# *Vorlesungstermin 4:*

# *Graphenalgorithmen II*

Markus Püschel
David Steurer

talks2.dsteurer.org/graphenalgorithmen2.pdf

Algorithmen und Datenstrukturen, Herbstsemester 2018, ETH Zürich

## *Plan für heute*

**allgemeines Thema:** Graphen durchlaufen

## *Plan für heute*

**allgemeines Thema:** Graphen durchlaufen

**zwei Ansätze:** Tiefensuche und Breitensuche

## *Plan für heute*

**allgemeines Thema:** Graphen durchlaufen

**zwei Ansätze:** Tiefensuche und Breitensuche

*wichtige Bausteine um Graphenprobleme zu lösen*

## *Plan für heute*

**allgemeines Thema:** Graphen durchlaufen

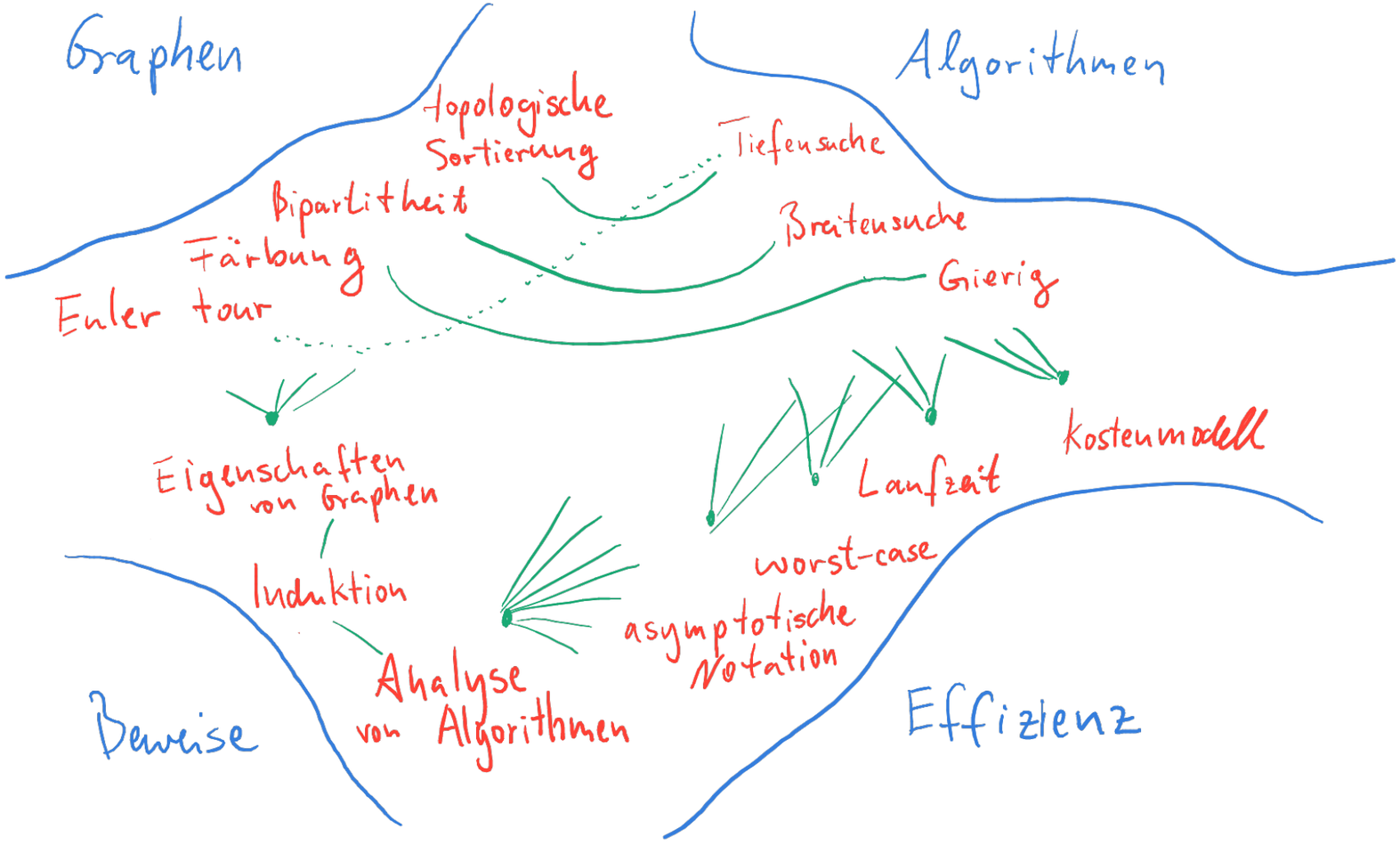
**zwei Ansätze:** Tiefensuche und Breitensuche

