Bäume und Wälder

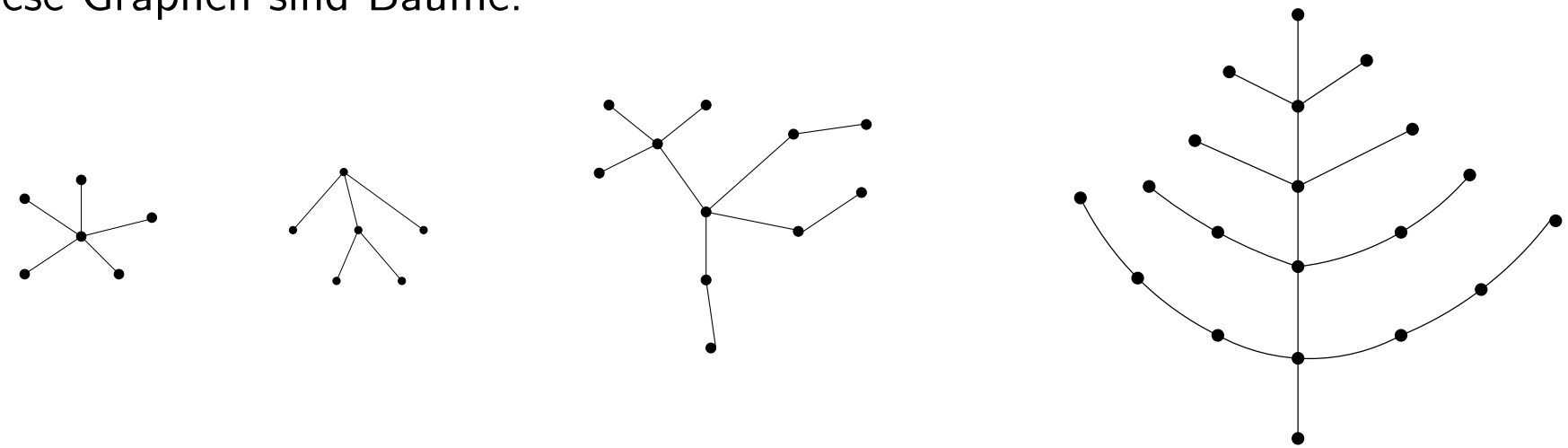
Bäume

Ein (ungerichteter) **Baum** ist ein ungerichteter Graph $G = (V, E)$, der zusammenhängend ist und keine einfachen Kreise enthält. V muss endlich sein.

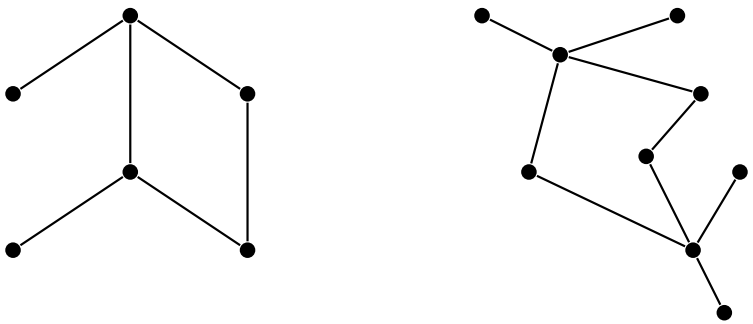
Bäume

Ein (ungerichteter) **Baum** ist ein ungerichteter Graph $G = (V, E)$, der zusammenhängend ist und keine einfachen Kreise enthält. V muss endlich sein.

Diese Graphen sind Bäume:



Diese aber nicht:



Wälder

Knoten-disjunkte Vereinigungen von Bäumen heißen Wälder. Präziser:

Die Graphen $B_1 = (V_1, E_1), \ldots, (V_k, E_k)$ seien Bäume und es gelte

$$V_i \cap V_j = \emptyset \text{ für } i \neq j.$$

Dann heißt $G = (V, E)$ ein **Wald**, falls

$$V = \bigcup_{i=1}^{k} V_i \text{ und } E = \bigcup_{i=1}^{k} E_k.$$

.

Wälder

Knoten-disjunkte Vereinigungen von Bäumen heißen Wälder. Präziser:

Die Graphen $B_1 = (V_1, E_1), \ldots, (V_k, E_k)$ seien Bäume und es gelte

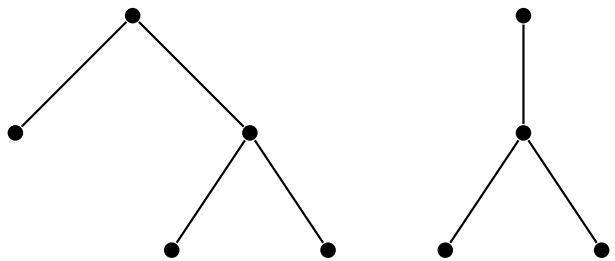
$$V_i \cap V_j = \emptyset \text{ für } i \neq j.$$

Dann heißt $G = (V, E)$ ein **Wald**, falls

$$V = \bigcup_{i=1}^{k} V_i \text{ und } E = \bigcup_{i=1}^{k} E_k.$$

.

Der Graph



ist kein Baum, da nicht zusammenhängend, wohl aber ein Wald.

Eigenschaften von Bäumen (1/2)

Sei $B = (V, E)$ ein Baum und seien $x, y \in V$ Knoten. Dann gibt es genau **einen** Weg von x nach y.

Beweis: Siehe Tafel.

Eigenschaften von Bäumen (1/2)

Sei $B = (V, E)$ ein Baum und seien $x, y \in V$ Knoten. Dann gibt es genau **einen** Weg von x nach y.

Beweis: Siehe Tafel.

Sei $B = (V, E)$ ein Baum. Wir nennen einen Knoten $b \in V$ ein

Blatt,

wenn b den Grad höchstens 1 besitzt. **Zeige**: B besitzt mindestens ein Blatt.

Beweis: Sei $(v_0, \ldots, v_\ell)$ ein einfacher Weg *maximaler* Länge in B. Wie stellt sich die Situation für v_ℓ dar?

Eigenschaften von Bäumen (2/2)

Sei $B = (V, E)$ ein Baum. Dann hat B genau einen Knoten mehr als Kanten,

d.h. es gilt $|E| = |V| - 1$.

Eigenschaften von Bäumen (2/2)

Sei $B = (V, E)$ ein Baum. Dann hat B genau einen Knoten mehr als Kanten,

$$\text{d.h. es gilt } |E| = |V| - 1.$$

Führe einen Beweis durch

vollständige Induktion nach der Anzahl n der Knoten von B.

(a) Der **Induktionsanfang** für $n = 1$:

Eigenschaften von Bäumen (2/2)

Sei $B = (V, E)$ ein Baum. Dann hat B genau einen Knoten mehr als Kanten,

d.h. es gilt $|E| = |V| - 1$.

Führe einen Beweis durch

vollständige Induktion nach der Anzahl n der Knoten von B.

(a) Der **Induktionsanfang** für $n = 1$: Ein ungerichteter Graph B mit einem Knoten v hat keine Kanten.

Eigenschaften von Bäumen (2/2)

Sei $B = (V, E)$ ein Baum. Dann hat B genau einen Knoten mehr als Kanten,

d.h. es gilt $|E| = |V| - 1$.

Führe einen Beweis durch

vollständige Induktion nach der Anzahl n der Knoten von B.

(a) Der **Induktionsanfang** für $n = 1$: Ein ungerichteter Graph B mit einem Knoten v hat keine Kanten.

(b) Der **Induktionsschritt**: Sei $B = (V, E)$ ein Baum mit $|V| = n + 1$ Knoten. Zeige, dass B genau n Kanten besitzt.

- Die **Induktionsannahme**: Jeder Baum mit n Knoten hat $n - 1$ Kanten.

Eigenschaften von Bäumen (2/2)

Sei $B = (V, E)$ ein Baum. Dann hat B genau einen Knoten mehr als Kanten,

d.h. es gilt $|E| = |V| - 1$.

Führe einen Beweis durch

vollständige Induktion nach der Anzahl n der Knoten von B.

(a) Der **Induktionsanfang** für $n = 1$: Ein ungerichteter Graph B mit einem Knoten v hat keine Kanten.

(b) Der **Induktionsschritt**: Sei $B = (V, E)$ ein Baum mit $|V| = n + 1$ Knoten. Zeige, dass B genau n Kanten besitzt.

- Die **Induktionsannahme**: Jeder Baum mit n Knoten hat $n - 1$ Kanten.
- **Entferne ein Blatt** b aus B und die eine mit b inzidente Kante.
  - ⋆ Der neue Baum B^* hat n Knoten und deshalb nach Induktionsannahme genau $n - 1$ Kanten.

Eigenschaften von Bäumen (2/2)

Sei $B = (V, E)$ ein Baum. Dann hat B genau einen Knoten mehr als Kanten,

d.h. es gilt $|E| = |V| - 1$.

Führe einen Beweis durch

vollständige Induktion nach der Anzahl n der Knoten von B.

(a) Der **Induktionsanfang** für $n = 1$: Ein ungerichteter Graph B mit einem Knoten v hat keine Kanten.

(b) Der **Induktionsschritt**: Sei $B = (V, E)$ ein Baum mit $|V| = n + 1$ Knoten. Zeige, dass B genau n Kanten besitzt.

- Die **Induktionsannahme**: Jeder Baum mit n Knoten hat $n - 1$ Kanten.
- **Entferne ein Blatt** b aus B und die eine mit b inzidente Kante.
  - ⋆ Der neue Baum B^* hat n Knoten und deshalb nach Induktionsannahme genau $n - 1$ Kanten.
  - ⋆ B hat genau einen Knoten und eine Kante mehr als $B^* \Longrightarrow$ $|E| - |V| = (n + 1) - (n - 1 + 1) = 1$.

Spannbäume

Bäume, die „ihren“ Graphen aufspannen

Sei $G = (V, E)$ ein ungerichteter Graph. Ein **Baum** $B = (V', E')$ heißt

Spannbaum von G,

falls $V' = V$ und $E' \subseteq E$ gilt.

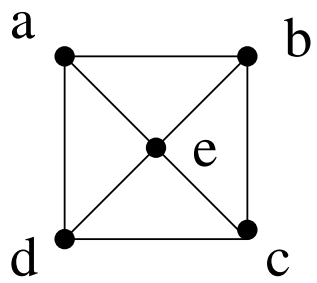
Bäume, die „ihren“ Graphen aufspannen

Sei $G = (V, E)$ ein ungerichteter Graph. Ein **Baum** $B = (V', E')$ heißt

Spannbaum von G,

falls $V' = V$ und $E' \subseteq E$ gilt.

Der Graph



hat u.a. folgende Spannbäume:

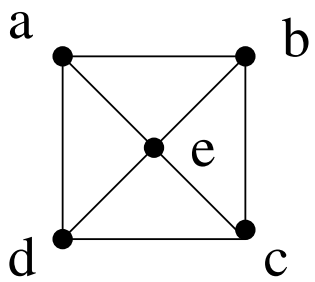
Bäume, die „ihren“ Graphen aufspannen

Sei $G = (V, E)$ ein ungerichteter Graph. Ein **Baum** $B = (V', E')$ heißt

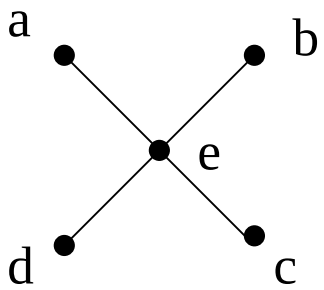
Spannbaum von G,

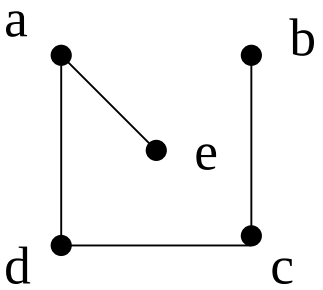
falls $V' = V$ und $E' \subseteq E$ gilt.

