

5. Graphentheorie

Graphen

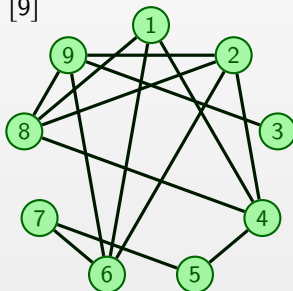
Definition (Graph)

Ein **Graph** ist ein geordnetes Paar $G = (V, E)$, wobei

- die **Ecken-/Knotenmenge** V eine endliche Menge ist
- und die **Kantenmenge** E eine Teilmenge der 2-elementigen Teilmengen von V ist, d.h., $E \subseteq V^{(2)} := \{M \in \wp(V) : |M| = 2\}$.

Wir schreiben $V(G)$ und $E(G)$ für die Ecken- und Kantenmenge eines Graphen G .

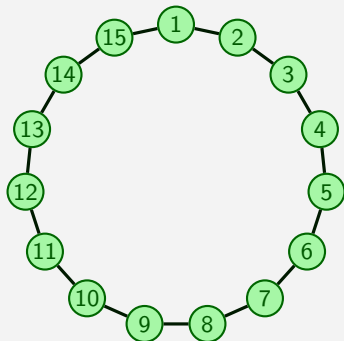
Beispiel: $V = \{1, \dots, 9\} = [9]$



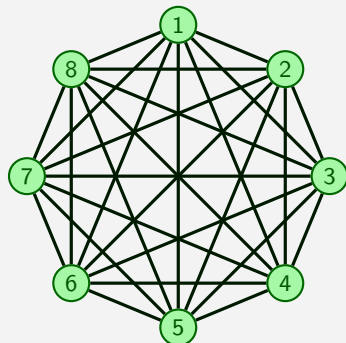
Graph $G = (V, E)$ entspricht symmetrischer, irreflexiver Relation E auf V

Kreise, Cliques und Wege

C_{15} – Kreis der Länge 15



K_8 – Clique/vollständiger Graph



P_{11} – Weg der Länge 11



Länge = Anzahl Kanten

Teilgraphen und induzierte Teilgraphen

Definition (Teilgraphen)

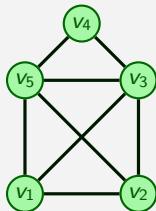
Sei $G = (V, E)$ ein Graph. Ein Graph H ist ein **Teilgraph**, falls $V(H) \subseteq V$ und $E(H) \subseteq E$ und wir schreiben dann $H \subseteq G$.

Ein Teilgraph $H = (U, E_H)$ von G heißt **induziert**, falls zusätzlich gilt

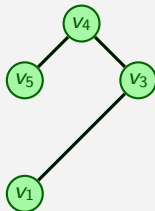
$$E_H = \{e \in E : e \subseteq U\},$$

d. h. H enthält **alle** Kanten von G die durch die in $U \subseteq V$ enthalten sind. Wir schreiben dann auch $H = G[U]$.

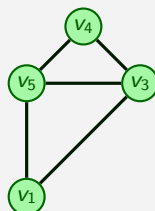
Graph G



Teilgraph von G



$G[\{v_1, v_3, v_4, v_5\}]$



Kopien in Graphen

Definition (Graphenisomorphismus)

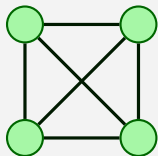
Eine **bijektive** Abbildung $\varphi: V(G) \rightarrow V(H)$ ist ein **Graphenisomorphismus** zwischen Graphen G und H , falls gilt

$$\{x, y\} \in E(G) \iff \{\varphi(x), \varphi(y)\} \in E(H).$$

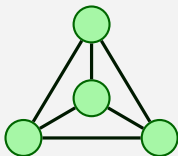
Dann sind G und H **isomorph** und wir schreiben $G \simeq H$ (oder einfach $G = H$).

Wir sagen G **enthält eine Kopie von H** und schreiben einfach $H \subseteq G$, falls H isomorph zu einem Teilgraphen von G ist.

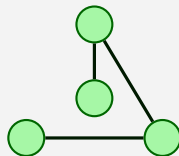
Clique K_4




Isomorpher K_4



Kopie von P_3 in K_4



Zusammenhängende Graphen

- Weg P  hat **Endecken** u und v und solch einen Weg P nennen wir **u - v -Weg**
- Wege der Länge $\ell \geq 1$ haben immer zwei unterschiedliche Endecken
- ein u - v -Weg der Länge $\ell \geq 2$ zusammen mit der Kante $\{u, v\}$ bildet einen Kreis der Länge $\ell + 1$

Definition (Zusammenhang)

Ein Graph $G = (V, E)$ ist **zusammenhängend**, falls es für je zwei Ecken $u, v \in V$ einen u - v -Weg in G gibt.

Komponenten eines Graphen sind maximale zusammenhängende Teilgraphen.

Bemerkungen:

- die Relation $u \sim v$ definiert durch „es existiert ein u - v -Weg in G “, ist eine Äquivalenzrelation auf $V(G)$ (**Warum?**)
- die Äquivalenzklassen induzieren genau die Komponenten von G

Eckengrade

Definition (Grad)

Für eine Ecke v eines Graphen $G = (V, E)$ ist die **Nachbarschaft** die Menge der Ecken $u \in V$, die mit v eine Kante in G bilden, d. h.

$$N(v) := \{u \in V : \{u, v\} \in E\}.$$

Der **Grad** von v bezeichnet die Anzahl der Nachbarn von v

$$d(v) := |N(v)|.$$

Der **Minimal-** und **Maximalgrad** von G sind die Extremwerte über alle Eckengrade

$$\delta(G) := \min_{v \in V} d(v) \quad \text{und} \quad \Delta(G) := \max_{v \in V} d(v).$$

- es gilt immer $\delta(G) \leq d(v) \leq \Delta(G)$ für alle Ecken $v \in V(G)$
- G heißt **k -regulär**, falls $d(v) = k$ für alle Ecken $v \in V(G)$
- Clique K_n ist $(n - 1)$ -regulär, Kreis C_ℓ ist 2-regulär
- für Wege P_ℓ mit $\ell \geq 1$ gilt $\delta(P_\ell) = 1$ und $\Delta(P_\ell) = 2$

Handshaking Lemma

Satz

Für jeden Graphen $G = (V, E)$ gilt

$$\sum_{v \in V} d(v) = 2|E|.$$

Beweis: In der Summe wird jede Kante $\{x, y\}$ genau **zweimal** gezählt – nämlich einmal im Summanden $d(x)$ und einmal im Summanden $d(y)$. \square

Korollar

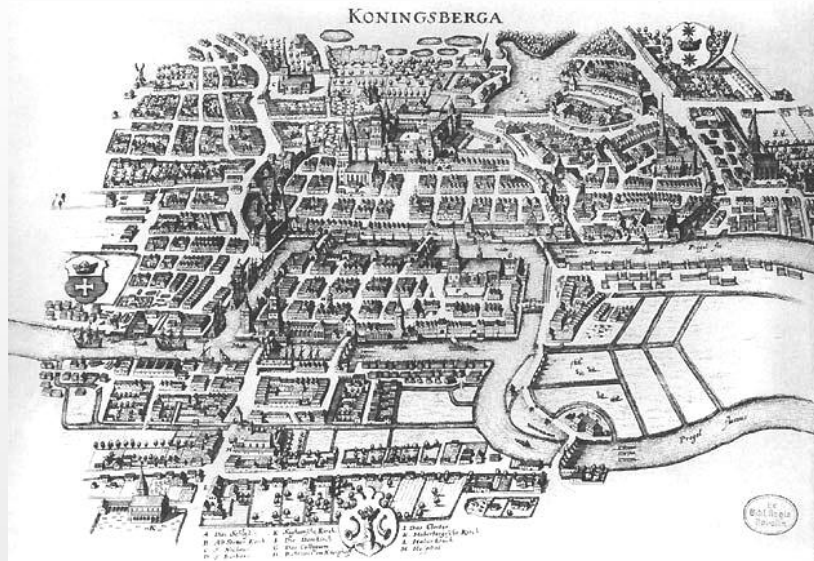
In jedem Graphen ist die Anzahl der Ecken mit ungeradem Grad gerade.

Beweis: (Paritätsargument)

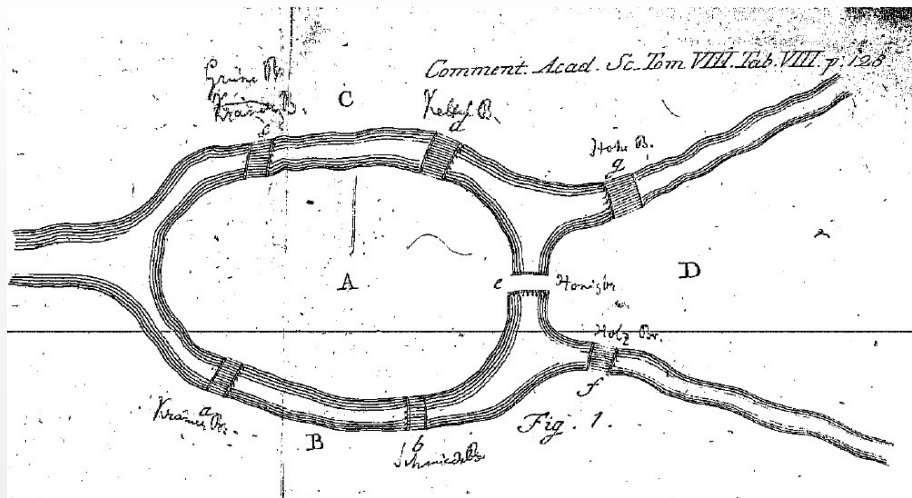
$$2|E| = \sum_{v \in V} d(v) = \sum_{\substack{v \in V \\ d(v) \text{ gerade}}} d(v) + \sum_{\substack{v \in V \\ d(v) \text{ ungerade}}} d(v)$$

$2|E|$ und \sum sind gerade $\implies \sum$ ist gerade \implies Anzahl der Summanden in \sum ist gerade, da jeder Summand ungerade ist \square

Königsberger Brückenproblem – anno 1736

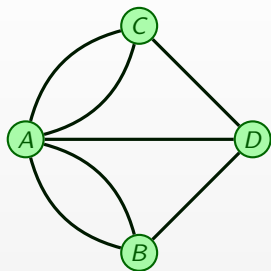


Königsberger Brückenproblem – für EULER



Gibt es einen Spaziergang, der alle sieben Brücken genau einmal durchläuft und am Ausgangspunkt endet?

Königsberger Brückenproblem – heute



Definition (Kantenzug)

Ein **Kantenzug** W in $G = (V, E)$ ist eine abwechselnde Folge von Ecken und Kanten

$$W = (v_0, e_1, v_1, e_2, v_2, \dots, v_{m-1}, e_m, v_m)$$

mit $e_i = \{v_{i-1}, v_i\}$ für $i \in [m]$. Hierbei dürfen sich Ecken und Kanten **wiederholen**.

Die **Länge des Kantenzuges** W ist m und W heißt **geschlossen**, falls $v_m = v_0$.

Frage/Definition

Welche (Multi)Graphen enthalten einen geschlossenen Kantenzug, der alle Kanten genau einmal durchläuft?

Solche Graphen heißen **EULERSch** und solch ein Kantenzug ist ein **EULERScher Kantenzug/geschlossener EULERzug/EULERkreis** bzw. eine **EULERSche Linie**.

Ein Kantenzug der alle Kanten genau einmal durchläuft, aber **nicht notwendigerweise** geschlossen ist, heißt **EULERzug** bzw. **EULERweg**.

Satz von EULER

Satz (EULER 1736)

Ein zusammenhängender Graph ist genau dann EULERSch, wenn alle Eckengrade gerade sind.

Bemerkung: Satz gilt auch für Multigraphen, wenn Grad v die Anzahl der Kanten ist, die v enthalten.

Beweis

„ \implies “ (EULERSch \implies alle Grade gerade)

klar (warum?)



„ \impliedby “ (alle Grade gerade \implies EULERSch)

- sei $W = (v_0, e_1, v_1, e_2, v_2, \dots, v_{m-1}, e_m, v_m)$ ein längster Kantenzug **ohne** Kantenwiederholungen in $G = (V, E)$ und sei $E_W = \{e_1, \dots, e_m\}$
- Maximalität von $W \implies$ alle Kanten die v_0 oder v_m enthalten liegen in E_W , da sonst W an einem Ende verlängert werden könnte
- $\implies v_0 = v_m$, d. h. W ist geschlossen, da sonst v_0 und v_m ungeraden Grad ≥ 1 im „Restgraphen“ $H = (V, E \setminus E_W)$ hätten
- wäre $E \setminus E_W \neq \emptyset$, dann gäbe es wegen dem Zusammenhang von G eine Kante $e = \{x, v_i\} \in E \setminus E_W$ und $W' := (x, e, v_i, e_{i+1}, v_{i+1}, \dots, v_m = v_0, e_1, v_1, \dots, v_{i-1}, e_i, v_i)$ wäre ein Kantenzug der Kanten nur einmal enthält und der zusätzlich zu den Kanten aus E_W auch noch die Kante $e = \{x, v_i\}$ enthält
- \implies Widerspruch zur Wahl von W als **längster** solcher Kantenzug
- \implies also ist $E_W = E$ und somit ein geschlossener EULERScherzug



Kreise durch jede Ecke

- Satz von EULER charakterisiert Graphen $G = (V, E)$ mit geschlossenem Kantenzug, der jede Kante genau einmal enthält
- Charakterisierung (alle Grade gerade und zshg.) ist einfach (Linearzeit in $|V| + |E|$) algorithmisch nachprüfbar
→ Entscheidungsproblem „ G EULERSch?“ ist in **P**
- für jeden gegebenen zusammenhängenden Graphen G kann man effizient folgendes finden/berechnen:
 - entweder einen EULERSchen Kantenzug
→ Zertifikat für „ G ist EULERSch“
 - oder eine Ecke mit ungeradem Grad
→ Zertifikat für „ G ist nicht EULERSch“

Frage/Definition

Was passiert, wenn man „Kante“ durch „Ecke“ ersetzt?

Ein HAMILTONkreis in einem Graphen $G = (V, E)$ ist ein Kreis, der jede Ecke (genau einmal) enthält. Ein solcher Graph G heißt HAMILTONisch.

KARPS 21 NP-vollständige Probleme von 1972

Richard M. Karp

University of California at Berkeley

Abstract: A large class of computational problems involve the determination of properties of graphs, digraphs, integers, arrays of integers, finite families of finite sets, boolean formulas and elements of other countable domains. Through simple encodings from such domains into the set of words over a finite alphabet these problems can be converted into language recognition problems, and we can inquire into their computational complexity. It is reasonable to consider such a problem satisfactorily solved when an algorithm for its solution is found which terminates within a number of steps bounded by a polynomial in the length of the input. We show that a large number of classic unsolved problems of covering, matching, packing, routing, assignment and sequencing are equivalent, in the sense that either each of them possesses a polynomial-bounded algorithm or none of them does.

