorlesung 19: Graphen und Bäume

Sebastian Küpper

Graphen als Modell für nicht-lineare Strukturen





Definition des Graphen

Definition (Graph)

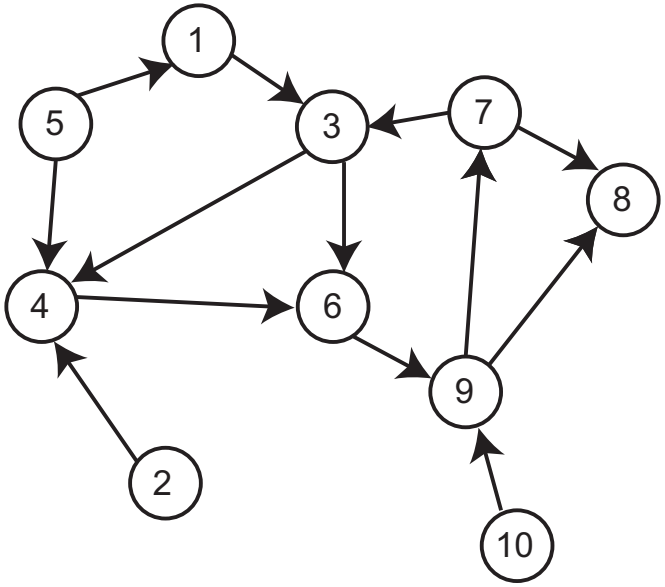
Ein Graph G ist ein Tupel:

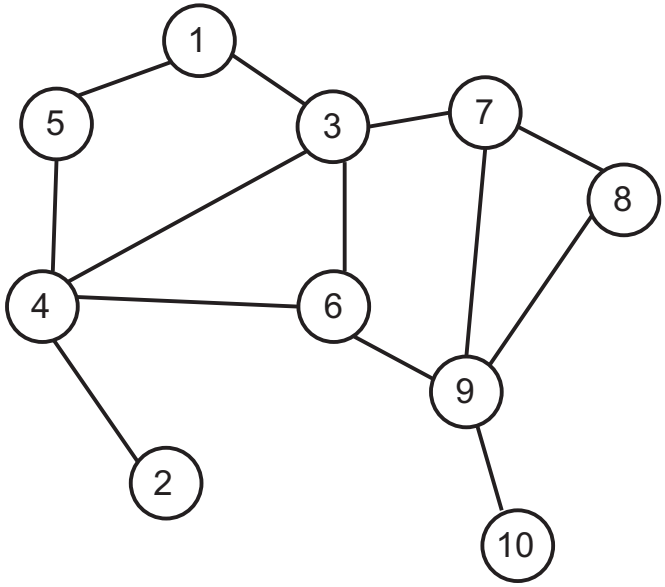
$$G = (V, E)$$

wobei

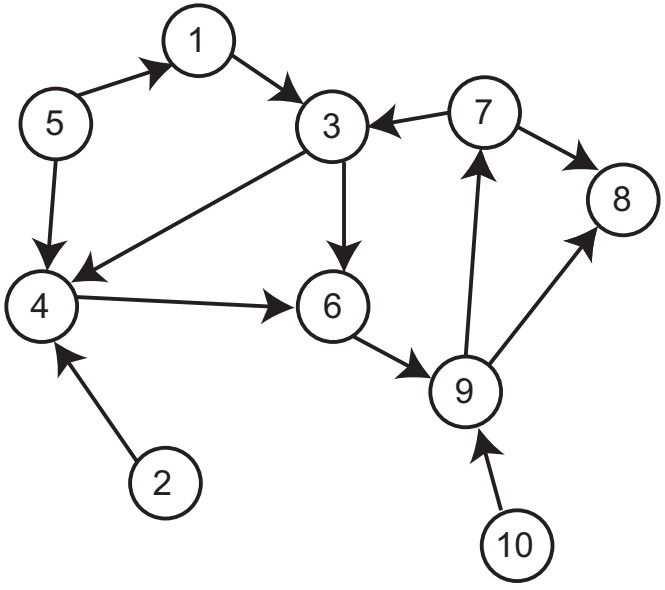
- V, die Menge der Knoten und
- $E \subseteq V \times V$, die Menge der Kanten ist, eine Kante ist also ein Paar von Knoten.

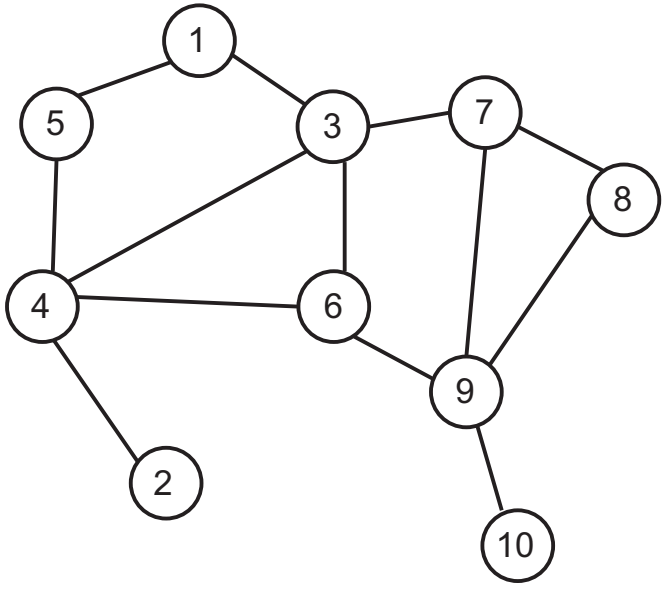
Gerichtete und ungerichtete Graphen





Gerichtete und ungerichtete Graphen





eingehende Kanten
Eingangsgrad
ungerichteter Kantenzug
gerichteter Kantenzug
Kreis / Zyklus

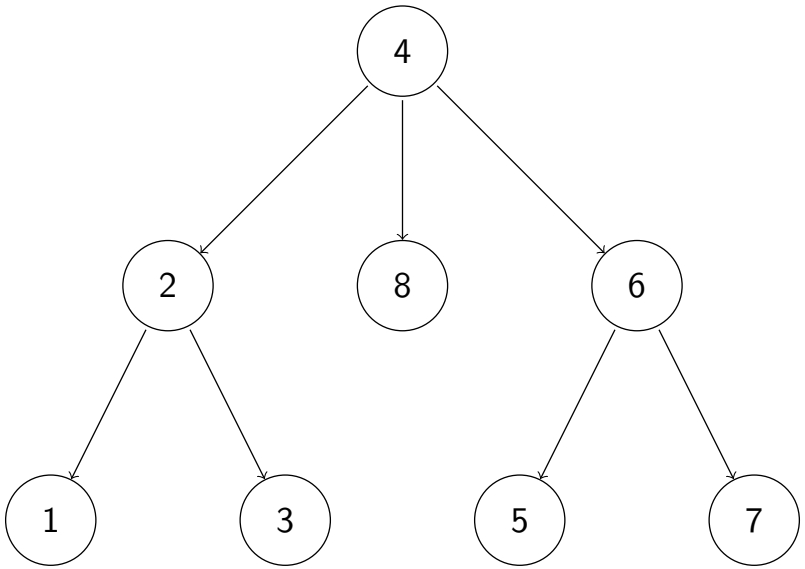
ausgehende Kanten
Ausgangsgrad
ungerichteter Pfad
gerichteter Pfad
DAG

Grad
Zusammenhang
starker Zusammenhang

Baum

Definition (Baum)

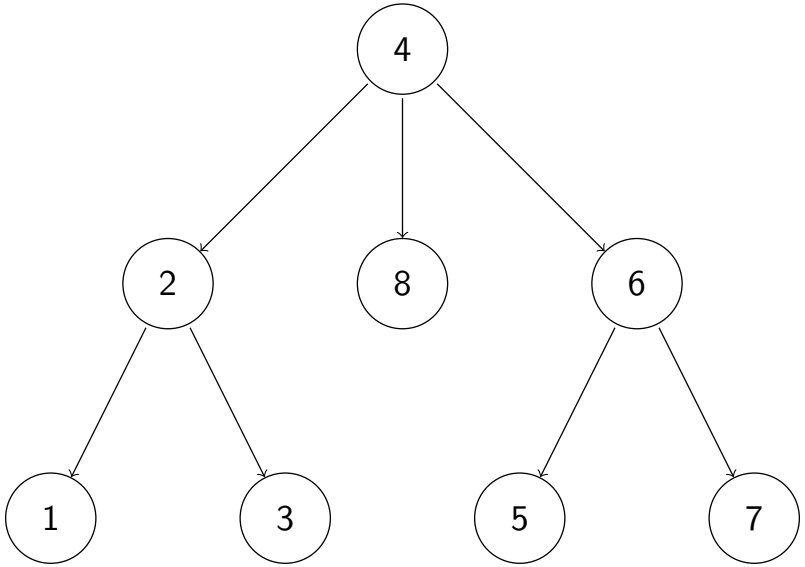
Ein Baum ist (schwach) zusammenhängender DAG in dem jeder Knoten maximal Eingangsgrad 1 hat.



Baum

Definition (Baum)

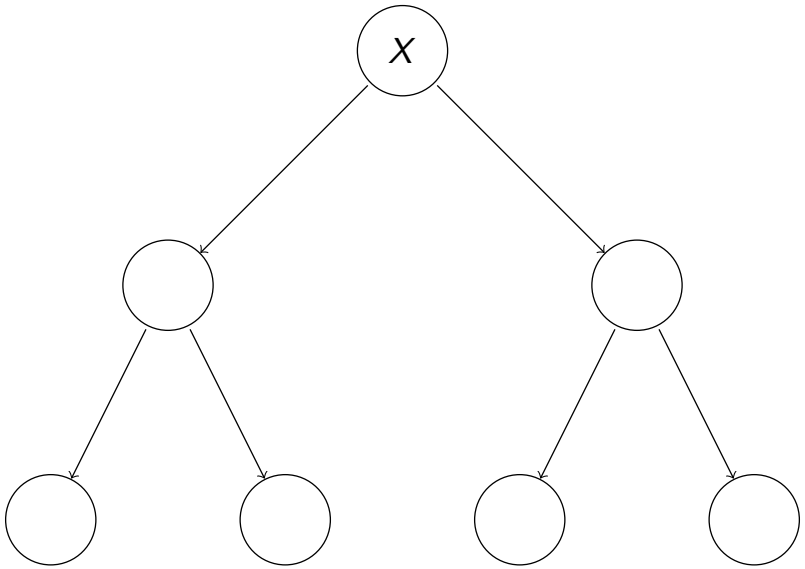
Ein Baum ist (schwach) zusammenhängender DAG in dem jeder Knoten maximal Eingangsgrad 1 hat.



Wurzel	Blatt	Höhe
Elternknoten	Kindknoten	Geschwisterknoten
Vorgänger	Nachfolger	Unterbaum
binärer Baum	balancierter Baum	Suchbaum

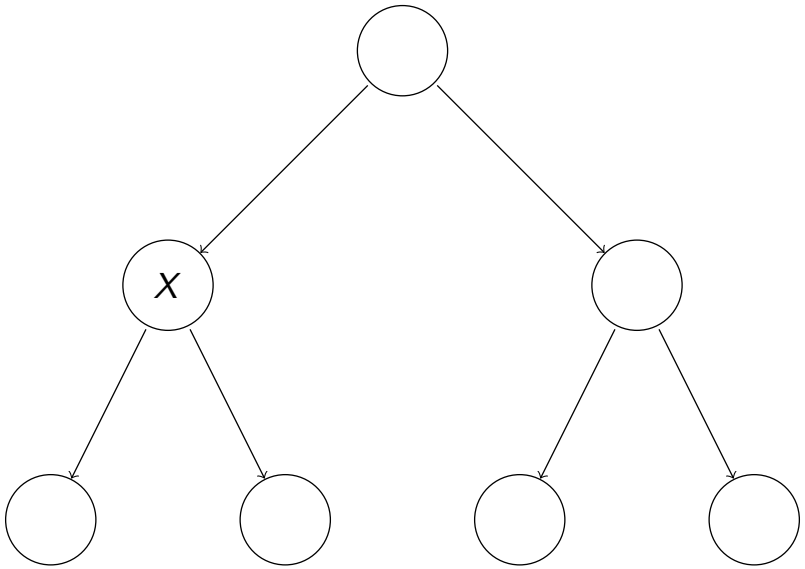
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



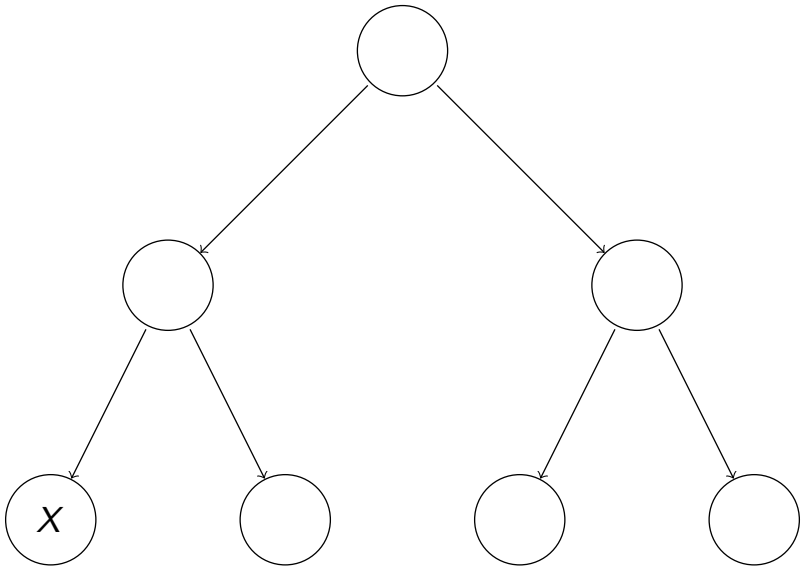
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



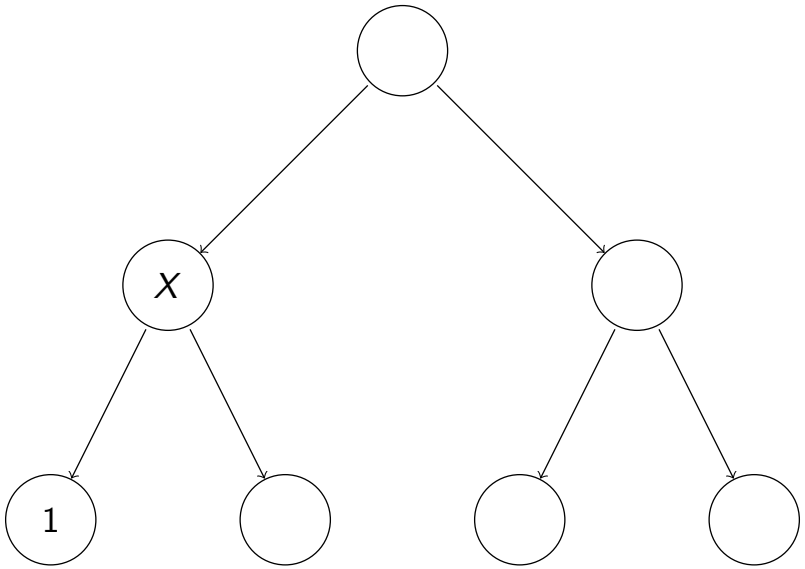
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



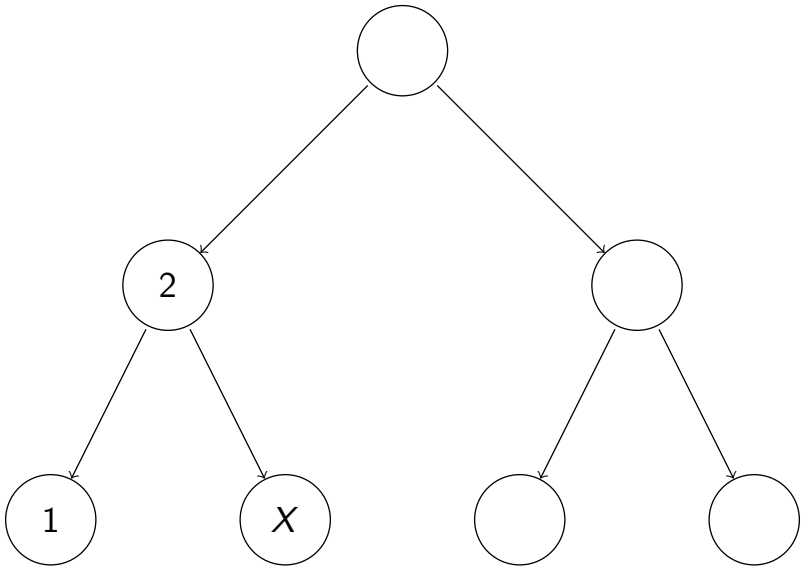
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



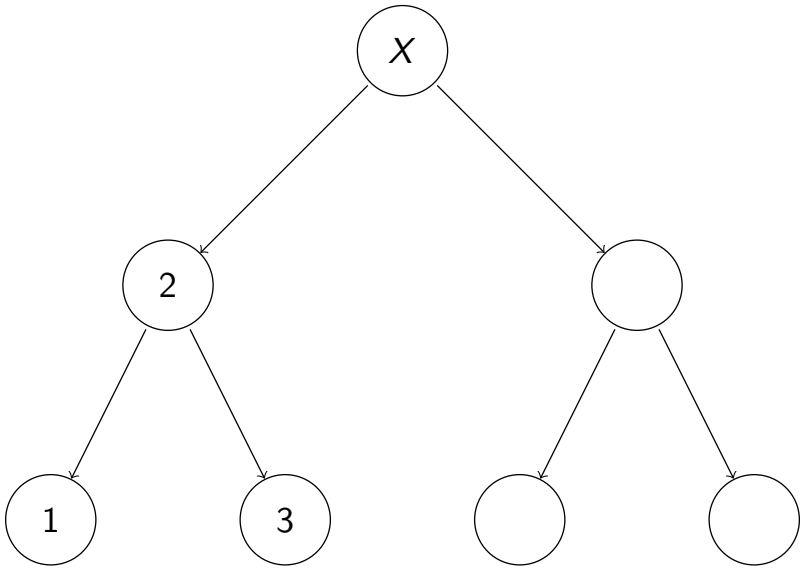
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



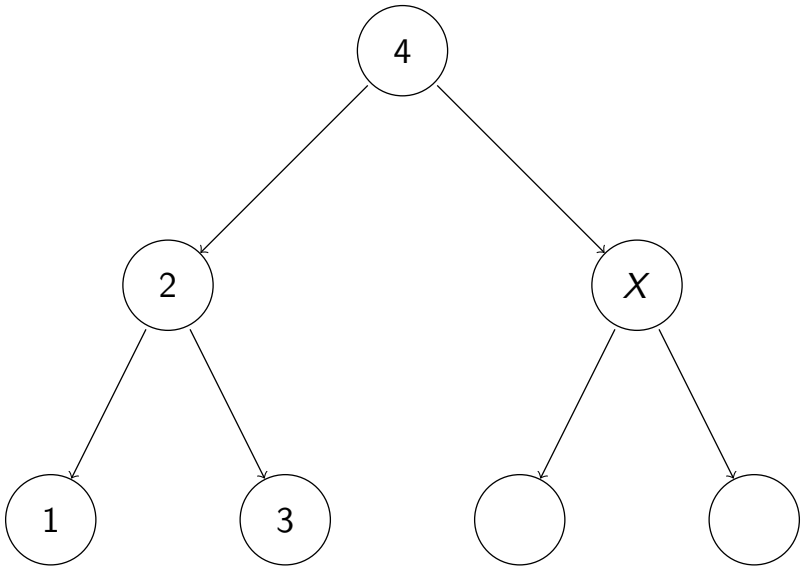
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



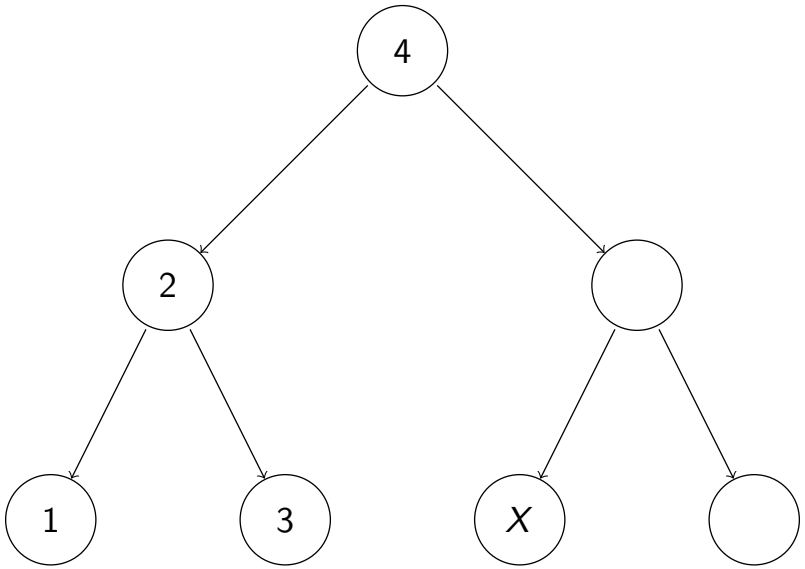
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



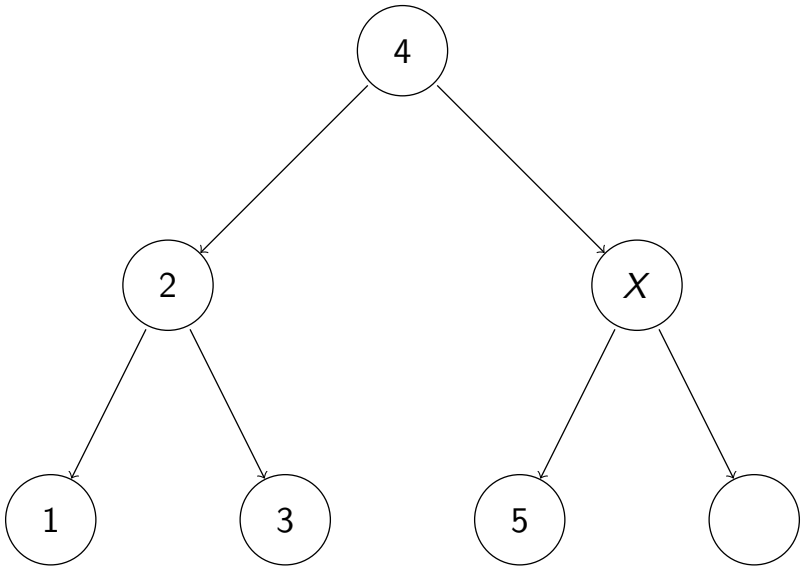
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



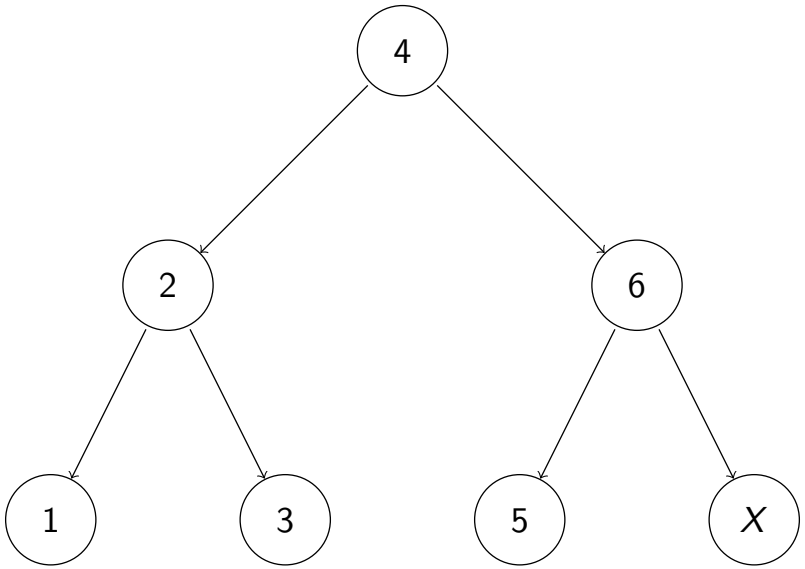
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



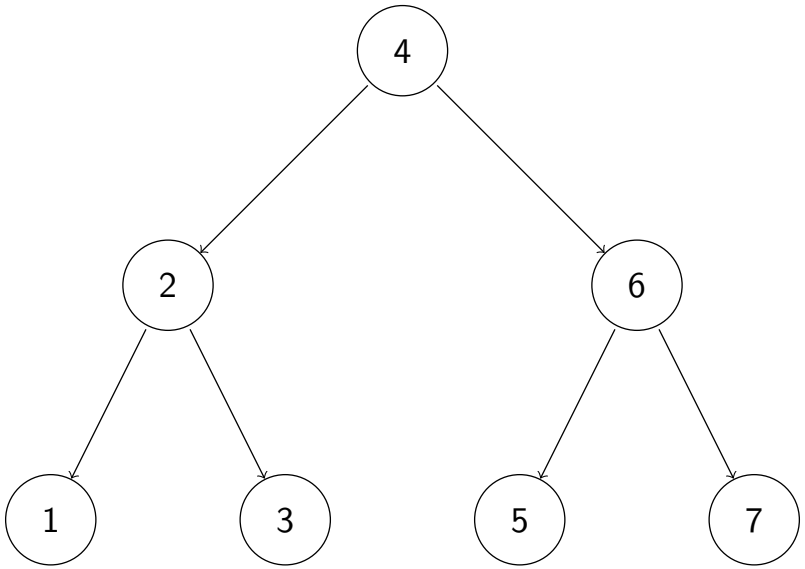
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



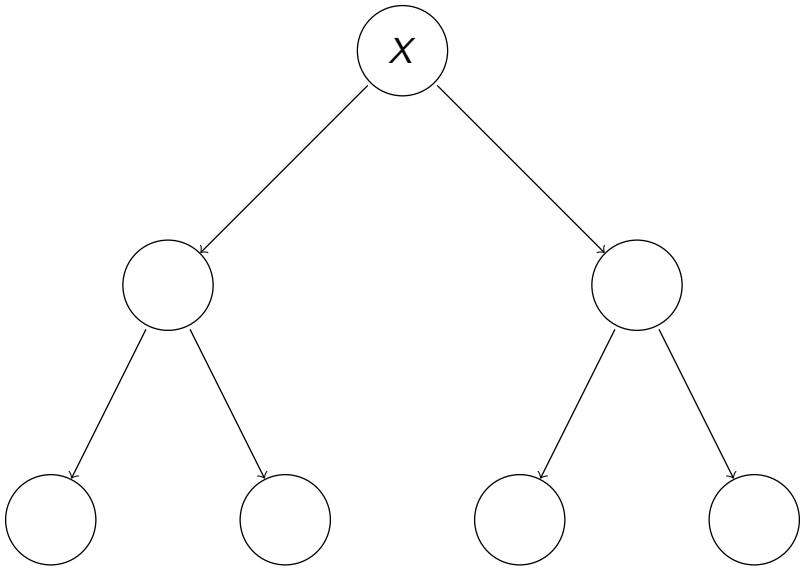
Durchlaufstrategien für Bäume: In-Order

Wir besuchen zuerst den linken Nachfolger, anschließend verarbeiten wir den Eintrag des aktuellen Knotens, danach besuchen wir den rechten Nachfolger.