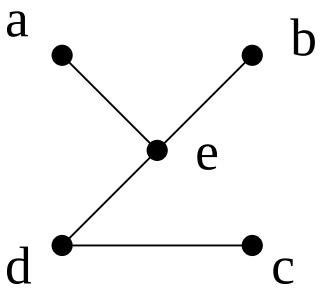
Der Graph



hat u.a. folgende Spannbäume:







Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von $G \iff G$ ist zusammenhängend.

Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von $G \iff G$ ist zusammenhängend.

$\Longrightarrow$ G ist zusammenhängend, denn

Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von $G \iff G$ ist zusammenhängend.

$\Longrightarrow$ G ist zusammenhängend, denn G hat nach Annahme einen Spannbaum und Bäume sind nach Definition zusammenhängend.

Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von G $\iff$ G ist zusammenhängend.

$\Longrightarrow$ G ist zusammenhängend, denn G hat nach Annahme einen Spannbaum und Bäume sind nach Definition zusammenhängend.

$\Longleftarrow$ Wir konstruieren einen Spannbaum:

Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von $G \iff G$ ist zusammenhängend.

$\Longrightarrow$ G ist zusammenhängend, denn G hat nach Annahme einen Spannbaum und Bäume sind nach Definition zusammenhängend.

$\Longleftarrow$ Wir konstruieren einen Spannbaum: Entferne eine Kante e aus G, wenn e zu einem Kreis gehört.

Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von $G \iff G$ ist zusammenhängend.

$\Longrightarrow$ G ist zusammenhängend, denn G hat nach Annahme einen Spannbaum und Bäume sind nach Definition zusammenhängend.

$\Longleftarrow$ Wir konstruieren einen Spannbaum: Entferne eine Kante e aus G, wenn e zu einem Kreis gehört.

- Der neue Graph hat eine Kante weniger, ist aber noch immer zusammenhängend.

Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von $G \iff G$ ist zusammenhängend.

$\Longrightarrow$ G ist zusammenhängend, denn G hat nach Annahme einen Spannbaum und Bäume sind nach Definition zusammenhängend.

$\Longleftarrow$ Wir konstruieren einen Spannbaum: Entferne eine Kante e aus G, wenn e zu einem Kreis gehört.

- Der neue Graph hat eine Kante weniger, ist aber noch immer zusammenhängend.
- Entferne solange Kanten, bis alle Kreise zerstört sind: Wir haben einen (Spann-)Baum erhalten!

Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von $G \iff G$ ist zusammenhängend.

$\Longrightarrow$ G ist zusammenhängend, denn G hat nach Annahme einen Spannbaum und Bäume sind nach Definition zusammenhängend.

$\Longleftarrow$ Wir konstruieren einen Spannbaum: Entferne eine Kante e aus G, wenn e zu einem Kreis gehört.

- Der neue Graph hat eine Kante weniger, ist aber noch immer zusammenhängend.
- Entferne solange Kanten, bis alle Kreise zerstört sind: Wir haben einen (Spann-)Baum erhalten!

Spannbäume sind die **kleinsten** zusammenhängenden Teilgraphen[a] eines zusammenhängenden Graphen.

[a] $G = (V, E)$ ist **Teilgraph** von $H = (V, F)$, wenn $E \subseteq F$ gilt.

Warum?

Spannbäume: Kleinste zusammenhängende Teilgraphen

Sei $G = (V, E)$ ein ungerichteter Graph. Dann gilt:

Es gibt mindestens einen Spannbaum von $G \iff G$ ist zusammenhängend.

$\Longrightarrow$ G ist zusammenhängend, denn G hat nach Annahme einen Spannbaum und Bäume sind nach Definition zusammenhängend.

$\Longleftarrow$ Wir konstruieren einen Spannbaum: Entferne eine Kante e aus G, wenn e zu einem Kreis gehört.

- Der neue Graph hat eine Kante weniger, ist aber noch immer zusammenhängend.
- Entferne solange Kanten, bis alle Kreise zerstört sind: Wir haben einen (Spann-)Baum erhalten!

Spannbäume sind die **kleinsten** zusammenhängenden Teilgraphen[a] eines zusammenhängenden Graphen.

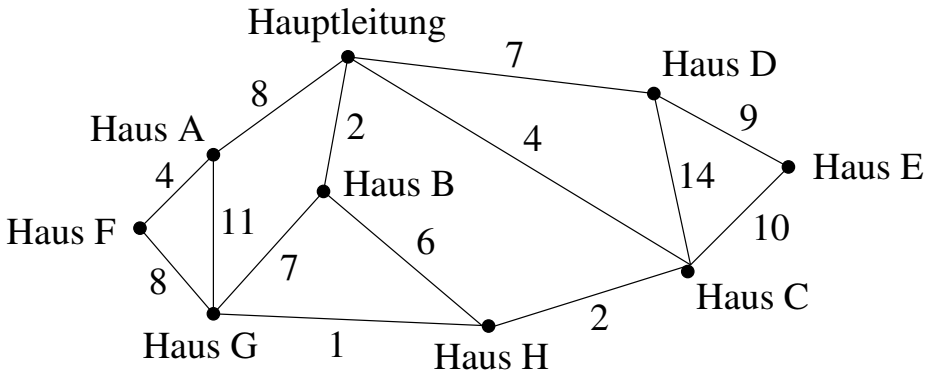
[a] $G = (V, E)$ ist **Teilgraph** von $H = (V, F)$, wenn $E \subseteq F$ gilt.

Warum? Ein kleinster zusammenhängender Teilgraph enthält keine Kreise und muss deshalb ein Baum sein!

Kabelfernsehen (1/2)

Eine Firma will Leitungen zum Empfang von Kabelfernsehen in einem neuen Wohngebiet verlegen.

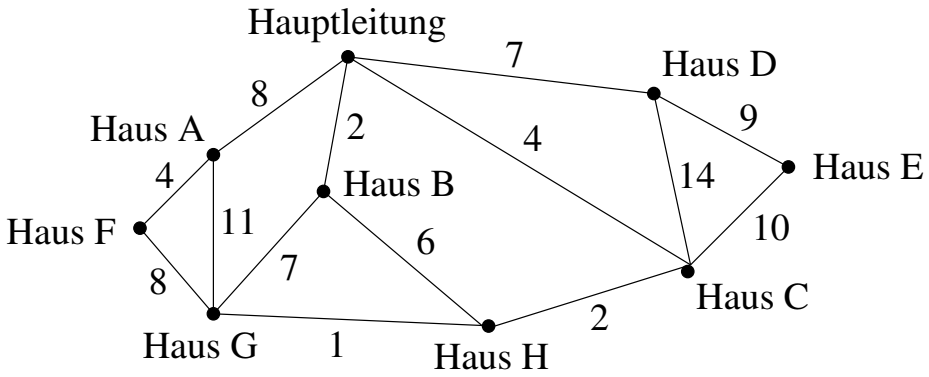
Der folgende Graph beschreibt die Leitungskosten:



Kabelfernsehen (1/2)

Eine Firma will Leitungen zum Empfang von Kabelfernsehen in einem neuen Wohngebiet verlegen.

Der folgende Graph beschreibt die Leitungskosten:



- Knoten entsprechen einzelnen Häusern bzw. der Hauptleitung, die aus einem bereits verkabelten Gebiet herangeführt wird.
- Eine Kante zwischen zwei Knoten zeigt, dass eine Direktleitung gelegt werden kann. Die Kante ist mit den Kosten der Verlegung (in TEuro) beschriftet.

Bestimme eine billigste Verkabelung, die alle Häuser, (möglicherweise über eine Folge von Kabeln) an die Hauptleitung anschließt!

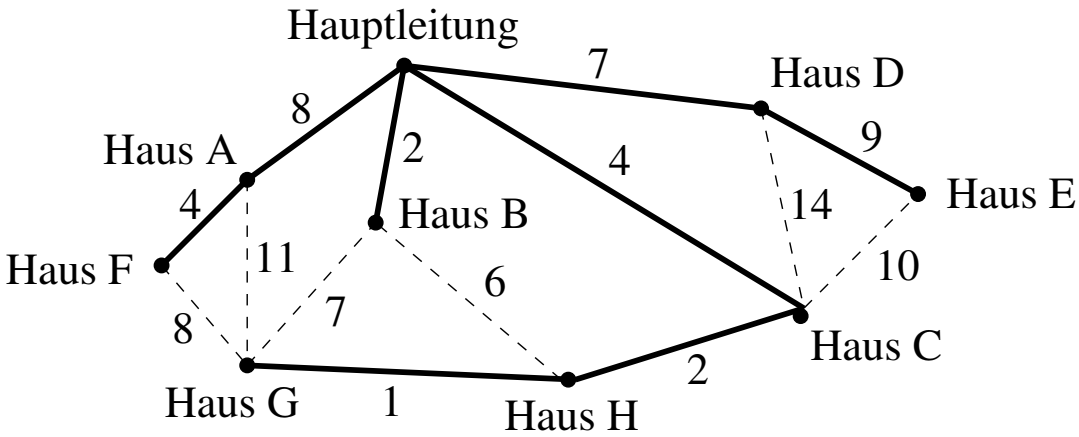
Kabelfernsehen (2/2)

Wir suchen einen **minimalen Spannbaum**, also einen Spannbaum mit möglichst kleiner Summe von Kantengewichten.

Kabelfernsehen (2/2)

Wir suchen einen **minimalen Spannbaum**, also einen Spannbaum mit möglichst kleiner Summe von Kantengewichten.

Die **fett** gezeichneten Kanten geben die Kanten eines minimalen Spannbaums an:



Verlegt die Firma genau diese Leitungen, so hat sie das neue Wohngebiet mit den geringstmöglichen Kosten ans Kabelfernsehen angeschlossen.

In den Veranstaltungen „**Datenstrukturen**“ und „**Theoretische Informatik 1**“ werden schnelle Algorithmen zur Bestimmung minimaler Spannbäume besprochen.

Gewurzelte Bäume
(„gerichtete Bäume“)

Bäume mit Wurzeln

Wir erhalten einen

gewurzelten Baum,

also einen gerichteten Baum mit Wurzel, indem man

1. in einem ungerichteten Baum einen Knoten als „**Wurzel**“ auswählt und
2. alle Kanten in die Richtung orientiert, die von der Wurzel weg führt.

Bäume mit Wurzeln

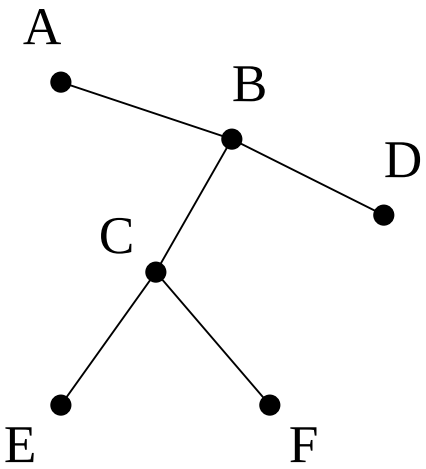
Wir erhalten einen

gewurzelten Baum,

also einen gerichteten Baum mit Wurzel, indem man

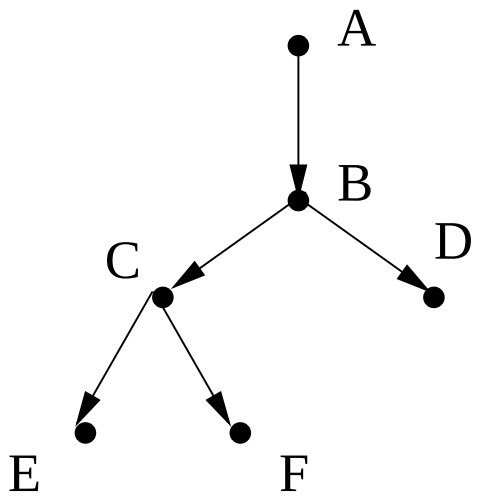
1. in einem ungerichteten Baum einen Knoten als „**Wurzel**“ auswählt und
2. alle Kanten in die Richtung orientiert, die von der Wurzel weg führt.