KARPs 21 NP-vollständige Probleme von 1972

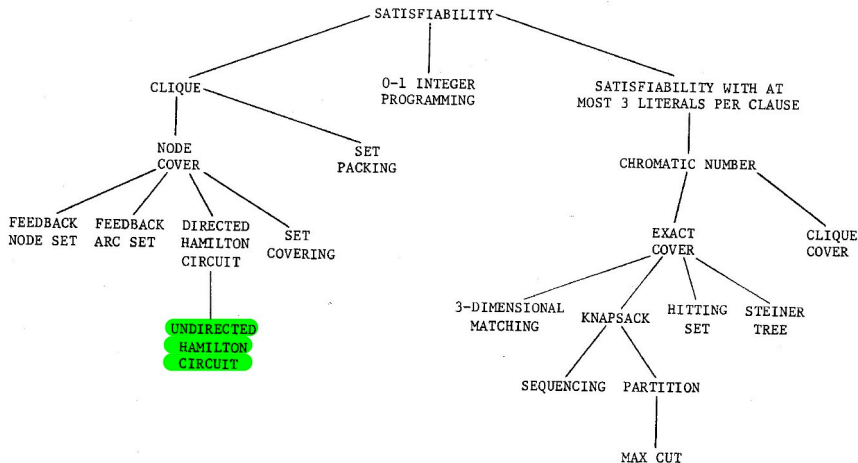


FIGURE 1 - Complete Problems



The Millennium Prize Problems

In order to celebrate mathematics in the new millennium, The Clay Mathematics Institute of Cambridge, Massachusetts (CMI) established seven *Prize Problems*. The Prizes were conceived to record some of the most difficult problems with which mathematicians were grappling at the turn of the second millennium; to elevate in the consciousness of the general public the fact that in mathematics, the frontier is still open and abounds in important unsolved problems; to emphasize the importance of working towards a solution of the deepest, most difficult problems; and to recognize achievement in mathematics of historical magnitude.

The prizes were announced at a meeting in Paris, held on May 24, 2000 at the Collège de France. Three lectures were presented: Timothy Gowers spoke on *The Importance of Mathematics*; Michael Atiyah and John Tate spoke on the problems themselves.

The seven Millennium Prize Problems were chosen by the founding Scientific Advisory Board of CMI, which conferred with leading experts worldwide. The focus of the board was on important classic questions that have resisted solution for many years.

Following the decision of the Scientific Advisory Board, the Board of Directors of CMI designated a \$7 million prize fund for the solutions to these problems, with \$1 million allocated to the solution of each problem.

Bounty challenge des Clay Mathematics Institute

[ABOUT](#)[PROGRAMS](#)[MILLENNIUM PROBLEMS](#)[PEOPLE](#)[PUBLICATIONS](#)[EVENTS](#)[EUCLID](#)

Millennium Problems

Yang–Mills and Mass Gap

Experiment and computer simulations suggest the existence of a "mass gap" in the solution to the quantum versions of the Yang–Mills equations. But no proof of this property is known.

Riemann Hypothesis

The prime number theorem determines the average distribution of the primes. The Riemann hypothesis tells us about the deviation from the average. Formulated in Riemann's 1859 paper, it asserts that all the 'non-obvious' zeros of the zeta function are complex numbers with real part $1/2$.

P vs NP Problem

If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

Für effizient (Polynomialzeit in $|V|$) prüfbare Charakterisierung HAMILTONischer Graphen oder den Beweis, dass keine solche existiert, gibt es eine Million Dollar.

Einfache notwendige und hinreichende Kriterien

Idee:

- da eine effiziente vollständige Charakterisierung ein schwieriges Problem und eventuell unlösbares Problem ist, untersucht man:
 - **notwendige Kriterien**, die jeder HAMILTONische Graph erfüllen muss, aber deren Erfüllung nicht in jedem Fall einen HAMILTONkreis erzwingt
Beispiele: $|V(G)| \geq 3$ oder G zshg. oder $\delta(G) \geq 2$
 - **hinreichende Kriterien**, deren Erfüllung einen HAMILTONkreis erzwingt, die aber nicht jeder HAMILTONische Graph erfüllt
Beispiel: $\delta(G) \geq |V(G)| - 1$ (und $|V(G)| \geq 3$)

Satz (G. A. DIRAC 1952)

Jeder Graph $G = (V, E)$ mit $n = |V| \geq 3$ und $\delta(G) \geq n/2$, ist HAMILTONisch.

Bemerkungen:

- DIRACs Kriterium (Gradbedingung) ist effizient überprüfbar
- die Gradbedingung ist **nicht** notwendig (Bsp.: C_ℓ mit $\ell \geq 5$)
- die Gradbedingung ist hinreichend (Aussage des Satzes)

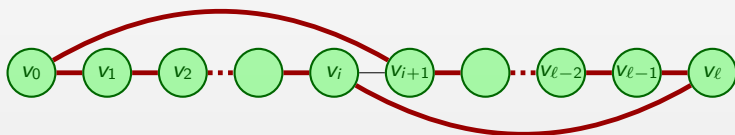
Beweis: $\delta(G) \geq n/2 \implies$ HAMILTONKREIS

Sei $G = (V, E)$ ein Graph mit $n \geq 3$ Ecken.

- G ist zusammenhängend: $\delta(G) \geq n/2 \implies$ je zwei unbenachbarte Ecken x und y haben mindestens 2 gemeinsame Nachbarn ($|N(x) \cap N(y)| \geq 2$)

Sei $P = v_0 - v_1 - v_2 - \dots - v_{\ell-2} - v_{\ell-1} - v_\ell$ ein längster Weg in G .

- \implies alle Nachbarn von v_0 und v_ℓ liegen auf P (Maximalität von P)
- \implies mind. $d(v_0) \geq n/2$ der Ecken $v_0, \dots, v_{\ell-1}$ sind Vorgänger eines Nachbarn von v_0 **und** mindestens $d(v_\ell) \geq n/2$ dieser Ecken sind ein Nachbar von v_ℓ
- \implies nach dem Schubfachprinzip gibt es also unter den $\ell \leq n - 1$ Ecken $v_0, \dots, v_{\ell-1}$ ein v_i sodass



- $\implies v_0, \dots, v_\ell$ liegen auf einem **Kreis C**
- \implies da G zusammenhängend ist muss C wegen der Maximalität von P alle Ecken enthalten (sonst gäbe es einen Weg P' der eine Ecke außerhalb von C und alle Ecken von C enthalten würde) □

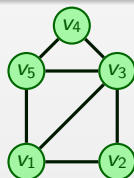
Datenstrukturen für Graphen

Adjazenzmatrix

Sei $G = (V, E)$ ein Graph mit $V = \{v_1, \dots, v_n\}$, dann ist die **Adjazenzmatrix** $A(G)$ ein quadratisches Zahlenschema mit n Zeilen und n Spalten, d. h. $A(G) = (a_{ij})_{i \in [n], j \in [n]}$, wobei a_{ij} der Eintrag in der i -ten Zeile und j -ten Spalte ist,

$$a_{ij} = \begin{cases} 1, & \text{falls } \{v_i, v_j\} \in E, \\ 0, & \text{sonst.} \end{cases}$$

$$A(G) = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

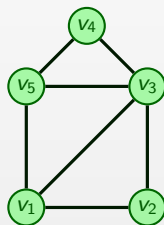
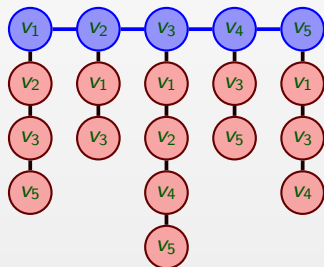


- **Vorteil:** Kantenabfrage durch einfachen und direkten Zugriff
- **Nachteil:** viel Speicherplatz, falls G nur wenige Kanten hat

Datenstrukturen für Graphen: Adjazenzlisten

Adjazenzlisten

Sei $G = (V, E)$ ein Graph, dann ist die **Adjazenzliste** L_v einer Ecke $v \in V$ eine Liste der Nachbarn von v . Die Listen L_v verwaltet man dann entweder über ein Array (undynamisch) oder über eine **Liste**, deren Elemente die Listen L_v sind.



- **Vorteil:** speichereffizient
- **Nachteil:** für die Kantenabfrage müssen die Listen traversiert werden

Bäume und Wälder

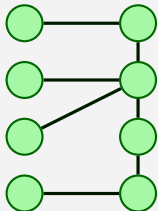
Definition

Ein kreisfreier Graph heißt **Wald** und ein zusammenhängender kreisfreier Graph heißt **Baum**.

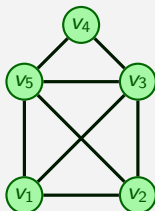
Ein **Spannbaum** eines zusammenhängenden Graphen G ist ein Teilgraph $T \subseteq G$ mit:

- T ist ein Baum
- und $V(T) = V(G)$.

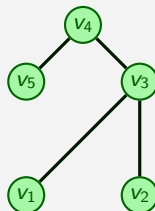
ein Baum



Graph G



Spannbaum von G



Satz

Für einen Graphen T mit $|V(T)| \geq 1$ sind die folgenden Aussagen äquivalent:

- 1 T ist ein Baum.
- 2 zwischen je zwei Ecken enthält T **genau** einen Weg.
- 3 T ist **minimal zusammenhängend**, d. h. T ist zusammenhängend, aber für jede Kante $e \in E(T)$ ist

$$T' = T - e := (V(T), E(T) \setminus \{e\})$$

nicht zusammenhängend.

- 4 T ist **maximal kreisfrei**, d. h. T ist kreisfrei, aber für jedes Paar $\{x, y\} \subseteq V(T)$ mit $\{x, y\} \notin E(T)$ ist

$$T'' = T + \{x, y\} := (V(T), E(T) \cup \{\{x, y\}\})$$

nicht kreisfrei.

- 5 T ist zusammenhängend und hat genau $|V(T)| - 1$ Kanten.
- 6 T ist kreisfrei und hat genau $|V(T)| - 1$ Kanten.

Baumcharakterisierungen - Beweis

- Äquivalenz von Aussagen **1** und **2** ist Hausaufgabe
- Für die restlichen Äquivalenzen zeigen wir die Implikationen

$$\mathbf{1} \Rightarrow \mathbf{3} \Rightarrow \mathbf{4} \Rightarrow \mathbf{1} \quad \text{und} \quad \mathbf{1} \Rightarrow \mathbf{5} \Rightarrow \mathbf{6} \Rightarrow \mathbf{1}.$$

„**1** \Rightarrow **3**“ (Baum \Rightarrow minimal zusammenhängend)

• Jeder Baum T ist nach Definition zusammenhängend.

• Angenommen ein Baum T ist nicht minimal zusammenhängend.

\Rightarrow es existiert eine Kante $\{x, y\}$, sodass $T - \{x, y\}$ zusammenhängend ist

$\Rightarrow T - \{x, y\}$ enthält x - y -Weg $P \Rightarrow T$ enthält Kreis aus P und $\{x, y\}$ ⚡ ✓

„**3** \Rightarrow **4**“ (minimal zusammenhängend \Rightarrow maximal kreisfrei)

• T minimal zusammenhängend $\Rightarrow T$ kreisfrei

• T zusammenhängend und $\{x, y\} \notin E(T) \Rightarrow$ es existiert x - y -Weg in T

$\Rightarrow T + \{x, y\}$ enthält Kreis $\Rightarrow T$ ist maximal kreisfrei ✓

„**4** \Rightarrow **1**“ (maximal kreisfrei \Rightarrow Baum)

• Sei T maximal kreisfrei. Wir zeigen, dass T zusammenhängend ist.

• Falls $\{x, y\} \notin E(T)$, dann schließt $\{x, y\}$ einen Kreis in $T + \{x, y\}$.

\Rightarrow es existiert x - y -Weg in T

\Rightarrow für je zwei Ecken $x, y \in V(T)$ existiert x - y -Weg

$\Rightarrow T$ ist zusammenhängend ✓

Vorbereitung für $1 \Rightarrow 5 \Rightarrow 6 \Rightarrow 1$

Lemma

Jeder (nichtleere) Baum T hat genau $|V(T)| - 1$ Kanten.

Beweis: (Induktion nach $n = |V(T)|$ mit allen Vorgängern)

- **Induktionsanfang $n = 1$:** jeder Graph mit einer Ecke hat 0 Kanten ✓
- **Induktionsschritt von $m < n$ nach n :**

Sei $n \geq 2$ und es gelte das Lemma für alle Bäume mit weniger als n Ecken.

$\Rightarrow T$ enthält mindestens eine Kante $\{x, y\}$

$\Rightarrow T - \{x, y\}$ besteht aus genau zwei Komponenten T_1 und T_2 mit

$$t_1 := |V(T_1)| \geq 1, \quad t_2 := |V(T_2)| \geq 1 \quad \text{und} \quad t_1 + t_2 = n.$$

\Rightarrow nach Induktionsannahme gilt $|E(T_1)| = t_1 - 1$ und $|E(T_2)| = t_2 - 1$

\Rightarrow da $E(T)$ genau aus den Kanten von T_1 , von T_2 und $\{x, y\}$ besteht, folgt

$$|E(T)| = |E(T_1)| + |E(T_2)| + 1 = (t_1 - 1) + (t_2 - 1) + 1 = t_1 + t_2 - 1 = n - 1.$$

$\Rightarrow T$ hat also genau $|V(T)| - 1$ Kanten □

Baumcharakterisierungen - Beweis von **1** \Rightarrow **5** \Rightarrow **6** \Rightarrow **1**

„**1** \Rightarrow **5**“ (Baum \Rightarrow zusammenhängend und genau $|V(T)| - 1$ Kanten)

- Jeder Baum T ist nach Definition zusammenhängend.
- Das Lemma besagt, dass jeder (nichtleere) Baum T genau $|V(T)| - 1$ Kanten hat. ✓

„**5** \Rightarrow **6**“ (zusammenhängend und genau $|V(T)| - 1$ Kanten \Rightarrow kreisfrei)

Angenommen T enthält einen Kreis, der die Kante $\{x, y\}$ enthält.

$\Rightarrow T - \{x, y\}$ enthält noch einen x - y -Weg P und für jeden a - b -Weg Q in T , der die Kante $\{x, y\}$ benutzt, kann P als „Umleitung“ genutzt werden.

\Rightarrow Es gibt auch einen a - b -Weg in $T - \{x, y\}$.

$\Rightarrow T - \{x, y\}$ ist zusammenhängend.

$\Rightarrow \exists$ minimal zusammenhängender Teilgraph $S \subseteq (T - \{x, y\})$ mit $V(S) = V(T)$

\Rightarrow wegen dem Lemma hat S genau $|V(S)| - 1 = |V(T)| - 1$ Kanten

\Rightarrow da T noch zusätzlich die Kante $\{x, y\}$ enthält, ergibt sich ein Widerspruch

$\Rightarrow T$ muss kreisfrei sein ✓

„**6** \Rightarrow **1**“ (kreisfrei und genau $|V(T)| - 1$ Kanten \Rightarrow Baum)

Angenommen T ist nicht zusammenhängend und besteht aus Komponenten C_1, \dots, C_k

für ein $k \geq 2$. Dann können wir in jeder Komponente C_i eine Ecke v_i beliebig wählen und die Kanten $\{v_1, v_j\}$ für $j = 2, \dots, k$ zu T hinzufügen. Der entstehende Graph T' ist zusammenhängend und weiterhin kreisfrei (warum?). D. h. T' ist ein Baum mit

$|E(T)| + k - 1 = |V(T)| - 1 + k - 1 \geq |V(T)|$ Kanten auf $V(T)$ Ecken. ⚡ (Lemma) ✓

Damit sind alle Äquivalenzen (bis auf die Hausaufgabe) bewiesen. □

Bäume haben Blätter

Korollar

Jeder Baum T mit $|V(T)| \geq 2$ enthält mindestens zwei Ecken mit Grad 1.

Definition

Ecken vom Grad höchstens 1 in Bäumen heißen **Blätter**.