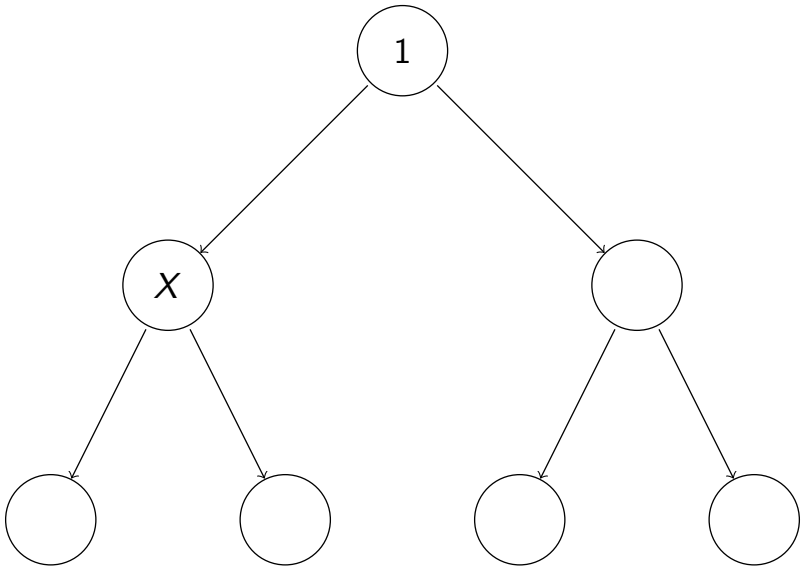
Durchlaufstrategien für Bäume: Pre-Order

Wir verarbeiten zuerst den Eintrag des aktuellen Knotens, anschließend besuchen wir den linken Nachfolger, danach den rechten Nachfolger



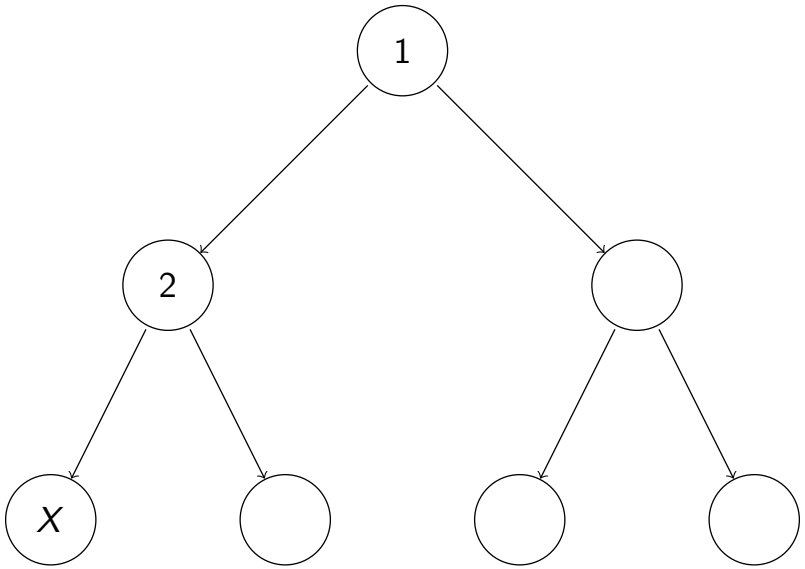
Durchlaufstrategien für Bäume: Pre-Order

Wir verarbeiten zuerst den Eintrag des aktuellen Knotens, anschließend besuchen wir den linken Nachfolger, danach den rechten Nachfolger



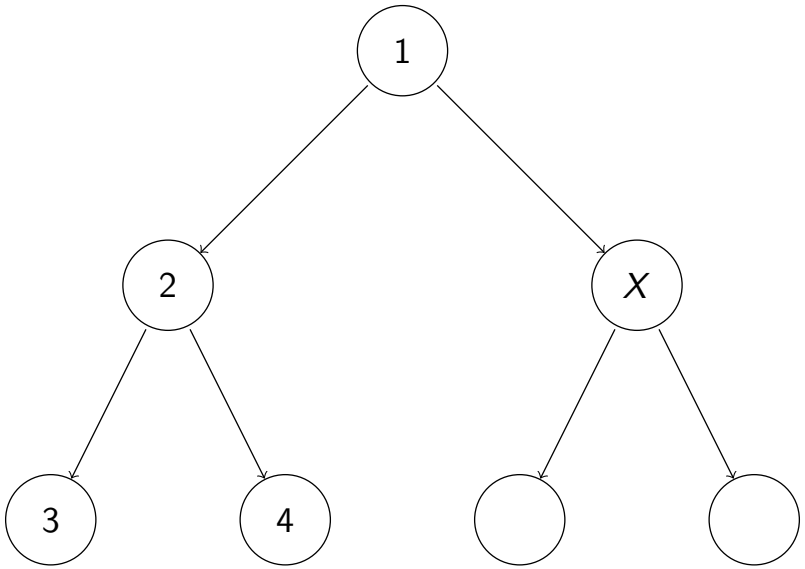
Durchlaufstrategien für Bäume: Pre-Order

Wir verarbeiten zuerst den Eintrag des aktuellen Knotens, anschließend besuchen wir den linken Nachfolger, danach den rechten Nachfolger



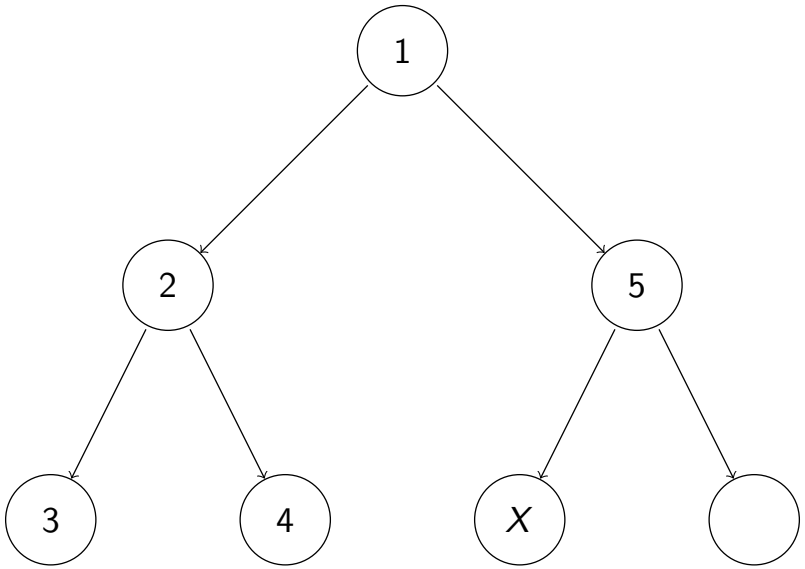
Durchlaufstrategien für Bäume: Pre-Order

Wir verarbeiten zuerst den Eintrag des aktuellen Knotens, anschließend besuchen wir den linken Nachfolger, danach den rechten Nachfolger



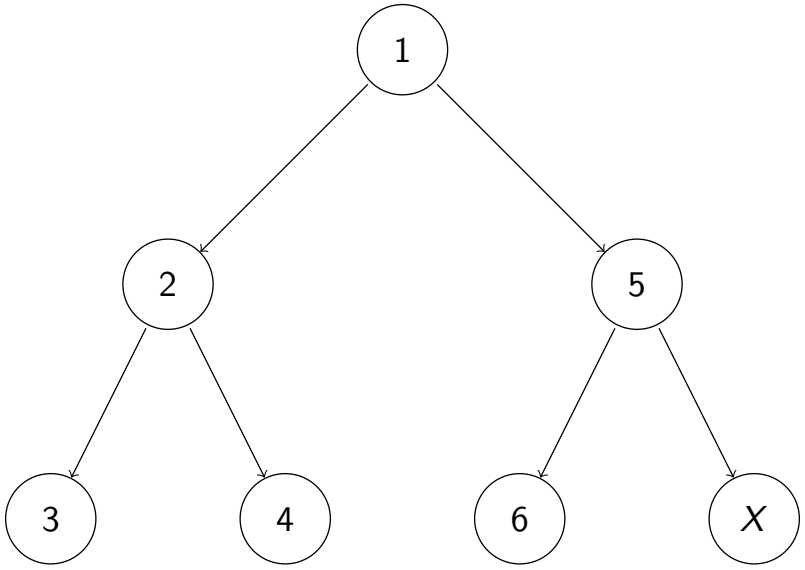
Durchlaufstrategien für Bäume: Pre-Order

Wir verarbeiten zuerst den Eintrag des aktuellen Knotens, anschließend besuchen wir den linken Nachfolger, danach den rechten Nachfolger



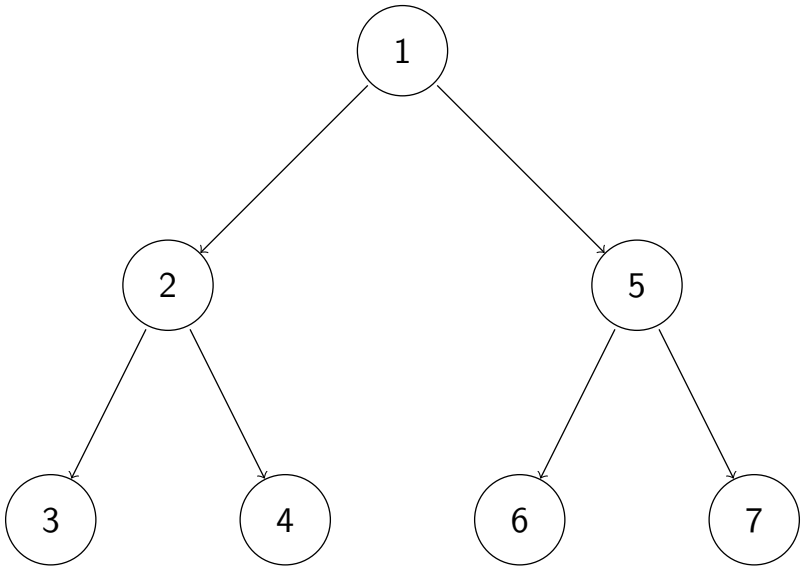
Durchlaufstrategien für Bäume: Pre-Order

Wir verarbeiten zuerst den Eintrag des aktuellen Knotens, anschließend besuchen wir den linken Nachfolger, danach den rechten Nachfolger



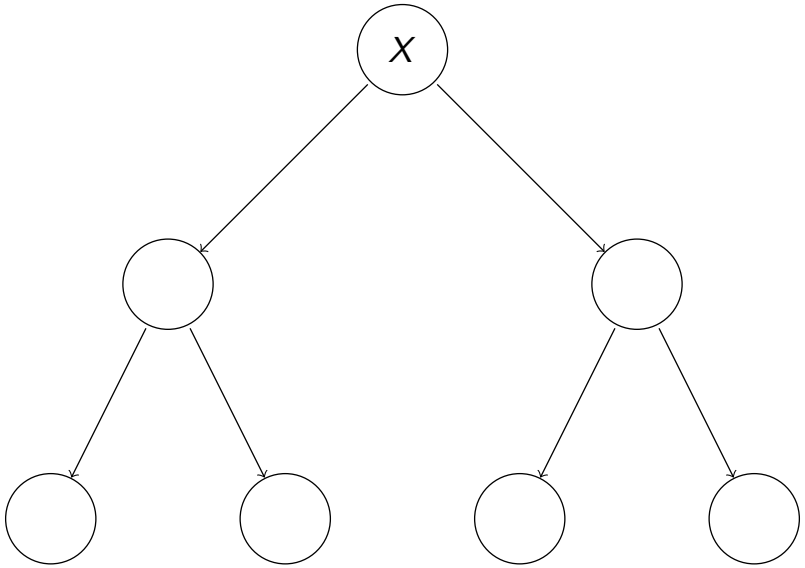
Durchlaufstrategien für Bäume: Pre-Order

Wir verarbeiten zuerst den Eintrag des aktuellen Knotens, anschließend besuchen wir den linken Nachfolger, danach den rechten Nachfolger



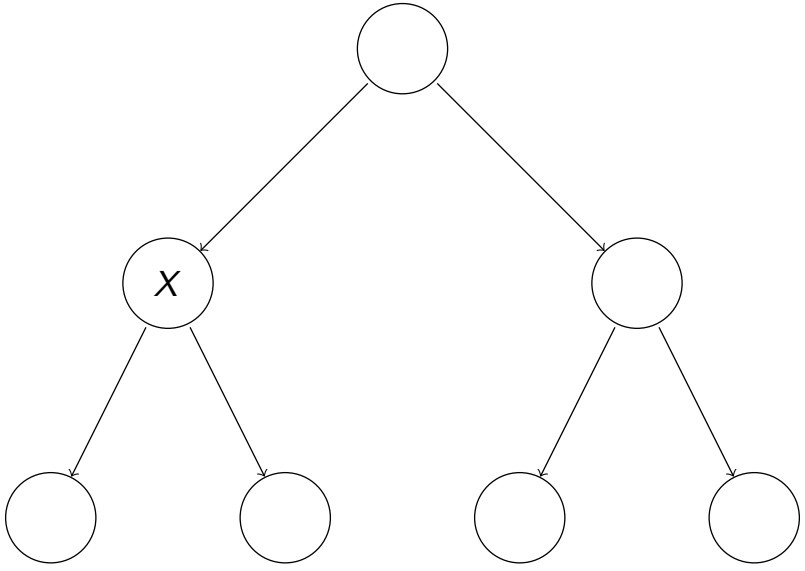
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



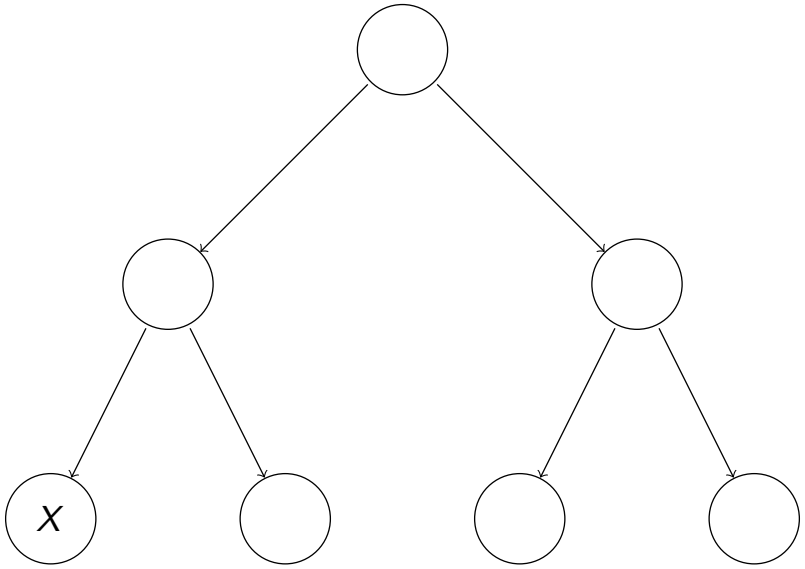
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



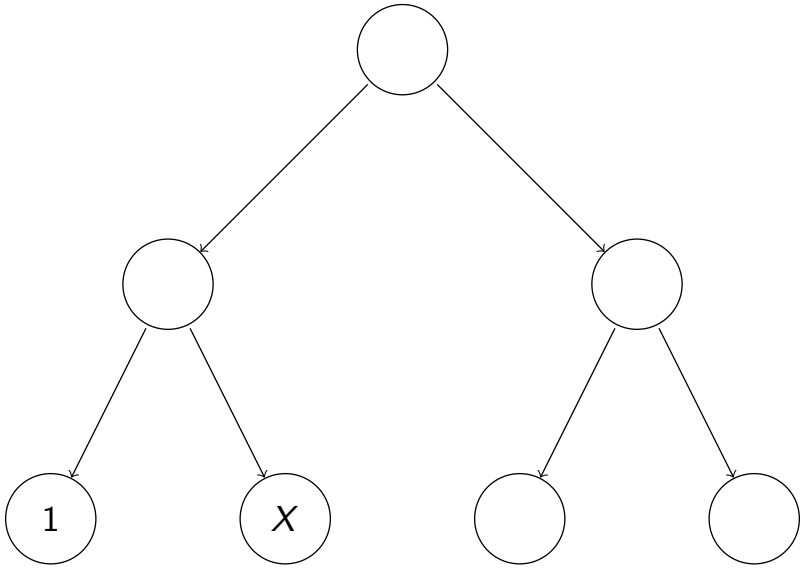
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



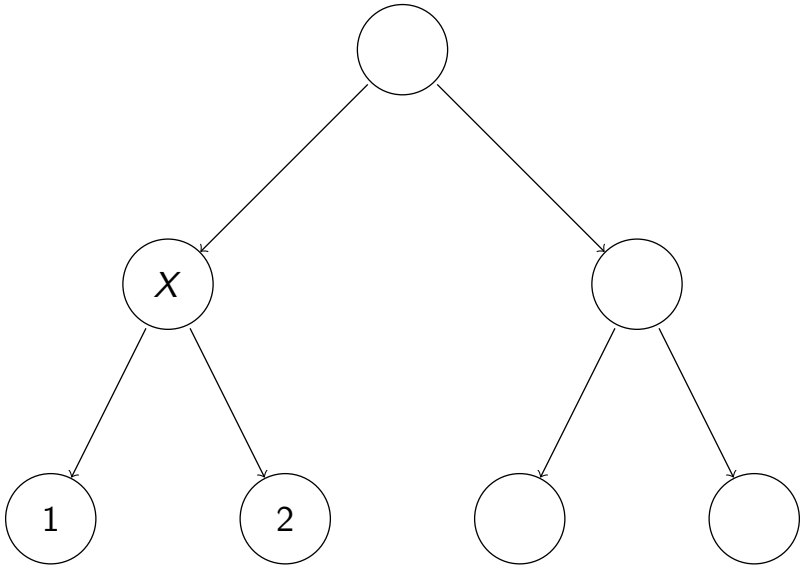
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



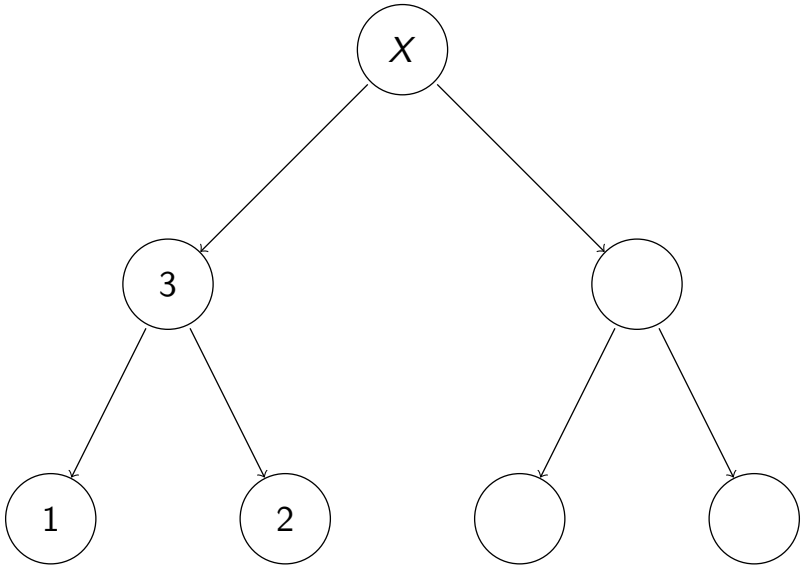
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



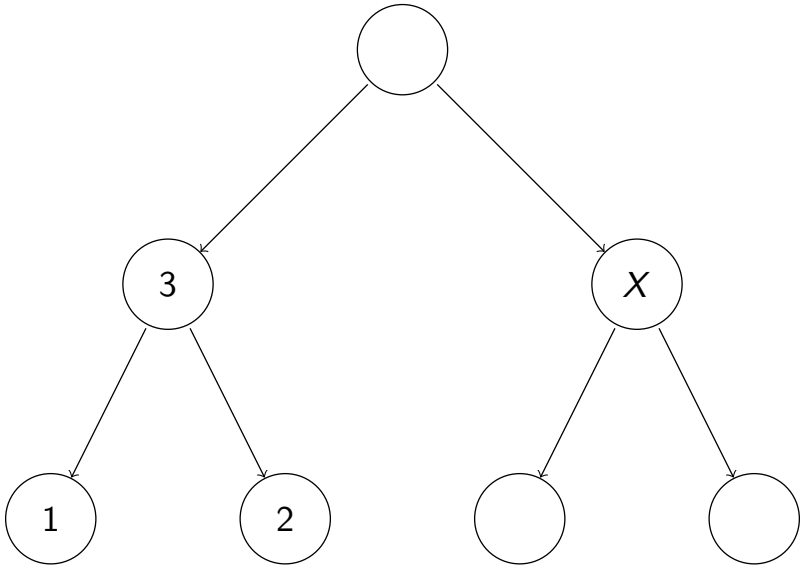
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