Ein ungerichteter Baum



und der zugehörige gerichtete Baum mit Wurzel A

$G_A :=$



Gewurzelte Bäume: Die formale Definition

Die präzise Definition des Begriffs „gewurzelter Baum“:

Ein gerichteter Graph $G = (V, E)$ mit einer endlichen Anzahl von Knoten heißt **gewurzelter Baum**, falls er folgende Eigenschaften hat:

Gewurzelte Bäume: Die formale Definition

Die präzise Definition des Begriffs „gewurzelter Baum“:

Ein gerichteter Graph $G = (V, E)$ mit einer endlichen Anzahl von Knoten heißt **gewurzelter Baum**, falls er folgende Eigenschaften hat:

(1) G besitzt genau einen Knoten $w \in V$ mit Ein-Grad$_G(w) = 0$.
Dieser Knoten wird **Wurzel** genannt.

(2) Für jeden Knoten $v \in V$ gilt:
Es gibt in G einen Weg von der Wurzel zum Knoten v.

(3) Für jeden Knoten $v \in V$ gilt: Ein-Grad$_G(v) \leqslant 1$.

Blätter, innere Knoten, Tiefe, Höhe, Eltern und Kinder

Blätter, innere Knoten und Kinder

Sei $B = (V, E)$ ein gewurzelter Baum mit Wurzel w.

(a) Blätter und innere Knoten:

- Knoten mit Aus-Grad 0 heißen **Blätter**,
- Knoten, die weder Wurzel noch Blätter sind, heißen **innere Knoten**.

Blätter, innere Knoten und Kinder

Sei $B = (V, E)$ ein gewurzelter Baum mit Wurzel w.

(a) Blätter und innere Knoten:

- Knoten mit Aus-Grad 0 heißen **Blätter**,
- Knoten, die weder Wurzel noch Blätter sind, heißen **innere Knoten**.

(b) Sei $v \in V$ ein Knoten von B.

- Die **Höhe von** v ist die Länge eines längsten Weges von v zu einem Blatt.
  Die **Höhe von** B ist die Höhe der Wurzel w.
- Die **Tiefe von** v ist die Länge des Weges von der Wurzel w zu v.
  Die **Tiefe von** B ist die maximale Tiefe eines Blatts.

Blätter, innere Knoten und Kinder

Sei $B = (V, E)$ ein gewurzelter Baum mit Wurzel w.

(a) Blätter und innere Knoten:

- Knoten mit Aus-Grad 0 heißen **Blätter**,
- Knoten, die weder Wurzel noch Blätter sind, heißen **innere Knoten**.

(b) Sei $v \in V$ ein Knoten von B.

- Die **Höhe von** v ist die Länge eines längsten Weges von v zu einem Blatt.
  Die **Höhe von** B ist die Höhe der Wurzel w.
- Die **Tiefe von** v ist die Länge des Weges von der Wurzel w zu v.
  Die **Tiefe von** B ist die maximale Tiefe eines Blatts.

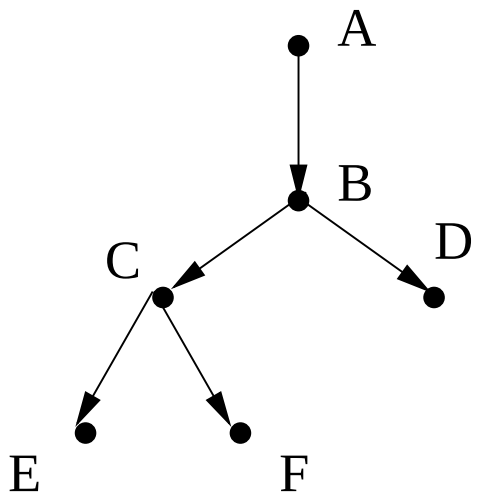
(c) Für jede Kante $(u, v) \in E$ heißt

- u der **Elternknoten** von v und
- v ein **Kind** von u.

Beispiele

Im Graphen

$G_A =$

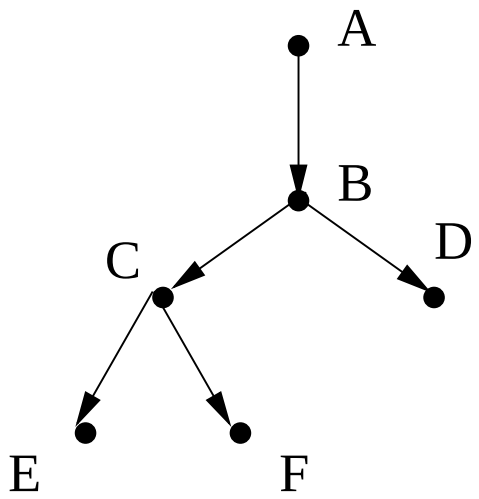


gilt:

- Knoten A hat Tiefe

Beispiele

Im Graphen

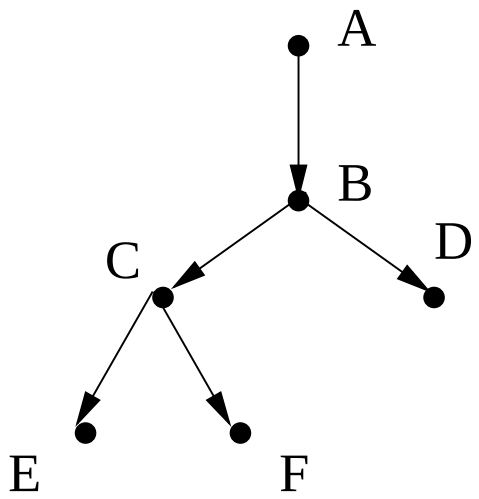
$G_A =$ 

gilt:

- Knoten A hat Tiefe 0, Höhe

Beispiele

Im Graphen

$G_A =$ 

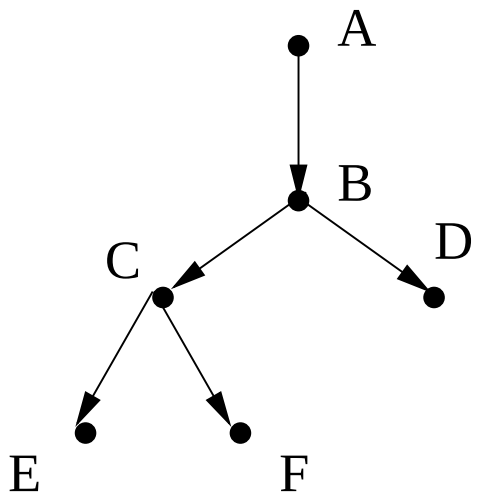
gilt:

- Knoten A hat Tiefe 0, Höhe 3 und A hat genau ein Kind, nämlich B;
- Knoten B hat Tiefe

Beispiele

Im Graphen

$G_A =$



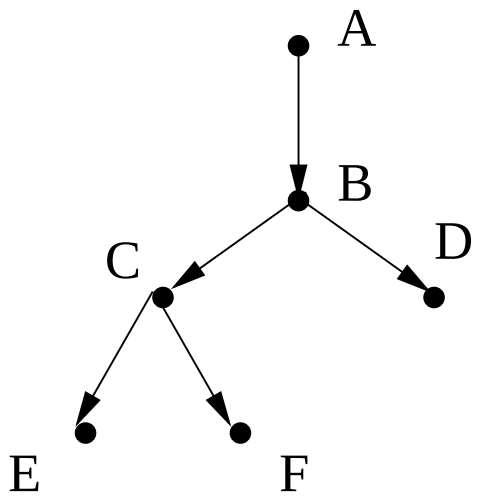
gilt:

- Knoten A hat Tiefe 0, Höhe 3 und A hat genau ein Kind, nämlich B;
- Knoten B hat Tiefe 1, Höhe

Beispiele

Im Graphen

$G_A =$

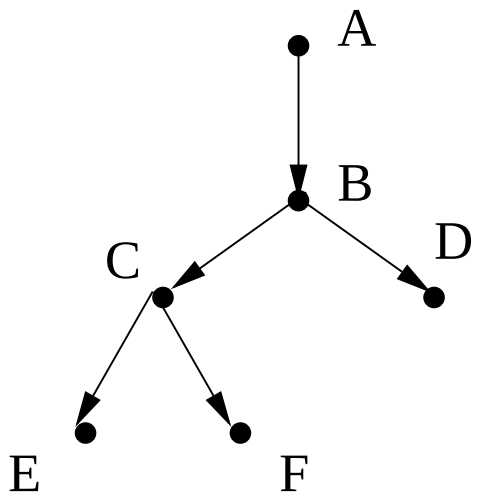


gilt:

- Knoten A hat Tiefe 0, Höhe 3 und A hat genau ein Kind, nämlich B;
- Knoten B hat Tiefe 1, Höhe 2 und B hat genau zwei Kinder, nämlich C und D;
- Knoten C hat Tiefe

Beispiele

Im Graphen

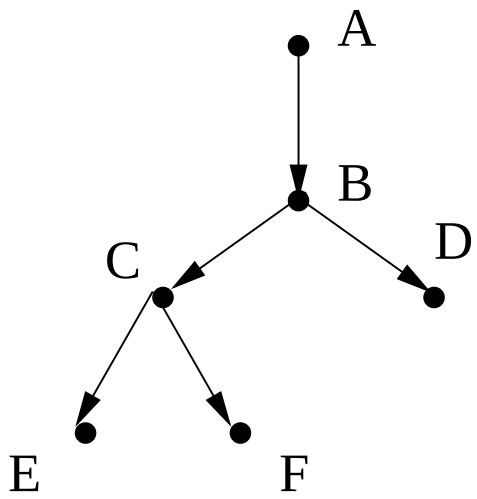
$G_A =$ 

gilt:

- Knoten A hat Tiefe 0, Höhe 3 und A hat genau ein Kind, nämlich B;
- Knoten B hat Tiefe 1, Höhe 2 und B hat genau zwei Kinder, nämlich C und D;
- Knoten C hat Tiefe 2, Höhe

Beispiele

Im Graphen

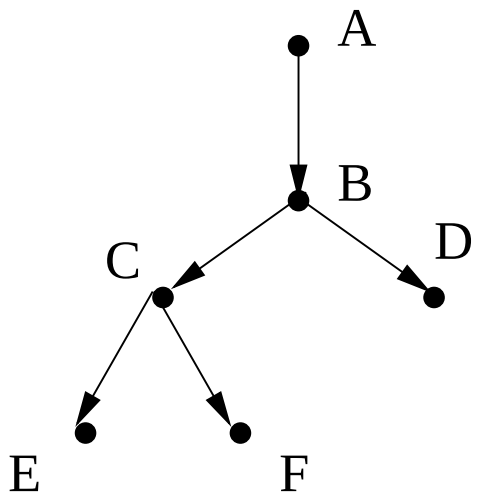
$G_A =$ 

gilt:

- Knoten A hat Tiefe 0, Höhe 3 und A hat genau ein Kind, nämlich B;
- Knoten B hat Tiefe 1, Höhe 2 und B hat genau zwei Kinder, nämlich C und D;
- Knoten C hat Tiefe 2, Höhe 1 und C hat genau zwei Kinder, nämlich C und F;

Beispiele

Im Graphen

$G_A =$ 

gilt:

- Knoten A hat Tiefe 0, Höhe 3 und A hat genau ein Kind, nämlich B;
- Knoten B hat Tiefe 1, Höhe 2 und B hat genau zwei Kinder, nämlich C und D;
- Knoten C hat Tiefe 2, Höhe 1 und C hat genau zwei Kinder, nämlich C und F;
- die Knoten D (Tiefe 2), E (Tiefe 3), F (Tiefe 3) haben keine Kinder.
  Als Blätter haben alle drei die Höhe 0.

Binärbäume

Binärbäume

(a) Ein gewurzelter Baum $B = (V, E)$ heißt **Binärbaum**, falls für jeden Knoten $v \in V$ gilt: Aus-Grad$_B(v) \leqslant 2$.

Binärbäume

(a) Ein gewurzelter Baum $B = (V, E)$ heißt **Binärbaum**, falls für jeden Knoten $v \in V$ gilt: Aus-Grad$_B(v) \leqslant 2$.

(b) $B = (V, E)$ heißt ein **voller Binärbaum**, falls stets Aus-Grad$_B(v) = 2$ gilt, wenn v kein Blatt ist.

Binärbäume

(a) Ein gewurzelter Baum $B = (V, E)$ heißt **Binärbaum**, falls für jeden Knoten $v \in V$ gilt: Aus-Grad$_B(v) \leqslant 2$.