Beweis (Korollar): Sei $T = (V, E)$ ein Baum mit $|V| \geq 2$. Wegen der Charakterisierung **5** gilt

$$|E| = |V| - 1 \quad (*)$$

und T ist zusammenhängend. Da $|V| \geq 2$ und T zusammenhängend ist, hat jede Ecke mindestens Grad 1. Angenommen, höchstens **eine Ecke hat Grad genau 1**, dann erhalten wir mit dem **Handshaking Lemma** den Widerspruch

$$2|V| - 2 = 2(|V| - 1) \stackrel{(*)}{=} 2|E| \stackrel{\text{H.L.}}{=} \sum_{v \in V} d(v) \geq 1 + 2(|V| - 1) = 2|V| - 1.$$

Dies ist ein Widerspruch, da $2|V| - 2 < 2|V| - 1$. □

Zusammenhängende Baumordnung

Korollar

Jeder Baum T mit $n \geq 1$ Ecken hat eine Aufzählung v_1, \dots, v_n seiner Ecken, sodass $T_i := T[\{v_1, \dots, v_i\}]$ für jedes $i \in [n]$ zusammenhängend ist.

Beweis: (Induktion nach n)

- Induktionsanfang $n = 1$: klar
- Induktionsschritt von n nach $n + 1$:



Sei $T = (V, E)$ ein Baum mit $n + 1 \geq 2$ Ecken.

$\Rightarrow T$ enthält ein Blatt v_{n+1} und sei u der Nachbar von v_{n+1}

\Rightarrow der induzierte Teilgraph

$$T' = T - v_{n+1} := T[V \setminus \{v_{n+1}\}],$$

der aus T durch Entfernen der Ecke v_{n+1} und der Kante $\{v_{n+1}, u\}$ entsteht, ist immer noch zusammenhängend (und kreisfrei).

$\Rightarrow T'$ ist ein Baum mit n Ecken

\Rightarrow wegen der Induktionsannahme gibt es eine Aufzählung v_1, \dots, v_n von $V \setminus \{v_{n+1}\}$, sodass $T'_i = T_i$ zusammenhängend ist für jedes $i \in [n]$

\Rightarrow da $T_{n+1} = T$ ebenfalls zusammenhängend ist, folgt das Korollar



Zusammenhängende Graphen haben Spannbäume

Korollar

Jeder zusammenhängende Graph $G = (V, E)$ enthält einen Spannbaum.

Beweis: Entferne Kanten aus $G = (V, E)$ solange, bis der resultierende (aufspannende) Teilgraph $T = (V, E_T)$ minimal zusammenhängend ist. Wegen der Charakterisierung **3**, ist T ein Baum und wegen $V(T) = V = V(G)$, ist T ein Spannbaum von G . □

Korollar

Jeder zusammenhängende Graph G mit $n \geq 1$ Ecken hat eine Aufzählung v_1, \dots, v_n seiner Ecken, sodass $G_i := G[\{v_1, \dots, v_i\}]$ für jedes $i \in [n]$ zusammenhängend ist.

Beweis: Wende gleiche Aussage für Bäume auf Spannbaum $T \subseteq G$ an. □

Bemerkungen:

- G zusammenhängend aber kein Baum \Rightarrow verschiedene Spannbäume
- in Anwendungen werden oft „spezielle“ Spannbäume genutzt

Wurzelbäume

Definition

Ein Baum $T = (V, E)$ mit einer ausgezeichneten Ecke $w \in V$ heißt **Wurzelbaum** und w ist dann die **Wurzel** von T .

Wurzelbäume definieren mit der Charakterisierung **2** eine Ordnung \leq_T auf V definiert durch

$$v \leq_T u \quad :\iff \quad v \text{ liegt auf dem eindeutig bestimmten } w\text{-}u\text{-Weg in } T.$$

Falls $v \leq_T u$, dann heißt v **Vorgänger** von u und u ist ein **Nachfolger** von v im Baum T (mit Wurzel w). Der größte Vorgänger v (unter \leq_T) von u mit $v \neq u$ heißt **direkter Vorgänger/Vater** von u und u ist ein **Kind** von v .

Beweis (\leq_T definiert Ordnung):

- **reflexiv**: Da jede Ecke v auf dem w - v -Weg in T liegt. ✓
- **antisymmetrisch**: Falls v auf dem w - u -Weg P liegt, dann ist der eindeutig bestimmte w - v -Weg Q ein Anfangsstück von P und wenn Q auch u enthält, dann müssen $P = Q$ und $u = v$ gelten. ✓
- **transitiv**: Falls x auf dem w - y -Weg P und y auf dem w - z -Weg Q in T liegt, dann folgt aus der Eindeutigkeit der Wege, dass P ein Anfangsstück von Q sein muss und somit liegt z auch auf Q . ✓ □

Reguläre Bäume

Definition

Sei $T = (V, E)$ ein Baum mit Wurzel w :

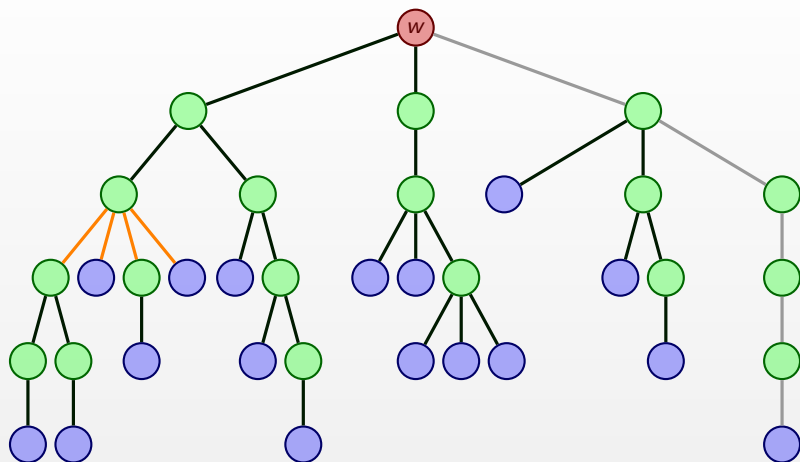
- der **Abstand** einer Ecke v zur Wurzel w ist die Länge des w - v -Weges in T ,
- die **Höhe** von T ist der maximale Abstand den eine Ecke $v \in V$ zu w hat,
- **Grad** von T ist die maximale Anzahl von Kindern, die eine Ecke $v \in V$ hat,
- T ist k -när, falls der Grad von T höchstens k ist.

Achtung: k -näre Bäume haben oft viele Ecken mit Grad $k + 1$

Konventionen:

- eine Wurzel kann auch ein Blatt sein (manchmal wird das ausgeschlossen)
- 2-näre Bäume heißen **binäre** und 3-näre heißen **trinär**
- Ecken, die keine Blätter sind, heißen **innere** Ecken
- ein Wurzelbaum ist **k -regulär**, wenn jede innere Ecke (inklusive Wurzel) genau k Kinder hat
- die Menge der Ecken mit Abstand i zur Wurzel heißt **i -tes Level/ i -te Stufe** von T

Beispiel



Nicht regulärer Baum mit **Wurzel** w , **17 Blättern**, **19 inneren Ecken** (inklusive **Wurzel**), **Grad 4**, Höhe 5 und 6 Leveln mit Größen

$$|L_0| = 1, |L_1| = 3, |L_2| = 6, |L_3| = 12, |L_4| = 10 \text{ und } |L_5| = 4.$$

Reguläre binäre Bäume

Satz

Jeder 2-reguläre Wurzelbaum $T = (V, E)$ mit $n \geq 3$ Ecken hat $\frac{n+1}{2}$ Blätter und $\frac{n-1}{2}$ innere Ecken.

Beweis: Sei $n \geq 3$ und sei $I \subseteq V$ die Menge der inneren Ecken und $B \subseteq V$ die Menge der Blätter. Es gilt also

$$I \cup B = V \quad \text{und} \quad |I| = n - |B|.$$

Aufgrund des Handshaking Lemma und der Baumcharakterisierung erhalten wir

$$2(n-1) = 2|E| = |B| + 3|I| - 1,$$

da jede innere Ecke **bis auf die Wurzel** Grad 3 in T hat. Einsetzen von $|I| = n - |B|$ und Umstellen ergibt:

$$2|B| = 3n - 1 - 2(n-1) = n + 1$$

und die Aussage folgt. □

Höhe und Größe in k -ären Bäume

Satz

Jeder k -äre Wurzelbaum mit Höhe h hat höchstens $\frac{k^{h+1}-1}{k-1}$ Ecken.

Beweis

Sei $T = (V, E)$ ein k -ärer Wurzelbaum mit Höhe h und für $i = 0, \dots, h$ sei L_i das i -te Level von T .

- über Induktion nach i zeigt man $|L_i| \leq k^i$ und somit gilt

$$|V| = \sum_{i=0}^h |L_i| \leq \sum_{i=0}^h k^i = \frac{k^{h+1} - 1}{k - 1},$$

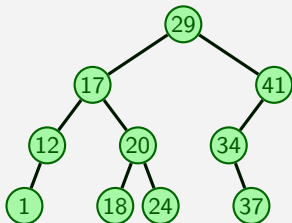
wobei die letzte Identität sich aus der geometrischen Summenformel (siehe Kapitel zur Induktion) ergibt. □

Bemerkungen:

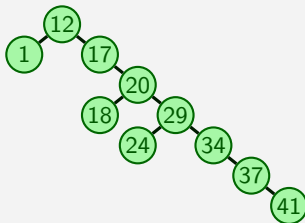
- Höhe k -ärer Bäume wächst mindestens logarithmisch in der Eckenanzahl
- Bäume mit logarithmischer Höhe heißen **balanciert** und sind Grundlage wichtiger Datenstrukturen und Algorithmen → binäre Suchbäume

Binäre Suchbäume

Balancierter binärer Baum



Unbalancierter binärer Baum



Binäre Suchbäume

Binäre Suchbäume sind eine Datenstruktur, die Datensätze mithilfe von Schlüsseln aus \mathbb{N} dynamisch verwaltet. Die Datensätze werden anhand der Schlüssel in einem binären Baum so angeordnet, dass für jede Ecke alle **linken Nachfolger** einen **kleineren** und alle **rechten Nachfolger** einen **größeren** Schlüssel haben.

Da der Zeitaufwand für die Such- und Verwaltungsoperationen in so einer Datenstruktur mit der **Höhe** des Baumes korrelieren, interessiert man sich für balancierte binäre Bäume mit Verwaltungsoperationen (Einfügen/Löschen), die die Balanciertheit erhalten.

→ **AVL-Bäume, Rot-Schwarz-Bäume**

Abstandserhaltender Spannbaum

Problem:

- Für einen Graphen $G = (V, E)$ und zwei Ecken $x, y \in V$ ist der **Abstand** $d_G(x, y)$ die Länge eines kürzesten x - y -Weges in G (bzw. ∞ falls kein solcher Weg existiert).
- Für eine Startecke/Quelle s berechne alle Abstände in einem zusammenhängenden Graphen $G = (V, E)$.
- Finde einen Spannbaum T mit Wurzel s der die Abstände zu s erhält, d. h. für jede Ecke $v \in V$ gilt:

$$d_G(s, v) = d_T(s, v).$$

Idee:

- Ausgehend von s , besuche zuerst alle Nachbarn (Abstand 1), dann deren Nachbarn (Abstand 2) usw., ohne Ecken zu wiederholen.
- Speichere dabei die Kanten durch die Ecken zuerst besucht wurden.
- Dabei entsteht ein Baum mit Wurzel s , der sogenannte **Breitensuchbaum** von G .

Breitensuche (breadth first search/BFS)

Algorithmus: Breitensuche

Gegeben: Zusammenhängender Graph $G = (V, E)$ und Startecke $s \in V$

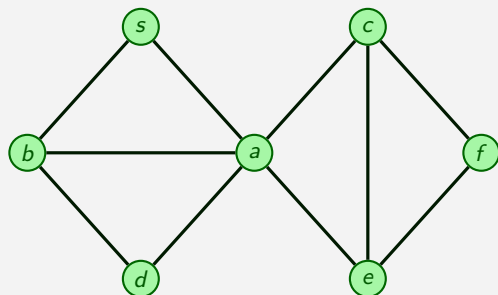
Gesucht: Breitensuchbaum T von G und Abstände $d(s, v)$ für alle $v \in V$

- 1 markiere s als **besucht**, setze $d(s, s) = 0$
- 2 initialisiere Wurzelbaum T , der nur die Wurzel s enthält
- 3 initialisiere eine Warteschlange/Queue Q , die nur s enthält
- 4 solange $Q \neq \emptyset$, sei v das erste Element in Q :
 - a Falls es keinen **unbesuchten** Nachbarn u von v in G gibt:
 - Entferne v als erstes Element von Q .
 - b Falls es einen **unbesuchten** Nachbarn u von v in G gibt:
 - Füge die Ecke u und die Kante $\{v, u\}$ zu T hinzu.
 - Markiere u als **besucht**.
 - Setze $d(s, u) = d(s, v) + 1$.
 - Füge u in Q am Ende ein.

Algorithmus mit Adjazenzlisten in einer Laufzeit linear in $|E|$ implementierbar

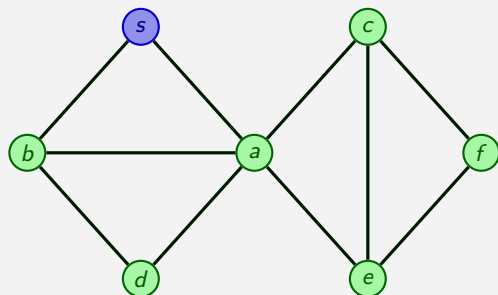
Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Beispiel: Breitensuche

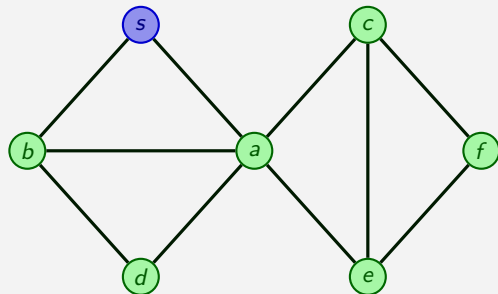
Gegebener Graph $G = (V, E)$ und $s \in V$



■ besuchte Ecken: s

Beispiel: Breitensuche

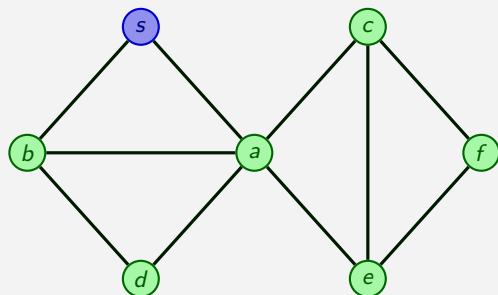
Gegebener Graph $G = (V, E)$ und $s \in V$