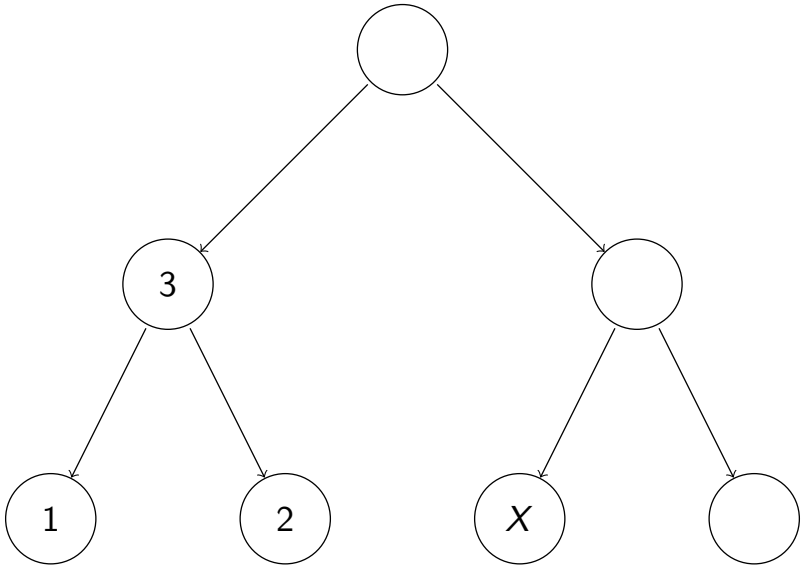
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



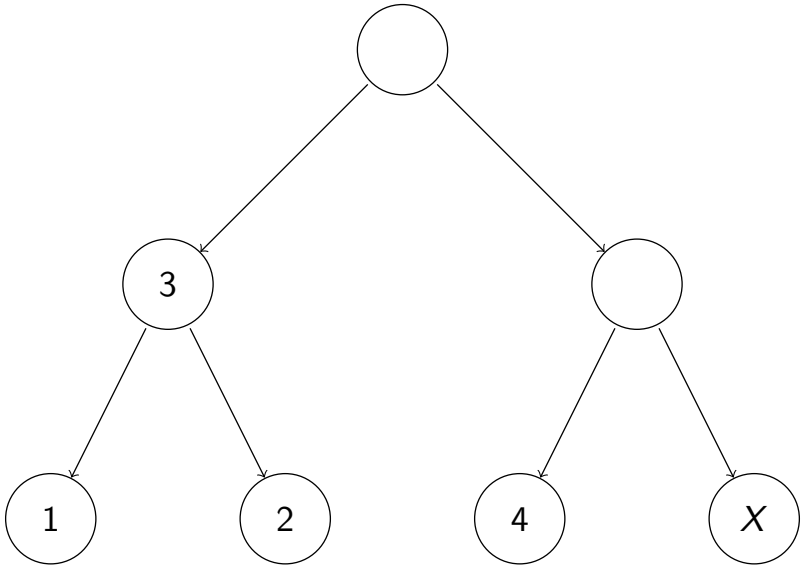
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



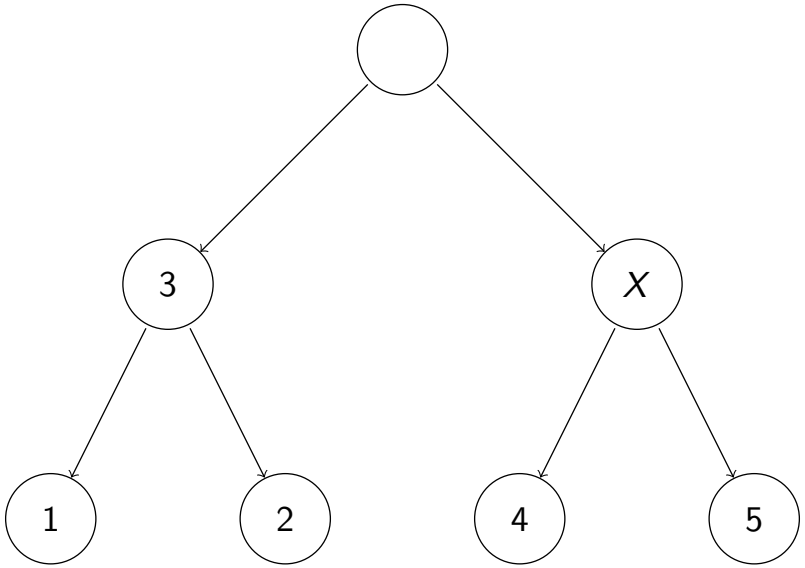
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



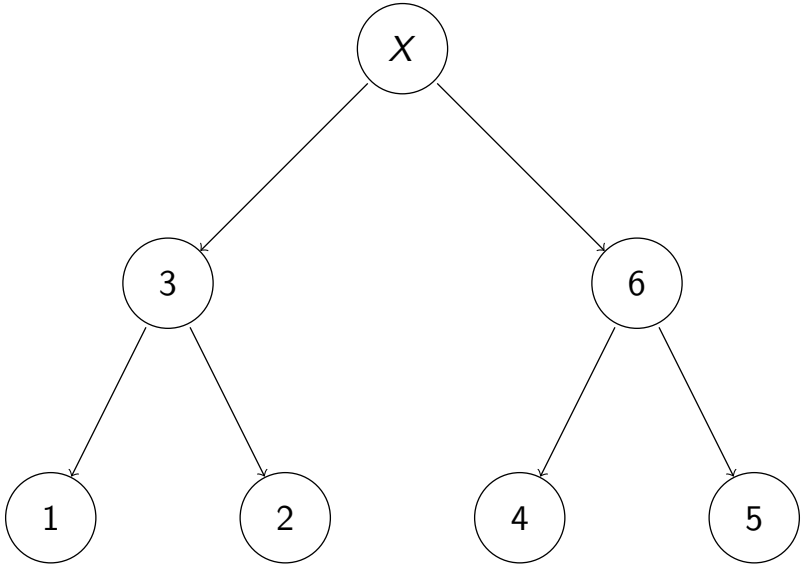
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



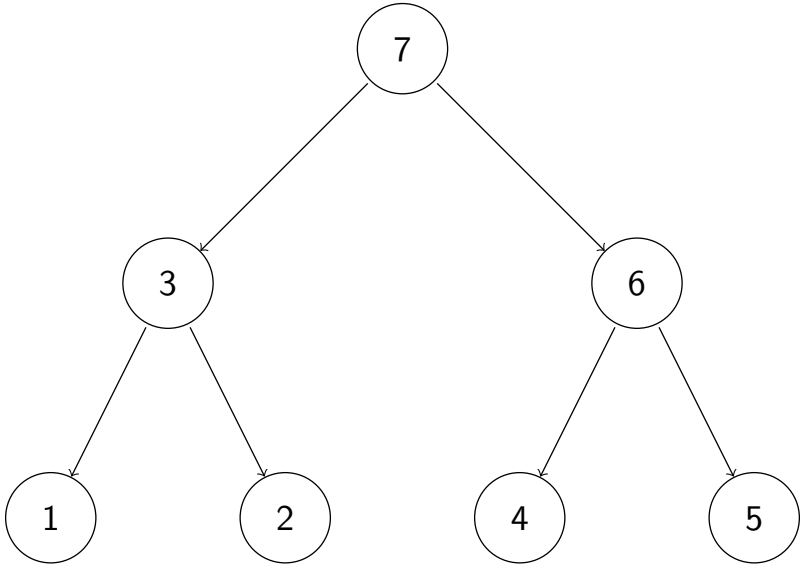
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



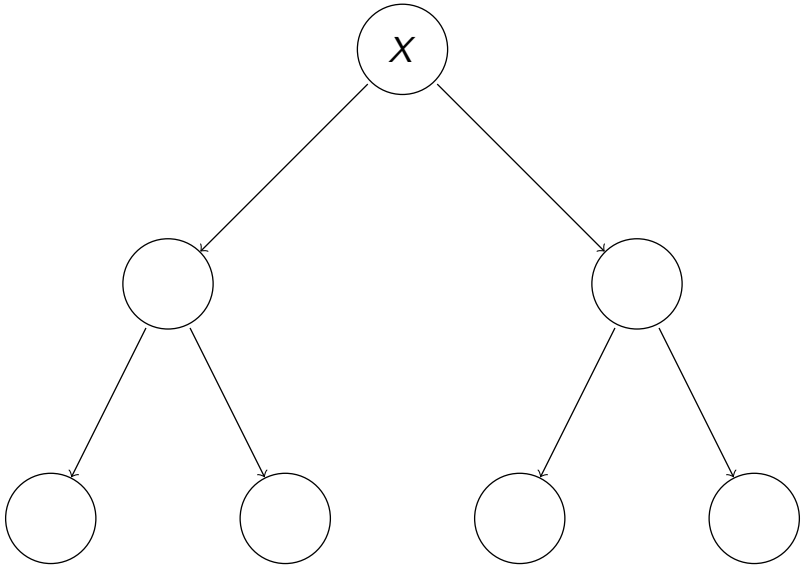
Durchlaufstrategien für Bäume: Post-Order

Wir besuchen zuerst den linken Nachfolger, danach den rechten Nachfolger und zuletzt verarbeiten wir den Eintrag des aktuellen Knotens



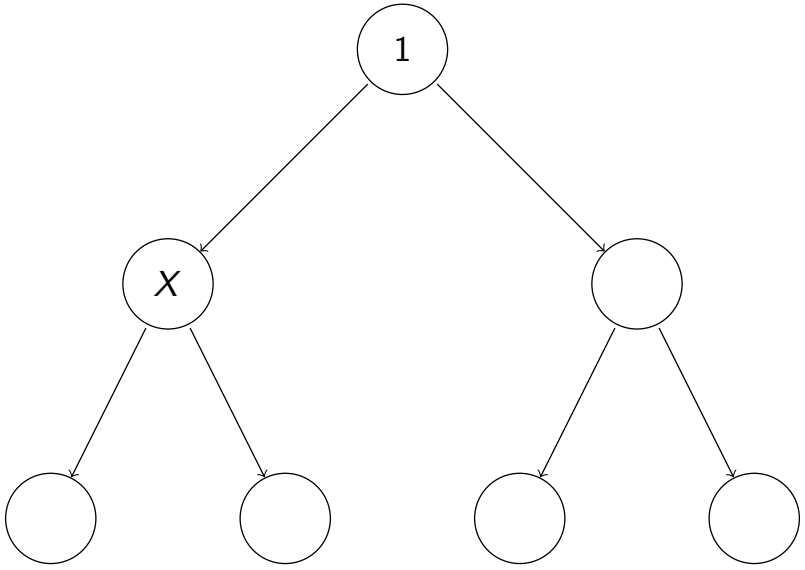
Durchlaufstrategien für Bäume: Level-Order

Ebenenweiser Besuch der Knoten: Wir besuchen erst alle Knoten der ersten Ebene, dann alle Knoten der zweiten Ebene...



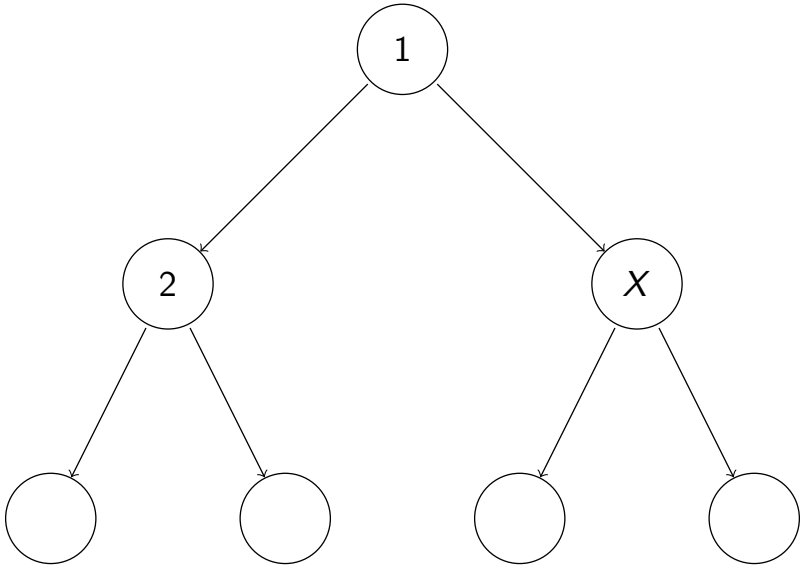
Durchlaufstrategien für Bäume: Level-Order

Ebenenweiser Besuch der Knoten: Wir besuchen erst alle Knoten der ersten Ebene, dann alle Knoten der zweiten Ebene...



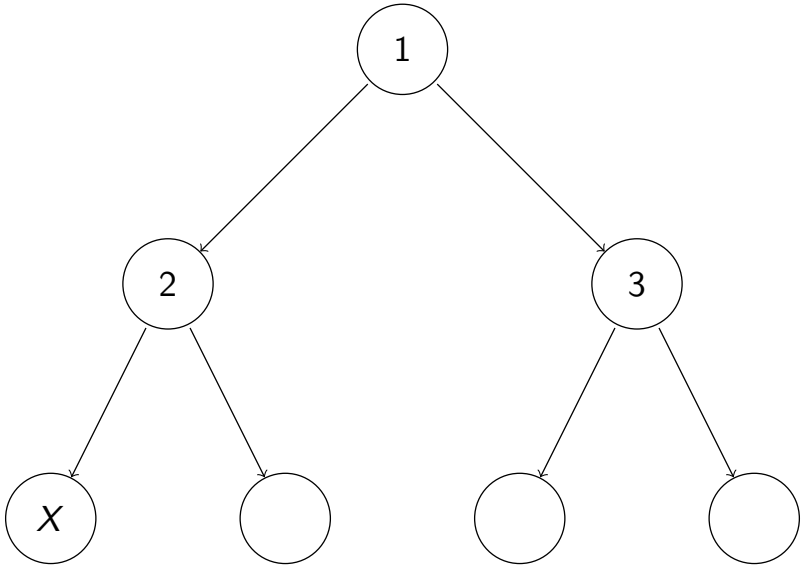
Durchlaufstrategien für Bäume: Level-Order

Ebenenweiser Besuch der Knoten: Wir besuchen erst alle Knoten der ersten Ebene, dann alle Knoten der zweiten Ebene...



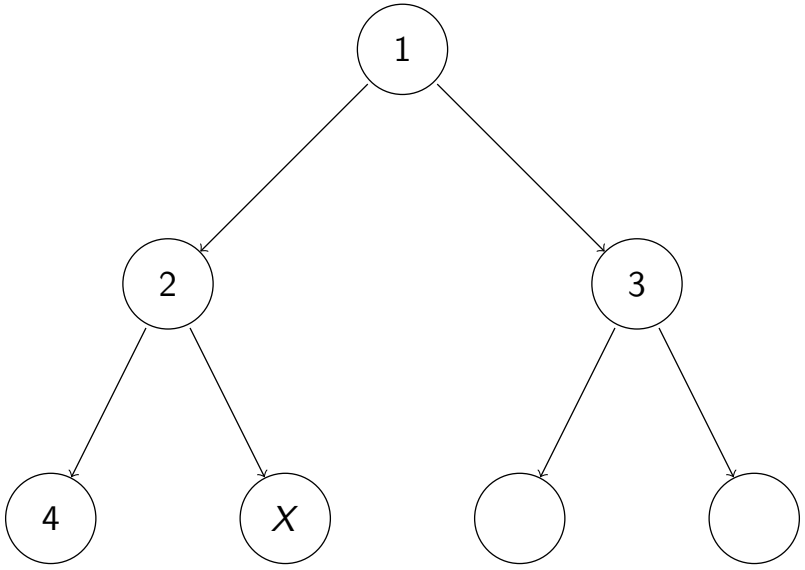
Durchlaufstrategien für Bäume: Level-Order

Ebenenweiser Besuch der Knoten: Wir besuchen erst alle Knoten der ersten Ebene, dann alle Knoten der zweiten Ebene...



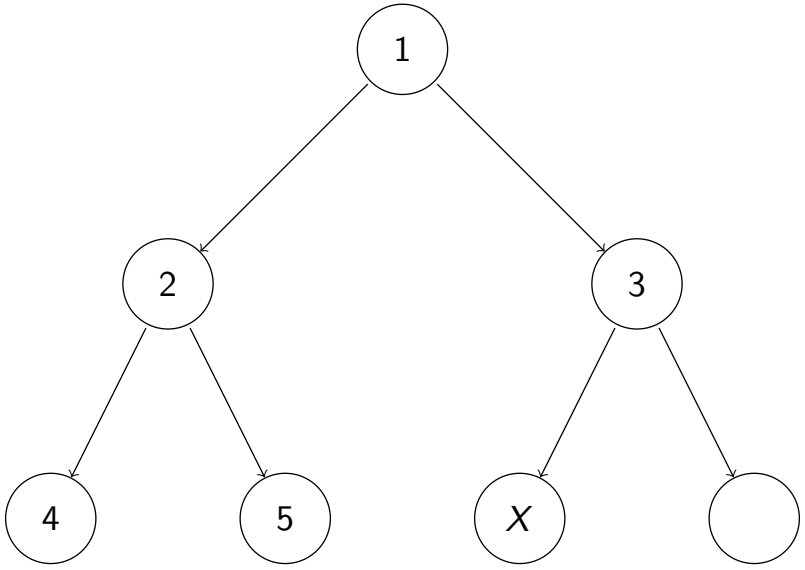
Durchlaufstrategien für Bäume: Level-Order

Ebenenweiser Besuch der Knoten: Wir besuchen erst alle Knoten der ersten Ebene, dann alle Knoten der zweiten Ebene...



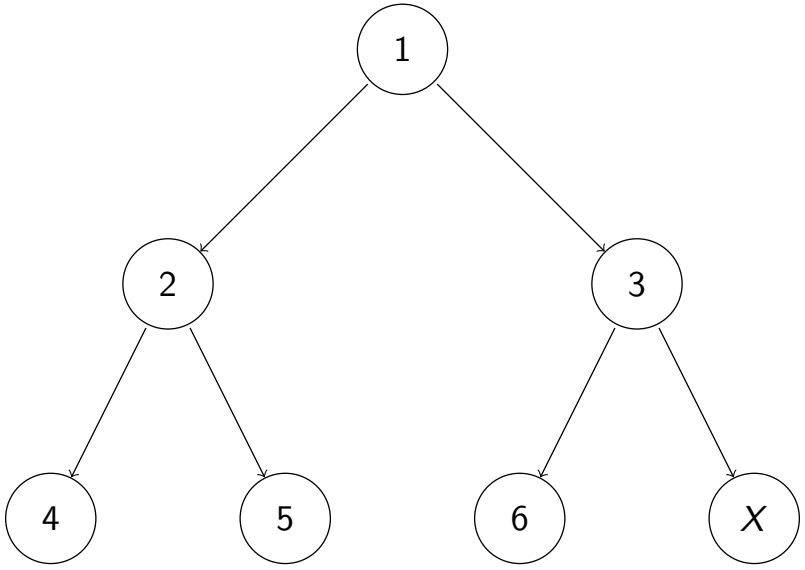
Durchlaufstrategien für Bäume: Level-Order

Ebenenweiser Besuch der Knoten: Wir besuchen erst alle Knoten der ersten Ebene, dann alle Knoten der zweiten Ebene...



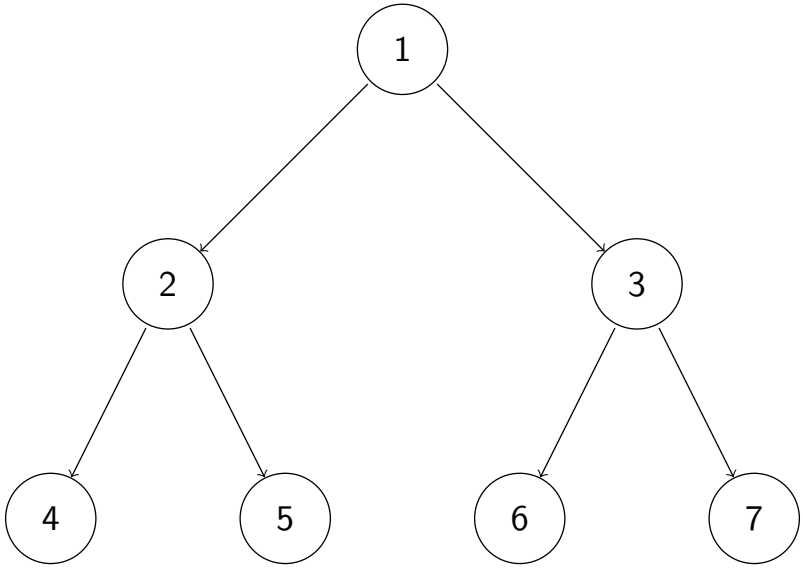
Durchlaufstrategien für Bäume: Level-Order

Ebenenweiser Besuch der Knoten: Wir besuchen erst alle Knoten der ersten Ebene, dann alle Knoten der zweiten Ebene...



Durchlaufstrategien für Bäume: Level-Order

Ebenenweiser Besuch der Knoten: Wir besuchen erst alle Knoten der ersten Ebene, dann alle Knoten der zweiten Ebene...



Implementierung: Binärbaumknoten

```
public class BinaryTreeNode {
   protected int entry;
   protected BinaryTreeNode leftChild;
   protected BinaryTreeNode rightChild;
   public BinaryTreeNode(int e) {
      this.entry = e;
   }
   public BinaryTreeNode(int e, BinaryTreeNode left,
       BinaryTreeNode right) {
      this(e);
      this.leftChild = left;
      this.rightChild = right;
   }
}
```

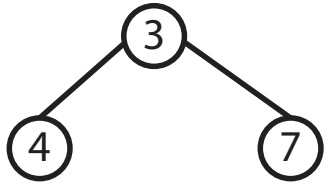
Hier unterschlagen: Getter und Setter für `leftChild` und `rightChild`.

Implementierung: Binärbaum

```
public class BinaryTree {
   private BinaryTreeNode root;
   public BinaryTree() {
   }
   public BinaryTree(BinaryTreeNode root) {
      this.root = root;
   }
   public BinaryTreeNode getRoot() {
      return root;
   }
}
```

Beispiel: Binärbaum erzeugen

```
BinaryTreeNode a = new BinaryTreeNode(4);
BinaryTreeNode b = new BinaryTreeNode(7);
BinaryTreeNode c = new BinaryTreeNode(3, a, b);
BinaryTree t = new BinaryTree(c);
```



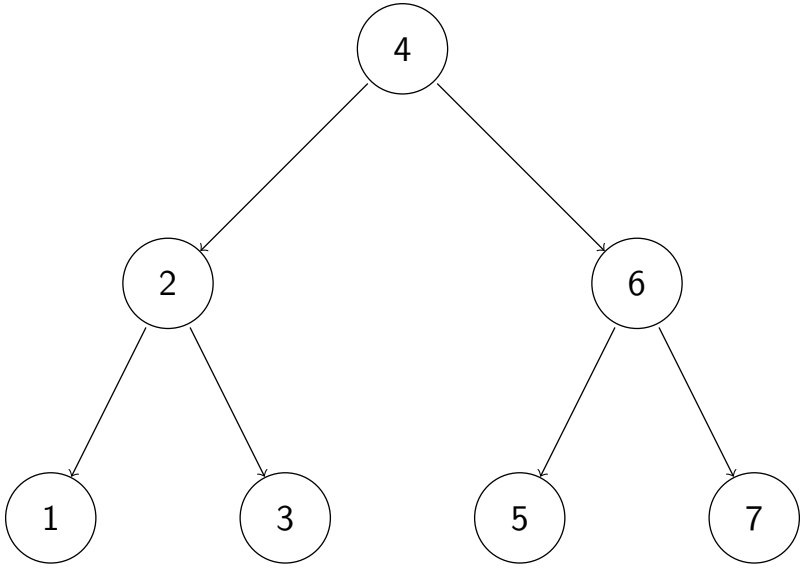
Implementierung: In-Order

```
public class BinaryTree {
private BinaryTreeNode root;
   ...
   public void printPreorder() {
      printPreorder(root); // start with root
   }
   private void printInorder(BinaryTreeNode tn) {
      if (tn == null) return; // base case: empty subtree
      printInorder(tn.getLeftChild()); // visit left child
      System.out.print(tn.getEntry() + " "); // visit
          current node
      printInorder(tn.getRightChild()); // visit right
          child
   }
}
```

Binärer Suchbaum und In-Order-Traversierung

Definition (Binärer Suchbaum)

Ein binärer Suchbaum ist ein binärer Baum (Ausgangsgrad eines jeden Knotens maximal 2) so dass für jeden Knoten v mit linkem Kind l gilt: $\text{label}(v) > \text{label}(l)$ und für jeden Knoten v mit linkem Kind r gilt: $\text{label}(v) < \text{label}(r)$



Einfügen in Suchbaum

Wir wollen n einfügen.