*wichtige Bausteine um Graphenprobleme zu lösen*

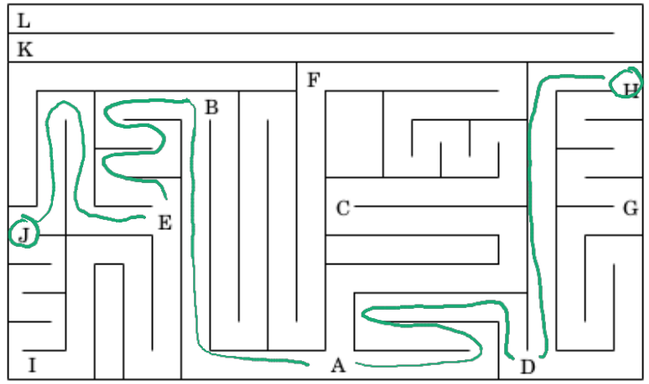
**Beispiele:** (heute)

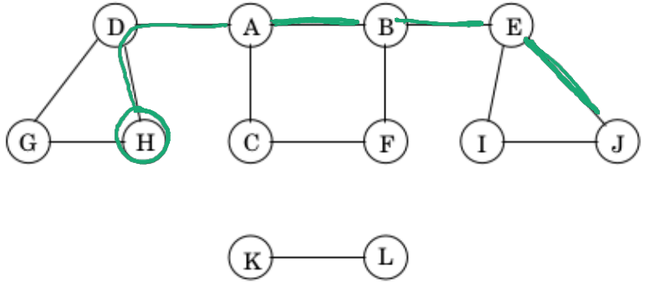
- *topologische Sortierung* (Tiefensuche)
- *kürzeste Wege* (Breitensuche)

# Übersicht: Vorlesung bis heute



# *Irrgärten erkunden*




mögliche Strategie:

- markiere besuchte Stellen mit *Kreide*
- verwende *Wollknäuel*, um zu Verzweigungen zurückzufinden, die noch nicht vollständig erkundet sind

als Graphproblem: gegeben ein Graph (z.B. als Adjazenzliste) und ein Startknoten $v$, welche Knoten sind von $v$ erreichbar?

## *Irrgärten erkunden*

mögliche Strategie:

- markiere besuchte Stellen mit *Kreide*
- verwende *Wollknäuel*, um zu Verzweigungen zurückzufinden, die noch nicht vollständig erkundet sind

als Graphproblem: gegeben ein Graph (z.B. als Adjazenzliste) und ein Startknoten $v$, welche Knoten sind von $v$ erreichbar?

*statt Kreide und Wollknäuel verwenden wir* ***Datenstrukturen***

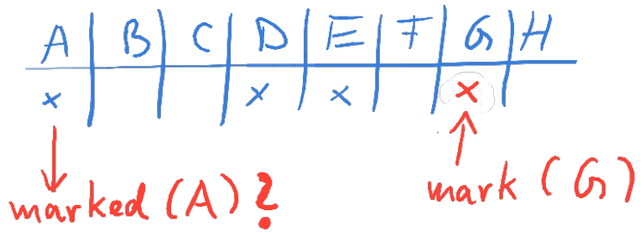
## *Irrgärten erkunden*

**mögliche Strategie:**

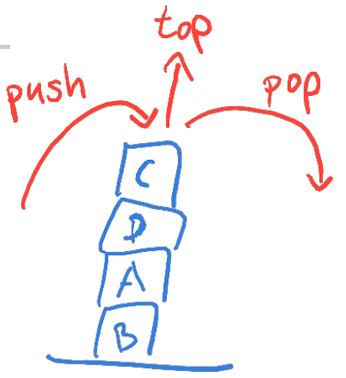
- markiere besuchte Stellen mit *Kreide*
- verwende *Wollknäuel*, um zu Verzweigungen zurückzufinden, die noch nicht vollständig erkundet sind

statt Kreide: **Tabelle** (en: array)
der Länge $n$ (Anzahl Knoten)

*Operationen:* (1) einen Knoten markieren, (2) testen ob ein Knoten markiert ist
*Laufzeiten:* jeweils $O(1)$




statt Wollknäuel: **Stapel** (en: stack)
*Operationen:* (1) einen Knoten oben auf Stapel legen, (2) auf obersten Knoten zugreifen, (3) obersten Knoten entfernen
*Laufzeiten:* jeweils $O(1)$



Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$

## *Tiefensuche: Beispiel*

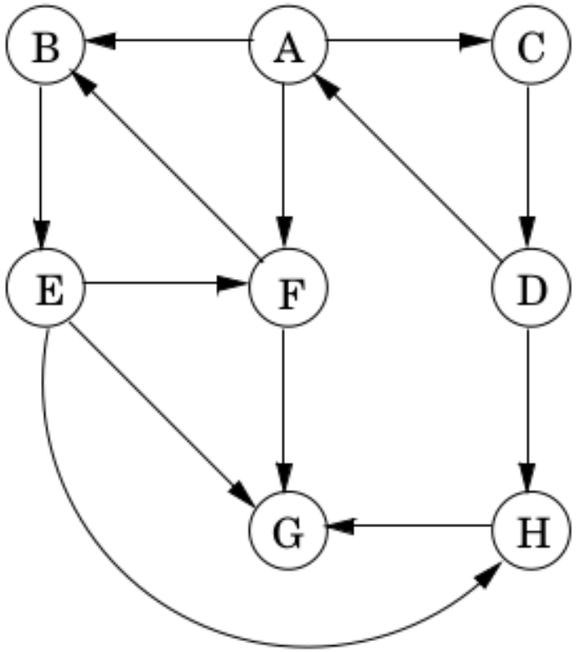
Startknoten: *A*

Iteration: 0

besuchte Knoten:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|

Stapel:




## *Tiefensuche: Beispiel*

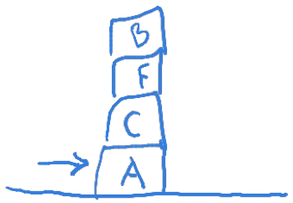
Startknoten: *A*

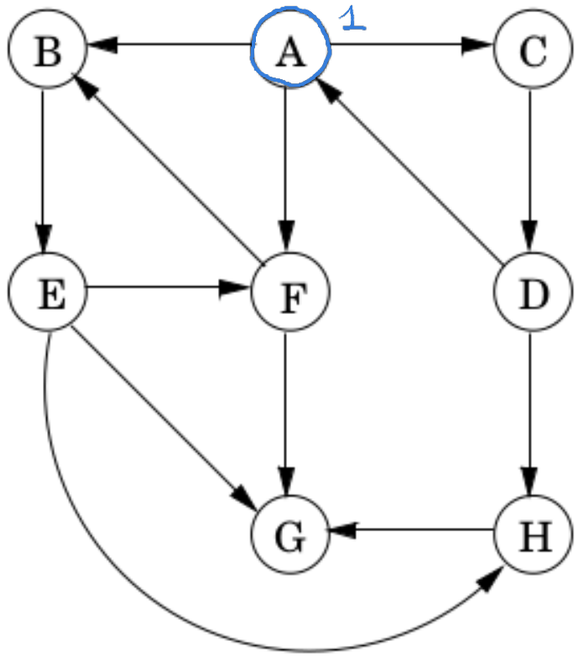
Iteration: 1

besuchte Knoten:

| ~~A~~ | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|

Stapel:



## *Tiefensuche: Beispiel*

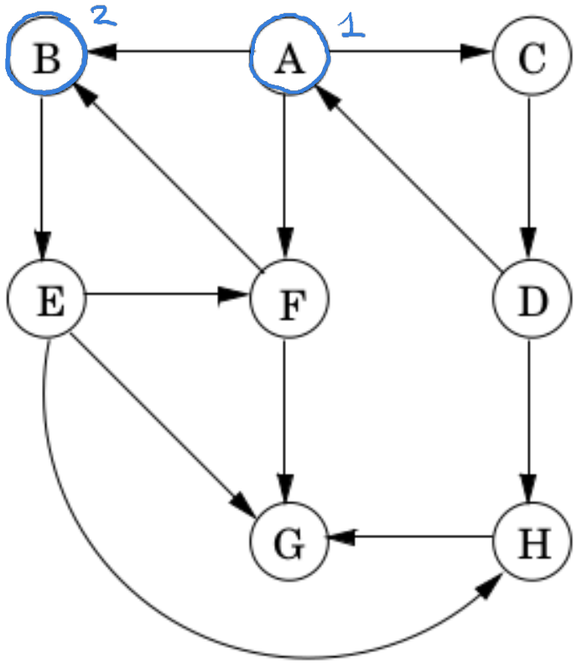
Startknoten: *A*

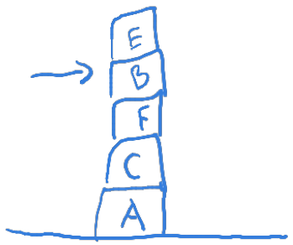
Iteration: 2

besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|

Stapel:



## *Tiefensuche: Beispiel*

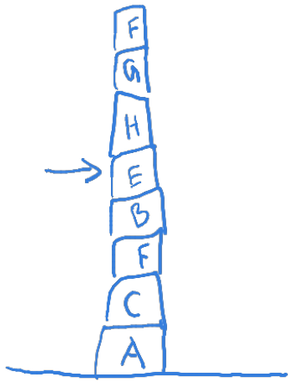
Startknoten: *A*

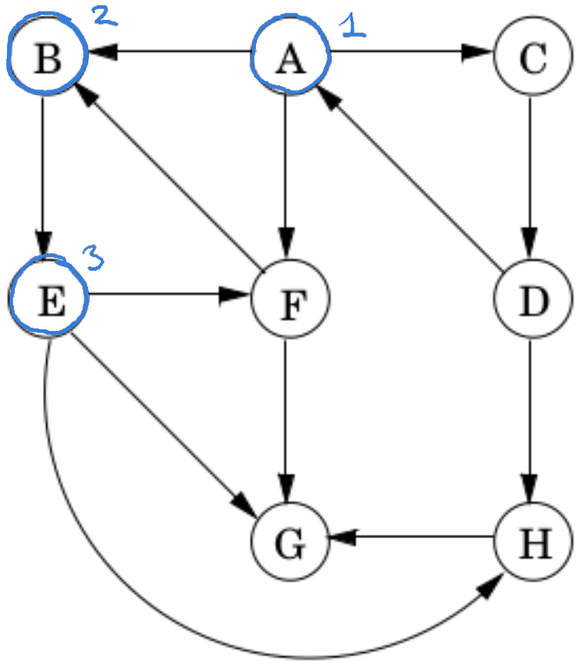
Iteration: 3

besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | F | G | H |
|---|---|---|---|---|---|---|---|

Stapel:



## *Tiefensuche: Beispiel*

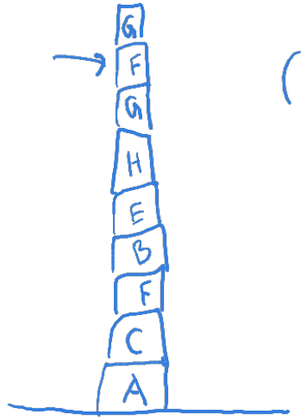
Startknoten: *A*

Iteration: 4

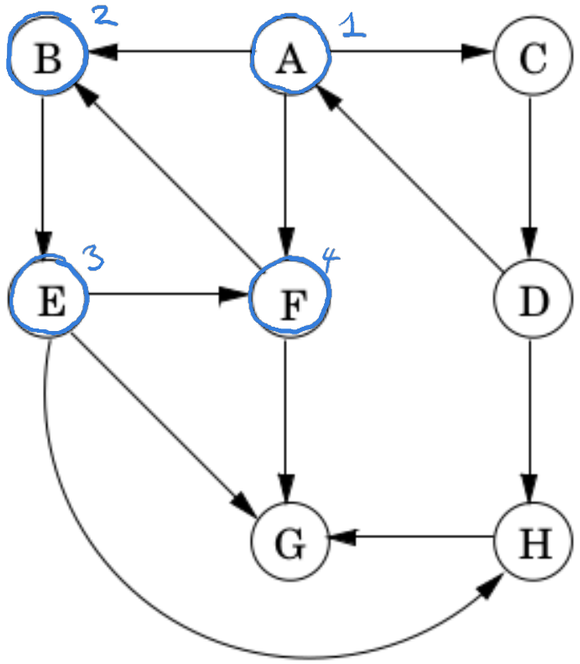
besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | ~~F~~ | G | H |
|---|---|---|---|---|---|---|---|

Stapel:




( Nachfolger B von F kommt nicht auf Stapel, da schon besucht )