(b) $B = (V, E)$ heißt ein **voller Binärbaum**, falls stets Aus-Grad$_B(v) = 2$ gilt, wenn v kein Blatt ist.

(c) Ein Binärbaum $B = (V, E)$ heißt **vollständiger Binärbaum**, falls gilt:

(1) Es ist Aus-Grad$(v) = 2$ für jeden Knoten v, der kein Blatt ist.

(2) Alle Blätter von B haben dieselbe Tiefe.

Binärbäume

(a) Ein gewurzelter Baum $B = (V, E)$ heißt **Binärbaum**, falls für jeden Knoten $v \in V$ gilt: Aus-Grad$_B(v) \leqslant 2$.

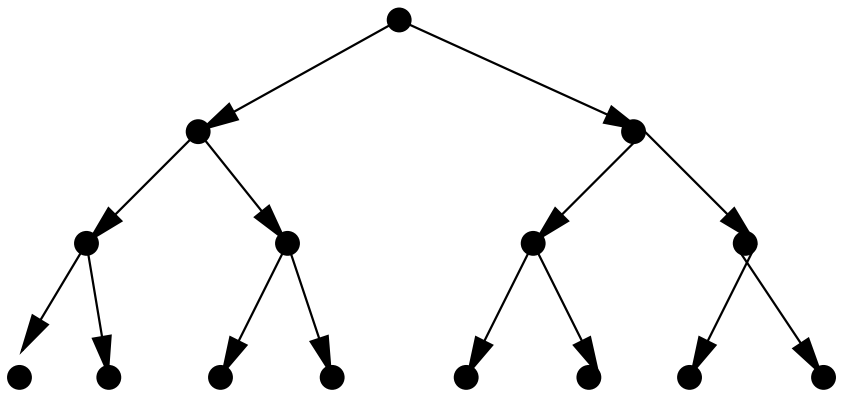
(b) $B = (V, E)$ heißt ein **voller Binärbaum**, falls stets Aus-Grad$_B(v) = 2$ gilt, wenn v kein Blatt ist.

(c) Ein Binärbaum $B = (V, E)$ heißt **vollständiger Binärbaum**, falls gilt:

(1) Es ist Aus-Grad$(v) = 2$ für jeden Knoten v, der kein Blatt ist.

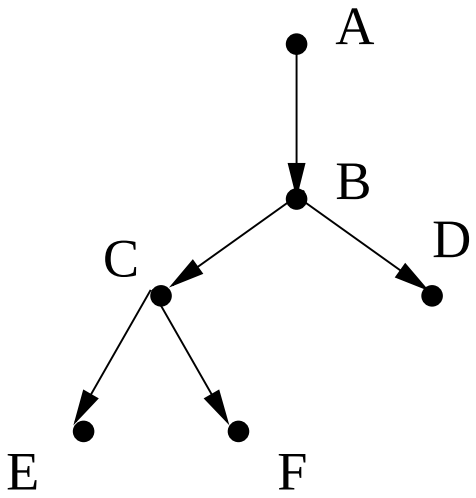
(2) Alle Blätter von B haben dieselbe Tiefe.

Der folgende Graph B_3 ist ein **vollständiger Binärbaum** der Höhe 3:

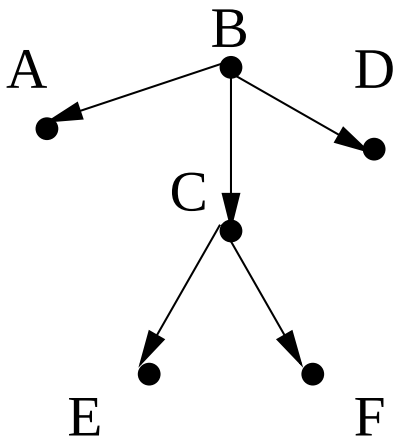


Binärbäume: Beispiele

$G_A =$

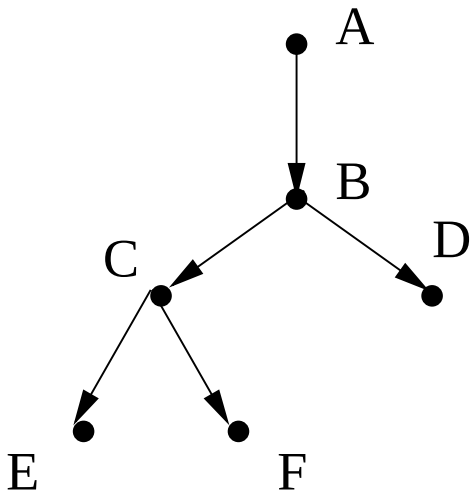


$G_B =$

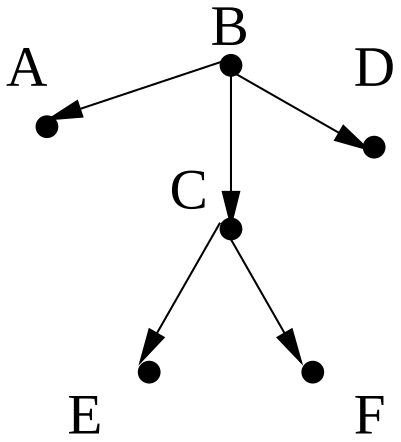


Binärbäume: Beispiele

$G_A =$



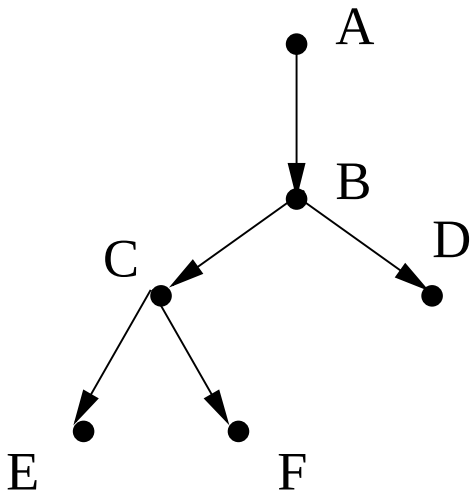
$G_B =$



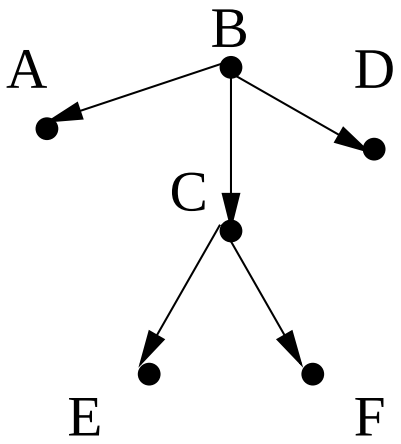
Nur G_A ist binär (aber G_A ist **kein** voller Binärbaum).
Beide Bäume stammen vom selben ungerichteten Baum ab, nämlich von

Binärbäume: Beispiele

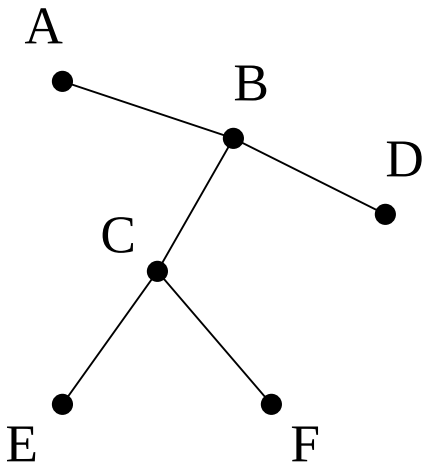
$G_A =$



$G_B =$



Nur G_A ist binär (aber G_A ist **kein** voller Binärbaum).
Beide Bäume stammen vom selben ungerichteten Baum ab, nämlich von



Wieviele Knoten hat ein Binärbaum der Tiefe t? (1/2)

Sei $t \in \mathbb{N}$.

(a) Jeder **vollständige Binärbaum der Tiefe** t hat genau 2^t Blätter und genau $2^{t+1} - 1$ Knoten.

(b) Jeder **Binärbaum der Tiefe t** hat höchstens 2^t Blätter und höchstens $2^{t+1} - 1$ Knoten.

Wieviele Knoten hat ein Binärbaum der Tiefe t? (1/2)

Sei $t \in \mathbb{N}$.

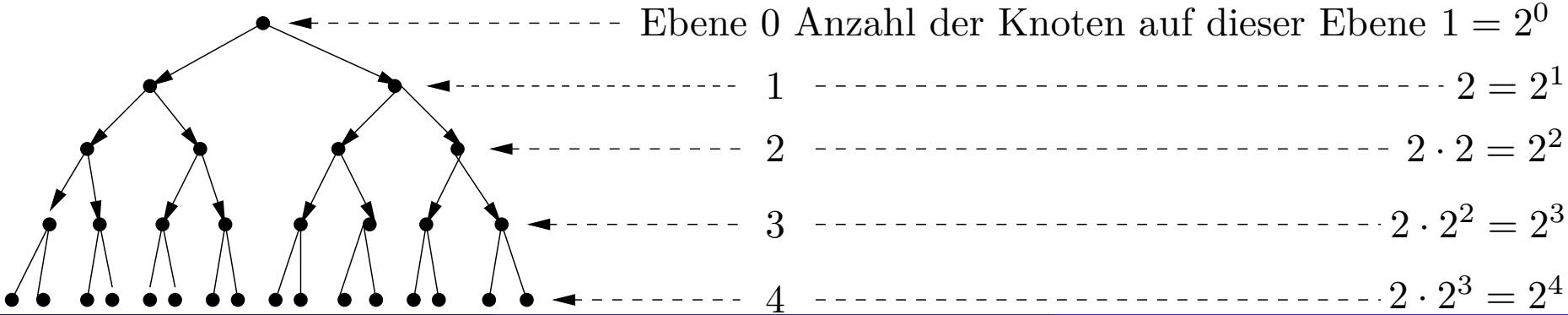
(a) Jeder **vollständige Binärbaum der Tiefe** t hat genau 2^t Blätter und genau $2^{t+1} - 1$ Knoten.

(b) Jeder **Binärbaum der Tiefe t** hat höchstens 2^t Blätter und höchstens $2^{t+1} - 1$ Knoten.

Intuition: Nach der Skizze sieht es so aus, dass ein vollständiger Binärbaum der Tiefe t genau 2^T Knoten der Tiefe T (für $T \leqslant t$) besitzt und deshalb gibt es genau

$$2^0 + 2^1 + 2^2 + \cdots + 2^t \;=\; 2^{t+1} - 1$$

Knoten.



Wieviele Knoten hat ein Binärbaum der Tiefe t? (2/2)

Aber wie formalisiert man die Intuition?

Wieviele Knoten hat ein Binärbaum der Tiefe t? (2/2)

Aber wie formalisiert man die Intuition?

Sei B ein vollständiger Binärbaum der *Tiefe* t. Wir bestimmen die Anzahl der Knoten der *Tiefe* T in B mit vollständiger Induktion nach T für alle $T \leqslant t$.

1. Wir beobachten, dass es (im Vergleich zur Anzahl der Knoten der Tiefe T), **genau** doppelt so viele Knoten der Tiefe $T+1$ gibt, falls $T+1 \leqslant t$.

Wieviele Knoten hat ein Binärbaum der Tiefe t? (2/2)

Aber wie formalisiert man die Intuition?

Sei B ein vollständiger Binärbaum der *Tiefe* t. Wir bestimmen die Anzahl der Knoten der *Tiefe* T in B mit vollständiger Induktion nach T für alle $T \leqslant t$.

1. Wir beobachten, dass es (im Vergleich zur Anzahl der Knoten der Tiefe T), **genau** doppelt so viele Knoten der Tiefe $T+1$ gibt, falls $T+1 \leqslant t$.
   - Wenn a_T die Anzahl der Knoten der Tiefe T ist, dann ist

$$a_0 = 1 \text{ und } a_{T+1} = 2a_T$$

Wieviele Knoten hat ein Binärbaum der Tiefe t? (2/2)

Aber wie formalisiert man die Intuition?