- Falls Baum leer: Das Element ist die Wurzel des Baums
- Sonst vergleiche Label m des aktuellen Knotens mit n

Einfügen in Suchbaum

Wir wollen n einfügen.

- Falls Baum leer: Das Element ist die Wurzel des Baums
- Sonst vergleiche Label m des aktuellen Knotens mit n
  - Falls $n < m$ füge n in den linken Teilbaum ein
  - Falls $n > m$ füge n in den rechten Teilbaum ein

Beispiel: Schrittweiser Aufbau von Suchbaum

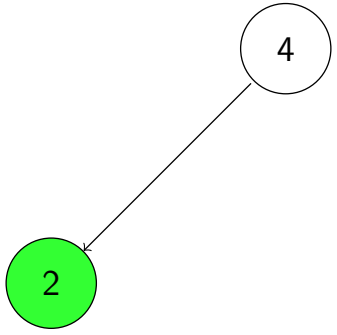
Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:

Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:

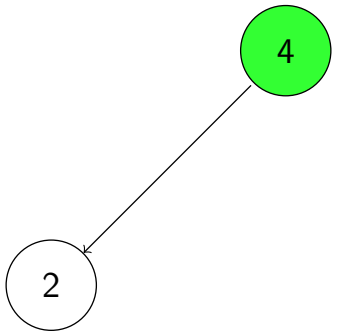
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



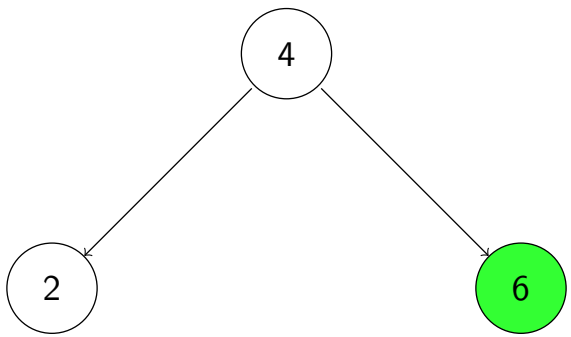
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



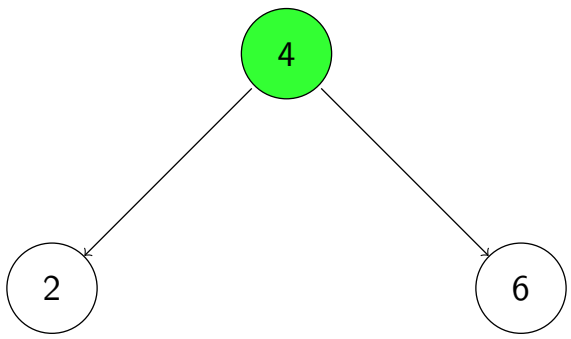
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



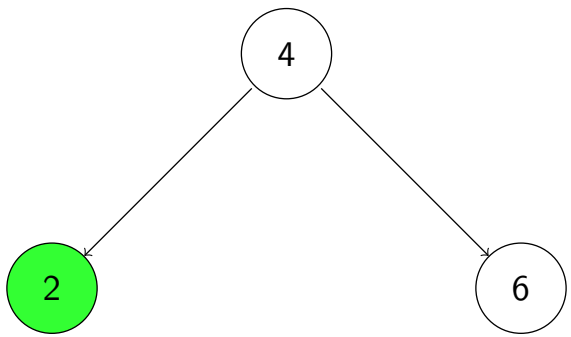
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



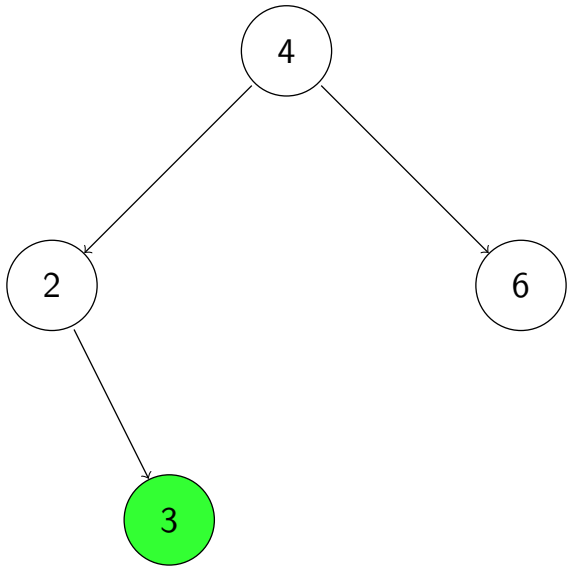
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



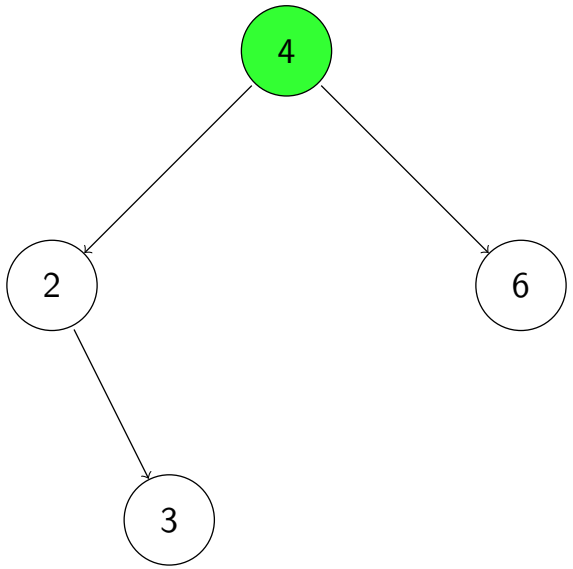
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



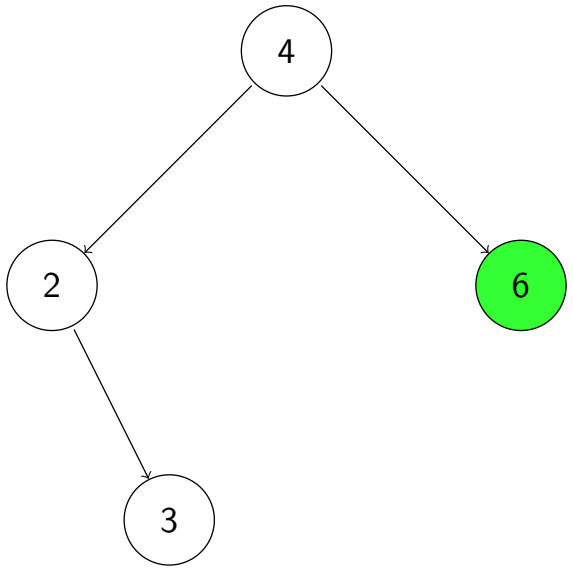
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



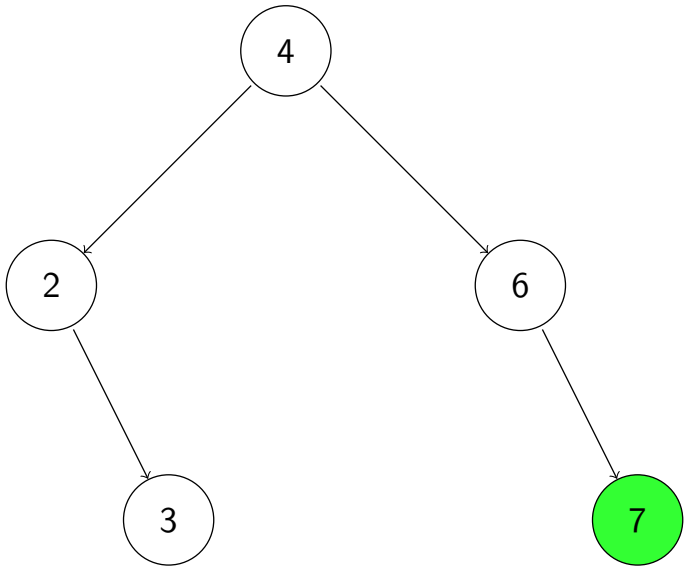
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



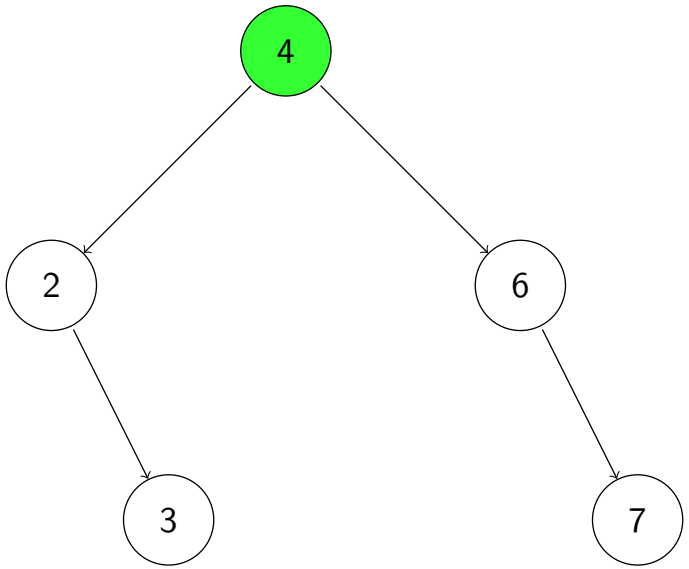
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



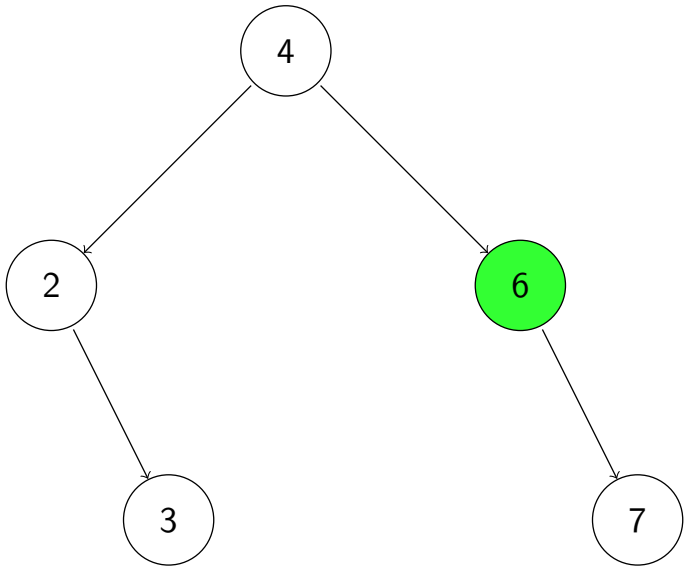
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



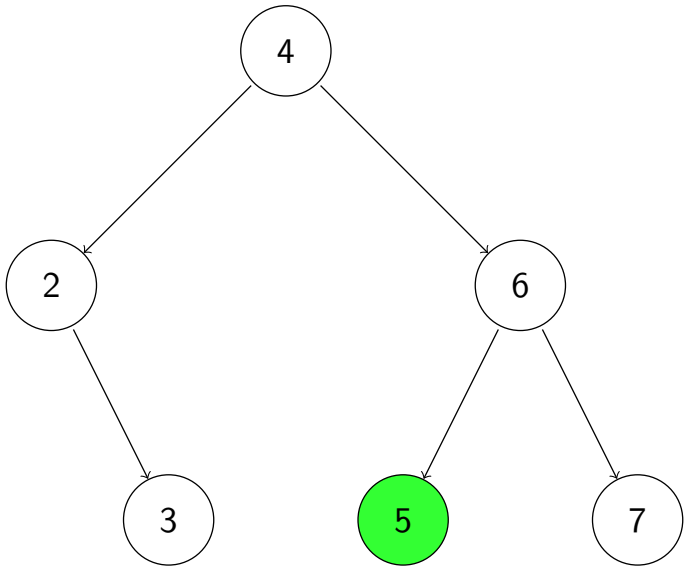
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



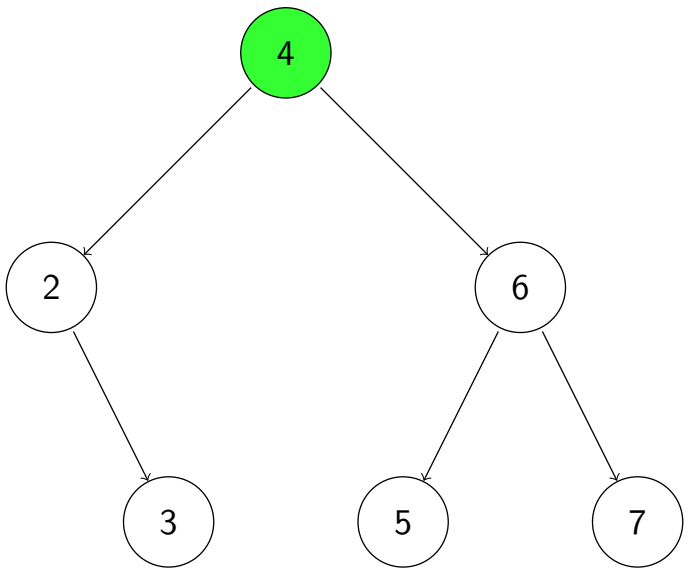
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



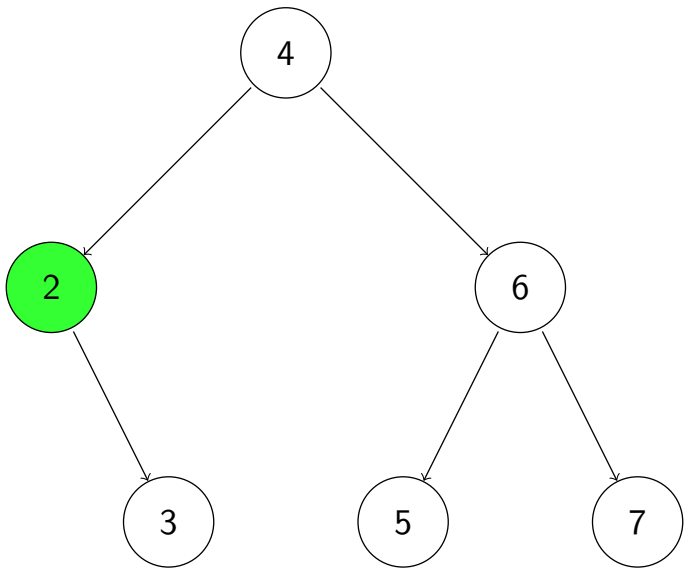
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



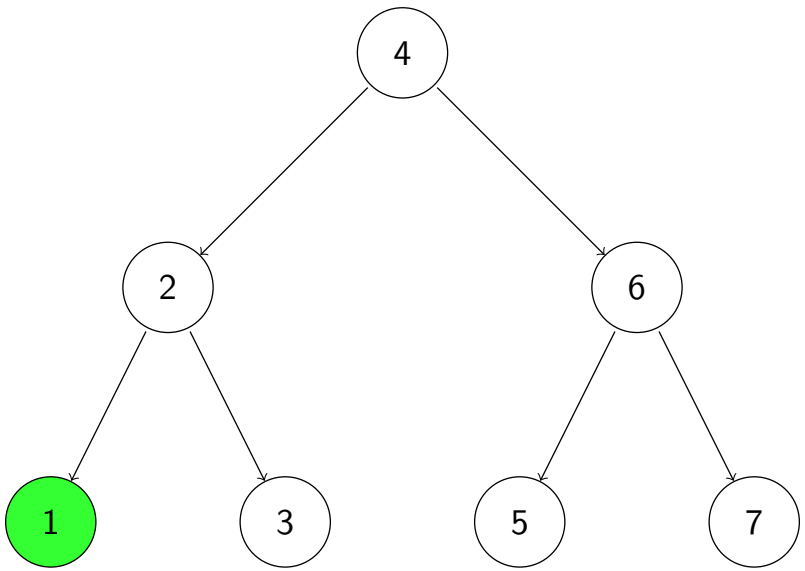
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



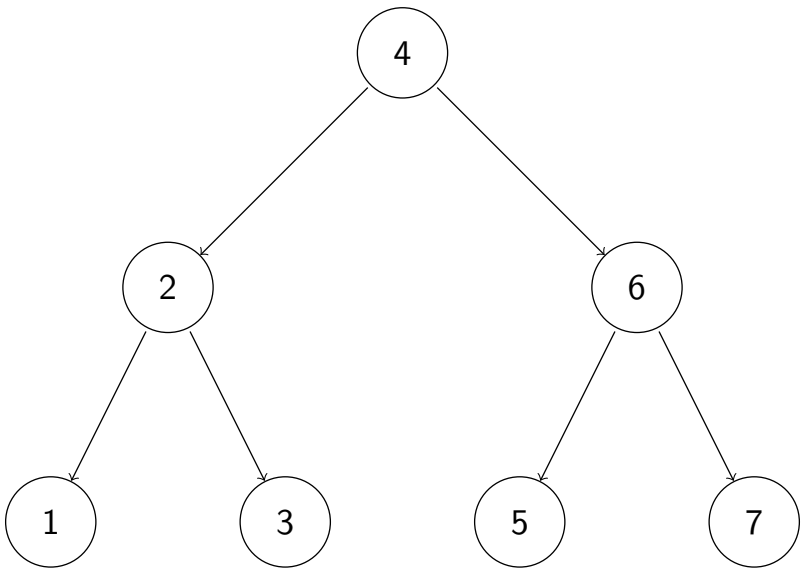
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



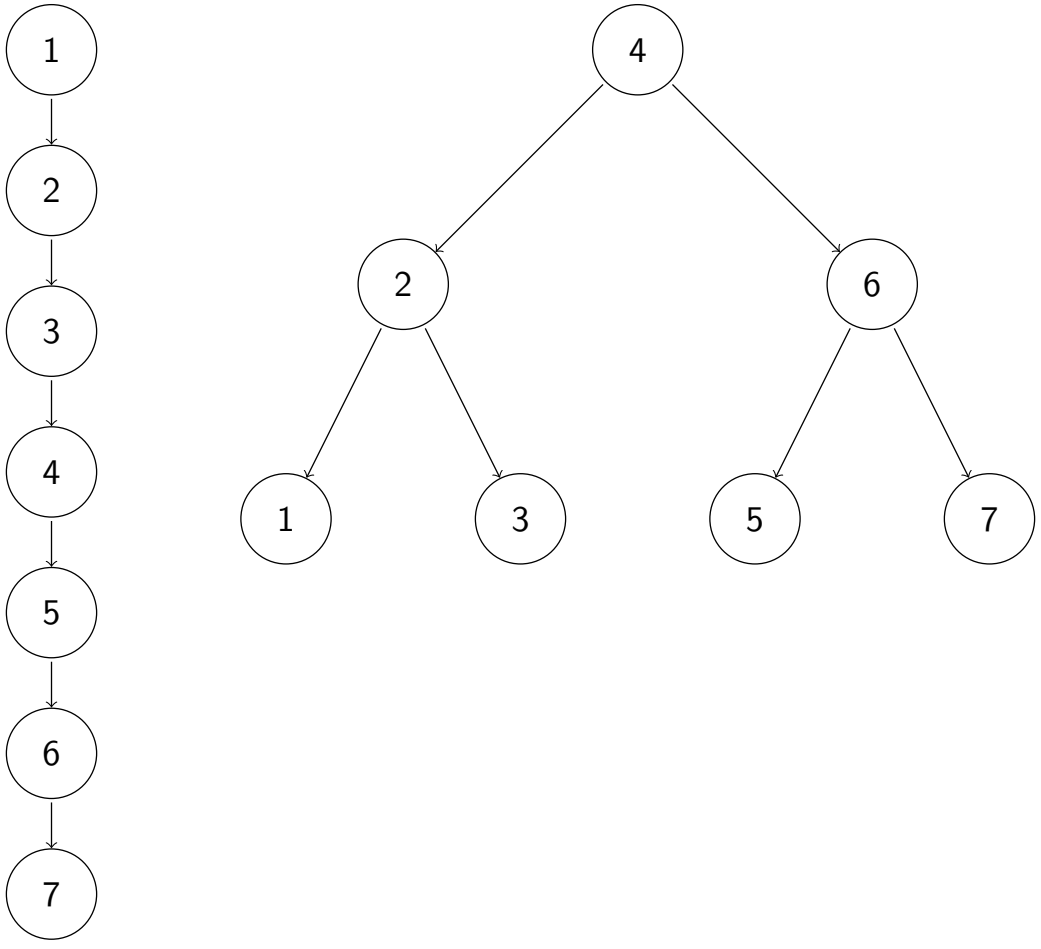
Beispiel: Schrittweiser Aufbau von Suchbaum

Wir wollen die Zahlen $1, 2, 3, 4, 5, 6, 7$ in der Reihenfolge $4, 2, 6, 3, 7, 5, 1$ in anfangs leeren Binärbaum einfügen:



Problem: Empfindlichkeit gegenüber Reihenfolge

Vergleichen wir den Baum mit dem Suchbaum, der bei der Reihenfolge $1, 2, 3, 4, 5, 6, 7$ entsteht:



Lösung: AVL-Baum

Definition (AVL-Baum)

Ein AVL-Baum ist ein binärer Suchbaum bei dem für jeden Knoten gilt: Die Tiefe des linken Teilbaums weicht um maximal 1 von der Tiefe des rechten Teilbaums ab.

Nun müssen wir Einfügeoperation anpassen: Einfügen in Binärbaum garantiert AVL-Eigenschaft nicht.

Lösung: AVL-Baum

Definition (AVL-Baum)

Ein AVL-Baum ist ein binärer Suchbaum bei dem für jeden Knoten gilt: Die Tiefe des linken Teilbaums weicht um maximal 1 von der Tiefe des rechten Teilbaums ab.