## *Tiefensuche: Beispiel*

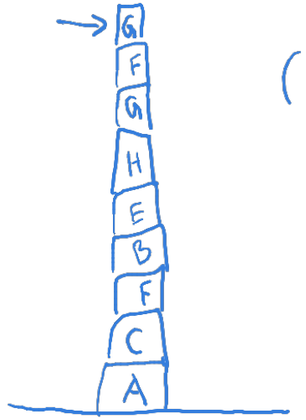
Startknoten: *A*

Iteration: 5

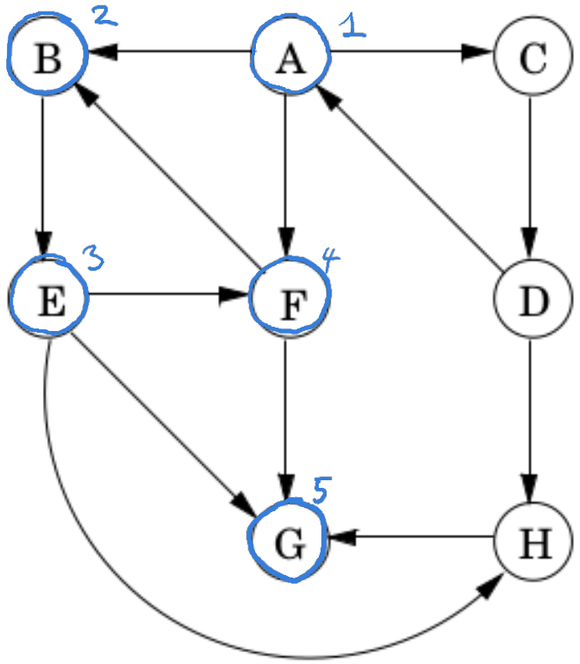
besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | ~~F~~ | ~~G~~ | H |
|---|---|---|---|---|---|---|---|

Stapel:




( Nachfolger B von F kommt nicht auf Stapel, da schon besucht )



## *Tiefensuche: Beispiel*

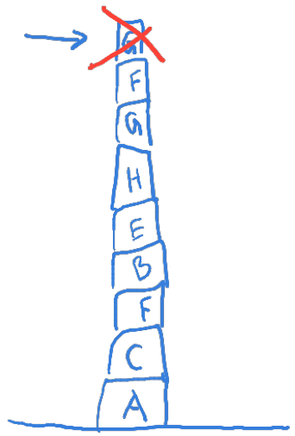
Startknoten: *A*

Iteration: 6

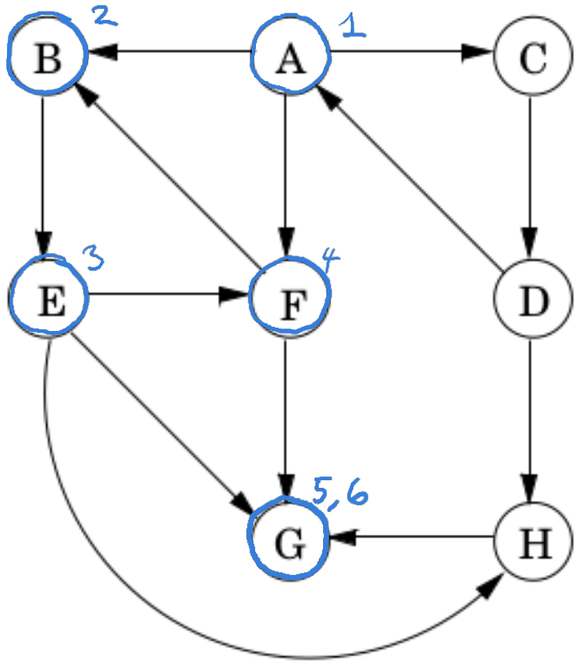
besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | ~~F~~ | ~~G~~ | H |
|---|---|---|---|---|---|---|---|

Stapel:




(G zum ersten Mal vom Stapel entfernt)



## *Tiefensuche: Beispiel*

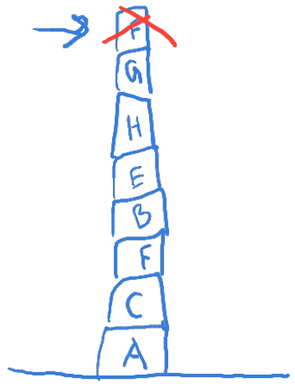
Startknoten: *A*

Iteration: 7

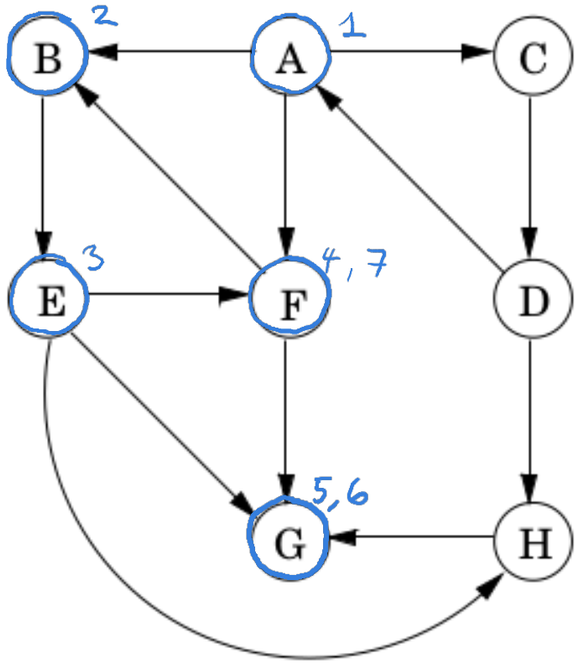
besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | ~~F~~ | ~~G~~ | H |
|---|---|---|---|---|---|---|---|