- besuchte Ecken: s
- Warteschlange Q: s

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



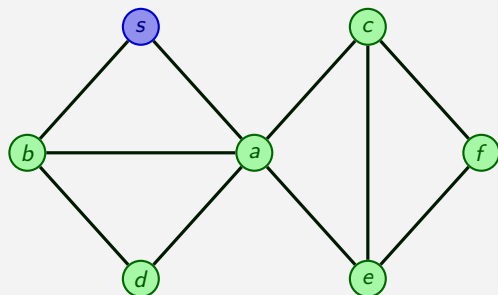
Breitensuchbaum T



- besuchte Ecken: s
- Warteschlange Q : s

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T

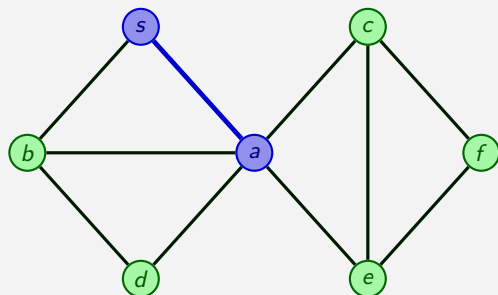


- besuchte Ecken: s
- Warteschlange Q : s

$$d(s, s) = 0$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T

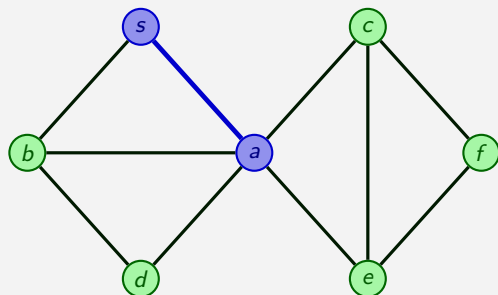


- besuchte Ecken: s
- Warteschlange Q : s
- Laufvariablen: $v = s, u = a$

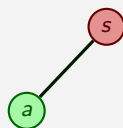
$$d(s, s) = 0$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



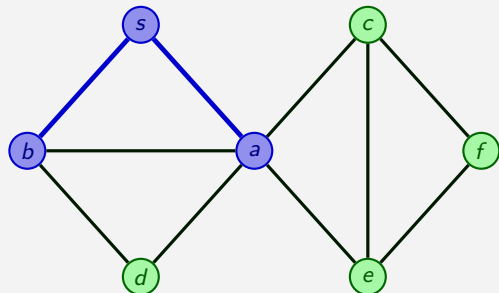
- besuchte Ecken: $s a$
- Warteschlange Q : $s a$
- Laufvariablen: $v = s, u = a$

$$d(s, s) = 0$$

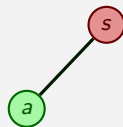
$$d(s, a) = 1$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



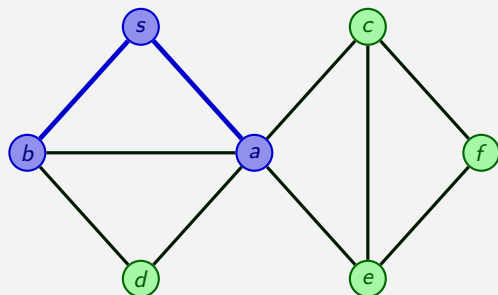
- besuchte Ecken: $s a$
- Warteschlange Q : $s a$
- Laufvariablen: $v = s, u = b$

$$d(s, s) = 0$$

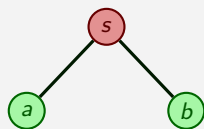
$$d(s, a) = 1$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b$
- Warteschlange Q : $s a b$
- Laufvariablen: $v = s, u = b$

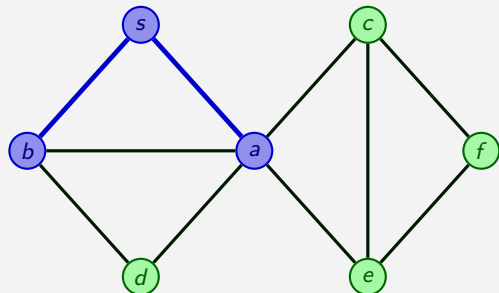
$$d(s, s) = 0$$

$$d(s, a) = 1$$

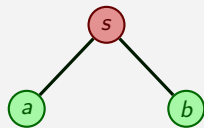
$$d(s, b) = 1$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b$
- Warteschlange Q : $\cancel{a} b$
- Laufvariablen: $v = s$

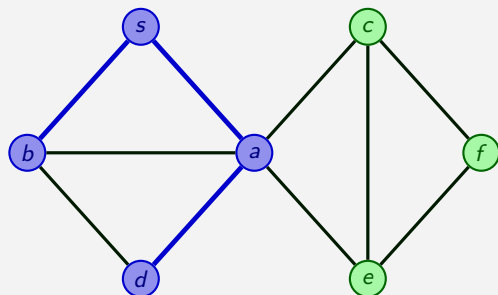
$$d(s, s) = 0$$

$$d(s, a) = 1$$

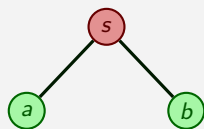
$$d(s, b) = 1$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b$
- Warteschlange Q : $a b$
- Laufvariablen: $v = a, u = d$

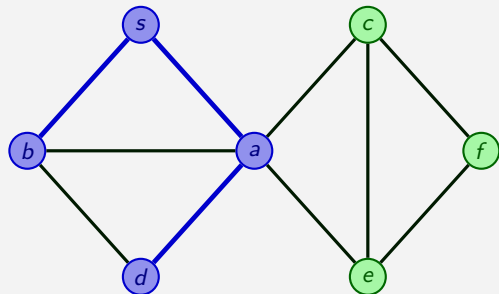
$$d(s, s) = 0$$

$$d(s, a) = 1$$

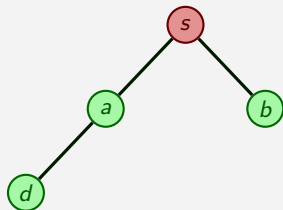
$$d(s, b) = 1$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b d$
- Warteschlange Q : $a b d$
- Laufvariablen: $v = a, u = d$

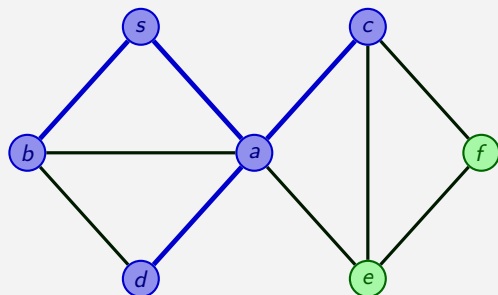
$$d(s, s) = 0, \quad d(s, d) = 2$$

$$d(s, a) = 1$$

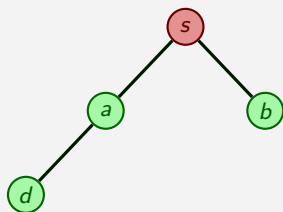
$$d(s, b) = 1$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b d$
- Warteschlange Q : $a b d$
- Laufvariablen: $v = a, u = c$

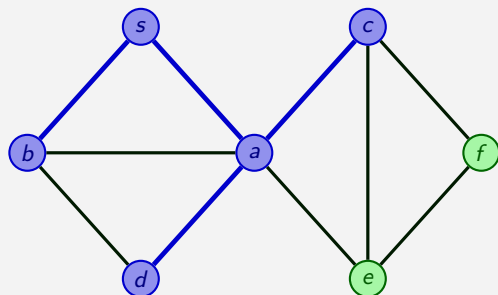
$$d(s, s) = 0, \quad d(s, d) = 2$$

$$d(s, a) = 1$$

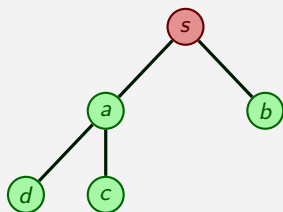
$$d(s, b) = 1$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T

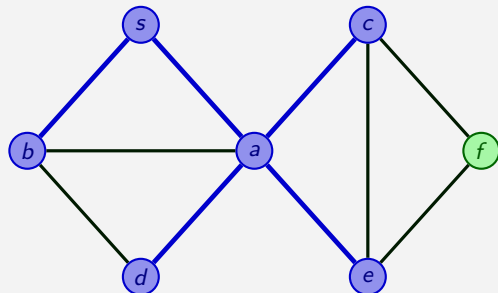


- besuchte Ecken: $s a b d c$
- Warteschlange Q : $a b d c$
- Laufvariablen: $v = a, u = c$

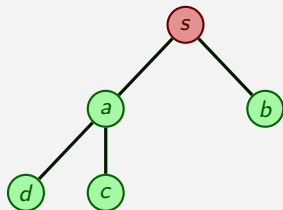
$$\begin{aligned}d(s, s) &= 0, & d(s, d) &= 2 \\d(s, a) &= 1, & d(s, c) &= 2 \\d(s, b) &= 1\end{aligned}$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T

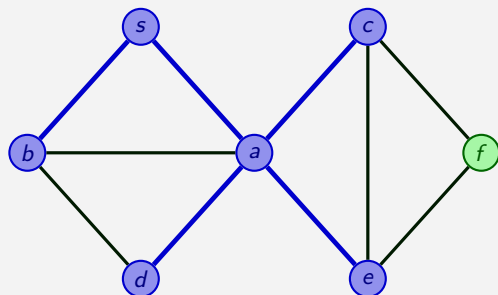


- besuchte Ecken: $s a b d c$
- Warteschlange Q : $a b d c$
- Laufvariablen: $v = a, u = e$

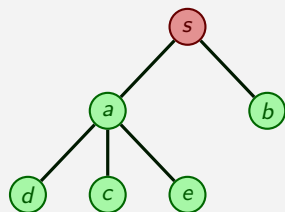
$$\begin{aligned}d(s, s) &= 0, & d(s, d) &= 2 \\d(s, a) &= 1, & d(s, c) &= 2 \\d(s, b) &= 1\end{aligned}$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b d c e$
- Warteschlange Q : $a b d c e$
- Laufvariablen: $v = a, u = e$

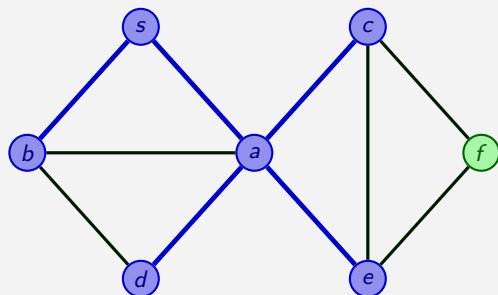
$$d(s, s) = 0, \quad d(s, d) = 2$$

$$d(s, a) = 1, \quad d(s, c) = 2$$

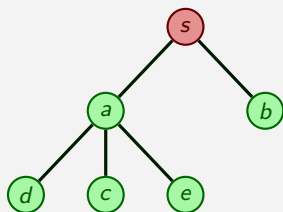
$$d(s, b) = 1, \quad d(s, e) = 2$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b d c e$
- Warteschlange Q : ~~$a b d c e$~~
- Laufvariablen: $v = a$

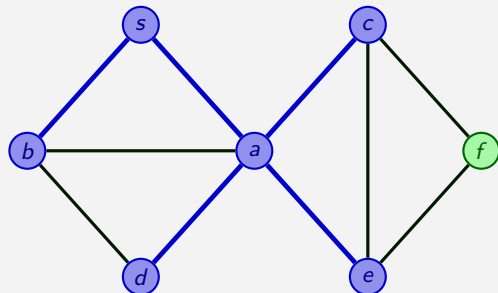
$$d(s, s) = 0, \quad d(s, d) = 2$$

$$d(s, a) = 1, \quad d(s, c) = 2$$

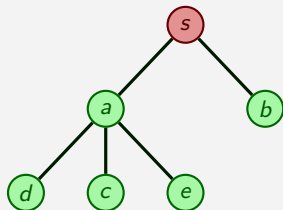
$$d(s, b) = 1, \quad d(s, e) = 2$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b d c e$
- Warteschlange Q : $\cancel{b} d c e$
- Laufvariablen: $v = b$

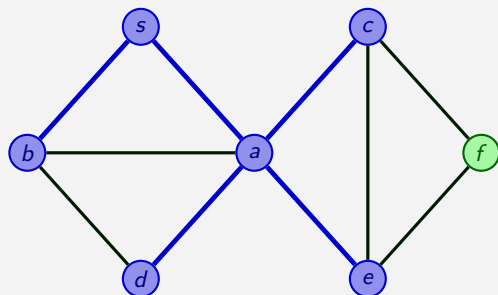
$$d(s, s) = 0, \quad d(s, d) = 2$$

$$d(s, a) = 1, \quad d(s, c) = 2$$

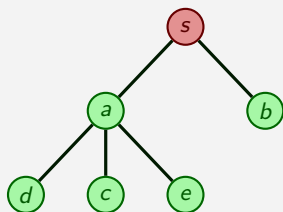
$$d(s, b) = 1, \quad d(s, e) = 2$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



■ besuchte Ecken: $s a b d c e$

■ Warteschlange Q : $\cancel{d} c e$

■ Laufvariablen: $v = d$

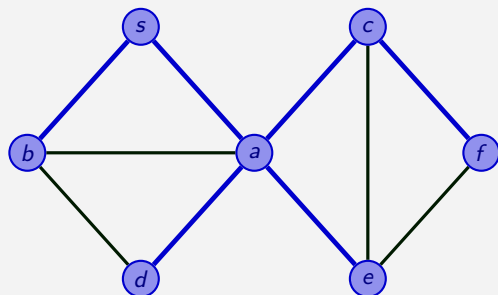
$$d(s, s) = 0, \quad d(s, d) = 2$$

$$d(s, a) = 1, \quad d(s, c) = 2$$

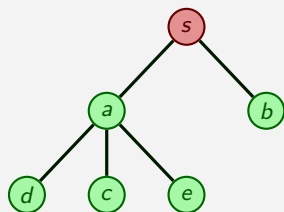
$$d(s, b) = 1, \quad d(s, e) = 2$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



- besuchte Ecken: $s a b d c e$
- Warteschlange Q : $c e$
- Laufvariablen: $v = c, u = f$

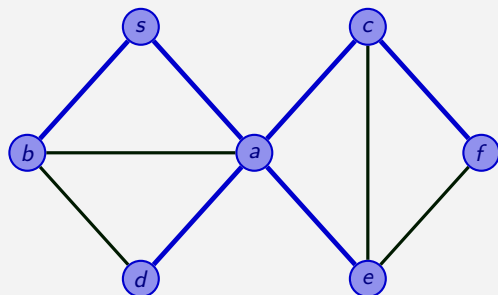
$$d(s, s) = 0, \quad d(s, d) = 2$$

$$d(s, a) = 1, \quad d(s, c) = 2$$

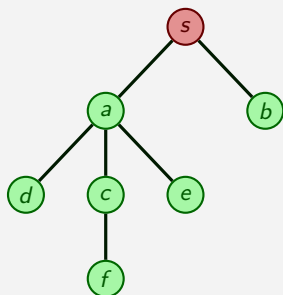
$$d(s, b) = 1, \quad d(s, e) = 2$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T

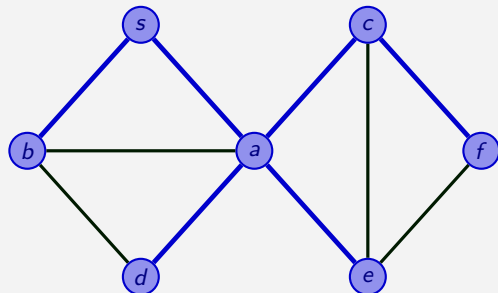


- besuchte Ecken: $s a b d c e f$
- Warteschlange Q : $c e f$
- Laufvariablen: $v = c, u = f$

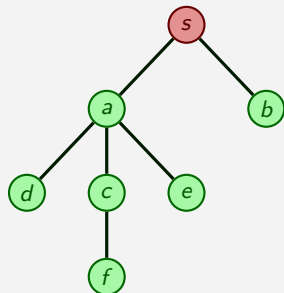
$$\begin{aligned}d(s, s) &= 0, & d(s, d) &= 2 \\d(s, a) &= 1, & d(s, c) &= 2 \\d(s, b) &= 1, & d(s, e) &= 2 \\& & d(s, f) &= 3\end{aligned}$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