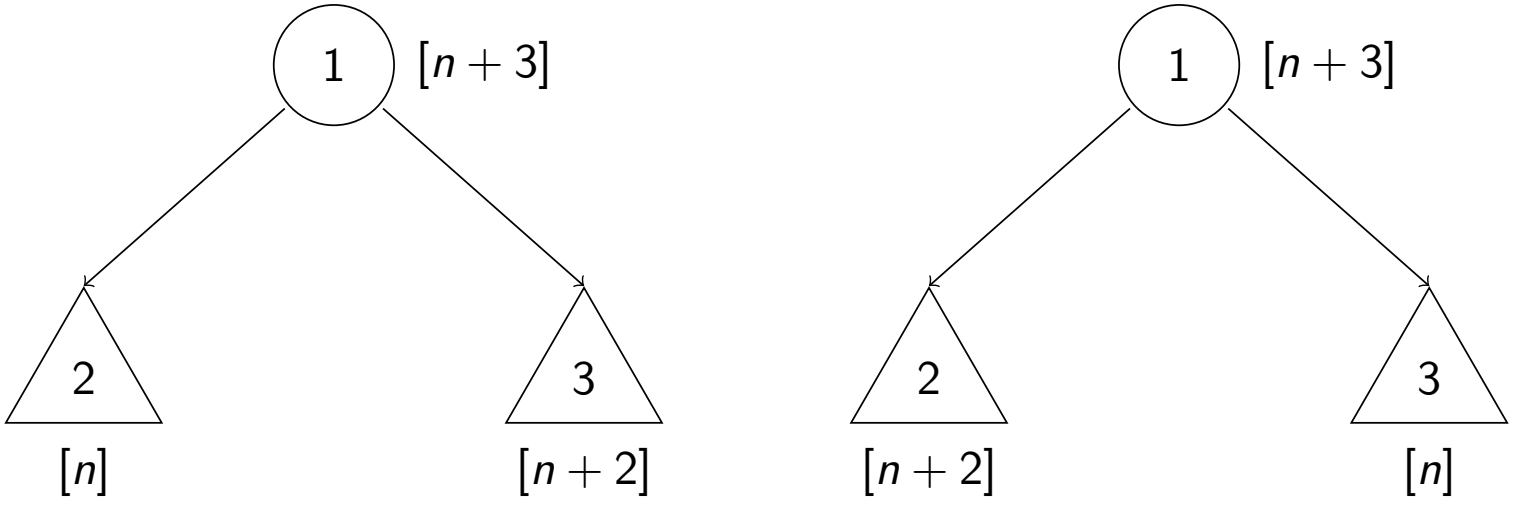
Nun müssen wir Einfügeoperation anpassen: Einfügen in Binärbaum garantiert AVL-Eigenschaft nicht.
Strategie: Füge wie in Binärbaum ein und stelle AVL-Eigenschaft anschließend wieder her.

Einfügen in AVL-Baum: Mögliche Abweichungen von AVL-Eigenschaft

Fallunterscheidung:

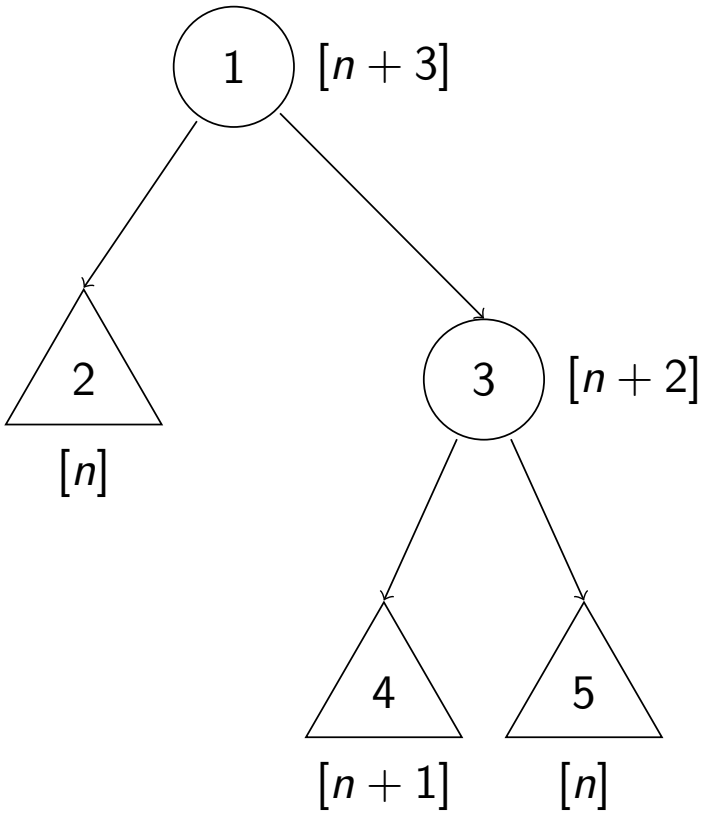
- Neuer Knoten ist Geschwister eines anderen Blattknotens $\Rightarrow$ AVL-Eigenschaft bleibt erhalten
- Neuer Knoten ist Kind eines vorherigen Blattknotens. Dann kann die AVL-Eigenschaft höchstens um 1 verletzt werden.

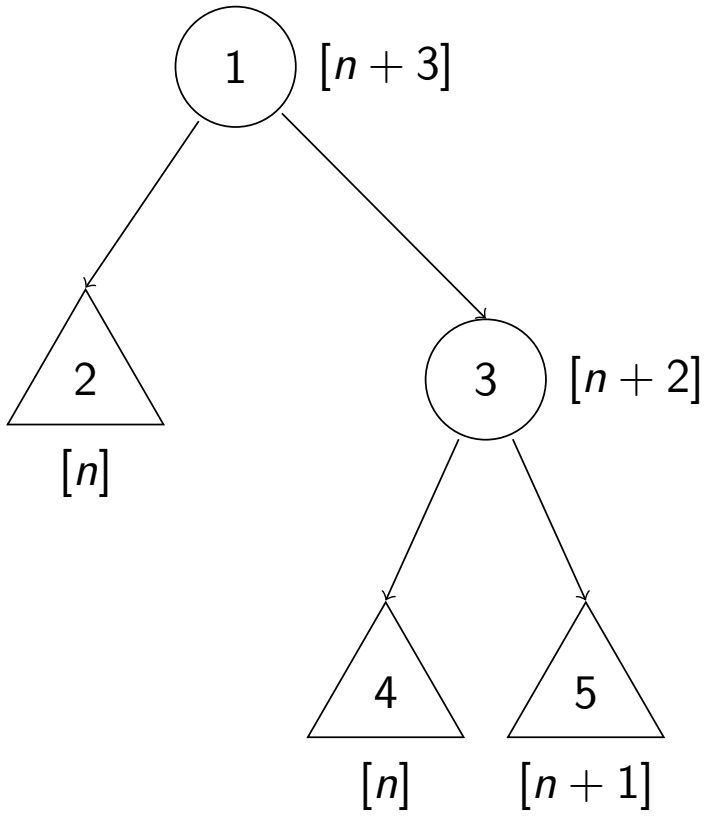
Abweichung nach links vs. Abweichung nach rechts



Wir lösen linke Situation auf, rechte Situation kann analog behandelt werden.

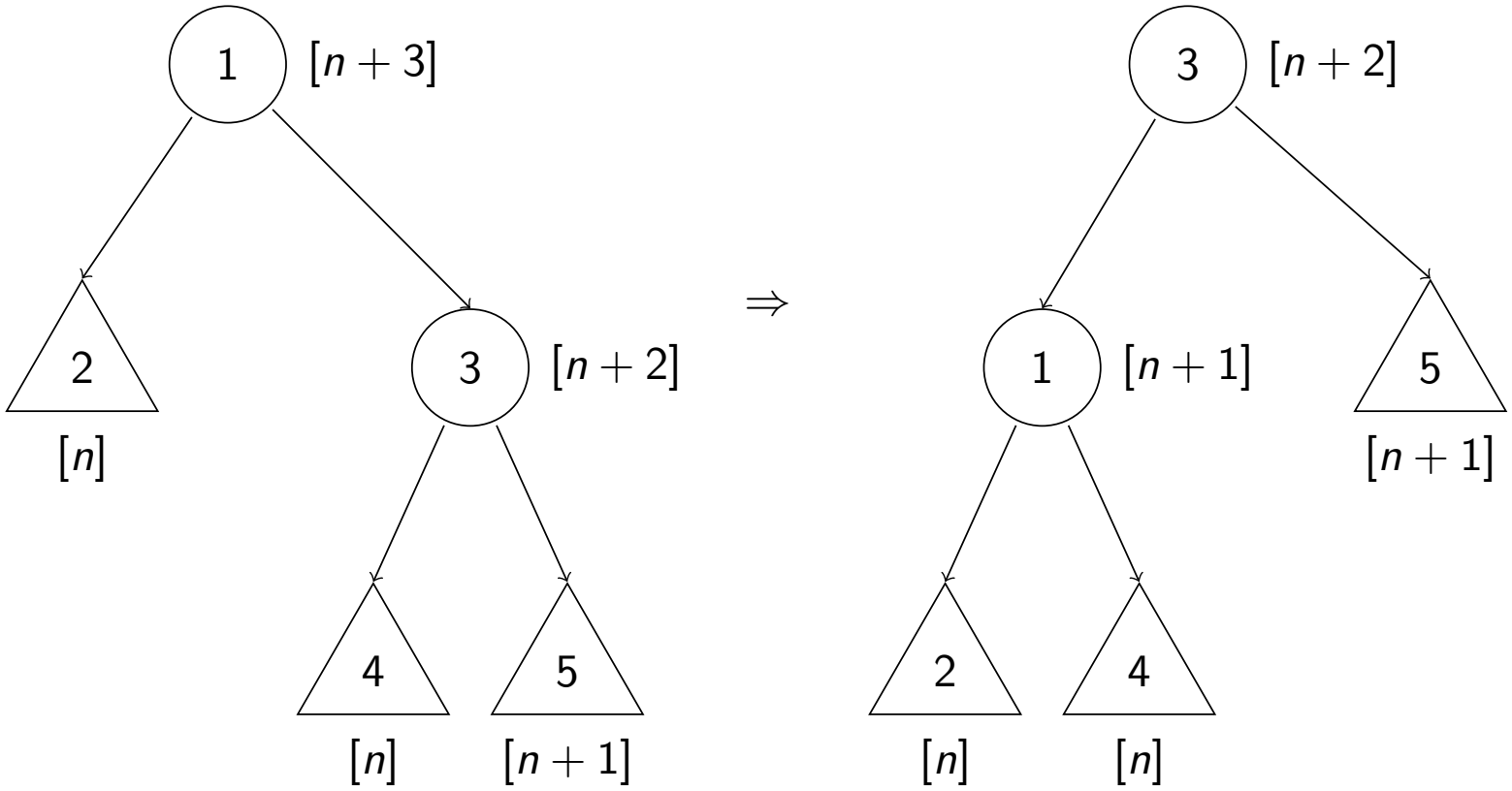
Ein genauerer Blick in den Baum



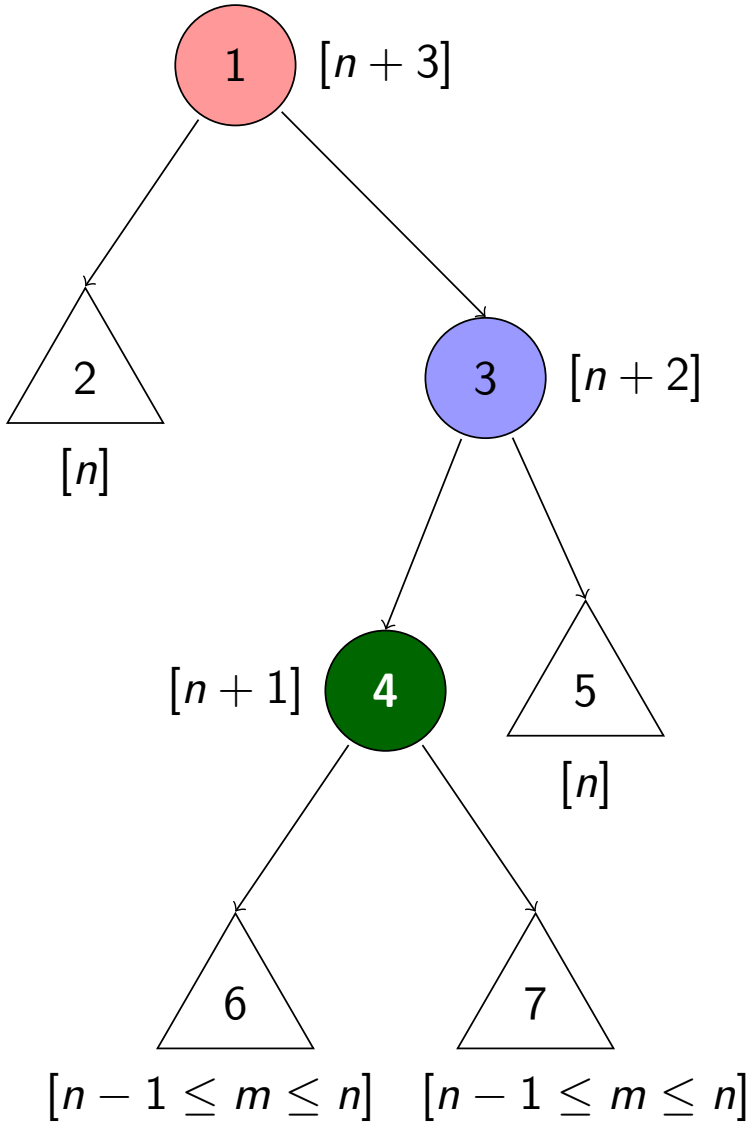


Die einfache Lösung: Linksrotation

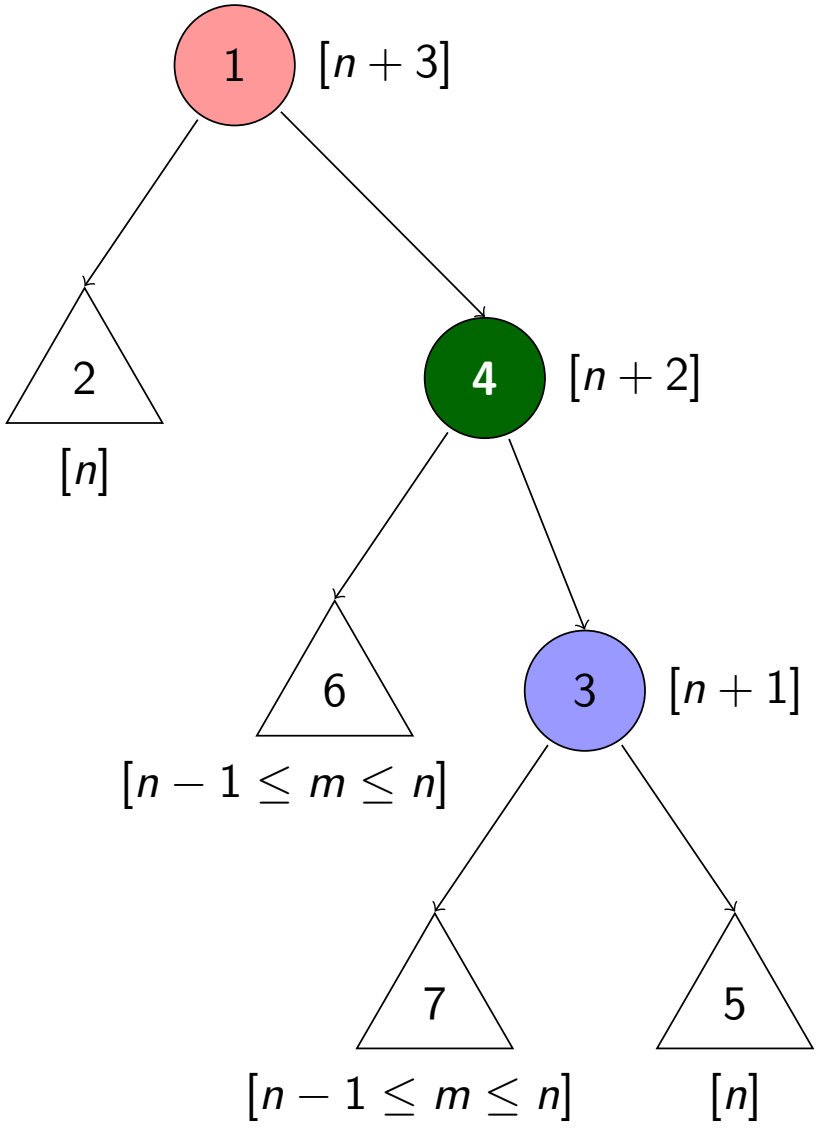
Der rechte Fall lässt sich mit einer einfachen Rotationsoperation lösen:



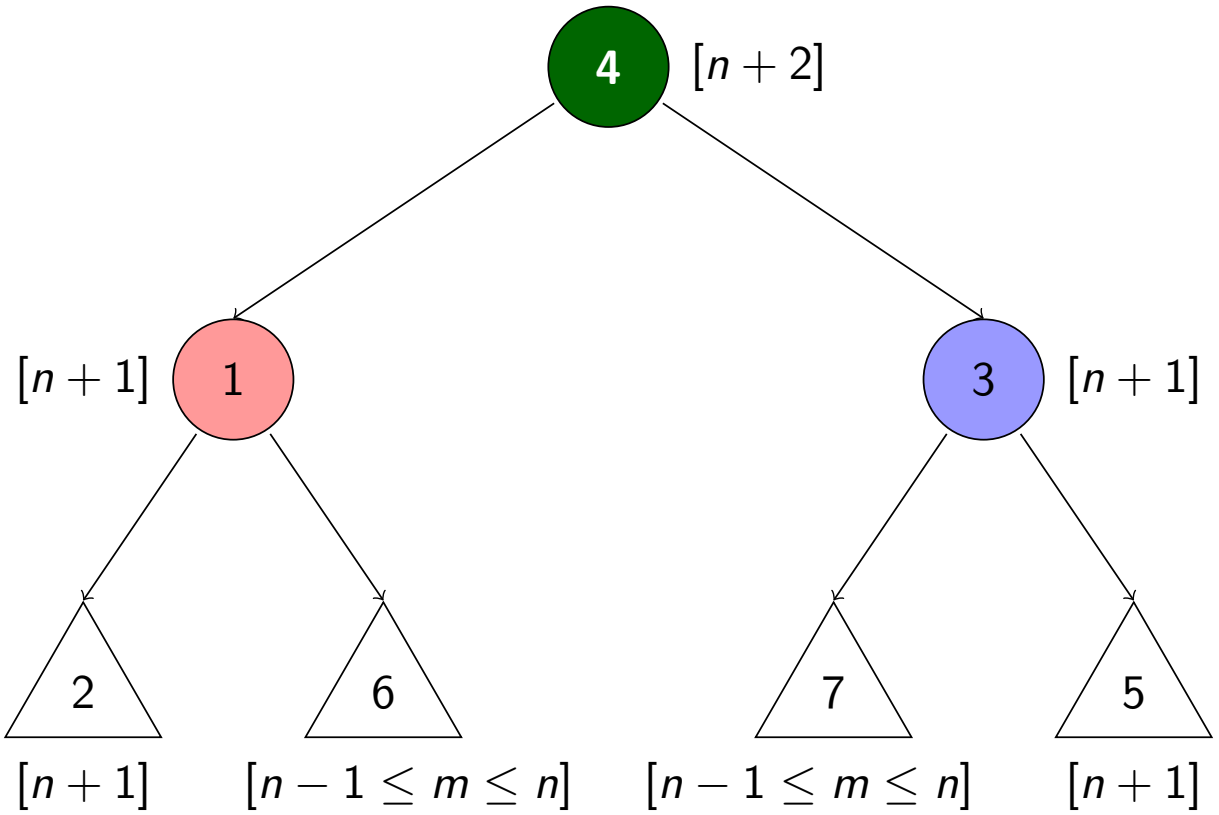
Die linke Situation: Ein noch tieferer Blick in den Baum



Erste Rotation: Rechtsrotation an 3

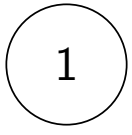


Zweite Rotation: Linksrotation an 1



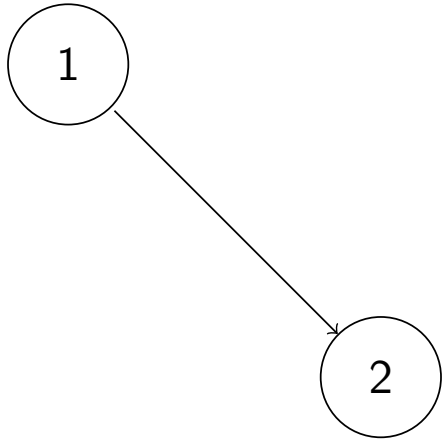
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



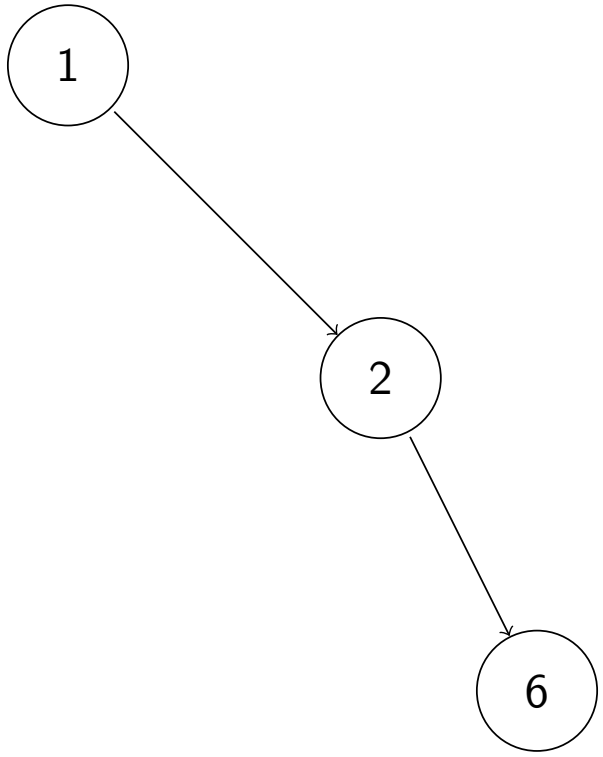
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



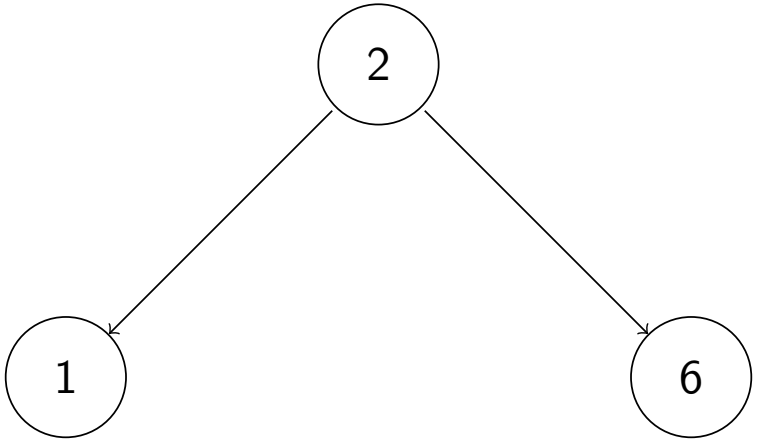
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



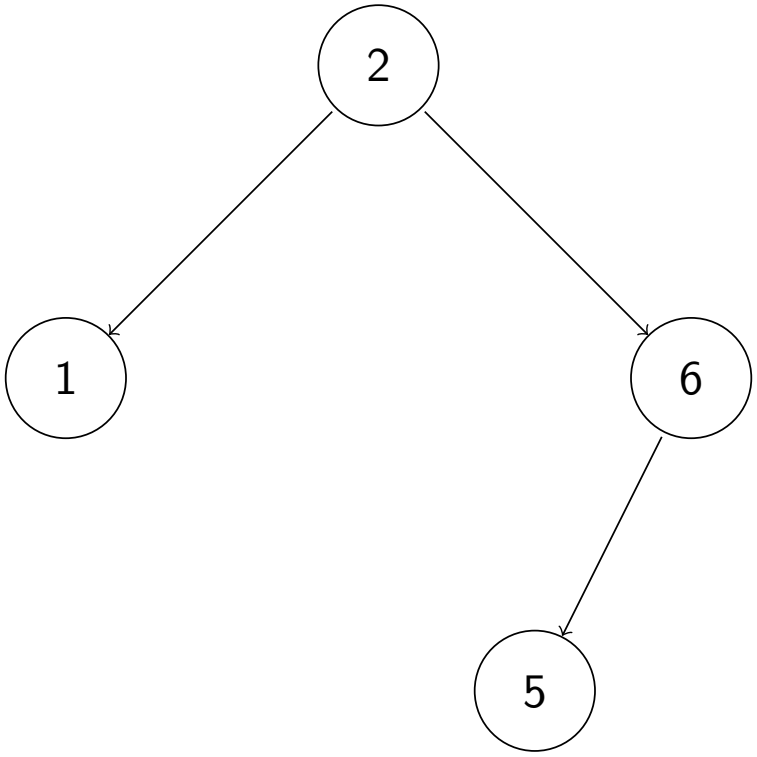
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