Stapel:




( F zum ersten Mal vom Stapel entfernt )



## *Tiefensuche: Beispiel*

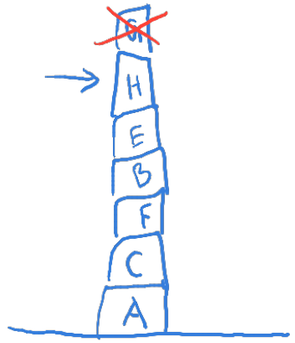
Startknoten: *A*

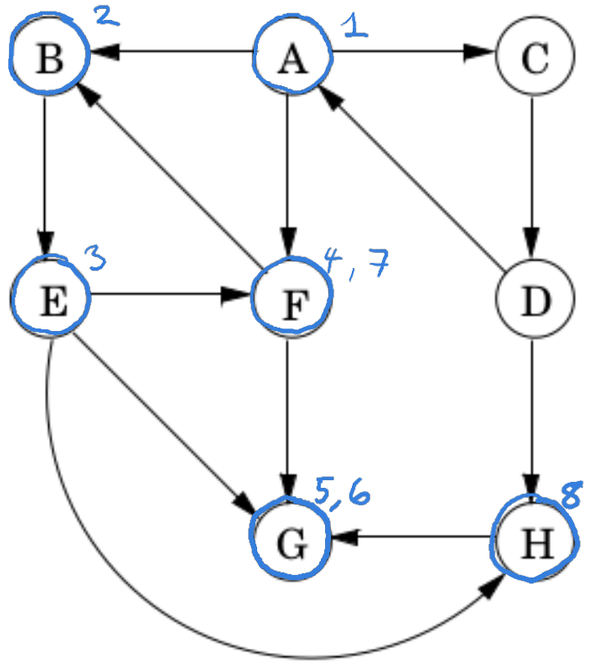
Iteration: 8

besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ |
|---|---|---|---|---|---|---|---|

Stapel:




(Iterationszähler nicht erhöht
für wiederholtes Entfernen von G vom Stack)

Nachfolger G von H nicht auf Stack gelegt,
da schon besucht

## *Tiefensuche: Beispiel*

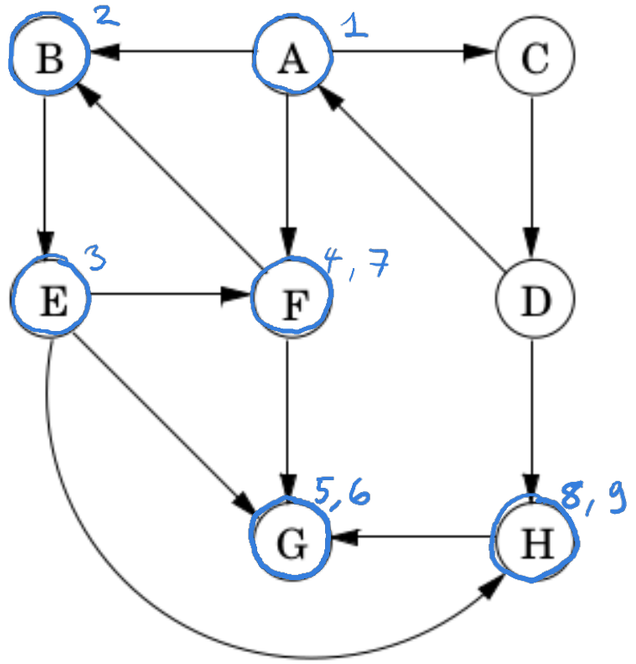
Startknoten: *A*

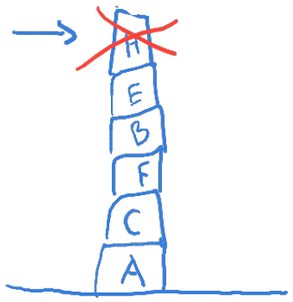
Iteration: 9

besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ |
|---|---|---|---|---|---|---|---|

Stapel:




( H zum ersten Mal vom Stack entfernt )