Sei B ein vollständiger Binärbaum der *Tiefe* t. Wir bestimmen die Anzahl der Knoten der *Tiefe* T in B mit vollständiger Induktion nach T für alle $T \leqslant t$.

1. Wir beobachten, dass es (im Vergleich zur Anzahl der Knoten der Tiefe T), **genau** doppelt so viele Knoten der Tiefe $T+1$ gibt, falls $T+1 \leqslant t$.
   - Wenn a_T die Anzahl der Knoten der Tiefe T ist, dann ist

$$a_0 = 1 \text{ und } a_{T+1} = 2a_T$$

2. Sodann zeige mit einer vollständigen Induktion nach T, dass gilt

$$a_T = 2^T.$$

Wieviele Knoten hat ein Binärbaum der Tiefe t? (2/2)

Aber wie formalisiert man die Intuition?

Sei B ein vollständiger Binärbaum der *Tiefe* t. Wir bestimmen die Anzahl der Knoten der *Tiefe* T in B mit vollständiger Induktion nach T für alle $T \leqslant t$.

1. Wir beobachten, dass es (im Vergleich zur Anzahl der Knoten der Tiefe T), **genau** doppelt so viele Knoten der Tiefe $T+1$ gibt, falls $T+1 \leqslant t$.
   - Wenn a_T die Anzahl der Knoten der Tiefe T ist, dann ist
   $$a_0 = 1 \text{ und } a_{T+1} = 2a_T$$
2. Sodann zeige mit einer vollständigen Induktion nach T, dass gilt
   $$a_T = 2^T.$$
3. B hat 2^t Blätter und $2^0 + 2^1 + \cdots + 2^t =$

Wieviele Knoten hat ein Binärbaum der Tiefe t? (2/2)

Aber wie formalisiert man die Intuition?

Sei B ein vollständiger Binärbaum der *Tiefe* t. Wir bestimmen die Anzahl der Knoten der *Tiefe* T in B mit vollständiger Induktion nach T für alle $T \leqslant t$.

1. Wir beobachten, dass es (im Vergleich zur Anzahl der Knoten der Tiefe T), **genau** doppelt so viele Knoten der Tiefe $T+1$ gibt, falls $T+1 \leqslant t$.
   - Wenn a_T die Anzahl der Knoten der Tiefe T ist, dann ist
   $$a_0 = 1 \text{ und } a_{T+1} = 2a_T$$
2. Sodann zeige mit einer vollständigen Induktion nach T, dass gilt
   $$a_T = 2^T.$$
3. B hat 2^t Blätter und $2^0 + 2^1 + \cdots + 2^t = 2^{t+1} - 1$ Knoten.

Wieviele Knoten hat ein Binärbaum der Tiefe t? (2/2)

Aber wie formalisiert man die Intuition?

Sei B ein vollständiger Binärbaum der *Tiefe* t. Wir bestimmen die Anzahl der Knoten der *Tiefe* T in B mit vollständiger Induktion nach T für alle $T \leqslant t$.

1. Wir beobachten, dass es (im Vergleich zur Anzahl der Knoten der Tiefe T), **genau** doppelt so viele Knoten der Tiefe $T+1$ gibt, falls $T+1 \leqslant t$.
   - Wenn a_T die Anzahl der Knoten der Tiefe T ist, dann ist
   $$a_0 = 1 \text{ und } a_{T+1} = 2a_T$$
2. Sodann zeige mit einer vollständigen Induktion nach T, dass gilt
   $$a_T = 2^T.$$
3. B hat 2^t Blätter und $2^0 + 2^1 + \cdots + 2^t = 2^{t+1} - 1$ Knoten.

Aber warum hat *jeder* Binärbaum der Tiefe t *höchstens* 2^t Blätter und *höchstens* $2^{t+1} - 1$ Knoten?

> *Unter allen Binärbäumen der Tiefe t hat der vollständige Binärbaum die meisten Blätter und Knoten.*

Modellierungsbeispiele für gewurzelte Bäume mit Knoten-/Kantenmarkierungen

Modellierung mit gewurzelten Bäumen

Gewurzelte Bäume werden eingesetzt, um

- **Entwicklungen** zu verfolgen:
  - Stammbäume (in der Genealogie) geben die Familiengeschichte wieder,
  - phylogenetische Bäume (in der Bioinformatik) zeigen evolutionäre Beziehungen zwischen verschiedenen Arten auf.
- **hierarchische Strukturen** zu repräsentieren wie etwa
  - eine Verzeichnisstruktur oder die
  - Organisationsstrukur einer Firma.
- **Datenstrukturen** für die Suche nach Daten zu entwickeln. (Siehe die gleichnamige Veranstaltung.)
  - Beispiele sind binäre Suchbäume, AVL-Bäume oder B-Bäume.
- **rekursive Definitionen** zu veranschaulichen, bzw. zu verstehen:
  - **Syntaxbäume** repräsentieren (aussagenlogische) Formeln, arithmetische Ausdrücke, XML-Dokumente oder die Syntax einer Programmiersprache,
  - **Rekursionsbäume** veranschaulichen rekursive Programme.
- komplexe Entscheidungen **systematisch** aufzulisten. Beispiele sind
  - Spielbäume,
  - Entscheidungsbäume.

Syntaxbäume

Wir haben aussagenlogische Formeln durch

eine rekursive Definition über den Aufbau der Formel

hergeleitet.

(a) Der **Syntaxbaum einer Formel** bildet die Rekursion nach:

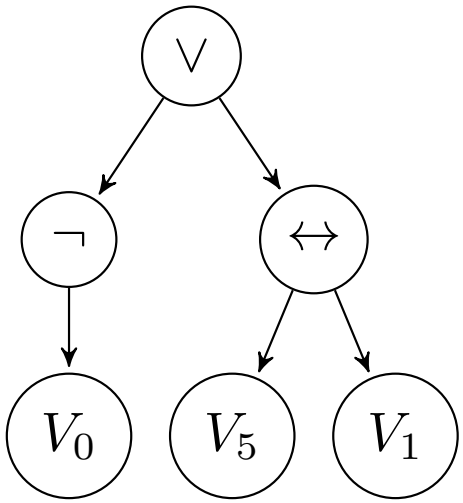
Innere Knoten wie auch die Wurzel sind jeweils mit einem Junktor **markiert**, die Blätter sind mit Variablen **markiert**.

(b) Syntaxbäume treten auch bei der Compilierung auf.

Um ein *Programm* auszuführen, erzeugt der Compiler den **Syntaxbaum des Programms**. (Mehr Details im Kapitel über „kontextfreie Grammatiken“.)

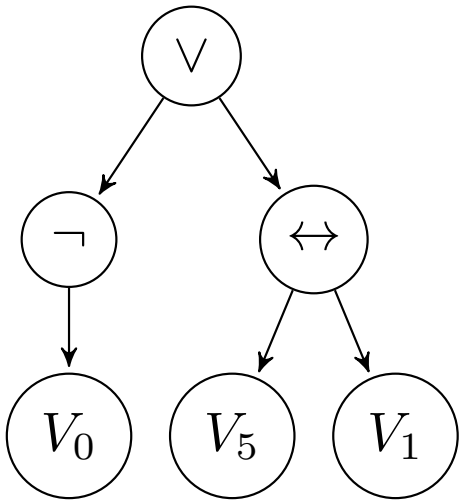
Die Syntax aussagenlogischer Formeln

Die Bedeutung der *aussagenlogischen Formel* $\big(\neg V_0 \vee (V_5 \leftrightarrow V_1)\big)$ wird sofort klar, wenn wir ihren Syntaxbaum betrachten:



Die Syntax aussagenlogischer Formeln

Die Bedeutung der *aussagenlogischen Formel* $\big(\neg V_0 \vee (V_5 \leftrightarrow V_1)\big)$ wird sofort klar, wenn wir ihren Syntaxbaum betrachten:



Für aussagenlogische Formeln ist der Syntaxbaum stets **binär**. Warum?

Rekursionsbäume

Rekursive Programme und ihre Rekursionsbäume

R($\vec{p}$) sei ein rekursives Programm mit den Parametern $\vec{p}$.
Wir führen den Rekursionsbaum **Baum($\vec{p}$)** nach den folgenden Regeln ein.

1. Beschrifte die Wurzel von **Baum($\vec{p}$)** mit den Parametern $\vec{\mathbf{p}}$ des Erstaufrufs.

Rekursive Programme und ihre Rekursionsbäume

R($\vec{p}$) sei ein rekursives Programm mit den Parametern $\vec{p}$.
Wir führen den Rekursionsbaum **Baum($\vec{p}$)** nach den folgenden Regeln ein.

1. Beschrifte die Wurzel von **Baum($\vec{p}$)** mit den Parametern $\vec{\mathbf{p}}$ des Erstaufrufs.
2. Sei v ein Knoten von **Baum($\vec{p}$)**, der mit den Parametern $\vec{\mathbf{q}}$ beschriftet ist.
   - Wenn in **R($\vec{q}$)** keine rekursiven Aufrufe getätigt werden, dann ist v ein Blatt.
   - Ansonsten erhält v für jeden rekursiven Aufruf innerhalb von **R($\vec{q}$)** ein neues Kind, das mit den Parametern des rekursiven Aufrufs beschriftet wird.

Rekursive Programme und ihre Rekursionsbäume

R($\vec{p}$) sei ein rekursives Programm mit den Parametern $\vec{p}$.
Wir führen den Rekursionsbaum **Baum($\vec{p}$)** nach den folgenden Regeln ein.

1. Beschrifte die Wurzel von **Baum($\vec{p}$)** mit den Parametern $\vec{\mathbf{p}}$ des Erstaufrufs.
2. Sei v ein Knoten von **Baum($\vec{p}$)**, der mit den Parametern $\vec{\mathbf{q}}$ beschriftet ist.
   - Wenn in **R($\vec{q}$)** keine rekursiven Aufrufe getätigt werden, dann ist v ein Blatt.
   - Ansonsten erhält v für jeden rekursiven Aufruf innerhalb von **R($\vec{q}$)** ein neues Kind, das mit den Parametern des rekursiven Aufrufs beschriftet wird.

Der Rekursionsbaum **Baum($\vec{p}$)** veranschaulicht die Arbeitsweise von **R($\vec{p}$)**:

Die Anzahl innerer Knoten von **Baum($\vec{p}$)** stimmt überein mit

Rekursive Programme und ihre Rekursionsbäume

R($\vec{p}$) sei ein rekursives Programm mit den Parametern $\vec{p}$.
Wir führen den Rekursionsbaum **Baum($\vec{p}$)** nach den folgenden Regeln ein.

1. Beschrifte die Wurzel von **Baum($\vec{p}$)** mit den Parametern $\vec{\mathbf{p}}$ des Erstaufrufs.
2. Sei v ein Knoten von **Baum($\vec{p}$)**, der mit den Parametern $\vec{\mathbf{q}}$ beschriftet ist.
   - Wenn in **R($\vec{q}$)** keine rekursiven Aufrufe getätigt werden, dann ist v ein Blatt.
   - Ansonsten erhält v für jeden rekursiven Aufruf innerhalb von **R($\vec{q}$)** ein neues Kind, das mit den Parametern des rekursiven Aufrufs beschriftet wird.

Der Rekursionsbaum **Baum($\vec{p}$)** veranschaulicht die Arbeitsweise von **R($\vec{p}$)**:

Die Anzahl innerer Knoten von **Baum($\vec{p}$)** stimmt überein mit der Anzahl rekursiver Aufrufe, die irgendwann während der Abarbeitung von **R($\vec{p}$)** getätigt werden.

Binärsuche

Binärsuche (1/2)

Binärsuche wird auf ein *aufsteigend sortiertes* Array A angewandt.

Es ist zu überprüfen, ob der Schlüssel y in A vorkommt.

Binärsuche (1/2)

Binärsuche wird auf ein *aufsteigend sortiertes* Array A angewandt.

Es ist zu überprüfen, ob der Schlüssel y in A vorkommt.

Das Programm **Binärsuche**(y, A):

1. Wenn A die Länge 1 hat, dann überprüfe direkt, ob y in A vorkommt.
   \# **Induktionsanfang** für Länge 1: Binärsuche findet y falls vorhanden.