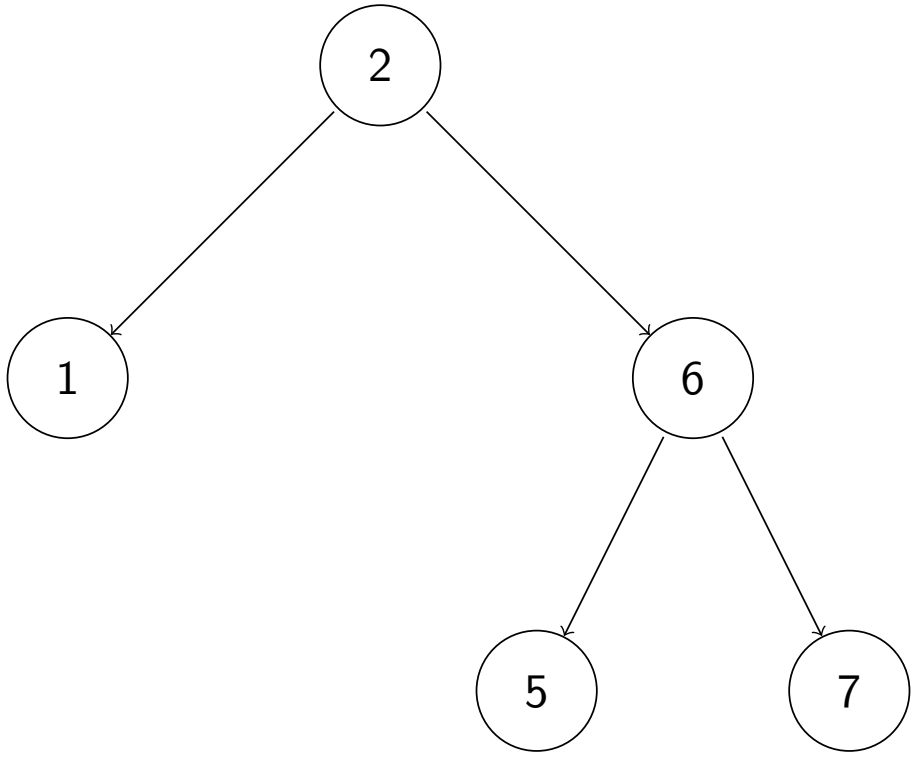
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



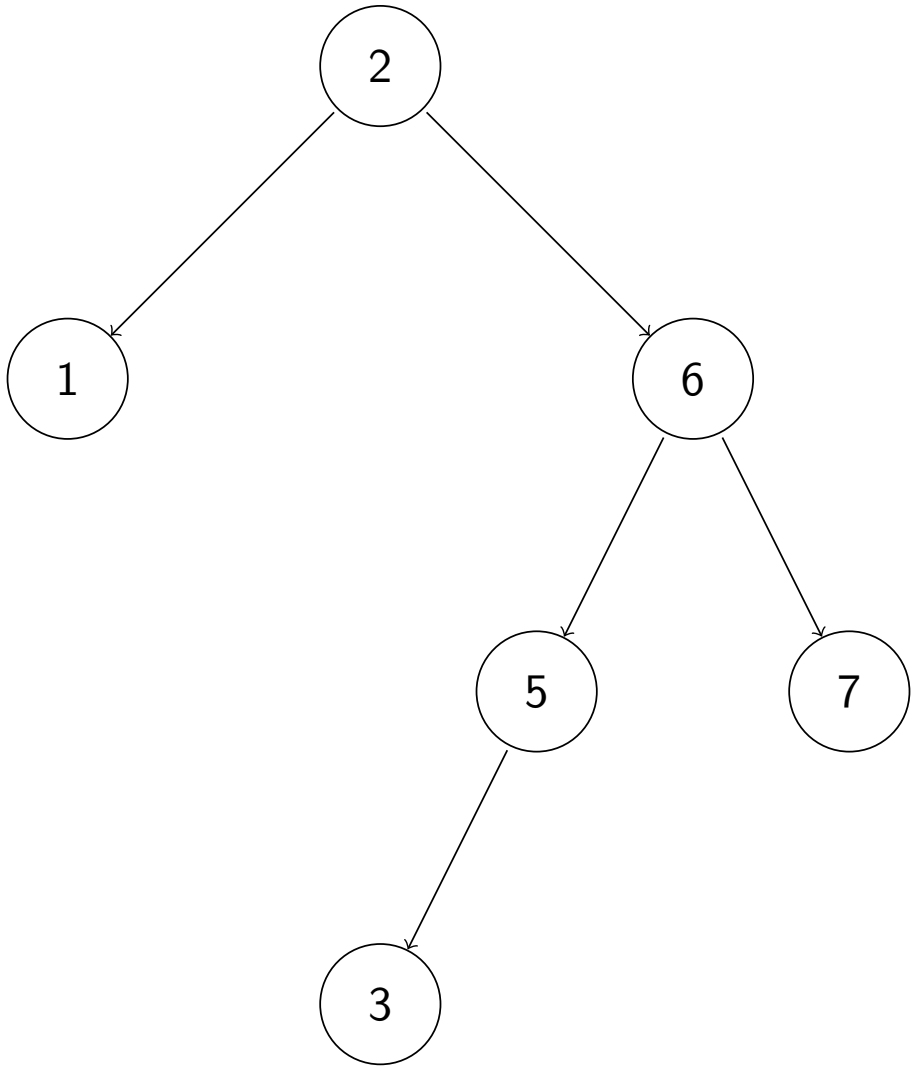
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



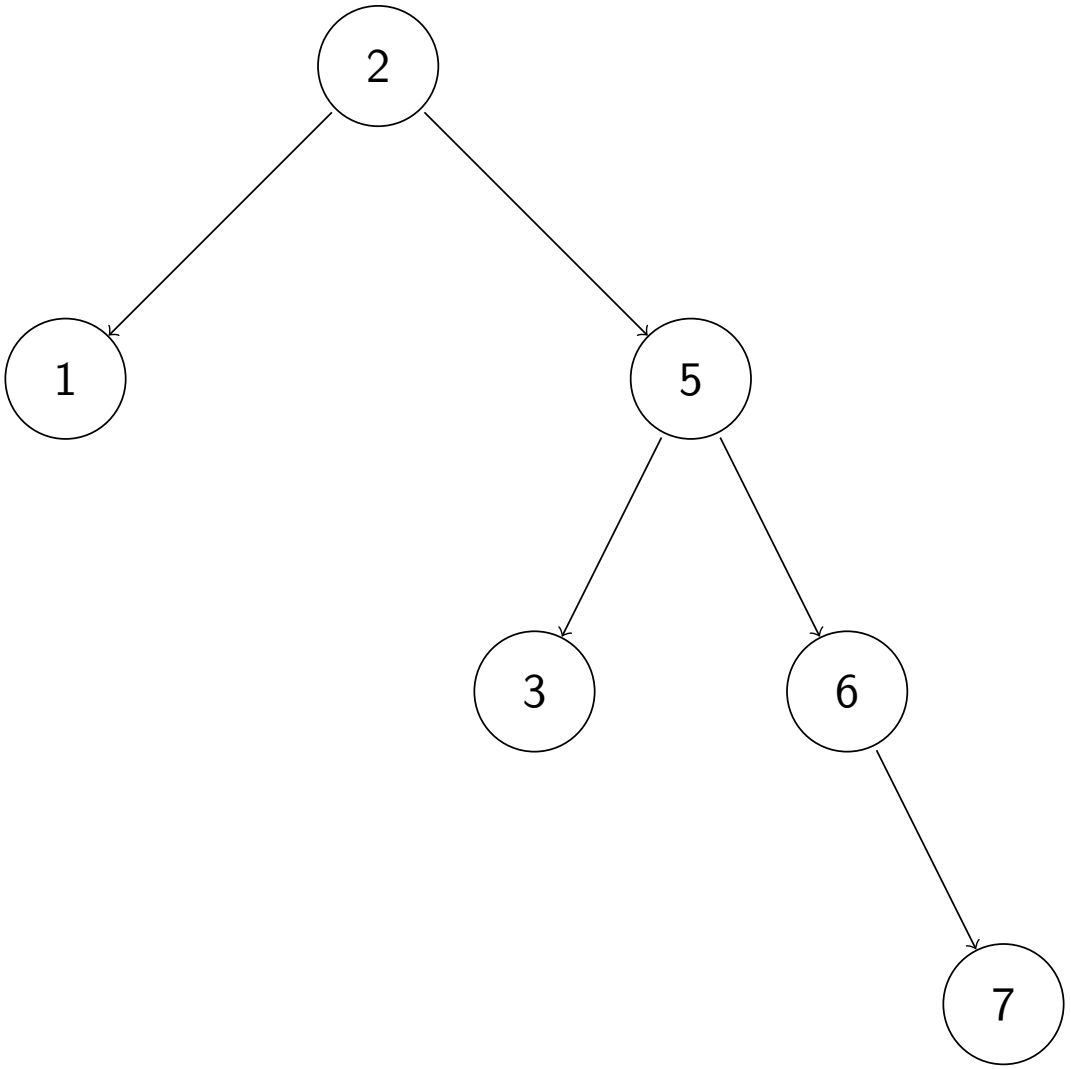
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



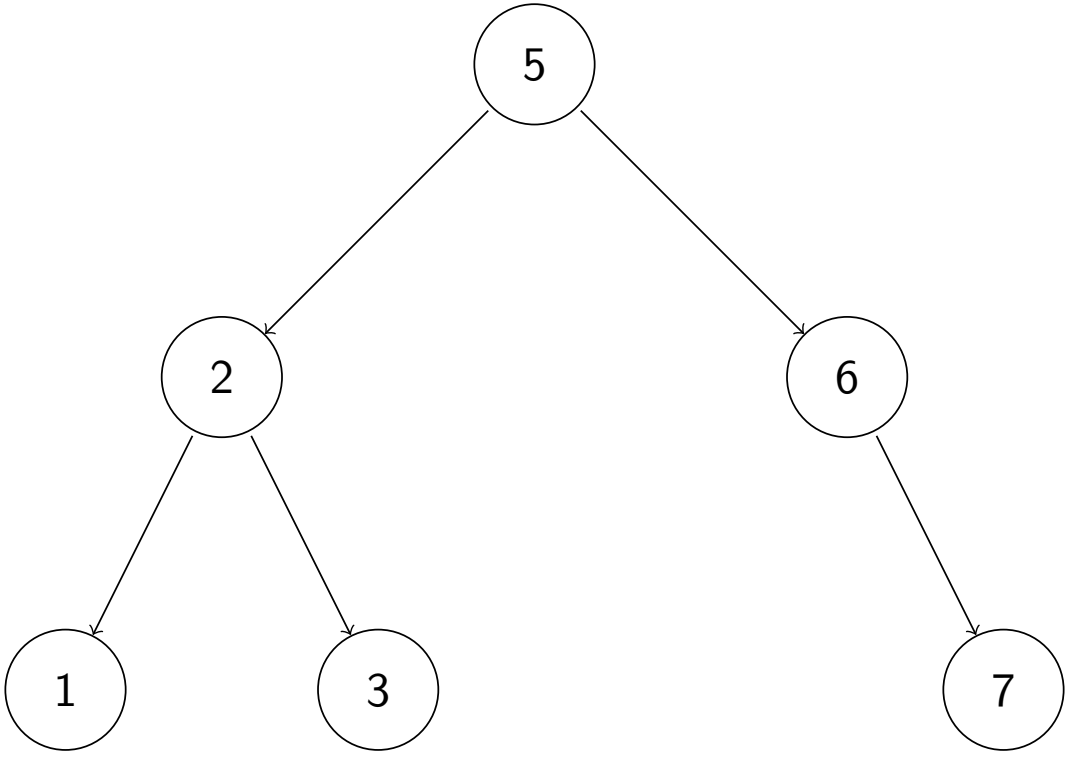
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



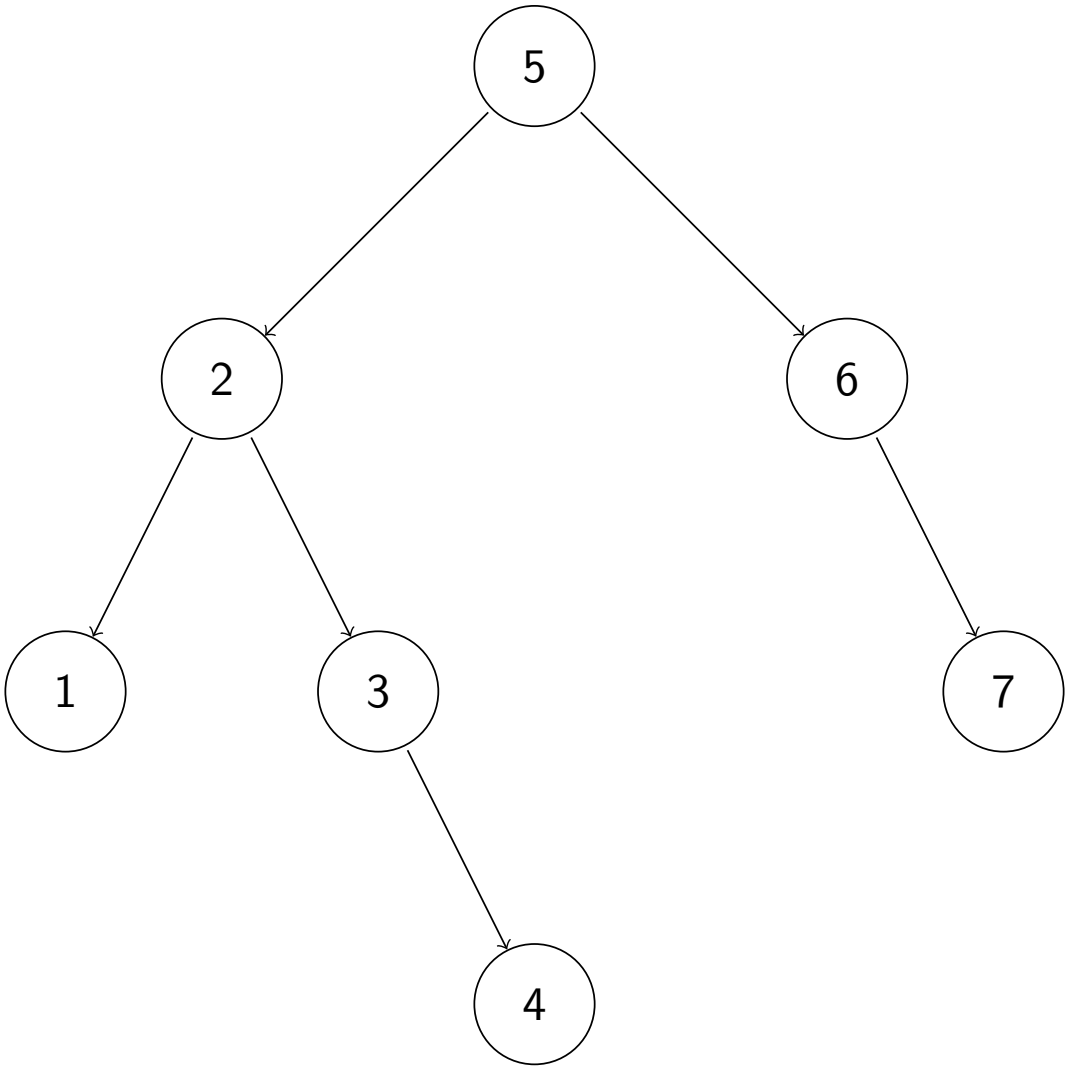
Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



Beispiel: Aufbau eines AVL-Baums

Wir fügen nacheinander die Zahlen $1, 2, 6, 5, 7, 3, 4$ in einen anfangs leeren AVL-Baum ein



Graphsuche

- In allgemeinen Graphen sucht man oft nach Wegen zwischen zwei Knoten
- Verschiedene Strategien:
  - Reicht irgendein Weg? Breiten- oder Tiefensuche
  - Muss es der kürzeste Weg sein? Dijkstra-Algorithmus (Optimierung durch A*-Algorithmus möglich, aber hier nicht thematisiert)
  - Suchverfahren nutzen Schlange (Breitensuche), Stapel (Tiefensuche), AVL-Bäume (Dijkstra)

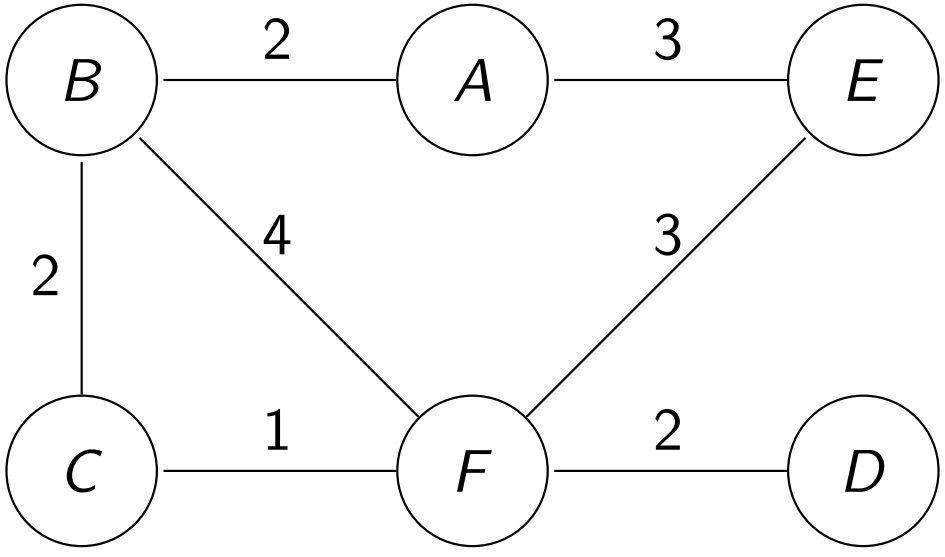
Breitensuche

Aufgabenstellung: Suche einen Knoten v von einem Knoten s aus

- Füge s in Warteschlange S ein
- Solange S nicht leer ist:
  - Entnimm das erste Element u der Warteschlange
  - Falls $u = v$ gib `true` zurück
  - Markiere u als besucht
  - Füge alle Nachbarn von u die nicht als besucht markiert sind (hinten) zu S hinzu
- Ist S leer, so wurde v nicht gefunden, gib `false` zurück