## *Tiefensuche: Beispiel*

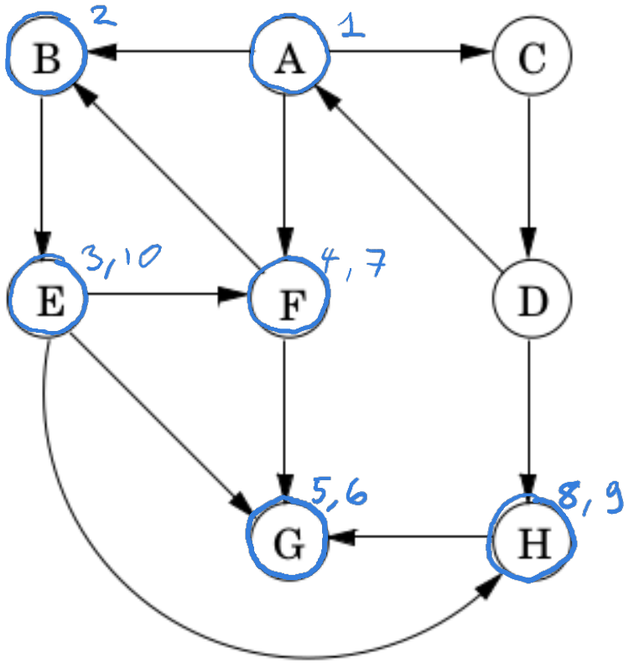
Startknoten: *A*

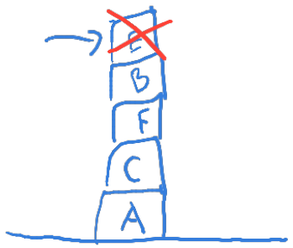
Iteration: 10

besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ |
|---|---|---|---|---|---|---|---|

Stapel:




( E zum ersten Mal vom Stack entfernt )

## *Tiefensuche: Beispiel*

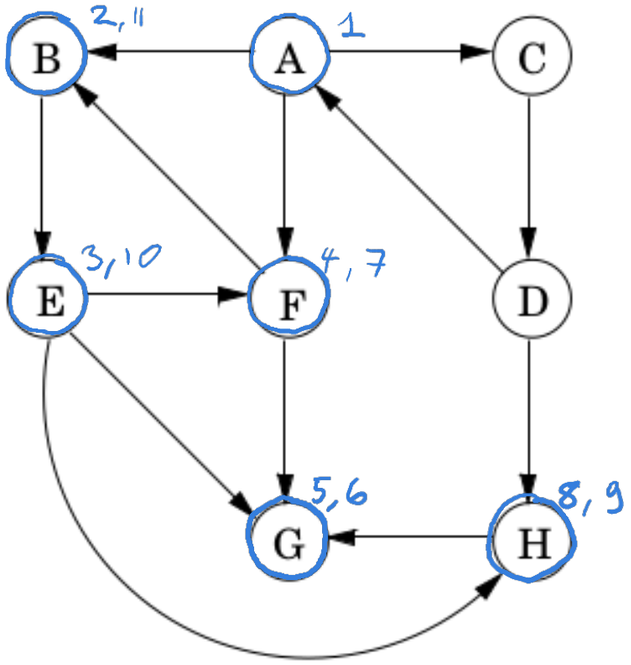
Startknoten: *A*

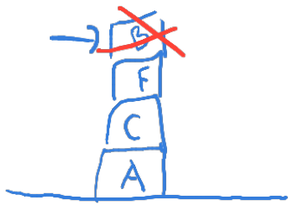
Iteration: 11

besuchte Knoten:

| ~~A~~ | ~~B~~ | C | D | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ |
|---|---|---|---|---|---|---|---|

Stapel:




( B zum ersten Mal vom Stack entfernt )

## *Tiefensuche: Beispiel*

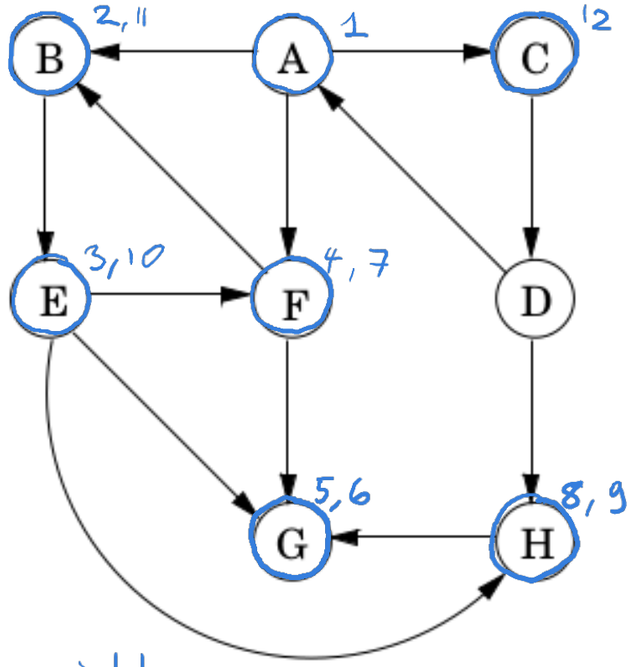
Startknoten: *A*

Iteration: 12

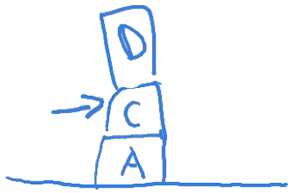
besuchte Knoten:

| ~~A~~ | ~~B~~ | ~~C~~ | D | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ |
|---|---|---|---|---|---|---|---|