Binärsuche (1/2)

Binärsuche wird auf ein *aufsteigend sortiertes* Array A angewandt.

Es ist zu überprüfen, ob der Schlüssel y in A vorkommt.

Das Programm **Binärsuche**(y, A):

1. Wenn A die Länge 1 hat, dann überprüfe direkt, ob y in A vorkommt.
   # **Induktionsanfang** für Länge 1: Binärsuche findet y falls vorhanden.
2. Sonst sei
$$A = A_{\text{links}}\, A[\text{mitte}]\, A_{\text{rechts}}.$$

   (a) Wenn $y = A[\text{mitte}]$, dann halte: y befindet sich in Position mitte“.

Binärsuche (1/2)

Binärsuche wird auf ein *aufsteigend sortiertes* Array A angewandt.

Es ist zu überprüfen, ob der Schlüssel y in A vorkommt.

Das Programm **Binärsuche**(y, A):

1. Wenn A die Länge 1 hat, dann überprüfe direkt, ob y in A vorkommt.
   # **Induktionsanfang** für Länge 1: Binärsuche findet y falls vorhanden.
2. Sonst sei
$$A = A_{\text{links}}\, A[\text{mitte}]\, A_{\text{rechts}}.$$
   (a) Wenn $y = A[\text{mitte}]$, dann halte: y befindet sich in Position mitte“.
   (b) Sonst, wenn $y < A[\text{mitte}]$, dann

Binärsuche (1/2)

Binärsuche wird auf ein *aufsteigend sortiertes* Array A angewandt.

Es ist zu überprüfen, ob der Schlüssel y in A vorkommt.

Das Programm **Binärsuche**(y, A):

1. Wenn A die Länge 1 hat, dann überprüfe direkt, ob y in A vorkommt.
   # **Induktionsanfang** für Länge 1: Binärsuche findet y falls vorhanden.
2. Sonst sei
$$A = A_{\text{links}}\, A[\text{mitte}]\, A_{\text{rechts}}.$$
   (a) Wenn $y = A[\text{mitte}]$, dann halte: y befindet sich in Position mitte“.
   (b) Sonst, wenn $y < A[\text{mitte}]$, dann rufe **Binärsuche**(y, A_{links}) auf.
   # *Induktionsvoraussetzung*: Binärsuche findet y in A_{links} falls vorhanden.

Binärsuche (1/2)

Binärsuche wird auf ein *aufsteigend sortiertes* Array A angewandt.

Es ist zu überprüfen, ob der Schlüssel y in A vorkommt.

Das Programm **Binärsuche**(y, A):

1. Wenn A die Länge 1 hat, dann überprüfe direkt, ob y in A vorkommt.
   # **Induktionsanfang** für Länge 1: Binärsuche findet y falls vorhanden.
2. Sonst sei
$$A = A_{\text{links}}\, A[\text{mitte}]\, A_{\text{rechts}}.$$
   (a) Wenn $y = A[\text{mitte}]$, dann halte: y befindet sich in Position mitte".
   (b) Sonst, wenn $y < A[\text{mitte}]$, dann rufe **Binärsuche**(y, A_{links}) auf.
   # *Induktionsvoraussetzung*: Binärsuche findet y in A_{links} falls vorhanden.
   (c) Sonst, wenn $y > A[\text{mitte}]$, dann rufe **Binärsuche**(y, A_{rechts}) auf.
   # *Induktionsvoraussetzung*: Binärsuche findet y in A_{rechts} falls vorhanden.

Binärsuche (1/2)

Binärsuche wird auf ein *aufsteigend sortiertes* Array A angewandt.

Es ist zu überprüfen, ob der Schlüssel y in A vorkommt.

Das Programm **Binärsuche**(y, A):

1. Wenn A die Länge 1 hat, dann überprüfe direkt, ob y in A vorkommt.
   # **Induktionsanfang** für Länge 1: Binärsuche findet y falls vorhanden.
2. Sonst sei
   $$A = A_{\text{links}}\, A[\text{mitte}]\, A_{\text{rechts}}.$$
   (a) Wenn $y = A[\text{mitte}]$, dann halte: y befindet sich in Position mitte“.
   (b) Sonst, wenn $y < A[\text{mitte}]$, dann rufe **Binärsuche**(y, A_{links}) auf.
   # *Induktionsvoraussetzung*: Binärsuche findet y in A_{links} falls vorhanden.
   (c) Sonst, wenn $y > A[\text{mitte}]$, dann rufe **Binärsuche**(y, A_{rechts}) auf.
   # *Induktionsvoraussetzung*: Binärsuche findet y in A_{rechts} falls vorhanden.

   # **Induktionsschritt**: Da Binärsuche in der richtigen Hälfte sucht, wird y gefunden falls vorhanden.

Binärsuche (2/2)

Wie sieht der Rekursionsbaum **Baum**(y, A) aus, wenn y nicht in A vorkommt?

Binärsuche (2/2)

Wie sieht der Rekursionsbaum **Baum**(y, A) aus, wenn y nicht in A vorkommt?

1. Der Rekursionsbaum Baum(A, y) ist ein Weg: Ein Knoten von B ist entweder ein Blatt oder verursacht `genau einen` rekursiven Aufruf.

Binärsuche (2/2)

Wie sieht der Rekursionsbaum **Baum**(y, A) aus, wenn y nicht in A vorkommt?

1. Der Rekursionsbaum Baum(A, y) ist ein Weg: Ein Knoten von B ist entweder ein Blatt oder verursacht `genau einen` rekursiven Aufruf.
2. Das Array A besitze genau $n = 2^k - 1$ Schlüssel.
   - Wenn Binärsuche ein Array der Länge $2^m - 1$ erhält, betrifft der nächste rekursive Aufruf ein Array der Länge

Binärsuche (2/2)

Wie sieht der Rekursionsbaum **Baum**(y, A) aus, wenn y nicht in A vorkommt?

1. Der Rekursionsbaum Baum(A, y) ist ein Weg: Ein Knoten von B ist entweder ein Blatt oder verursacht `genau einen` rekursiven Aufruf.
2. Das Array A besitze genau $n = 2^k - 1$ Schlüssel.
   - Wenn Binärsuche ein Array der Länge $2^m - 1$ erhält, betrifft der nächste rekursive Aufruf ein Array der Länge $2^{m-1} - 1$.
   - Der Rekursionsbaum Baum(y, A) ist also ein Weg der Länge

Binärsuche (2/2)

Wie sieht der Rekursionsbaum **Baum**(y, A) aus, wenn y nicht in A vorkommt?

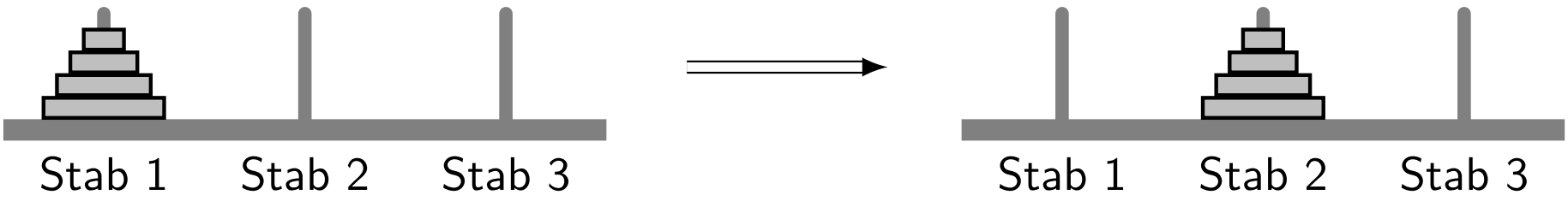
1. Der Rekursionsbaum Baum(A, y) ist ein Weg: Ein Knoten von B ist entweder ein Blatt oder verursacht `genau einen` rekursiven Aufruf.
2. Das Array A besitze genau $n = 2^k - 1$ Schlüssel.
   - Wenn Binärsuche ein Array der Länge $2^m - 1$ erhält, betrifft der nächste rekursive Aufruf ein Array der Länge $2^{m-1} - 1$.
   - Der Rekursionsbaum Baum(y, A) ist also ein Weg der Länge $k - 1$.

Binärsuche ist schnell, weil stets höchstens ein rekursiver Aufruf getätigt wird und weil die Länge des Abschnitts in dem gesucht wird, mindestens halbiert wird.

Türme von Hanoi

Die Türme von Hanoi (1/3)

- **Die Anfangskonfiguration**:
  - Wir haben drei Stäbe 1, 2 und 3.
  - Ursprünglich besitzt Stab 1 N Ringe, wobei die Ringe in absteigender Größe auf dem Stab aufgereiht sind:
    - ⋆ Der größte Ring von Stab 1 ist also der unterste Ring.
  - Die Stäbe 2 und 3 sind zu Anfang leer.
- **Die Züge**
  - Bewege einen zuoberst liegenden Ring von einem Stab zu einem anderen.
  - Der Zug ist nur dann erlaubt, wenn der Ring auf einen größeren Ring gelegt wird oder wenn der Stab leer ist.
- **Die Zielkonfiguration**:
  - Alle Ringe müssen nach Stab 2 bewegt werden.



Die Türme von Hanoi (2/3)

```
def Hanoi(N, stab1, stab2, stab3):
# Annahmen: (stab1,stab2,stab3) ist eine Permutation von {1, 2, 3}.
# Die obersten N Ringe auf Stab stab1 passen auf stab2 und stab3.
# Ziel: Bewege die obersten N Ringe von stab1 auf stab2.
```

Die Türme von Hanoi (2/3)

```
def Hanoi(N, stab1, stab2, stab3):
# Annahmen: (stab1,stab2,stab3) ist eine Permutation von {1,2,3}.
# Die obersten N Ringe auf Stab stab1 passen auf stab2 und stab3.
# Ziel: Bewege die obersten N Ringe von stab1 auf stab2.
  if (N==1):
      lege den obersten Ring von Stab stab1 auf Stab stab2
      # Induktionsanfang: Ziel erreicht!
```

Die Türme von Hanoi (2/3)

```
def Hanoi(N, stab1, stab2, stab3):
# Annahmen: (stab1,stab2,stab3) ist eine Permutation von {1,2,3}.
# Die obersten N Ringe auf Stab stab1 passen auf stab2 und stab3.
# Ziel: Bewege die obersten N Ringe von stab1 auf stab2.
  if (N==1):
      lege den obersten Ring von Stab stab1 auf Stab stab2
      # Induktionsanfang: Ziel erreicht!
   else:
      Hanoi(N-1, stab1, stab3, stab2)
      # Induktionsvoraussetzung ⟹ Die obersten N − 1 Ringe von Stab stab1
      # befinden sich jetzt auf Stab stab3.
```

Die Türme von Hanoi (2/3)

```
def Hanoi(N, stab1, stab2, stab3):
# Annahmen: (stab1,stab2,stab3) ist eine Permutation von {1,2,3}.
# Die obersten N Ringe auf Stab stab1 passen auf stab2 und stab3.
# Ziel: Bewege die obersten N Ringe von stab1 auf stab2.
  if (N==1):
      lege den obersten Ring von Stab stab1 auf Stab stab2
      # Induktionsanfang: Ziel erreicht!
   else:
      Hanoi(N-1, stab1, stab3, stab2)
      # Induktionsvoraussetzung ⟹ Die obersten N − 1 Ringe von Stab stab1
      # befinden sich jetzt auf Stab stab3.
      lege den obersten Ring von Stab stab1 auf Stab stab2
```

Die Türme von Hanoi (2/3)

```
def Hanoi(N, stab1, stab2, stab3):
# Annahmen: (stab1,stab2,stab3) ist eine Permutation von {1,2,3}.
# Die obersten N Ringe auf Stab stab1 passen auf stab2 und stab3.
# Ziel: Bewege die obersten N Ringe von stab1 auf stab2.
  if (N==1):
      lege den obersten Ring von Stab stab1 auf Stab stab2
      # Induktionsanfang: Ziel erreicht!
   else:
      Hanoi(N-1, stab1, stab3, stab2)
      # Induktionsvoraussetzung ⟹ Die obersten N − 1 Ringe von Stab stab1
      # befinden sich jetzt auf Stab stab3.
      lege den obersten Ring von Stab stab1 auf Stab stab2
      Hanoi(N-1, stab3, stab2, stab1)
      # Induktionsvoraussetzung ⟹ Die obersten N − 1 Ringe von Stab stab3
      # befinden sich jetzt auf Stab stab2 ⟹ Induktionsschritt: Ziel erreicht!
```

Türme von Hanoi (3/3)

Wie sieht der Rekursionsbaum **Baum**$(N, 1, 2, 3)$ für Hanoi(N,1,2,3) aus?