■ besuchte Ecken: $s a b d c e f$

■ Warteschlange Q : $\emptyset e f$

■ Laufvariablen: $v = c$

$$d(s, s) = 0, \quad d(s, d) = 2$$

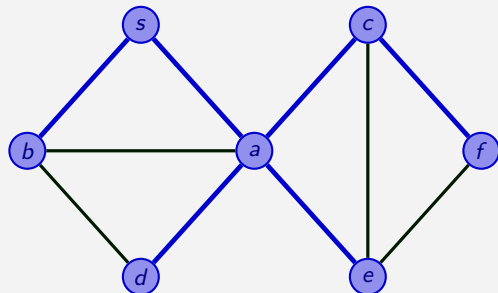
$$d(s, a) = 1, \quad d(s, c) = 2$$

$$d(s, b) = 1, \quad d(s, e) = 2$$

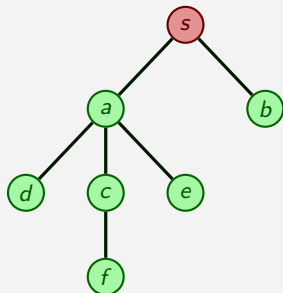
$$d(s, f) = 3$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



■ besuchte Ecken: $s a b d c e f$

■ Warteschlange Q : \emptyset

■ Laufvariablen: $v = e$

$$d(s, s) = 0, \quad d(s, d) = 2$$

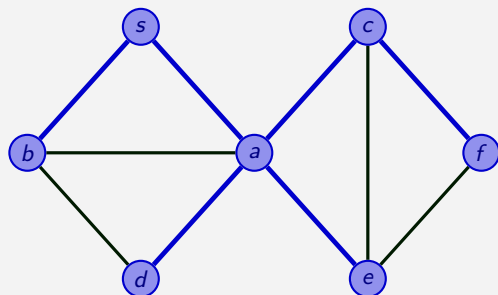
$$d(s, a) = 1, \quad d(s, c) = 2$$

$$d(s, b) = 1, \quad d(s, e) = 2$$

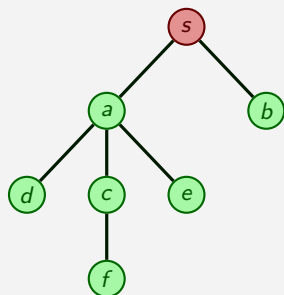
$$d(s, f) = 3$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



■ besuchte Ecken: $s a b d c e f$

■ Warteschlange Q : f

■ Laufvariablen: $v = f$

$$d(s, s) = 0, \quad d(s, d) = 2$$

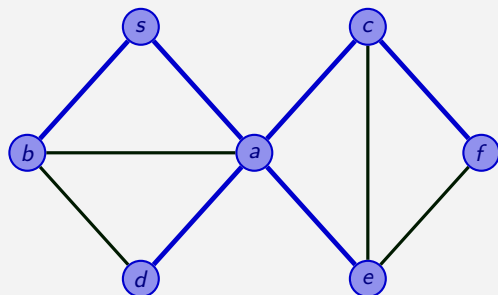
$$d(s, a) = 1, \quad d(s, c) = 2$$

$$d(s, b) = 1, \quad d(s, e) = 2$$

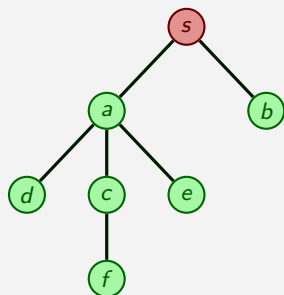
$$d(s, f) = 3$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



■ besuchte Ecken: $s a b d c e f$

■ Warteschlange Q : \emptyset

$$d(s, s) = 0, \quad d(s, d) = 2$$

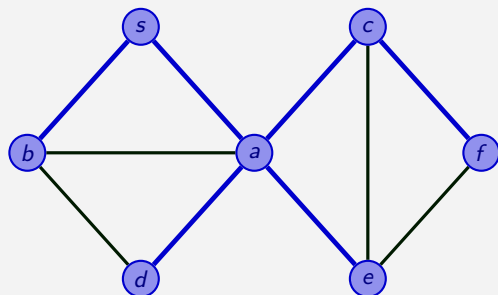
$$d(s, a) = 1, \quad d(s, c) = 2$$

$$d(s, b) = 1, \quad d(s, e) = 2$$

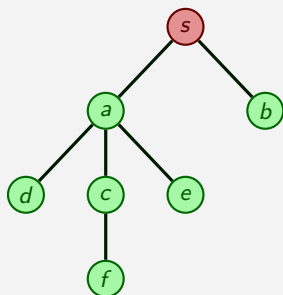
$$d(s, f) = 3$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



■ besuchte Ecken: $s a b d c e f$

■ Warteschlange Q : \emptyset

⇒ **Algorithmus terminiert**

$$d(s, s) = 0, \quad d(s, d) = 2$$

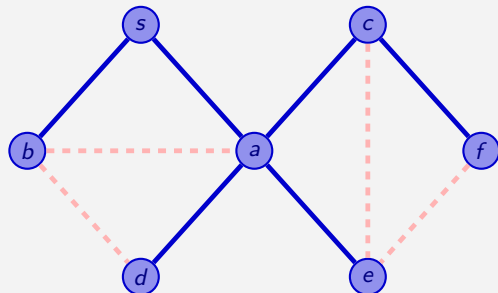
$$d(s, a) = 1, \quad d(s, c) = 2$$

$$d(s, b) = 1, \quad d(s, e) = 2$$

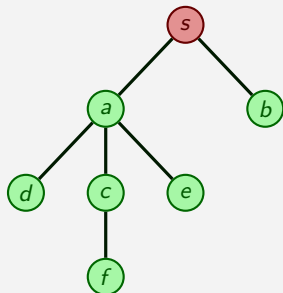
$$d(s, f) = 3$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



■ besuchte Ecken: $s a b d c e f$

■ Warteschlange Q : \emptyset

⇒ **Algorithmus terminiert**

$$d(s, s) = 0, \quad d(s, d) = 2$$

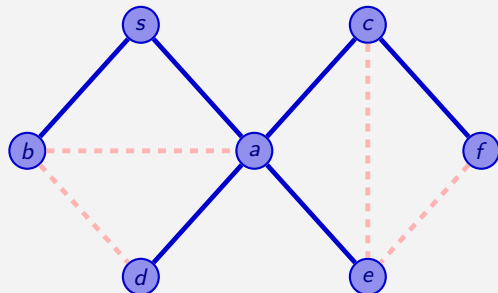
$$d(s, a) = 1, \quad d(s, c) = 2$$

$$d(s, b) = 1, \quad d(s, e) = 2$$

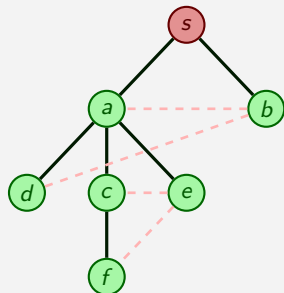
$$d(s, f) = 3$$

Beispiel: Breitensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Breitensuchbaum T



■ besuchte Ecken: $s a b d c e f$

■ Warteschlange Q : \emptyset

⇒ **Algorithmus terminiert**

$$d(s, s) = 0, \quad d(s, d) = 2$$

$$d(s, a) = 1, \quad d(s, c) = 2$$

$$d(s, b) = 1, \quad d(s, e) = 2$$

$$d(s, f) = 3$$

Korrektheit der Breitensuche

Beweis: Sei $s = v_1, \dots, v_n$ die Reihenfolge in der die Ecken im Algorithmus **besucht** werden.

- jede Ecke $v \in V$ wird höchstens einmal besucht, da nur unbesuchte Ecken in **b** **besucht** werden und dies nie rückgängig gemacht wird
- jede Ecke $v \in V$ wird tatsächlich auch besucht, d. h. $v = v_i$ für ein $i \in [n]$
→ einfacher Induktionsbeweis nach $d_G(s, v)$
- zu jedem Zeitpunkt im Algorithmus (**Schleifeninvariante**) ist T zusammenhängend und $|E(T)| = |V(T)| - 1$ Kanten
→ einfacher Induktionsbeweis
- ⇒ T ist zu jeder Zeit ein Baum (Baumcharakterisierung **5**)
- ⇒ T ist ein Spannbaum, sobald die letzte Ecke v_n besucht wird
- $d(s, v_i) \geq d_G(s, v_i)$ für $i = 1, \dots, n$
→ Induktionsbeweis nach i
- für alle $i = 2, \dots, n$ gilt $d(s, v_{i-1}) \leq d(s, v_i)$ für die vom Algorithmus definierte Abstandsfunktion $d(s, \cdot)$
→ Induktionsbeweis nach i

Schließlich zeigen wir $d(s, u) \leq d_G(s, u)$ für jede Ecke $u \in V$.

- Sei u mit minimalem Abstand $d_G(s, u)$ zu s , so dass $d(s, u) > d_G(s, u)$
- ⇒ insbesondere $u \neq s$, da $0 = d(s, s) = d_G(s, s)$
- sei P ein kürzester s - u -Weg in G und v die Ecke vor u auf P
- ⇒ $d_G(s, v) < d_G(s, u)$ und wegen der Wahl von u gilt $d(s, v) = d_G(s, v)$
- ⇒ $d(s, v) < d(s, u)$ und somit wurde v vor u besucht und in Q einsortiert
- ⇒ spätestens als v erstes Element von Q war wurde u besucht
- ⇒ $d(s, u) \leq d(s, v) + 1 = d_G(s, v) + 1 = d_G(s, u)$ □

Struktur von Breitensuchbäumen

- Breitensuchbäume sind im Allgemeinen nicht eindeutig, da nicht festgelegt wird in welcher Reihenfolge die Nachbarn von v in der Schleife 4 besucht werden
- allerdings sind die Levelmengen in jedem Breitensuchbaum T mit Startecke s gleich, da gilt

$$L_i = \{v \in V : d_G(s, v) = i\}$$

und für jedes v ist der s - v -Weg in T ein kürzester s - v -Weg in G

- für jede Kante $\{x, y\} \in E(G)$ gibt es $i, j \in \mathbb{N}$, sodass

$$x \in L_i, \quad y \in L_j \quad \text{und} \quad |i - j| \leq 1,$$

d. h. Kanten verlaufen nur innerhalb eines Levels oder zwischen zwei benachbarten Leveln, da wegen der Kante $\{x, y\}$ sofort $|d_G(s, x) - d_G(s, y)| \leq 1$ gelten muss

Tiefensuche (depth first search/DFS)

Wenn man in dem Breitensuche-Algorithmus die Warteschlange Q durch einen Stapel/Stack S ersetzt, erhält man die Tiefensuche.

Algorithmus: Tiefensuche

Gegeben: Zusammenhängender Graph $G = (V, E)$ und Startecke $s \in V$

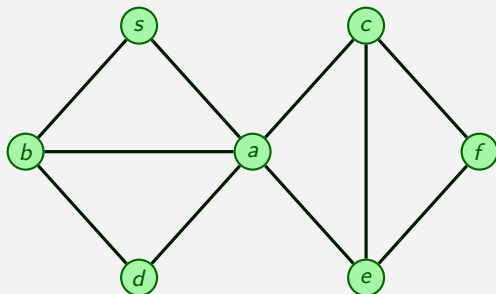
Gesucht: Tiefensuchbaum T von G

- 1 markiere s als **besucht**
- 2 initialisiere Wurzelbaum T , der nur die Wurzel s enthält
- 3 initialisiere einen **Stapel/Stack** S , der nur s enthält
- 4 solange $S \neq \emptyset$ sei v das oberste Element in S :
 - a Falls es keinen **unbesuchten** Nachbarn u von v in G gibt:
 - Entferne v als oberstes Element von S .
 - b Falls es einen **unbesuchten** Nachbarn u von v in G gibt:
 - Füge die Ecke u und die Kante $\{v, u\}$ zu T hinzu.
 - Markiere u als **besucht**.
 - Lege u oben auf den Stapel S .

Algorithmus mit Adjazenzlisten in einer Laufzeit linear in $|E|$ implementierbar

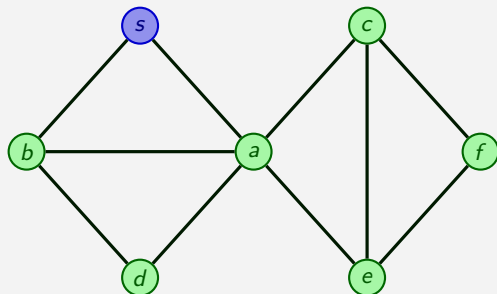
Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



Beispiel: Tiefensuche

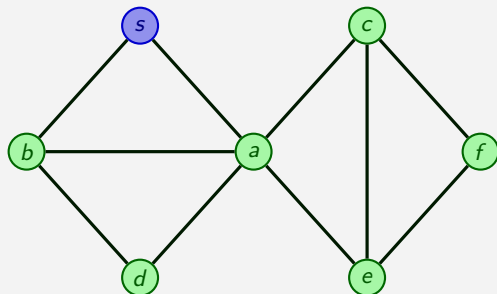
Gegebener Graph $G = (V, E)$ und $s \in V$



- besuchte Ecken: s

Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

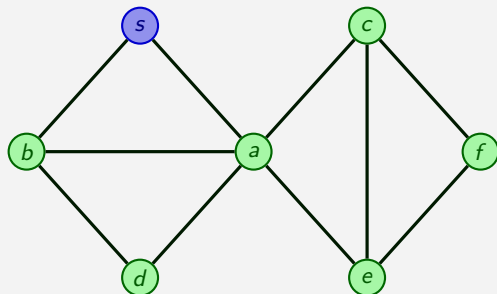


- besuchte Ecken: s
- Stapel S :

S

Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



- besuchte Ecken: s
- Stapel S :

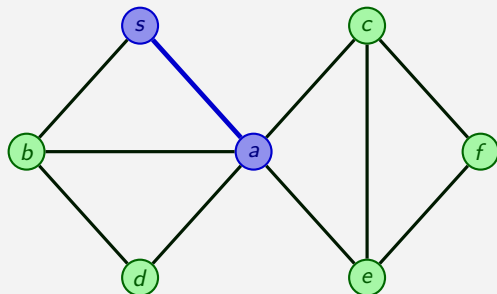
Tiefensuchbaum T



S

Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



■ besuchte Ecken: s a

■ Stapel S :

a

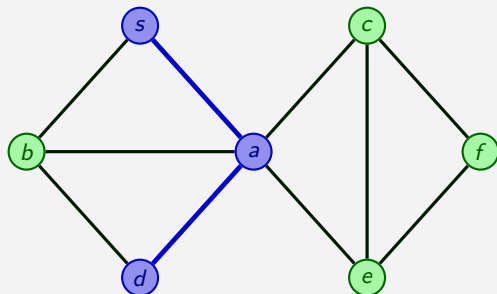
s

Tiefensuchbaum T



Beispiel: Tiefensuche

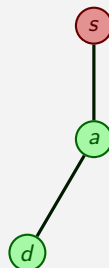
Gegebener Graph $G = (V, E)$ und $s \in V$



- besuchte Ecken: s a d
- Stapel S :

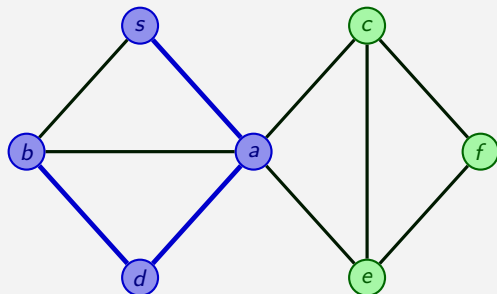
d
 a
 s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