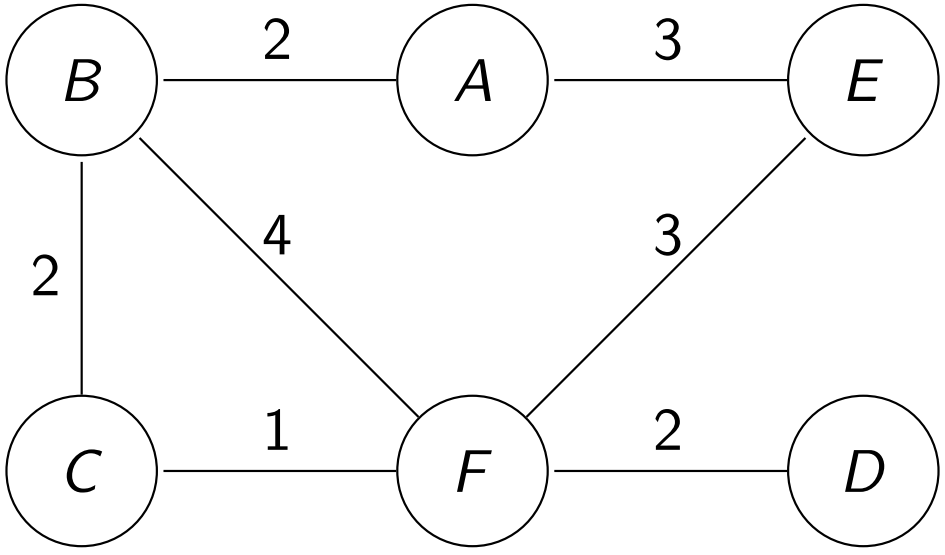
Beispiel Breitensuche

Ausgangsknoten A, suche nach Knoten D



Beispiel Breitensuche

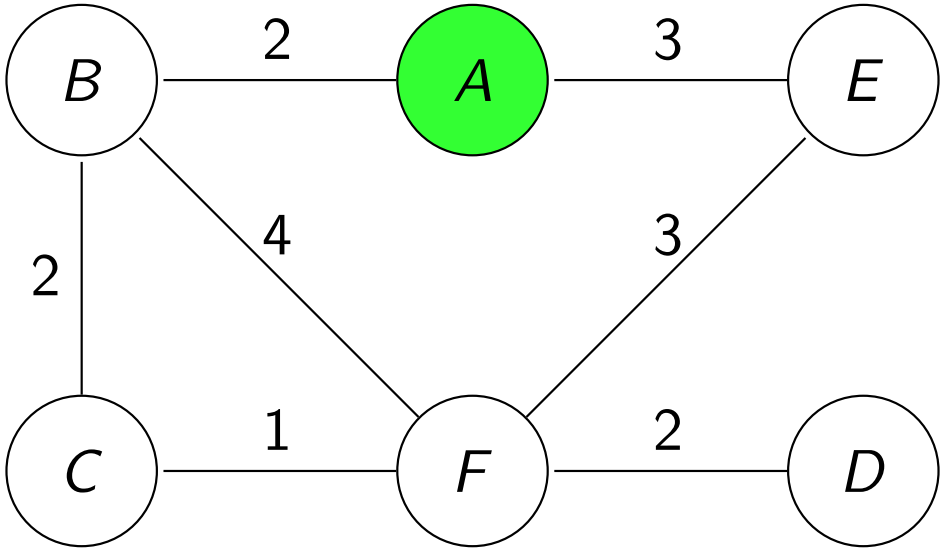
Ausgangsknoten A, suche nach Knoten D



Warteschlange: $[A]$

Beispiel Breitensuche

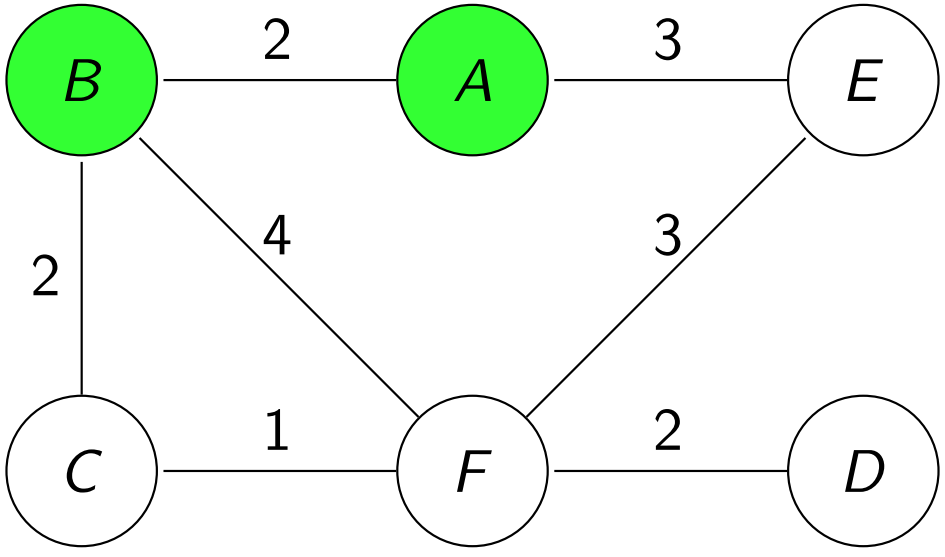
Ausgangsknoten A, suche nach Knoten D



Warteschlange: $[B, E]$

Beispiel Breitensuche

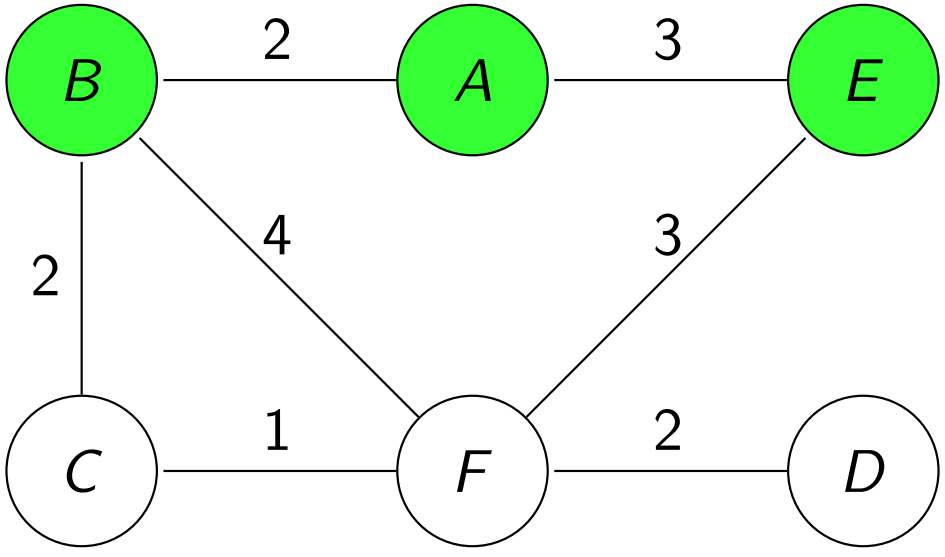
Ausgangsknoten A, suche nach Knoten D



Warteschlange: $[E, C, F]$

Beispiel Breitensuche

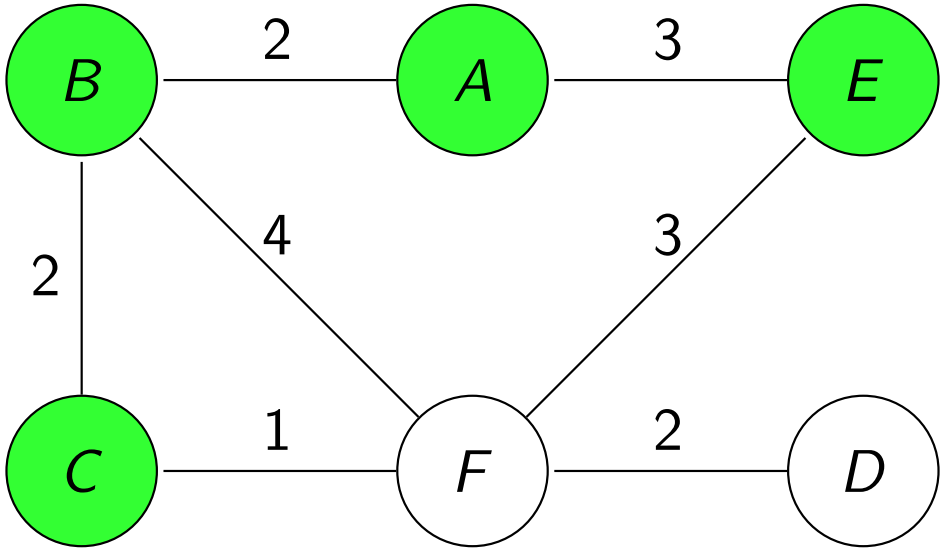
Ausgangsknoten A, suche nach Knoten D



Warteschlange: $[C, F, F]$

Beispiel Breitensuche

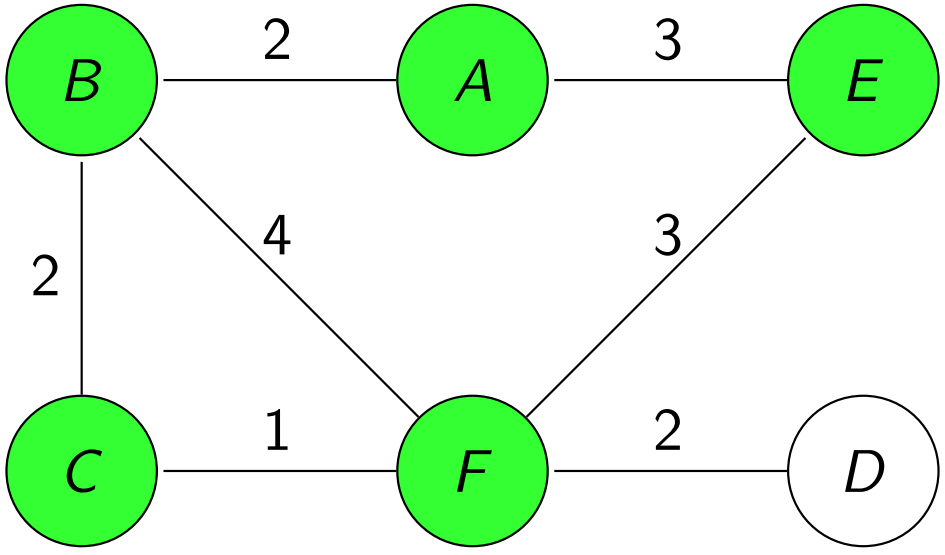
Ausgangsknoten A, suche nach Knoten D



Warteschlange: $[F, F, F]$

Beispiel Breitensuche

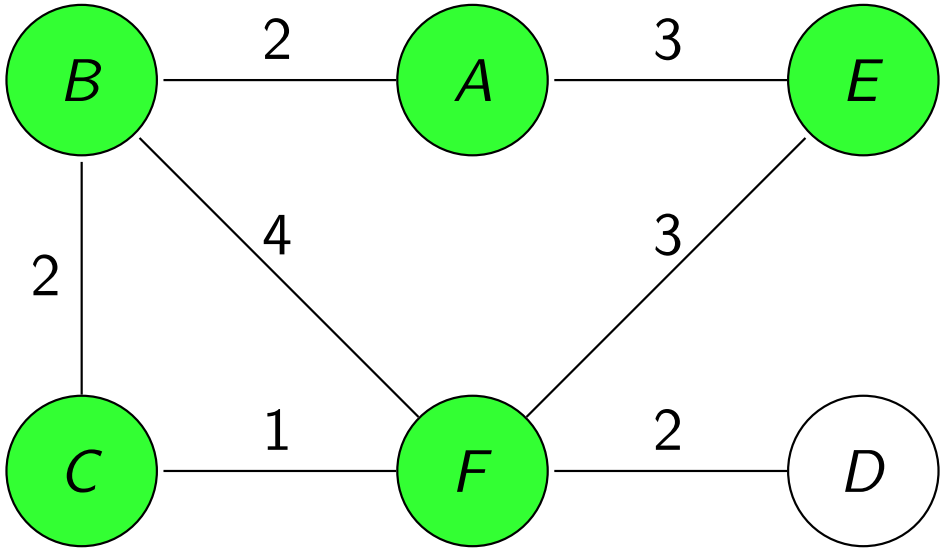
Ausgangsknoten A, suche nach Knoten D



Warteschlange: $[F, F, D]$

Beispiel Breitensuche

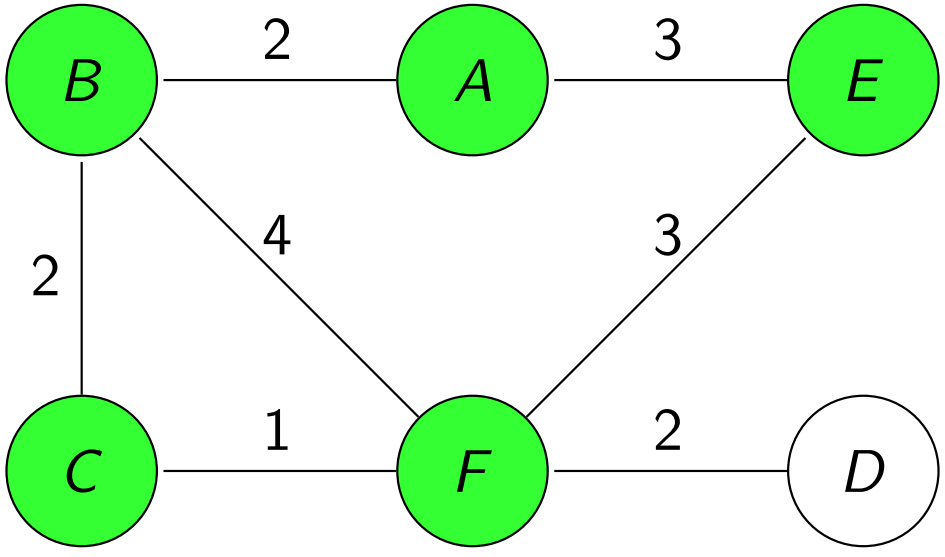
Ausgangsknoten A, suche nach Knoten D



Warteschlange: $[F, D]$

Beispiel Breitensuche

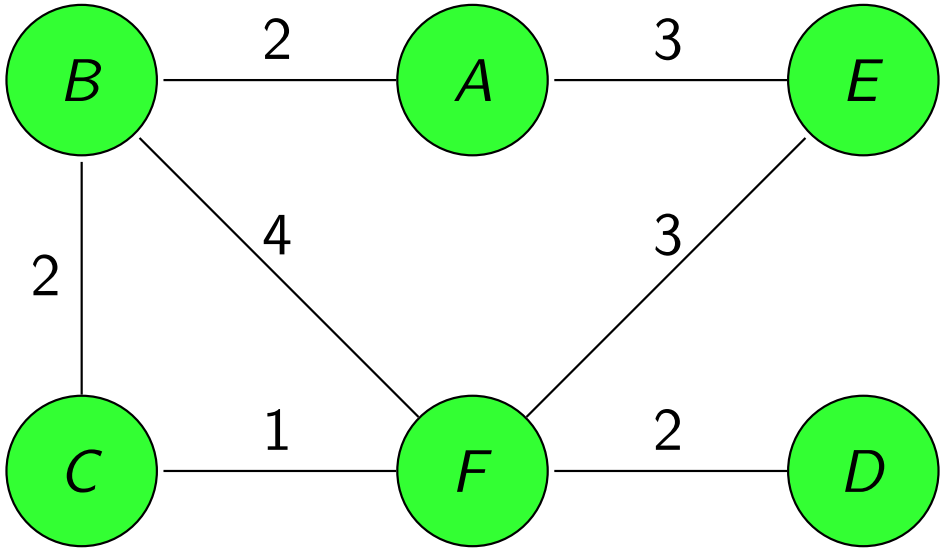
Ausgangsknoten A, suche nach Knoten D



Warteschlange: $[D]$

Beispiel Breitensuche

Ausgangsknoten A, suche nach Knoten D



Warteschlange: [], Rückgabe: `true`

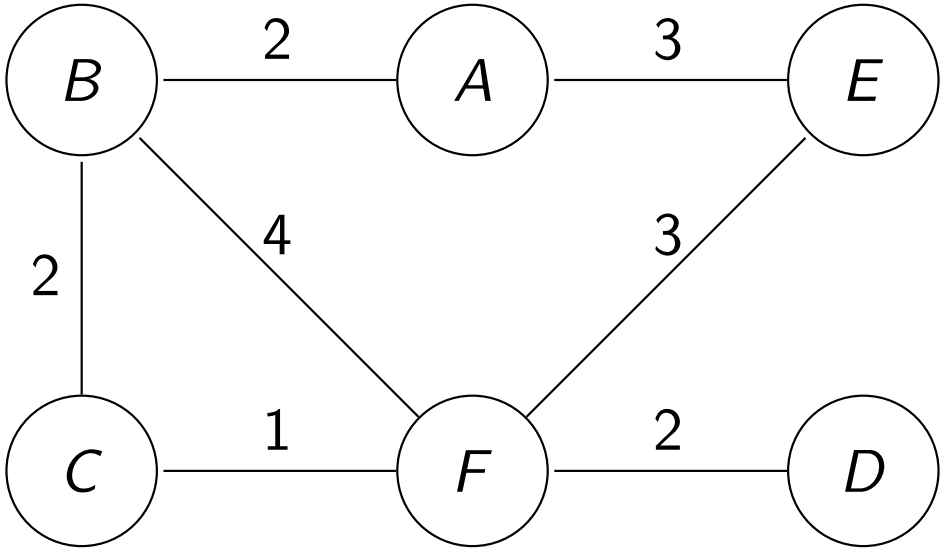
Tiefensuche

Aufgabenstellung: Suche einen Knoten v von einem Knoten s aus

- Füge s in Stapel S ein
- Solange S nicht leer ist:
  - Entnimm das erste Element u des Stapels
  - Falls $u = v$ gib `true` zurück
  - Markiere u als besucht
  - Füge alle Nachbarn von u die nicht als besucht markiert sind (vorne) zu S hinzu
- Ist S leer, so wurde v nicht gefunden, gib `false` zurück

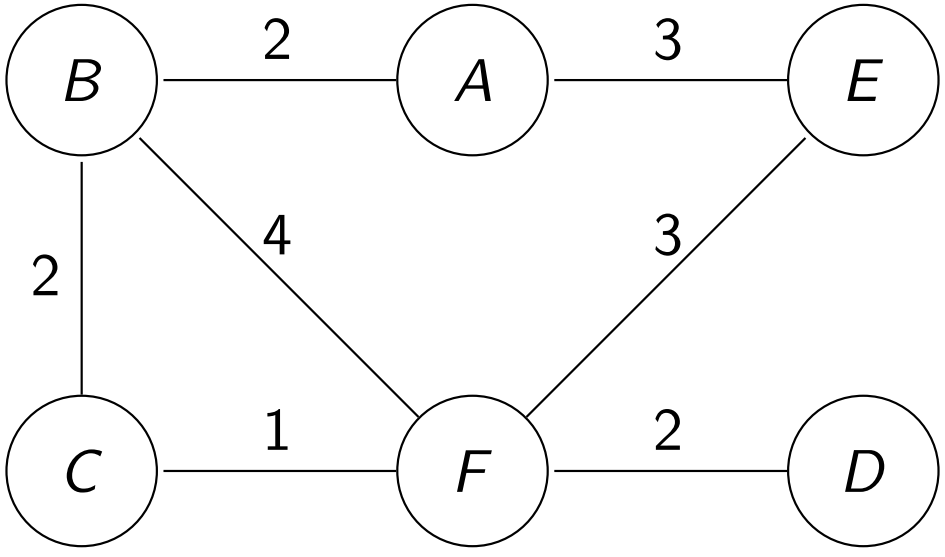
Beispiel Tiefensuche

Ausgangsknoten A, suche nach Knoten D



Beispiel Tiefensuche

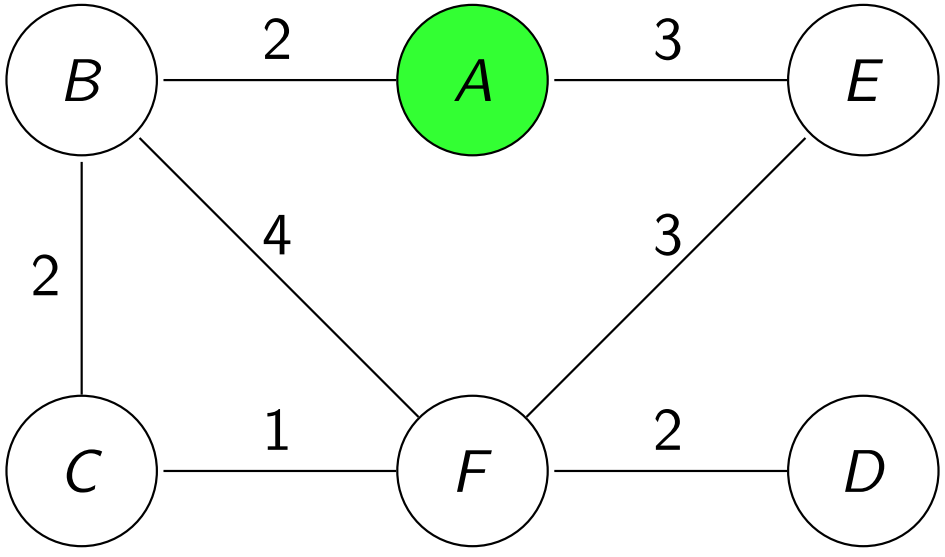
Ausgangsknoten A, suche nach Knoten D



Stapel: $[A]$

Beispiel Tiefensuche

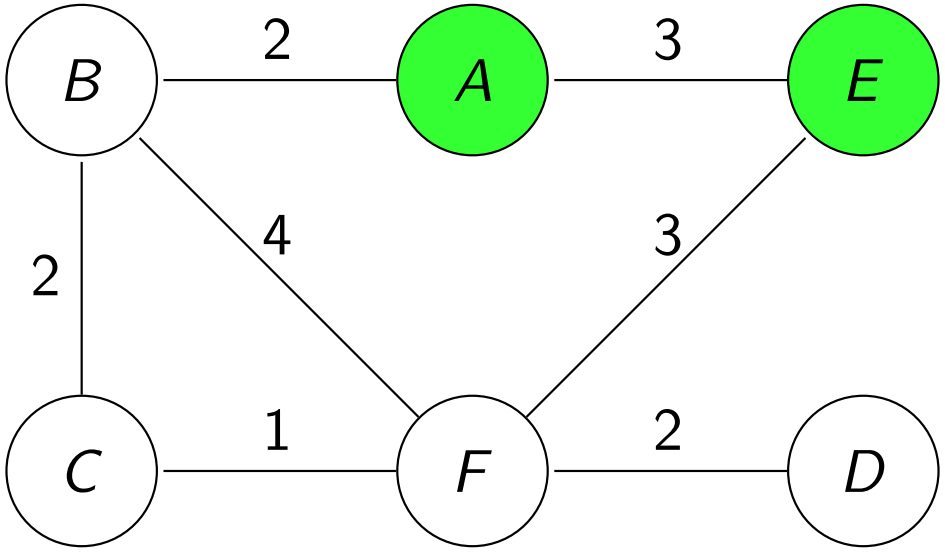
Ausgangsknoten A, suche nach Knoten D



Stapel: $[E, B]$

Beispiel Tiefensuche

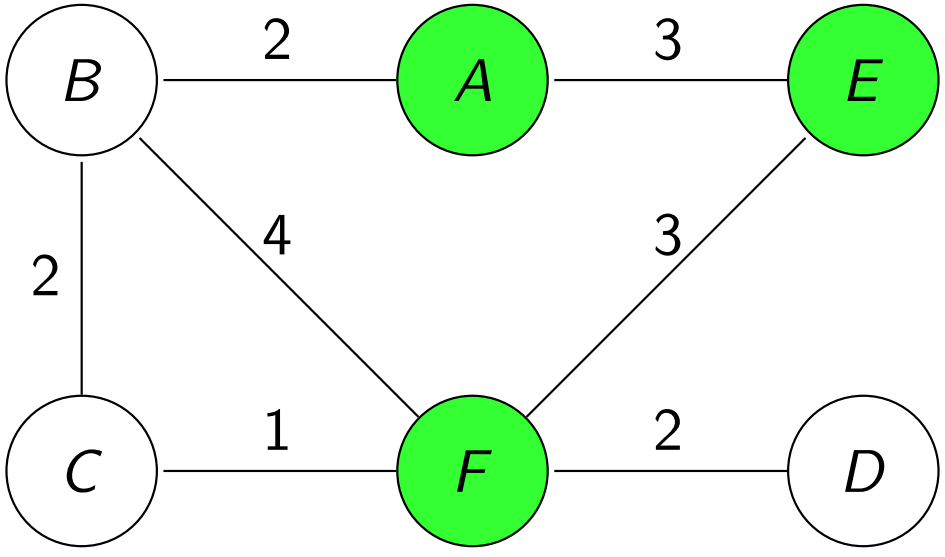
Ausgangsknoten A, suche nach Knoten D



Stapel: $[F, B]$

Beispiel Tiefensuche

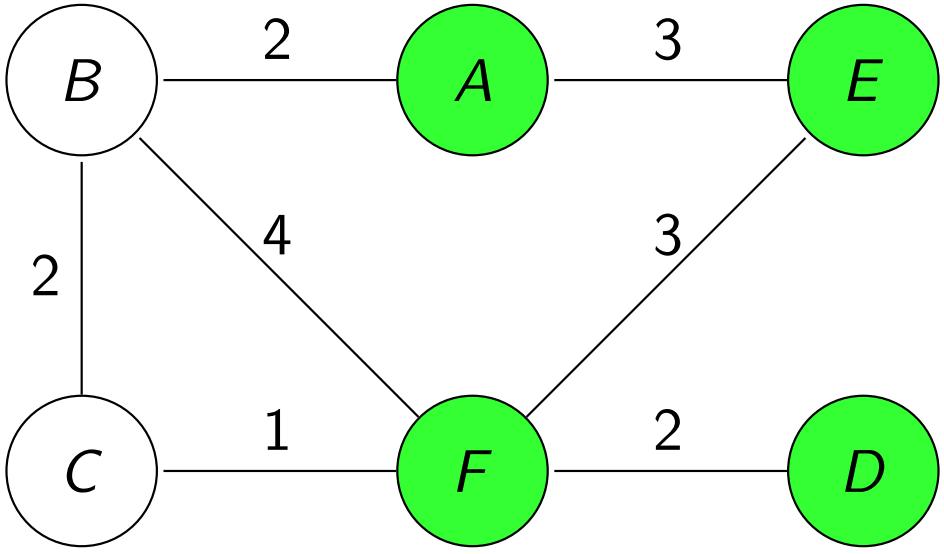
Ausgangsknoten A, suche nach Knoten D



Stapel: $[D, C, B, B]$

Beispiel Tiefensuche

Ausgangsknoten A, suche nach Knoten D



Stapel: $[C, B, B]$, Rückgabe: `true`

Vorrangwarteschleife

- Sortierte Liste die Zugriff auf Minimum (oder Maximum) zulässt und sortiert einfügt
- Effizient mithilfe eines AVL-Baums implementierbar

Dijkstra-Algorithmus

Aufgabenstellung: Suche einen kürzest möglichen Pfad zu u von einem Knoten s aus

1. Initialisiere V (Vorgängertripel) als leere Liste und P als leere Vorrangwarteschlange.
2. Füge $(s, 0, -)$ zu V hinzu

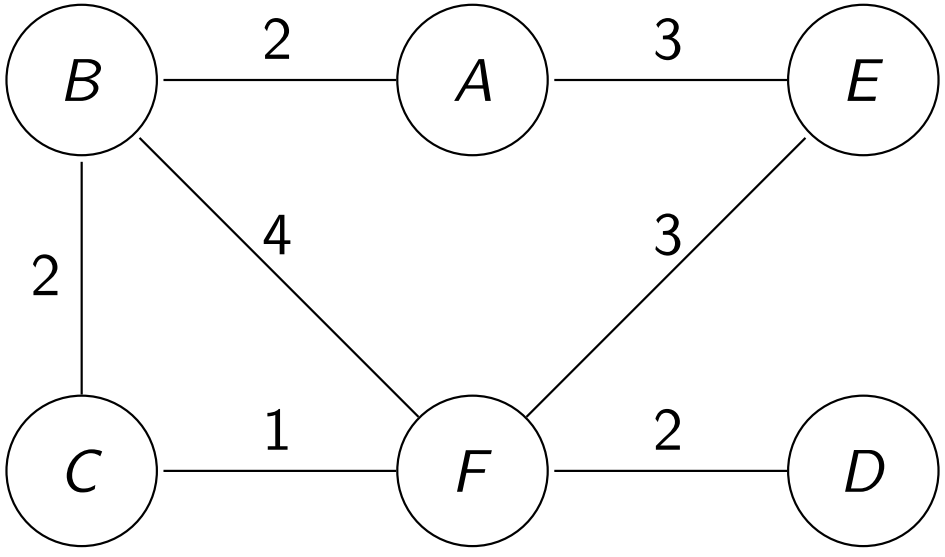
Dijkstra-Algorithmus

Aufgabenstellung: Suche einen kürzest möglichen Pfad zu u von einem Knoten s aus

1. Initialisiere V (Vorgängertripel) als leere Liste und P als leere Vorrangwarteschlange.
2. Füge $(s, 0, -)$ zu V hinzu
3. Falls P leer ist terminiere, sonst entnehme (v, d, v') aus P.
4. Falls v bereits besucht wurde, setze bei Schritt 3 fort.
5. Markiere v als besucht und füge (v, d, v') in V ein.
6. Betrachte alle noch nicht besuchten Nachbarn w des Knotens v und füge $(w, d + f(v, w), v)$ in P ein.
7. Fahre mit Schritt 3 fort.