1. Es werden zwei rekursive Aufrufe mit dem Parameter $N - 1$ getätigt. **Baum**$(N, 1, 2, 3)$ ist also

Türme von Hanoi (3/3)

Wie sieht der Rekursionsbaum **Baum**$(N, 1, 2, 3)$ für Hanoi(N,1,2,3) aus?

1. Es werden zwei rekursive Aufrufe mit dem Parameter $N - 1$ getätigt. **Baum**$(N, 1, 2, 3)$ ist also ein vollständiger Binärbaum der Tiefe

Türme von Hanoi (3/3)

Wie sieht der Rekursionsbaum **Baum**$(N, 1, 2, 3)$ für Hanoi(N,1,2,3) aus?

1. Es werden zwei rekursive Aufrufe mit dem Parameter $N - 1$ getätigt. **Baum**$(N, 1, 2, 3)$ ist also ein vollständiger Binärbaum der Tiefe $N - 1$.
2. Das sind ganz schlechte Nachrichten: Ein solcher Baum besitzt

$$2^{\text{Tiefe}+1} - 1 = 2^N - 1$$

Knoten.

- Die Anzahl der Ringbewegungen stimmt überein mit der Anzahl der Knoten des Rekursionsbaums.
- Hanoi(N,1,2,3) führt $2^N - 1$ Ringbewegungen aus.

Türme von Hanoi (3/3)

Wie sieht der Rekursionsbaum **Baum**$(N, 1, 2, 3)$ für Hanoi(N,1,2,3) aus?

1. Es werden zwei rekursive Aufrufe mit dem Parameter $N - 1$ getätigt. **Baum**$(N, 1, 2, 3)$ ist also ein vollständiger Binärbaum der Tiefe $N - 1$.
2. Das sind ganz schlechte Nachrichten: Ein solcher Baum besitzt

$$2^{\text{Tiefe}+1} - 1 = 2^N - 1$$

Knoten.

- Die Anzahl der Ringbewegungen stimmt überein mit der Anzahl der Knoten des Rekursionsbaums.
- Hanoi(N,1,2,3) führt $2^N - 1$ Ringbewegungen aus.

Die Anzahl der Ringbewegungen ist so groß, weil für $N > 1$ stets zwei rekursive Aufrufe getätigt werden, wobei der Parameter N nur um Eins reduziert wird.

Entscheidungsbäume: Spielbäume

Spielbäume

In einem zwei-Personen Spiel wie **Schach, Dame, Mühle, Go, Othello** ... spielen Alice und Bob gegeneinander. Können wir in einer vorgegebenen Spielsituation einen **gewinnenden Zug** für den ziehenden Spieler bestimmen?

Wir bauen einen **Spielbaum**.

Spielbäume

In einem zwei-Personen Spiel wie **Schach, Dame, Mühle, Go, Othello** ... spielen Alice und Bob gegeneinander. Können wir in einer vorgegebenen Spielsituation einen **gewinnenden Zug** für den ziehenden Spieler bestimmen?

Wir bauen einen **Spielbaum**.

(a) Nach der Zugfolge $(z_1, \ldots, z_m)$ sei Spieler $X \in \{$ **Alice**, **Bob** $\}$ am Zug.

- Der Knoten $v_{z_1,\ldots,z_m}$ modelliert die Zugfolge.
- $v_{z_1,\ldots,z_m}$ wird mit dem ziehenden Spieler X beschriftet.

Von $v_{z_1,\ldots,z_m}$ aus wird für jeden möglichen Zug z des Spielers X eine mit z beschriftete Kante zum Kind $v_{z_1,\ldots,z_m,z}$ eingefügt.

Spielbäume

In einem zwei-Personen Spiel wie **Schach, Dame, Mühle, Go, Othello** ... spielen Alice und Bob gegeneinander. Können wir in einer vorgegebenen Spielsituation einen **gewinnenden Zug** für den ziehenden Spieler bestimmen?

Wir bauen einen **Spielbaum**.

(a) Nach der Zugfolge $(z_1, \ldots, z_m)$ sei Spieler $X \in \{$ **Alice**, **Bob** $\}$ am Zug.

- Der Knoten $v_{z_1,\ldots,z_m}$ modelliert die Zugfolge.
- $v_{z_1,\ldots,z_m}$ wird mit dem ziehenden Spieler X beschriftet.

Von $v_{z_1,\ldots,z_m}$ aus wird für jeden möglichen Zug z des Spielers X eine mit z beschriftete Kante zum Kind $v_{z_1,\ldots,z_m,z}$ eingefügt.

(b) Ein Blatt wird mit 1 oder -1 beschriftet, falls **Alice** das Spiel gewinnt oder verliert. (Wir nehmen an, dass es kein Unentschieden gibt.)

- In partiellen Spielbäumen wird die Zugfolge eines Blattes mit einer reellen Zahl bewertet.

Der Minimax-Algorithmus

Das Ziel: Bestimme welcher Spieler eine Gewinnstrategie hat.

Der Minimax-Algorithmus

Das Ziel: Bestimme welcher Spieler eine Gewinnstrategie hat.

1. Der **Minimax-Algorithmus** beginnt mit den Blättern und arbeitet sich dann „aufwärts“ zur Wurzel vor. Für einen inneren Knoten u definiere den Spielwert

$$\text{wert}(u) = \begin{cases} 1 & \textbf{Alice} \text{ hat eine Gewinnstrategie,} \\ -1 & \textbf{Bob} \text{ hat eine Gewinnstrategie.} \end{cases}$$

Der Minimax-Algorithmus

Das Ziel: Bestimme welcher Spieler eine Gewinnstrategie hat.

1. Der **Minimax-Algorithmus** beginnt mit den Blättern und arbeitet sich dann „aufwärts“ zur Wurzel vor. Für einen inneren Knoten u definiere den Spielwert

$$\text{wert}(u) = \begin{cases} 1 & \textbf{Alice} \text{ hat eine Gewinnstrategie,} \\ -1 & \textbf{Bob} \text{ hat eine Gewinnstrategie.} \end{cases}$$

2. **Alice** sei in v am Zug und für jedes Kind w von v sei „wert(w)“ bekannt. Setze

$$\text{wert}(v) =$$

Der Minimax-Algorithmus

Das Ziel: Bestimme welcher Spieler eine Gewinnstrategie hat.

1. Der **Minimax-Algorithmus** beginnt mit den Blättern und arbeitet sich dann „aufwärts“ zur Wurzel vor. Für einen inneren Knoten u definiere den Spielwert

$$\text{wert}(u) = \begin{cases} 1 & \textbf{Alice} \text{ hat eine Gewinnstrategie,} \\ -1 & \textbf{Bob} \text{ hat eine Gewinnstrategie.} \end{cases}$$

2. **Alice** sei in v am Zug und für jedes Kind w von v sei „wert(w)“ bekannt. Setze

$$\text{wert}(v) = \textbf{max}_{w \text{ ist ein Kind von } v} \text{wert}(w),$$

   denn eine intelligent spielende **Alice** wird den besten möglichen Zug auswählen.
3. Ist hingegen **Bob** in v am Zug, setze

$$\text{wert}(v) =$$

Der Minimax-Algorithmus

Das Ziel: Bestimme welcher Spieler eine Gewinnstrategie hat.

1. Der **Minimax-Algorithmus** beginnt mit den Blättern und arbeitet sich dann „aufwärts“ zur Wurzel vor. Für einen inneren Knoten u definiere den Spielwert

$$\text{wert}(u) = \begin{cases} 1 & \textbf{Alice} \text{ hat eine Gewinnstrategie,} \\ -1 & \textbf{Bob} \text{ hat eine Gewinnstrategie.} \end{cases}$$

2. **Alice** sei in v am Zug und für jedes Kind w von v sei „wert(w)“ bekannt. Setze

$$\text{wert}(v) = \textbf{max}_{w \text{ ist ein Kind von } v} \text{wert}(w),$$

denn eine intelligent spielende **Alice** wird den besten möglichen Zug auswählen.

3. Ist hingegen **Bob** in v am Zug, setze

$$\text{wert}(v) = \textbf{min}_{w \text{ ist ein Kind von } v} \text{wert}(w) :$$

ein intelligent spielender **Bob** wählt seinerseits den bestmöglichen Zug aus.

Mensch gegen Maschine (1/2)

Schach:

- Für nicht-triviale Spiele ist der Spielbaum einfach viel zu groß:
  Siehe https://en.wikipedia.org/wiki/Game_complexity
  - Man schätzt die Anzahl erreichbarer Stellungen auf ca. 10^{47}.
  - Schätzungen für die kleinste Anzahl von Stellungen, die der Minimax-Algorithmus betrachten muss, liegen bei ca. 10^{123}.
- \+ Stattdessen spielen clevere Strategien nur „gute Züge“ durch.
  - Heuristiken werden eingesetzt, um zwischenzeitlich erhaltene Spielsituationen zu bewerten und „gute Züge“ zu bestimmen.
  - Für Schach: Eröffnungsbibiotheken, Endspiel-Datenbanken und Datenbanken vollständig gespielter Partien werden benutzt.
- \+ Neuronale Programme lernen „gute von schlechten Zügen zu unterscheiden“.
  - Google's AlphaZero ist das gegenwärtig stärkste Schachprogramm.
    Siehe: https://de.wikipedia.org/wiki/AlphaZero

Mensch gegen Maschine (1/2)

Schach:

- Für nicht-triviale Spiele ist der Spielbaum einfach viel zu groß:
  Siehe https://en.wikipedia.org/wiki/Game_complexity
  - Man schätzt die Anzahl erreichbarer Stellungen auf ca. 10^{47}.
  - Schätzungen für die kleinste Anzahl von Stellungen, die der Minimax-Algorithmus betrachten muss, liegen bei ca. 10^{123}.
- \+ Stattdessen spielen clevere Strategien nur „gute Züge“ durch.
  - Heuristiken werden eingesetzt, um zwischenzeitlich erhaltene Spielsituationen zu bewerten und „gute Züge“ zu bestimmen.
  - Für Schach: Eröffnungsbibiotheken, Endspiel-Datenbanken und Datenbanken vollständig gespielter Partien werden benutzt.
- \+ Neuronale Programme lernen „gute von schlechten Zügen zu unterscheiden“.
  - Google's AlphaZero ist das gegenwärtig stärkste Schachprogramm.
    Siehe: https://de.wikipedia.org/wiki/AlphaZero

Mensch gegen Maschine (2/2)

(a) Schach: Die Spielstärke eines Schachspielers wird mit der Elo-Zahl gemessen.
- Die besten 10 Schachprogramme erzielen Elo-Zahlen zwischen 3000 und 3450. Die Spielstärke von AlphaZero wird auf ca. 3700 geschätzt.
- Der Schachweltmeister „schafft“ eine Elo-Zahl von ca. 2850. :-((

(b) Othello:
- Das Othello-Programm Logistello hat in 1997 alle sechs Spiele gegen den damaligen Weltmeister gewonnen.
- Die besten Othello-Programme sind selbst Spitzenspielern weit überlegen :-((