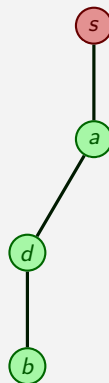


■ besuchte Ecken: $s a d b$

■ Stapel S :

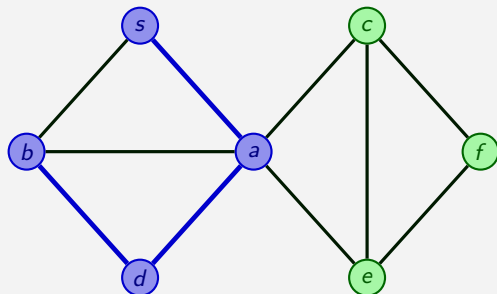
b
 d
 a
 s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

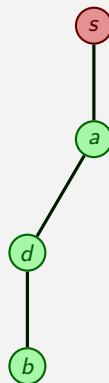


■ besuchte Ecken: $s a d b$

■ Stapel S :

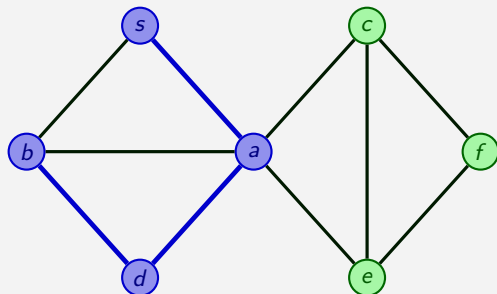
~~b~~
 d
 a
 s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$



■ besuchte Ecken: $s a d b$

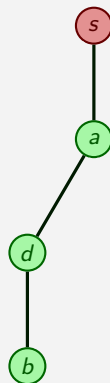
■ Stapel S :

a

a

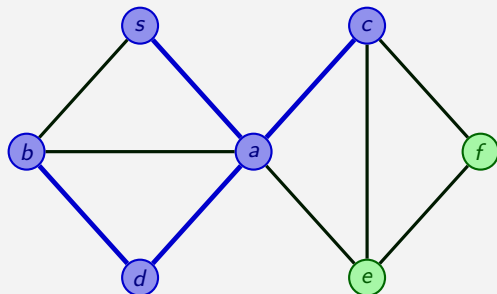
s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

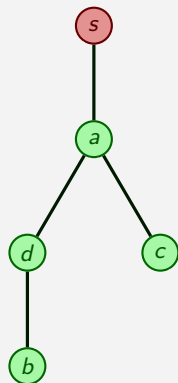


■ besuchte Ecken: $s a d b c$

■ Stapel S :

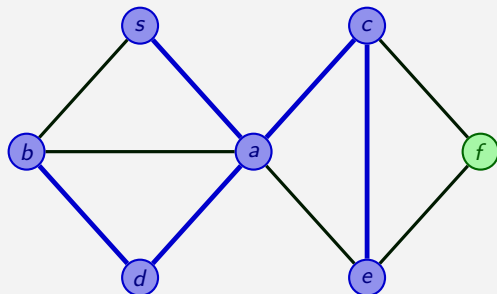
c
 a
 s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

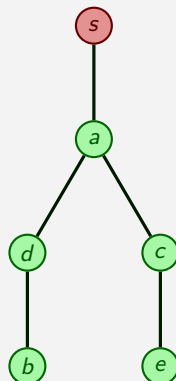


■ besuchte Ecken: $s a d b c e$

■ Stapel S :

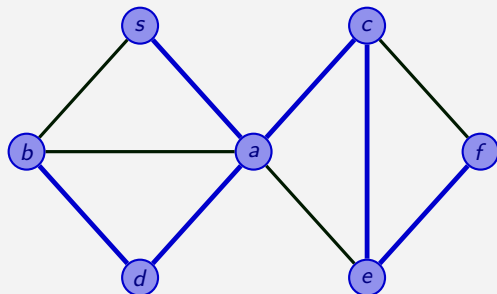
e
 c
 a
 s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

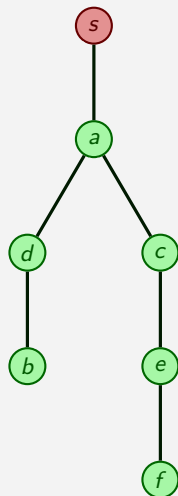


■ besuchte Ecken: $s a d b c e f$

■ Stapel S :

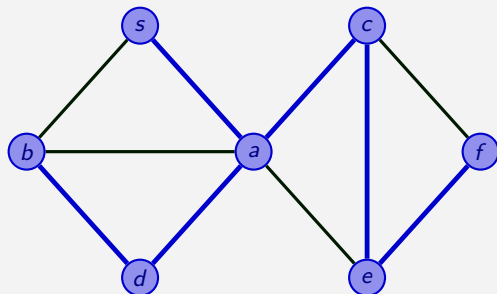
f
 e
 c
 a
 s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

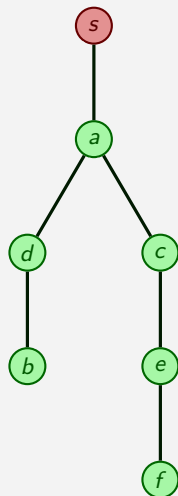


■ besuchte Ecken: $s a d b c e f$

■ Stapel S :

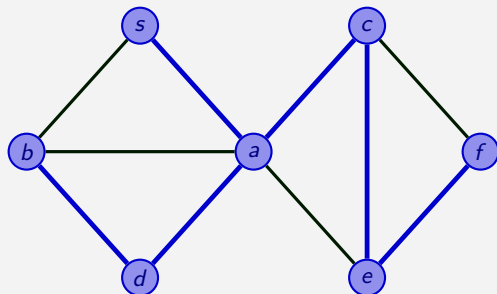
~~f~~
e
c
a
s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

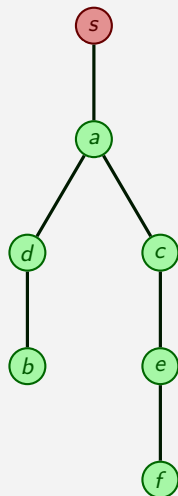


■ besuchte Ecken: $s a d b c e f$

■ Stapel S :

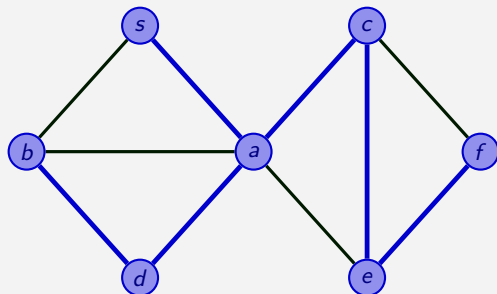
~~c~~
c
a
s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

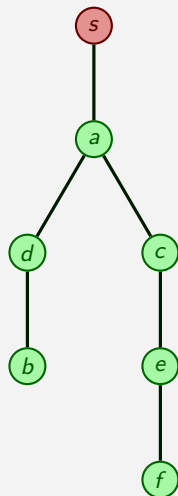


■ besuchte Ecken: $s a d b c e f$

■ Stapel S :

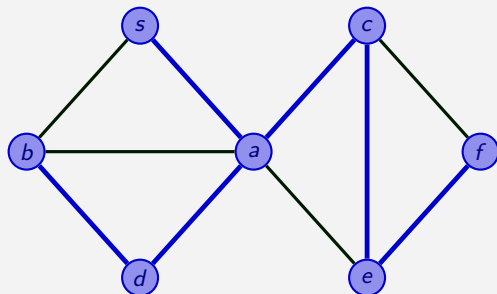
$\not\{$
 a
 s

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

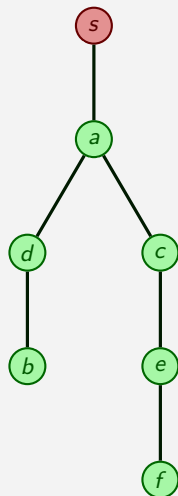


■ besuchte Ecken: $s a d b c e f$

■ Stapel S :

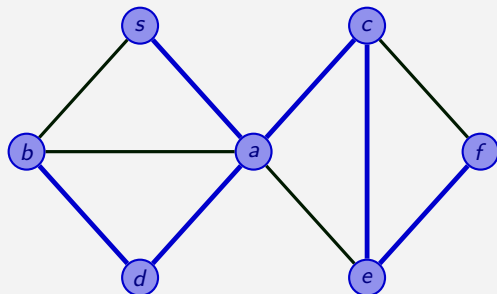
~~a~~
s

Tiefensuchbaum T



Beispiel: Tiefensuche

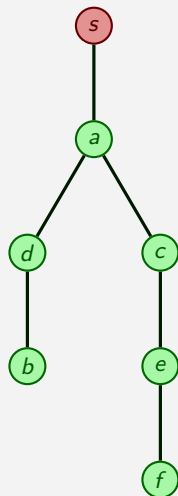
Gegebener Graph $G = (V, E)$ und $s \in V$



■ besuchte Ecken: $s a d b c e f$

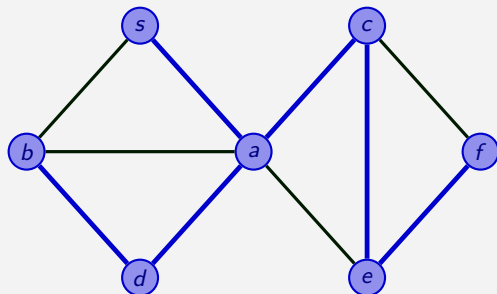
■ Stapel S :

Tiefensuchbaum T



Beispiel: Tiefensuche

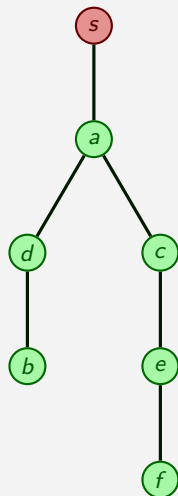
Gegebener Graph $G = (V, E)$ und $s \in V$



■ besuchte Ecken: $s a d b c e f$

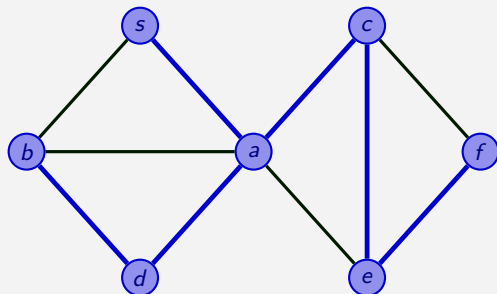
■ Stapel S : \emptyset

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

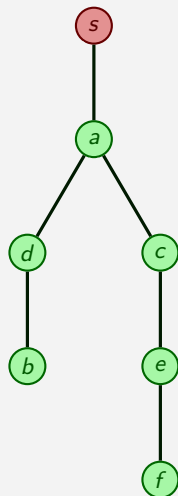


■ besuchte Ecken: $s a d b c e f$

■ Stapel S : \emptyset

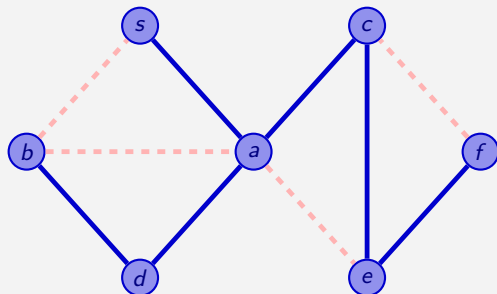
⇒ **Algorithmus terminiert**

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

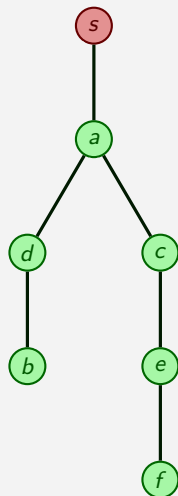


■ besuchte Ecken: $s a d b c e f$

■ Stapel S : \emptyset

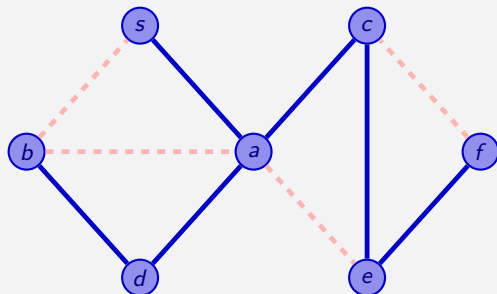
⇒ **Algorithmus terminiert**

Tiefensuchbaum T



Beispiel: Tiefensuche

Gegebener Graph $G = (V, E)$ und $s \in V$

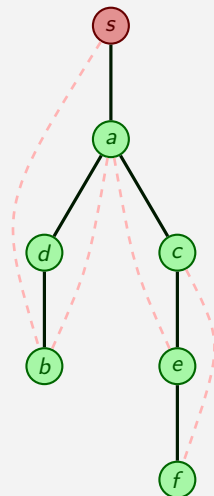


■ besuchte Ecken: $s a d b c e f$

■ Stapel S : \emptyset

⇒ **Algorithmus terminiert**

Tiefensuchbaum T



Struktur von Tiefensuchbäumen

- Tiefensuche liefert einen Spannbaum T für zusammenhängende Graphen G zurück (Warum?)
- wie bei der Breitensuche, ist T nicht eindeutig bestimmt

Satz

Sei T ein Spannbaum erzeugt durch die Tiefensuche für einen zusammenhängende Graphen $G = (V, E)$ mit Startecke/Wurzel s und sei \leq_T die entsprechende Baumordnung auf V . Für jede Kante $\{x, y\} \in E$ gilt entweder

$$x \leq_T y \quad \text{oder} \quad y \leq_T x,$$

d. h. x und y sind (nicht unbedingt direkte) Vorgänger und Nachfolger voneinander.

Bemerkung:

- Spannbäume mit dieser Eigenschaft heißen **normale Spannbäume**
- Algorithmus und Satz zeigen, dass jeder zusammenhängende Graph einen normalen Spannbaum hat

$\{x, y\} \in E \implies$ entweder $x \leq_T y$ oder $y \leq_T x$

Beweis

Sei $s = v_1, \dots, v_n$ die Reihenfolge in der die Ecken im Algorithmus besucht werden und sei $e = \{v_i, v_j\} \in E$ mit $1 \leq i < j \leq n$.

Falls $\{v_i, v_j\}$ eine Kante des Baumes T ist, dann ist $j = i + 1$ und somit ist v_i ein direkter Vorgänger von v_j in T ✓

Sei also $\{v_i, v_j\} \in E$ keine Kante des Baumes T . Da v_i vor v_j besucht und auf den Stapel gelegt wurde und erst vom Stapel in Schritt **a** entfernt wird, wenn alle Nachbarn besucht wurden, wurde v_j in der Zwischenzeit auf den Stapel (oberhalb von v_i) gelegt und irgendwann entfernt.

Mit Induktion zeigt man dann, dass alle Ecken u die besucht werden, während v_i irgendwo im Stapel S liegt, in einem Teilbaum von T unterhalb von v_i liegen.