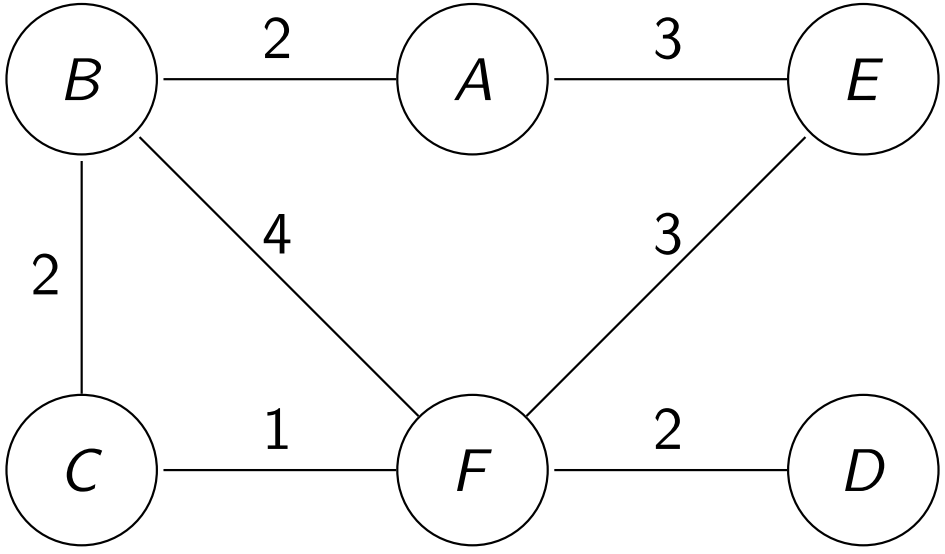
Beispiel Dijkstra-Algorithmus

Ausgangsknoten A, suche nach kürzestem Weg zu D



Beispiel Dijkstra-Algorithmus

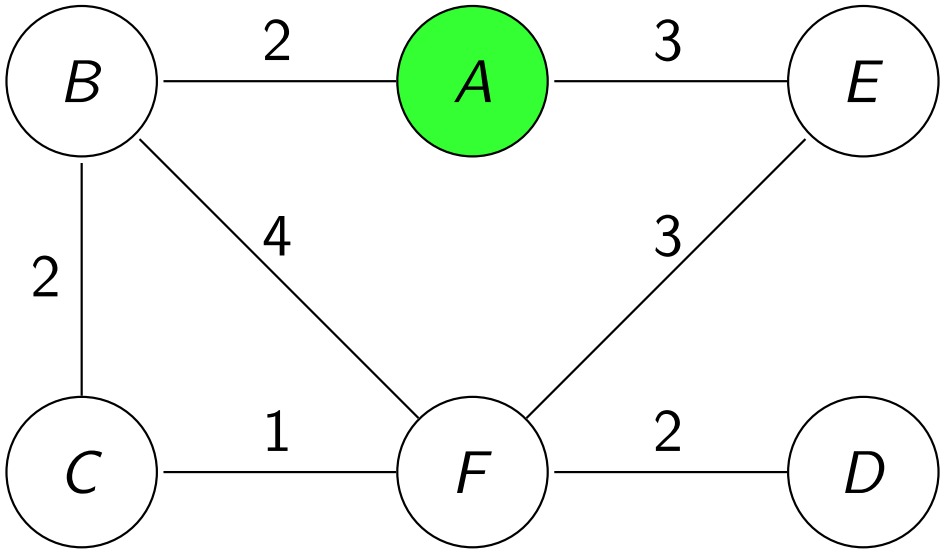
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = []$
$V = [(A, 0, -)]$

Beispiel Dijkstra-Algorithmus

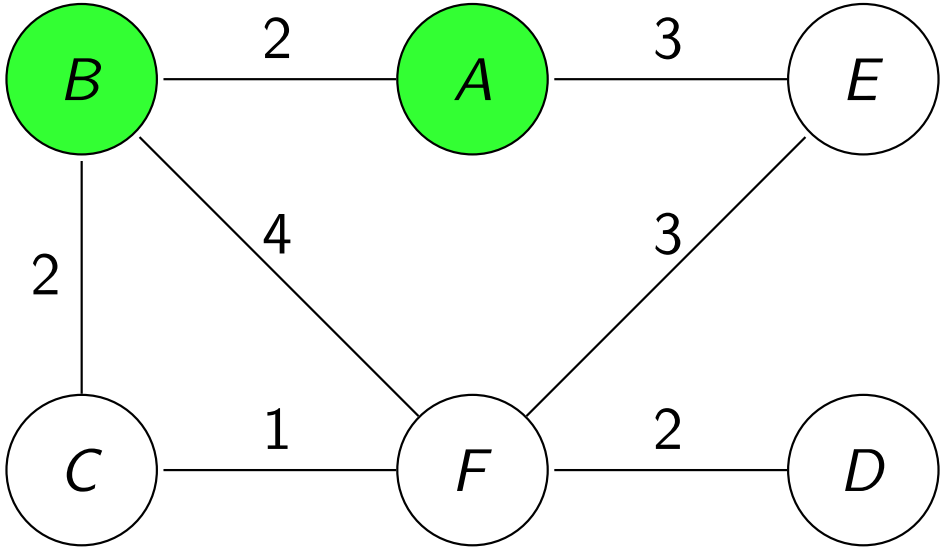
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -)]$
$V = [(B, 2, A), (E, 3, A)]$

Beispiel Dijkstra-Algorithmus

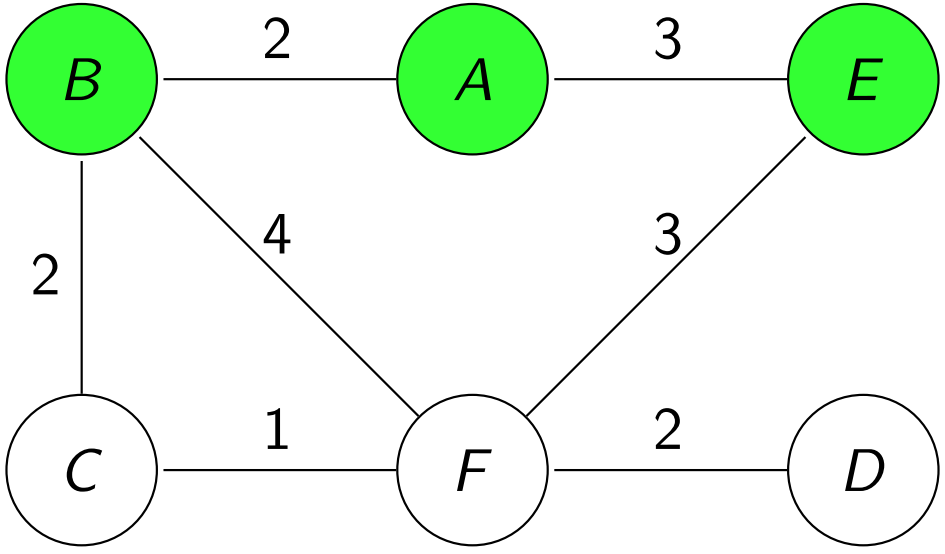
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A)]$
$V = [(E, 3, A), (C, 4, B), (F, 6, B)]$

Beispiel Dijkstra-Algorithmus

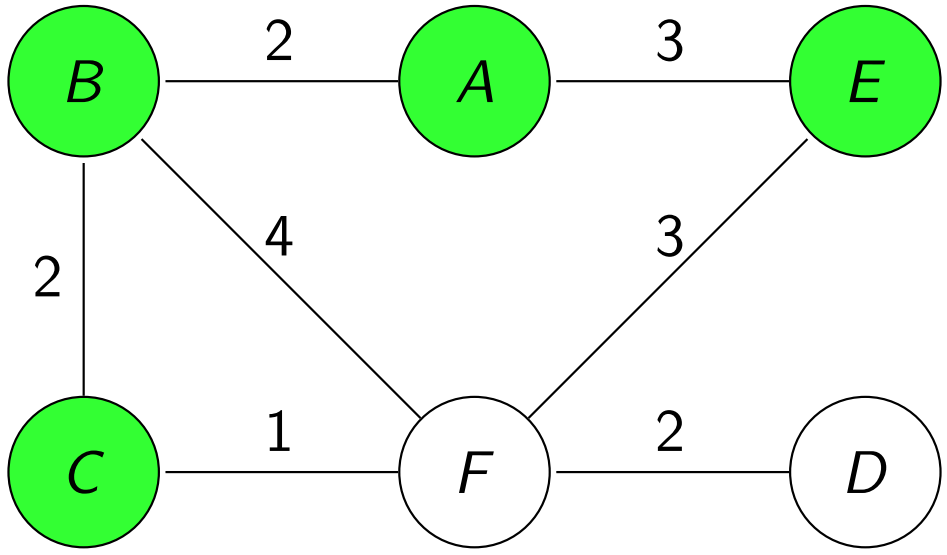
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A)]$
$V = [(C, 4, B), (F, 6, B), (F, 6, E)]$

Beispiel Dijkstra-Algorithmus

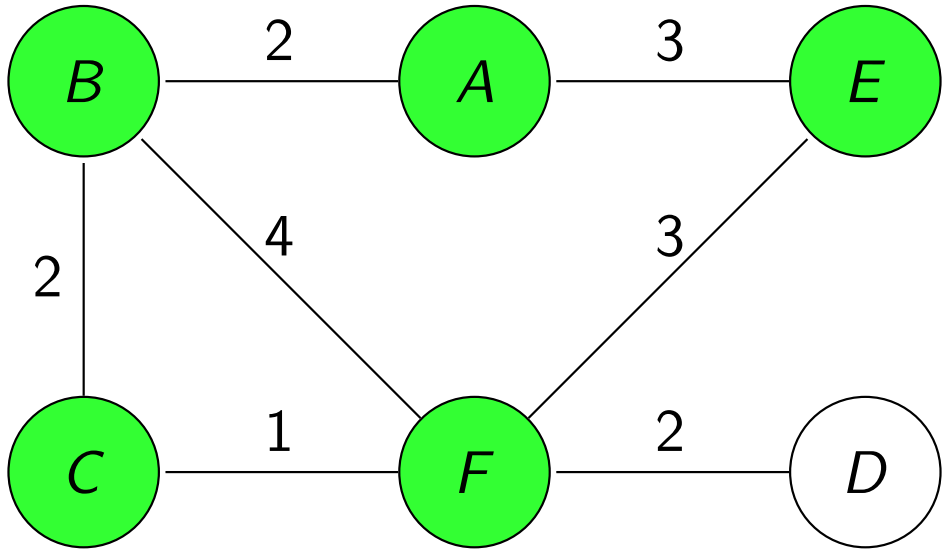
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B)]$
$V = [(F, 5, C), (F, 6, B), (F, 6, E)]$

Beispiel Dijkstra-Algorithmus

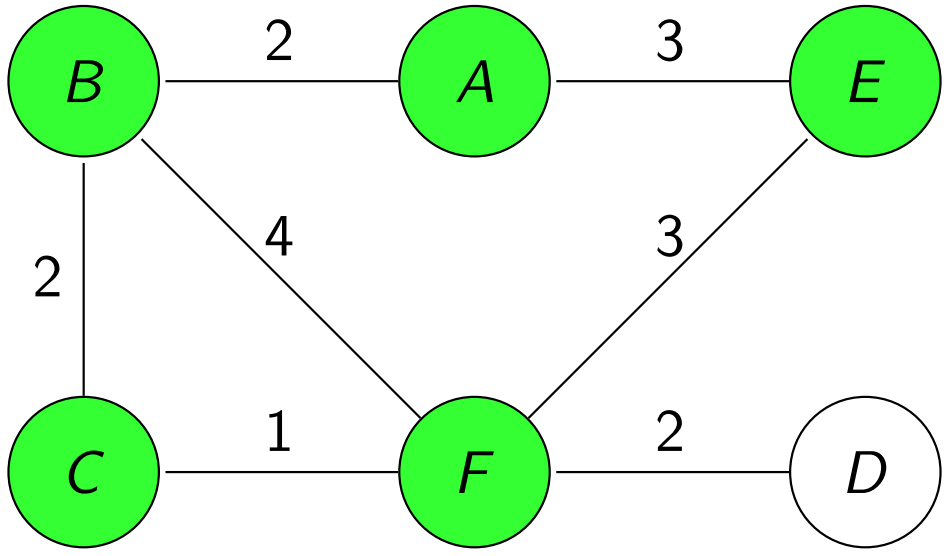
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B), (F, 5, C)]$
$V = [(F, 6, B), (F, 6, E), (D, 7, F)]$

Beispiel Dijkstra-Algorithmus

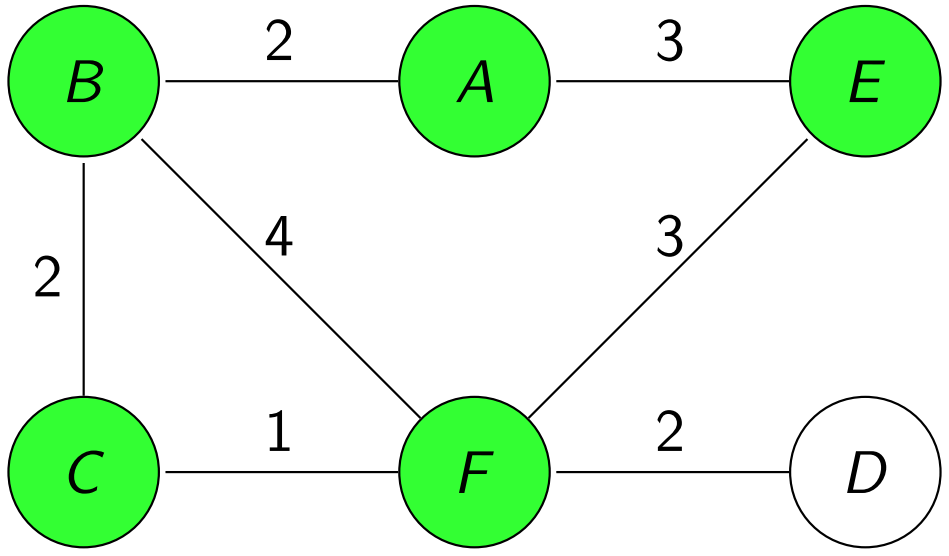
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B), (F, 5, C)]$
$V = [(F, 6, E), (D, 7, F)]$

Beispiel Dijkstra-Algorithmus

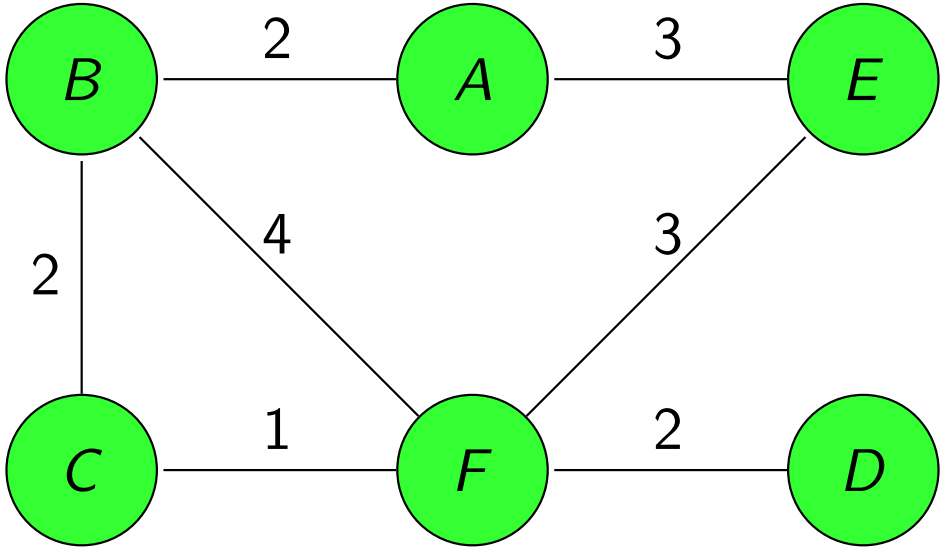
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B), (F, 5, C)]$
$V = [(D, 7, F)]$

Beispiel Dijkstra-Algorithmus

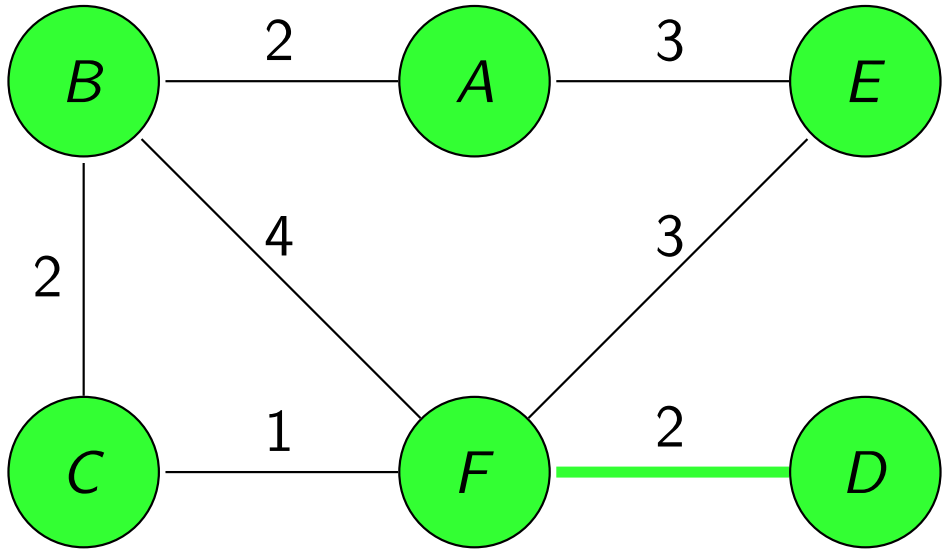
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B), (F, 5, C), (D, 7, F)]$
$V = []$

Beispiel Dijkstra-Algorithmus

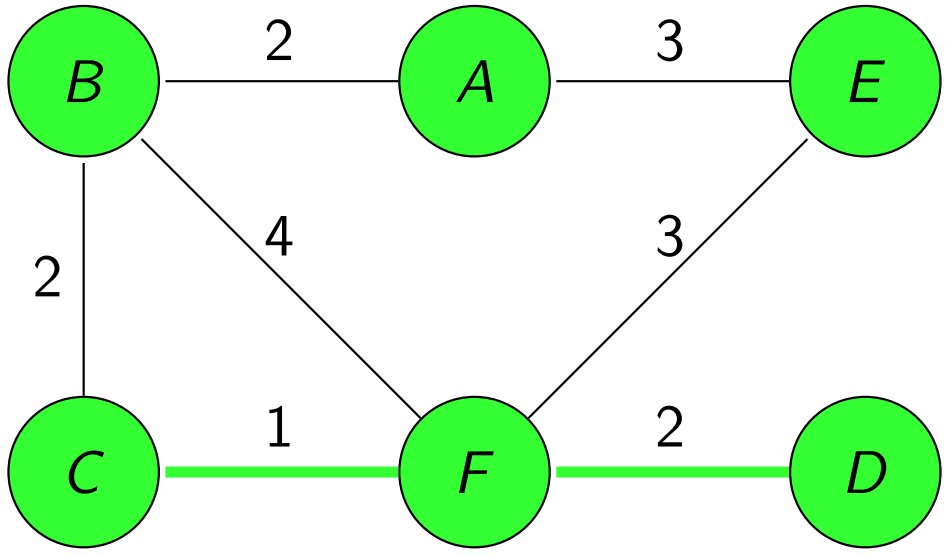
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B), (F, 5, C), (D, 7, F)]$
$V = []$

Beispiel Dijkstra-Algorithmus

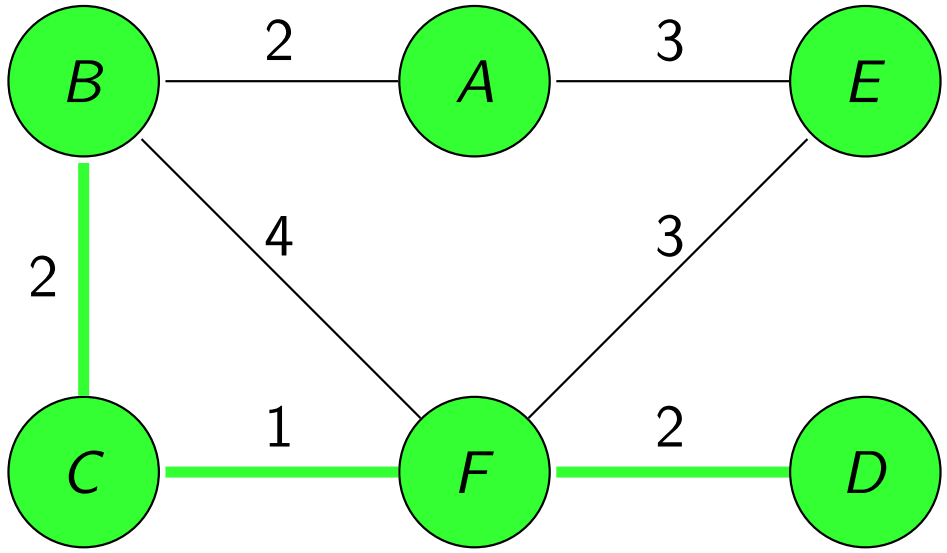
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B), (F, 5, C), (D, 7, F)]$
$V = []$

Beispiel Dijkstra-Algorithmus

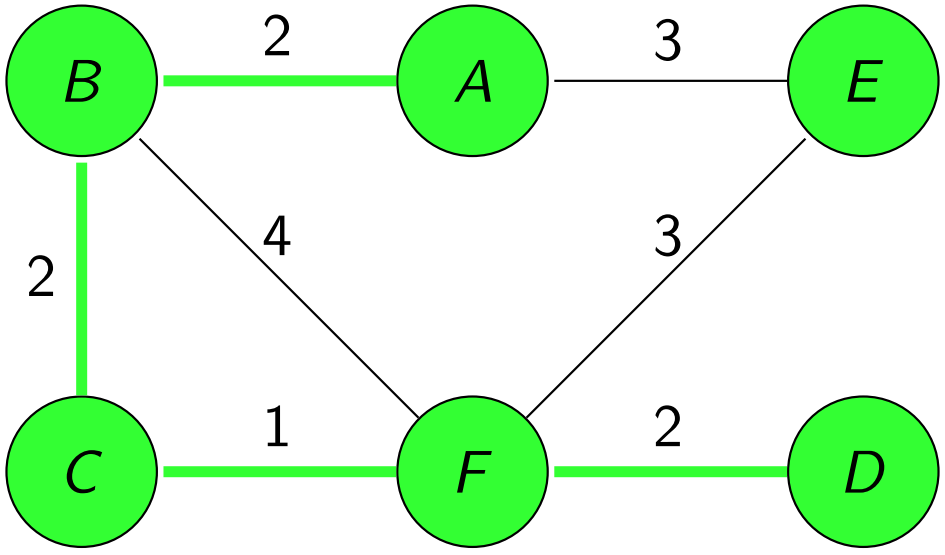
Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B), (F, 5, C), (D, 7, F)]$
$V = []$

Beispiel Dijkstra-Algorithmus

Ausgangsknoten A, suche nach kürzestem Weg zu D



$P = [(A, 0, -), (B, 2, A), (E, 3, A), (C, 4, B), (F, 5, C), (D, 7, F)]$
$V = []$

Fragen zur Vorlesungseinheit

1. Welche Durchlaufstrategien für Bäume gibt es?
2. Wie fügt man einen neuen Knoten in einen AVL-Baum ein?
3. Wie findet man den kürzesten Weg zwischen zwei Knoten in einem Graphen?