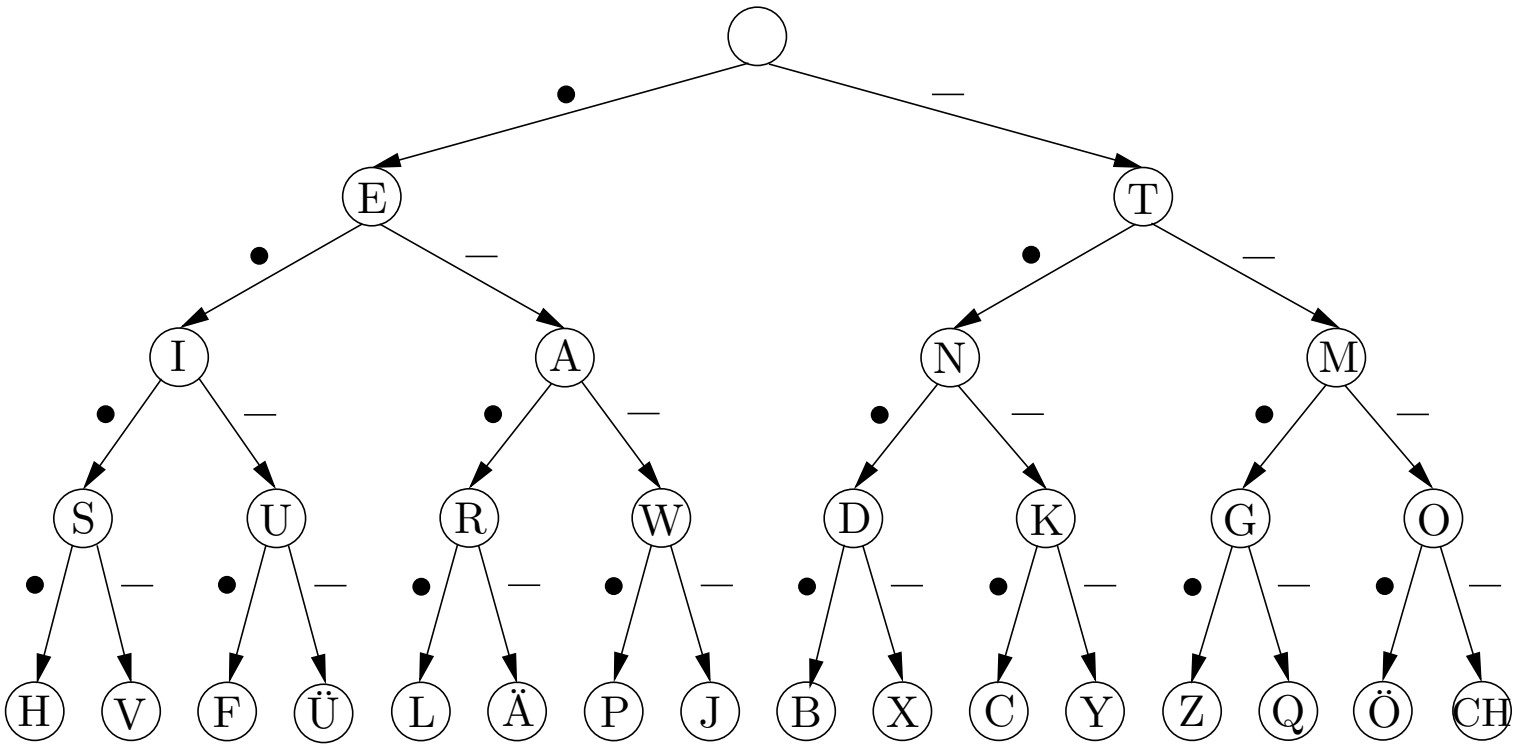
(c) Go: In 2016 wurde der Go-Weltmeister mit 4:1 geschlagen.

Entscheidungsbäume:
Bäume für die Darstellung von Entscheidungsfolgen

Entscheidungsbäume und der Morse-Code

Folgen von Entscheidungen können häufig durch gewurzelte, markierte Bäume modelliert werden. Diese Bäume nennt man auch **Entscheidungsbäume**.

Der Entscheidungsbaum für den Morse-Code: Im Morse-Code wird jeder Buchstabe durch eine Folge von kurzen und langen Signalen repräsentiert. Ein „kurzes Signal“ wird im folgenden Baum als Kantenmarkierung “•” dargestellt, ein „langes Signal“ als “—” .



Die Struktur von Entscheidungsbäumen

Entscheidungsbäume zählen systematisch alle endgültigen Entscheidungen auf.

(a) Die inneren Knoten modellieren einen **Zwischenstand** (oder eine partielle Entscheidung) bei der Entscheidungsfindung.
   - Im Beispiel des Morse-Code werden Knoten mit dem gerade kodierten Buchstaben beschriftet.
   - Blätter entsprechen vollständigen Entscheidungen.

Die Struktur von Entscheidungsbäumen

Entscheidungsbäume zählen systematisch alle endgültigen Entscheidungen auf.

(a) Die inneren Knoten modellieren einen **Zwischenstand** (oder eine partielle Entscheidung) bei der Entscheidungsfindung.
- Im Beispiel des Morse-Code werden Knoten mit dem gerade kodierten Buchstaben beschriftet.
- Blätter entsprechen vollständigen Entscheidungen.

(b) Die von einem Knoten des Entscheidungsbaums ausgehenden Kanten modellieren die "momentan" möglichen, einander ausschließenden **Alternativen**.
- Für jeden Knoten des Entscheidungsbaums muss **entschieden** werden, welche Alternative ausgewählt werden soll.
- Der Weg von der Wurzel zu einem Blatt ist mit allen gefällten Entscheidungen markiert.

Entscheidungsbäume: Anwendungsbeispiele

(a) In **Optimierungsproblemen der kombinatorischen Optimierung** gibt es viele Lösungen unterschiedlicher Qualität: Gesucht ist eine optimale Lösung.

- Alle „in Frage kommenden“ Lösungen sind **systematisch aufzuzählen**:
  - ⋆ Jede mögliche Lösung L muss irgendeiner endgültigen Entscheidung –und damit einem Blatt b_L im Baum– entsprechen.
  - ⋆ Das Blatt b_L wird mit der Qualität der Lösung L beschriftet.
- Wir schauen uns gleich das Problem des Handlungsreisenden an.

Entscheidungsbäume: Anwendungsbeispiele

(a) In **Optimierungsproblemen der kombinatorischen Optimierung** gibt es viele Lösungen unterschiedlicher Qualität: Gesucht ist eine optimale Lösung.

- Alle „in Frage kommenden“ Lösungen sind **systematisch aufzuzählen**:
  - ⋆ Jede mögliche Lösung L muss irgendeiner endgültigen Entscheidung –und damit einem Blatt b_L im Baum– entsprechen.
  - ⋆ Das Blatt b_L wird mit der Qualität der Lösung L beschriftet.
- Wir schauen uns gleich das Problem des Handlungsreisenden an.

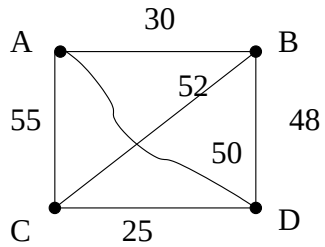
(b) Die in der technischen Informatik verwandten

binären Entscheidungsgraphen

(engl: binary decision diagrams) entstehen, wenn verschiedene Knoten eines Entscheidungsbaums zu einem Knoten zusammengefasst werden.

Das Problem des Handlungsreisenden (1/2)

Der folgende Graph gibt Entfernungen (in km) zwischen den Städten A,B,C und D an:

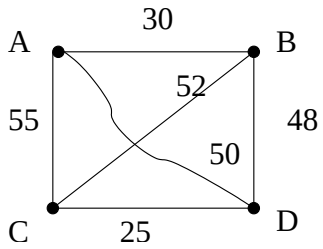


Ein Handlungsreisender soll einen **möglichst kurzen** Rundweg finden, auf dem er alle Städte A, B, C, D besucht.

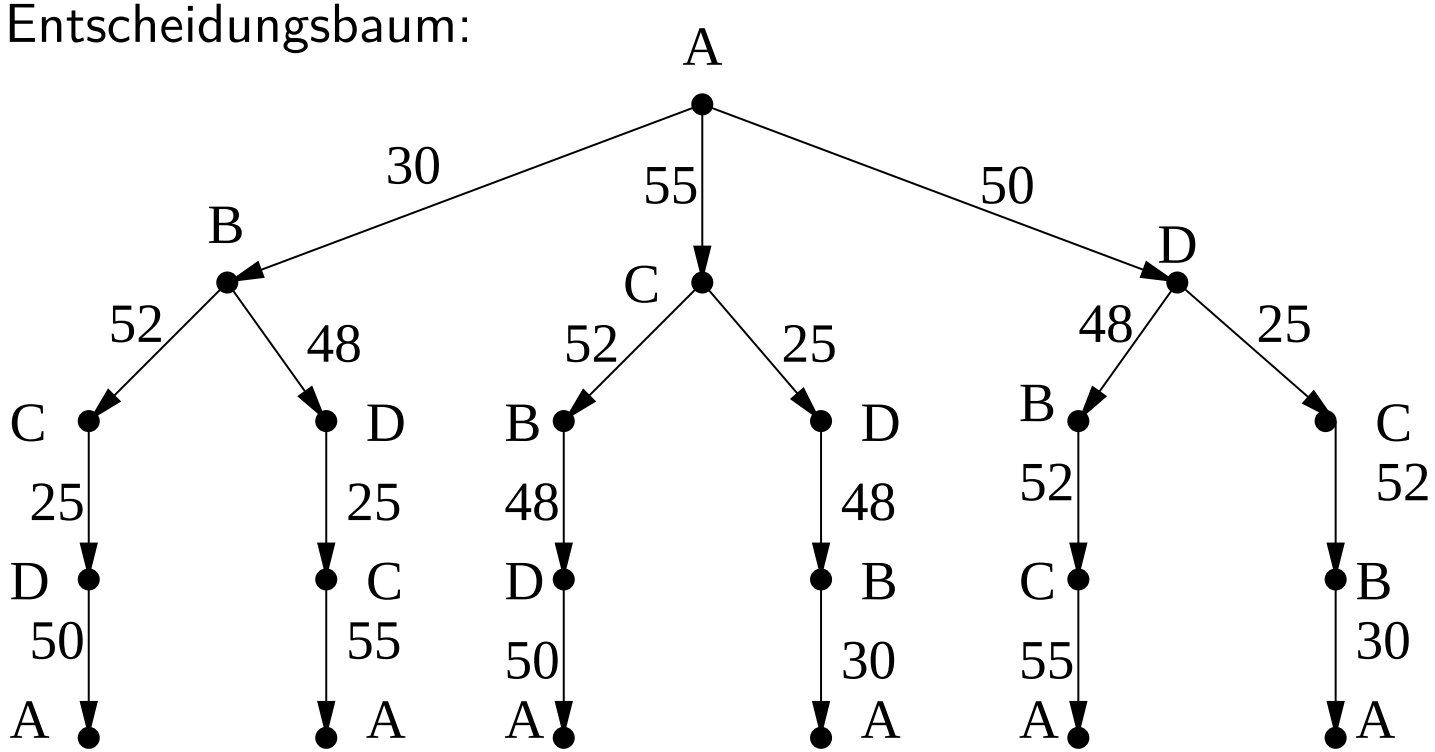
Baue einen Entscheidungsbaum, der systematisch alle Rundwege aufzählt.

Das Problem des Handlungsreisenden (2/2)

Zur Erinnerung:



Und hier ist der Entscheidungsbaum:



Und wenn es sehr viel mehr Städte gibt?

(a) Es gibt $(n-1)!$ viele verschiedene Rundwege, die alle in einer bestimmten Stadt „beginnen“.

(b) Für $n = 4$ haben wir gerade einmal $3! = 6$ Rundwege und unser Entscheidungsbaum ist recht klein.

Aber was können wir noch für $n = 50$ ausrichten?

Und wenn es sehr viel mehr Städte gibt?

(a) Es gibt $(n-1)!$ viele verschiedene Rundwege, die alle in einer bestimmten Stadt „beginnen“.

(b) Für $n = 4$ haben wir gerade einmal $3! = 6$ Rundwege und unser Entscheidungsbaum ist recht klein.

Aber was können wir noch für $n = 50$ ausrichten?

Branch & Bound-Verfahren schneiden möglichst große Teilbäume aus dem Baum heraus, sobald es klar ist, dass dort keine optimale Lösungen „sitzen“.

Branch & Bound Verfahren werden in der Bachelor-Veranstaltung „**Approximationsalgorithmen**“ untersucht.