Insbesondere gilt damit also

$$v_i \leq_T v_j$$



Minimale Spannbäume

Problem:

- für einen zusammenhängenden Graphen $G = (V, E)$ mit Kantengewichten

$$w: E \rightarrow \mathbb{R}$$

suchen wir einen Spannbaum T mit minimalem Gewicht (minimaler Spannbaum/MST)

$$w(T) := \sum_{e \in E(T)} w(e).$$

Idee:

- Greedy-Strategie: lokal bestmögliche Wahl ausführen, „ohne an die Konsequenzen zu denken“
- hier: wähle „leichteste“ noch nicht betrachtete Kante e und
 - füge e in den aktuellen Wald T ein, falls e darin keinen Kreis schließt
 - andernfalls, verwerfe e

Algorithmus von KRUSKAL

Algorithmus: MST-Kruskal

Gegeben: gewichteter, zusammenhängender Graph $G = (V, E)$ mit Kantengewichte $w: E \rightarrow \mathbb{R}$

Gesucht: minimaler Spannbaum T

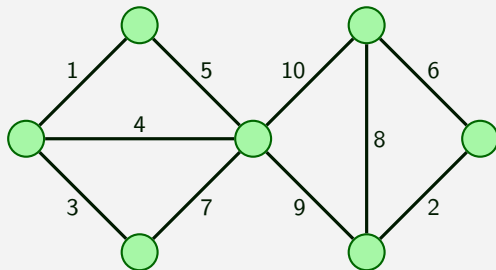
- 1 sortiere die Kanten $E = \{e_1, \dots, e_m\}$, sodass $w(e_1) \leq \dots \leq w(e_m)$
- 2 initialisiere aufspannenden Wald $T = (V, \emptyset)$ ohne Kanten
- 3 für $i = 1, \dots, m$:
 - a falls $T + e_i$ kreisfrei ist, füge e_i zu T hinzu, d. h. $T := T + e_i$

Historische Bemerkungen:

- bereits vor KRUSKAL (1956) wurden Algorithmen für das MST-Problem von BORŮVKA (1926) und JARNÍK (1930) publiziert
- JARNÍKs Algorithmus wurde 1957 von PRIM wiederentdeckt: Greedy-Algorithmus, wobei allerdings T die ganze Zeit ein Baum ist, d. h. in jedem Schritt wählt man eine leichteste Kante, die den aktuellen Baum T mit einer Ecke außerhalb von T verbindet

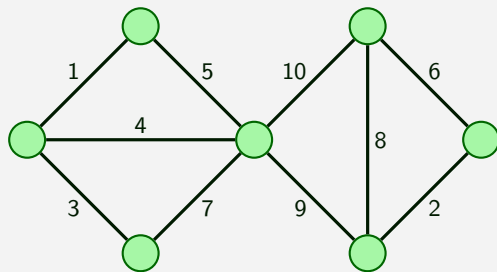
Beispiel: KRUSKALS Algorithmus

gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



Beispiel: KRUSKALS Algorithmus

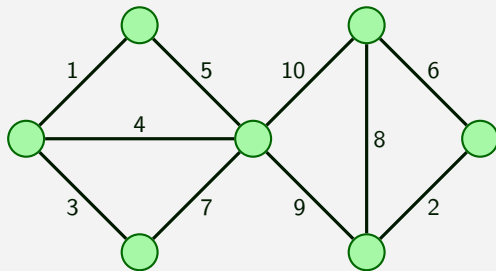
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Beispiel: KRUSKALS Algorithmus

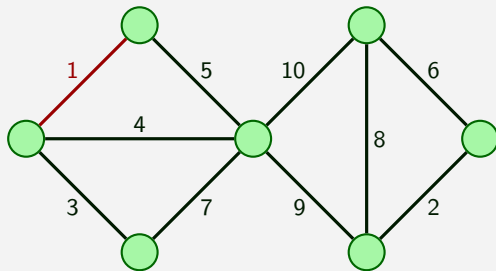
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- initialisiere $V(T) = V(G)$ und $E(T) = \emptyset$

Beispiel: KRUSKALS Algorithmus

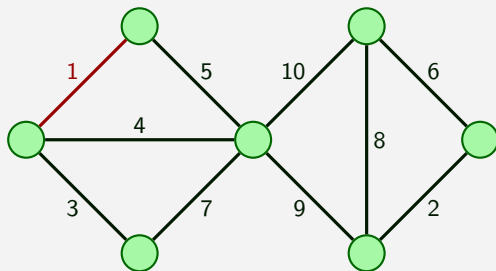
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- initialisiere $V(T) = V(G)$ und $E(T) = \emptyset$
- **aktuelle Kante:** hat Gewicht 1,

Beispiel: KRUSKALS Algorithmus

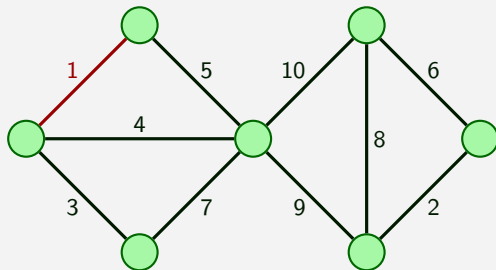
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- initialisiere $V(T) = V(G)$ und $E(T) = \emptyset$
- **aktuelle Kante:** hat Gewicht 1,
 - Kante 1 schließt keinen Kreis in T

Beispiel: KRUSKALS Algorithmus

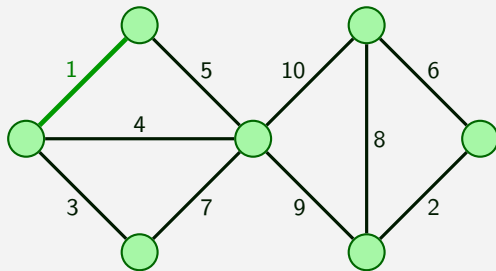
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- initialisiere $V(T) = V(G)$ und $E(T) = \emptyset$
- **aktuelle Kante:** hat Gewicht 1,
 - Kante 1 schließt keinen Kreis in T \Rightarrow Kante zu T hinzufügen

Beispiel: KRUSKALS Algorithmus

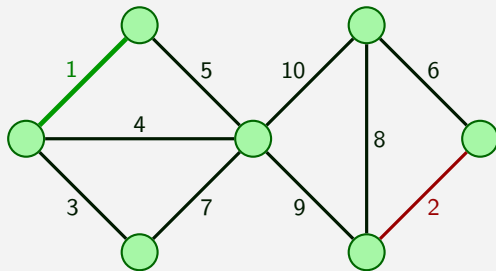
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: $\cancel{1}, 2, 3, 4, 5, 6, 7, 8, 9, 10$
- $V(T) = V(G)$ und $E(T) = \{1\}$

Beispiel: KRUSKALS Algorithmus

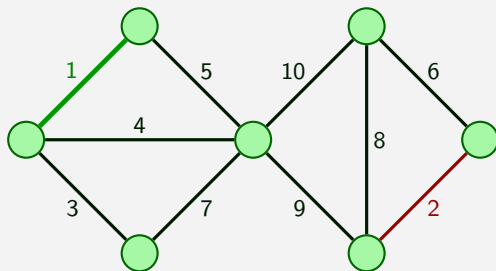
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, 2, 3, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1\}$
- **aktuelle Kante:** hat Gewicht 2,

Beispiel: KRUSKALS Algorithmus

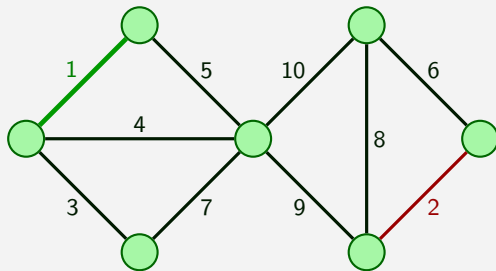
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: $\cancel{1}, 2, 3, 4, 5, 6, 7, 8, 9, 10$
- $V(T) = V(G)$ und $E(T) = \{1\}$
- **aktuelle Kante:** hat Gewicht 2,
 - Kante 2 schließt keinen Kreis in T

Beispiel: KRUSKALS Algorithmus

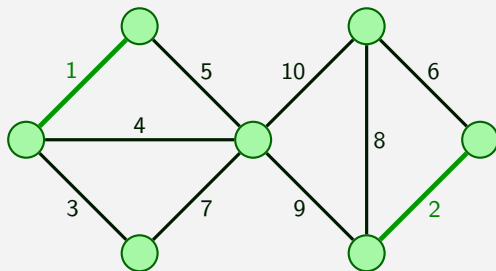
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, 2, 3, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1\}$
- **aktuelle Kante:** hat Gewicht 2,
 - Kante 2 schließt keinen Kreis in T \Rightarrow Kante zu T hinzufügen

Beispiel: KRUSKALS Algorithmus

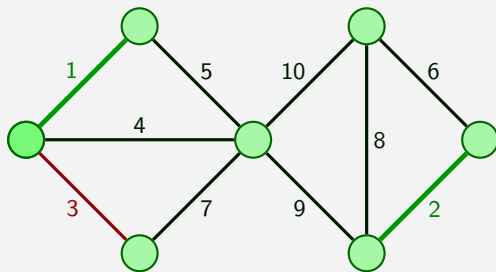
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, 3, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2\}$

Beispiel: KRUSKALS Algorithmus

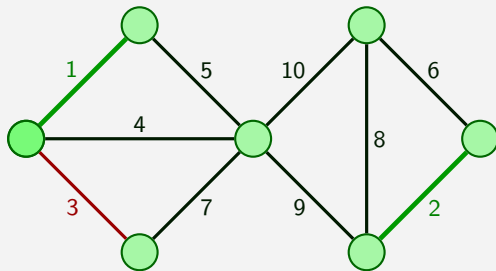
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, 3, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2\}$
- **aktuelle Kante:** hat Gewicht 3,

Beispiel: KRUSKALS Algorithmus

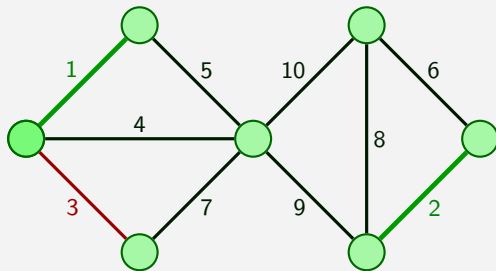
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, 3, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2\}$
- **aktuelle Kante:** hat Gewicht 3,
 - Kante 3 schließt keinen Kreis in T

Beispiel: KRUSKALS Algorithmus

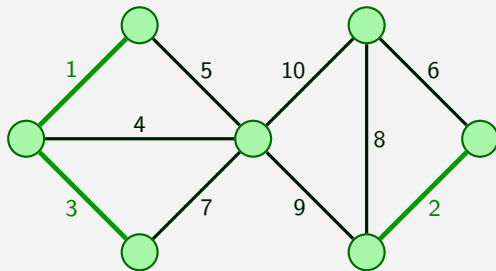
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, 3, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2\}$
- **aktuelle Kante:** hat Gewicht 3,
 - Kante 3 schließt keinen Kreis in T \Rightarrow Kante zu T hinzufügen

Beispiel: KRUSKALS Algorithmus

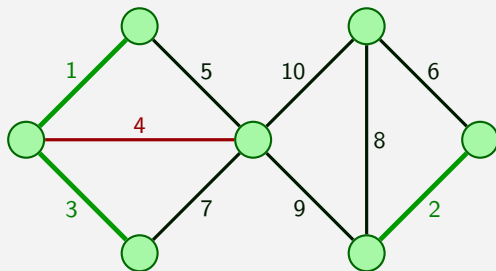
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3\}$

Beispiel: KRUSKALS Algorithmus

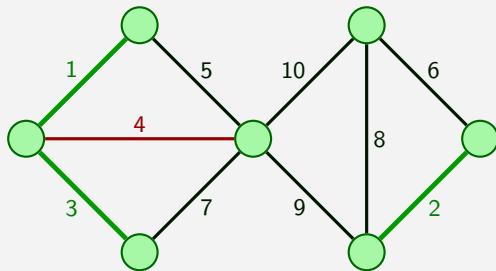
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3\}$
- **aktuelle Kante:** hat Gewicht 4,

Beispiel: KRUSKALS Algorithmus

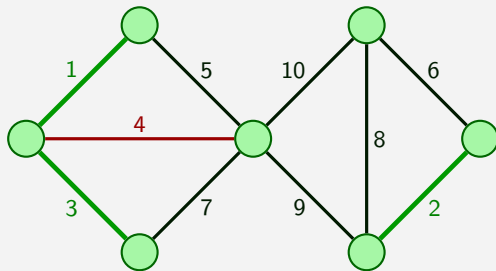
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3\}$
- **aktuelle Kante:** hat Gewicht 4,
 - Kante 4 schließt keinen Kreis in T

Beispiel: KRUSKALS Algorithmus

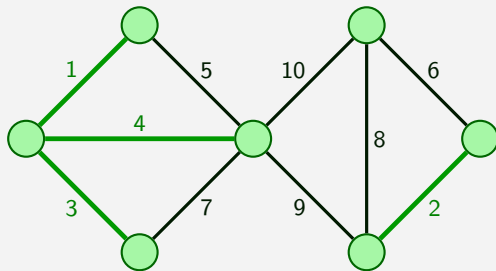
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, 4, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3\}$
- **aktuelle Kante:** hat Gewicht 4,
 - Kante 4 schließt keinen Kreis in T \Rightarrow Kante zu T hinzufügen

Beispiel: KRUSKALS Algorithmus

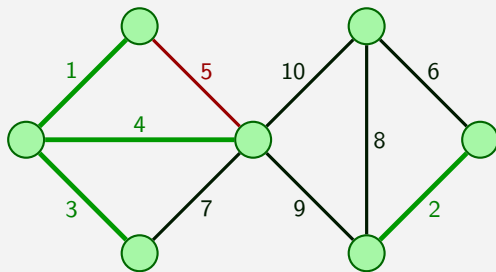
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4\}$

Beispiel: KRUSKALS Algorithmus

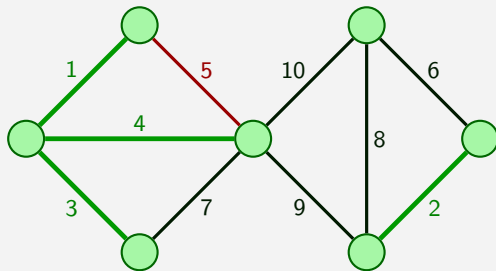
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4\}$
- **aktuelle Kante:** hat Gewicht 5,

Beispiel: KRUSKALS Algorithmus

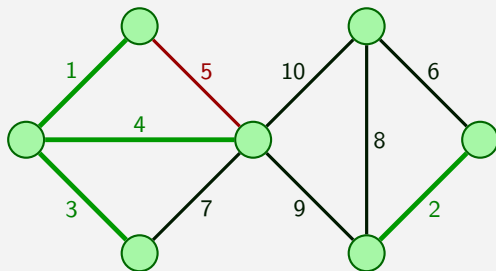
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4\}$
- **aktuelle Kante:** hat Gewicht 5,
 - Kante 5 **schließt einen Kreis** in T

Beispiel: KRUSKALS Algorithmus

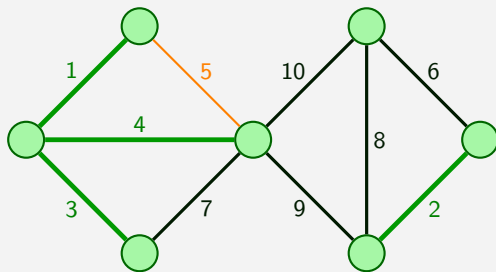
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, 5, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4\}$
- **aktuelle Kante:** hat Gewicht 5,
 - Kante 5 schließt einen Kreis in T \Rightarrow Kante verwerfen

Beispiel: KRUSKALS Algorithmus

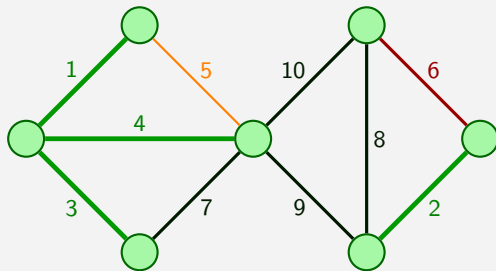
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4\}$

Beispiel: KRUSKALS Algorithmus

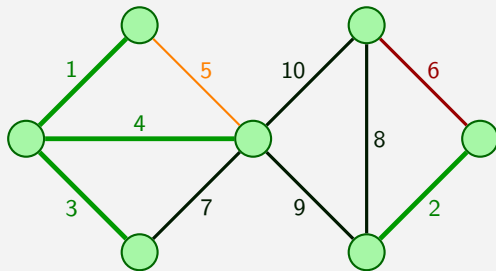
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4\}$
- **aktuelle Kante:** hat Gewicht 6,

Beispiel: KRUSKALS Algorithmus

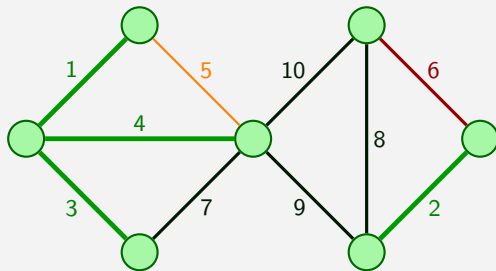
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4\}$
- **aktuelle Kante:** hat Gewicht 6,
 - Kante 6 schließt keinen Kreis in T

Beispiel: KRUSKALS Algorithmus

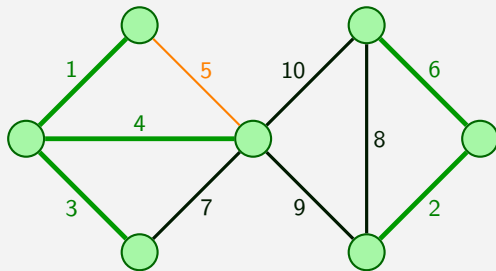
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, 6, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4\}$
- **aktuelle Kante:** hat Gewicht 6,
 - Kante 6 schließt keinen Kreis in T \Rightarrow Kante zu T hinzufügen

Beispiel: KRUSKALS Algorithmus

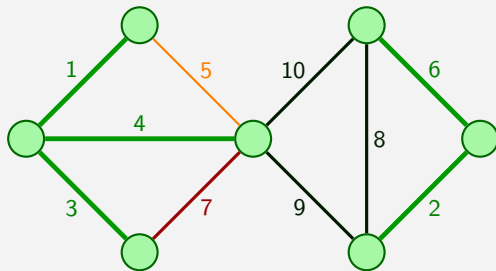
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$

Beispiel: KRUSKALS Algorithmus

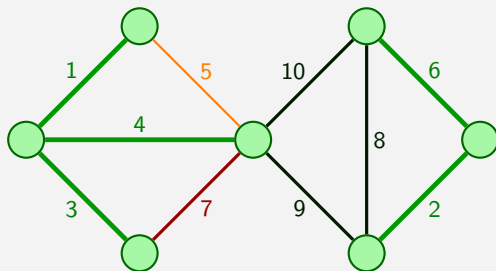
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 7,

Beispiel: KRUSKALS Algorithmus

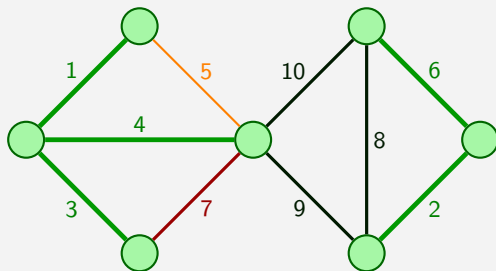
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 7,
 - Kante 7 **schließt einen Kreis** in T

Beispiel: KRUSKALS Algorithmus

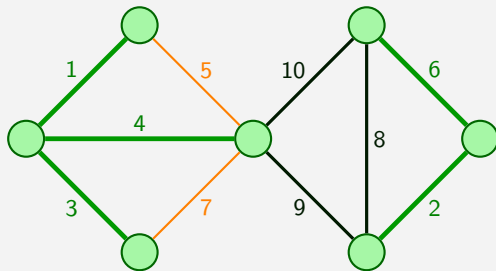
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 7,
 - Kante 7 **schließt einen Kreis** in T \Rightarrow **Kante verwerfen**

Beispiel: KRUSKALS Algorithmus

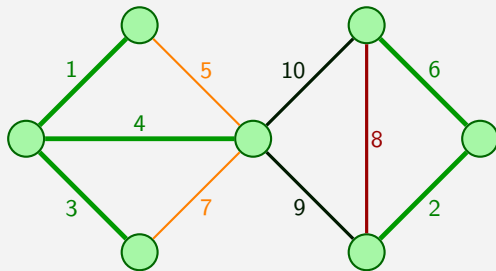
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$

Beispiel: KRUSKALS Algorithmus

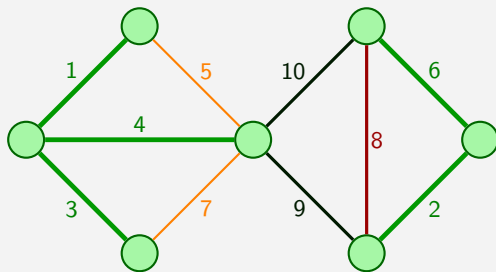
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 8,

Beispiel: KRUSKALS Algorithmus

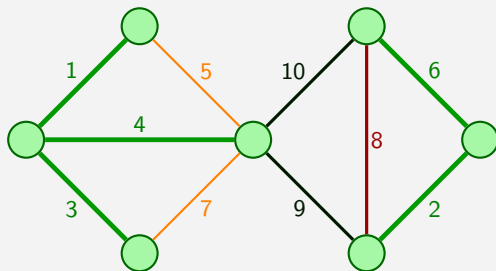
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 8,
 - Kante 8 **schließt einen Kreis** in T

Beispiel: KRUSKALS Algorithmus

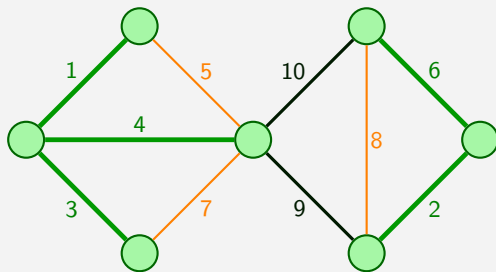
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, 8, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 8,
 - Kante 8 schließt einen Kreis in T \Rightarrow Kante verwerfen

Beispiel: KRUSKALS Algorithmus

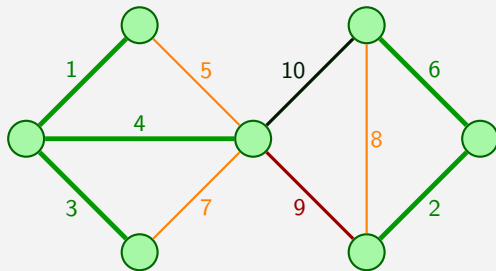
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, ~~8~~, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$

Beispiel: KRUSKALS Algorithmus

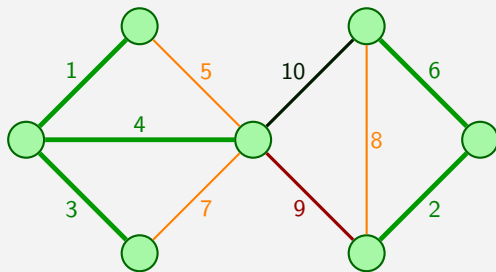
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, ~~8~~, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 9,

Beispiel: KRUSKALS Algorithmus

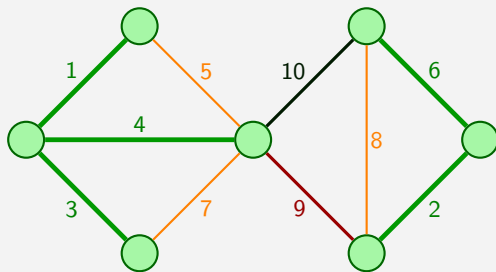
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, ~~7~~, ~~8~~, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 9,
 - Kante 9 schließt keinen Kreis in T

Beispiel: KRUSKALS Algorithmus

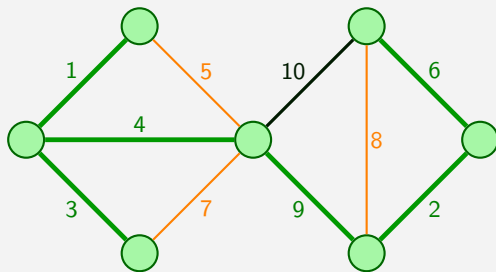
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, ~~7~~, ~~8~~, 9, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6\}$
- **aktuelle Kante:** hat Gewicht 9,
 - Kante 9 schließt keinen Kreis in T \Rightarrow Kante zu T hinzufügen

Beispiel: KRUSKALS Algorithmus

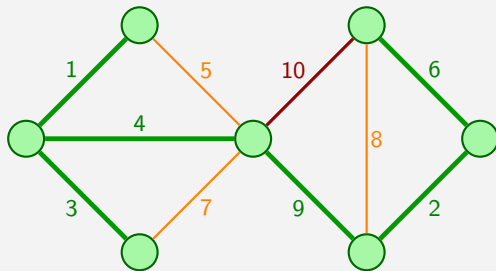
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, ~~8~~, ~~9~~, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6, 9\}$

Beispiel: KRUSKALS Algorithmus

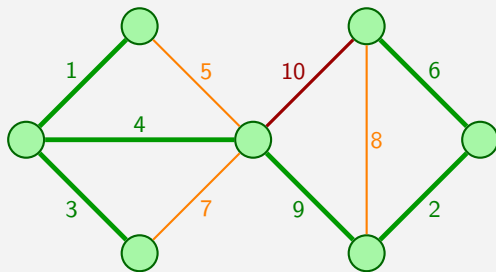
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, ~~8~~, ~~9~~, 10
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6, 9\}$
- **aktuelle Kante:** hat Gewicht 10,

Beispiel: KRUSKALS Algorithmus

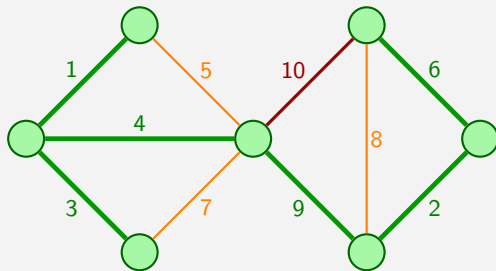
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6, 9\}$
- **aktuelle Kante:** hat Gewicht 10,
 - Kante 10 schließt keinen Kreis in T

Beispiel: KRUSKALS Algorithmus

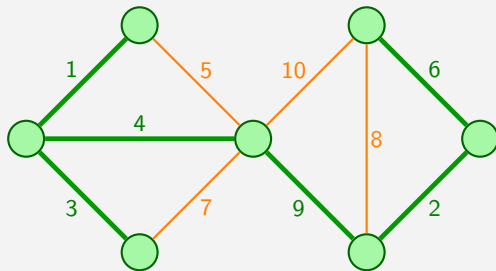
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$
- $V(T) = V(G)$ und $E(T) = \{1, 2, 3, 4, 6, 9\}$
- **aktuelle Kante:** hat Gewicht 10,
 - Kante 10 schließt keinen Kreis in T \Rightarrow Kante zu T hinzufügen

Beispiel: KRUSKALS Algorithmus

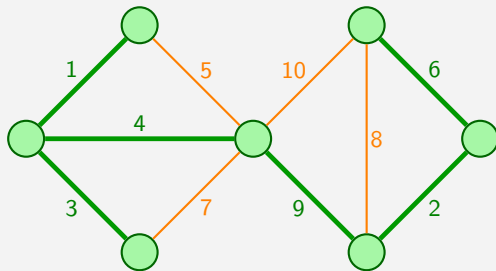
gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, 7, ~~8~~, ~~9~~, ~~10~~
- **Ausgabe:** $V(T) = V(G)$, $E(T) = \{1, 2, 3, 4, 6, 9\}$

Beispiel: KRUSKALS Algorithmus

gewichteter Eingabegraph G (Kanten identifiziert mit ihren Gewichten)



- geordnete Kantenliste: ~~1~~, ~~2~~, ~~3~~, ~~4~~, ~~5~~, ~~6~~, ~~7~~, ~~8~~, ~~9~~, ~~10~~
- **Ausgabe:** $V(T) = V(G)$, $E(T) = \{1, 2, 3, 4, 6, 9\}$ und $w(T) = 25$

Korrektheit von KRUSKALS Algorithmus

Beweis: Sei T die Ausgabe des Algorithmus für $G = (V, E)$ und $w: E \rightarrow \mathbb{R}$ mit $|E| = m$ und $w(e_1) \leq \dots \leq w(e_m)$.

- nach **2** ist $V(T) = V$ und nach Wahl der Kanten in Schritt **a** ist T **kreisfrei**
- T ist **zusammenhängend**:
 - Angenommen T enthält mindestens zwei Komponenten:
 - G zusammenhängend \Rightarrow es gibt eine Kante $e = \{x, y\} \in E(G)$ und zwei Komponenten C_1 und C_2 in T mit $x \in V(C_1)$ und $y \in V(C_2)$
 - e schließt keinen Kreis in T und hätte in **a** eingefügt werden müssen ⚡

$\Rightarrow T$ ist ein Spannbaum von G

Angenommen T ist kein minimaler Spannbaum. Wähle einen minimalen Spannbaum $T^* \neq T$, sodass der kleinste Index i einer Kante

$$e_i \in E(T) \setminus E(T^*)$$

größtmöglich ist.

- e_i schließt einen Kreis $C \subseteq T^* + e_i$
 - $C - e_i$ enthält eine Kante e_j , die nicht in T liegt
 - da T^* alle Kanten von T enthält, die vor e_i in T eingefügt wurden und der Algorithmus e_i aber nicht für e_j gewählt hat, muss gelten $j > i$ und $w(e_j) \geq w(e_i)$
- $\Rightarrow T^{**} = T^* + e_i - e_j := (V, (E(T^*) \cup \{e_i\}) \setminus \{e_j\})$ ist ein Spannbaum mit $w(T^{**}) \leq w(T^*)$
- $\Rightarrow T^{**}$ ist ein minimaler Spannbaum, der der minimalen Wahl des Index i mit $e_i \in E(T) \setminus E(T^*)$ widerspricht □

Laufzeit von KRUSKALS Algorithmus

- Sortieren der Kanten (Schritt **1**) kann in einer Laufzeit proportional in $|E| \log |V|$ implementiert werden
- der Kreistest (Schritt **a**) kann (elementar) über alle Kanten in einer Gesamtlaufzeit proportional in $|E| + |V| \log |V|$ implementiert werden

(Wie?)

⇒ Gesamtlaufzeit proportional zu $(|V| + |E|) \log |V|$

Bemerkungen:

- Schnellster MST-Algorithmus mit bekannter Laufzeit hat eine Laufzeit proportional zu $\alpha(|V|, |E|)(|V| + |E|)$, wobei $\alpha(\cdot, \cdot)$ die inverse ACKERMANN-Funktion ist
- $\alpha(|V|, |E|) \rightarrow \infty$ für $|V|, |E| \rightarrow \infty$, allerdings extrem langsam

Wichtige offene Frage

Gibt es einen MST-Algorithmus mit Laufzeit linear in $|V| + |E|$, d. h. mit Laufzeit

$$C \cdot (|V| + |E|)$$

für eine Konstante $C > 0$ unabhängig von $G = (V, E)$.