Stapel:




(Iterationszähler nicht erhöht beim wiederholten Entfernen von F)



## *Tiefensuche: Beispiel*

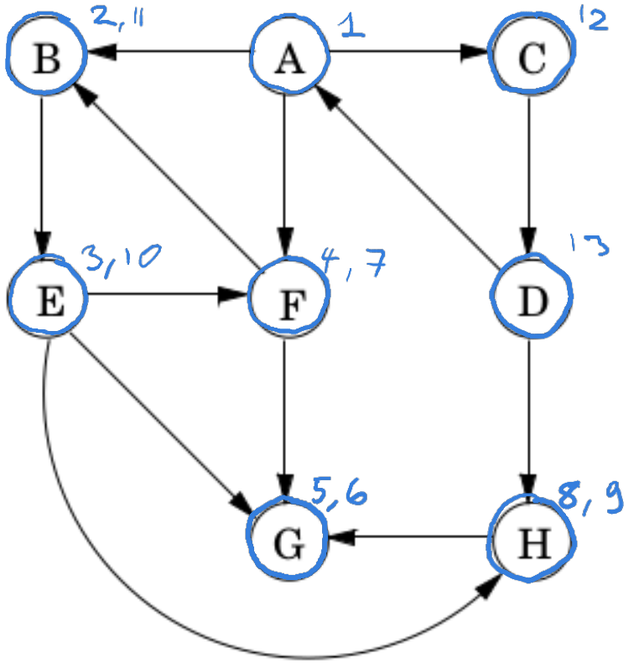
Startknoten: *A*

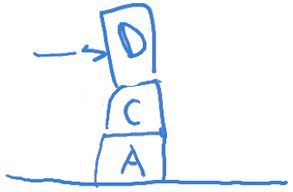
Iteration: 13

besuchte Knoten:

| ~~A~~ | ~~B~~ | ~~C~~ | ~~D~~ | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ |
|---|---|---|---|---|---|---|---|

Stapel:




( Nachfolger A, H von D kommen nicht auf Stapel da schon besucht)

## *Tiefensuche: Beispiel*

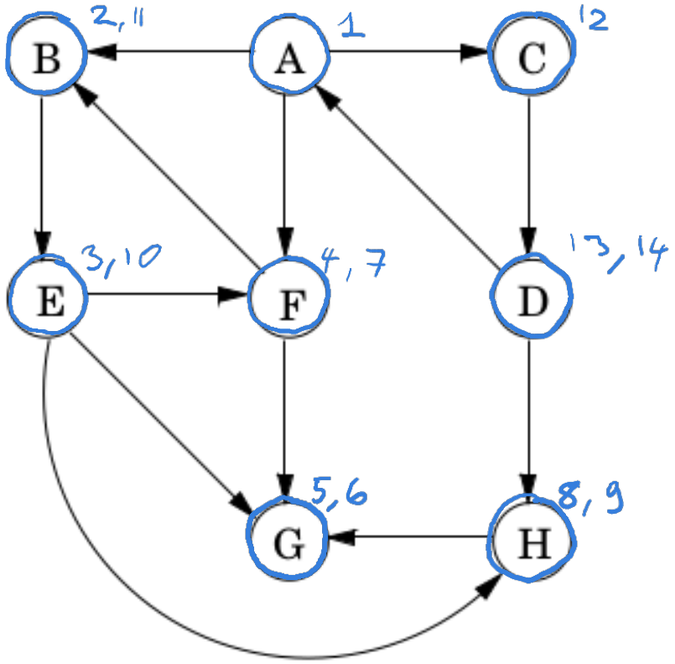
Startknoten: *A*

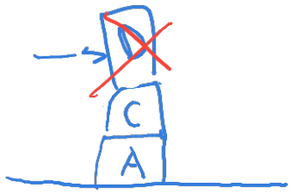
Iteration: 14

besuchte Knoten:

| ~~A~~ | ~~B~~ | ~~C~~ | ~~D~~ | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ |
|---|---|---|---|---|---|---|---|

Stapel:



## *Tiefensuche: Beispiel*

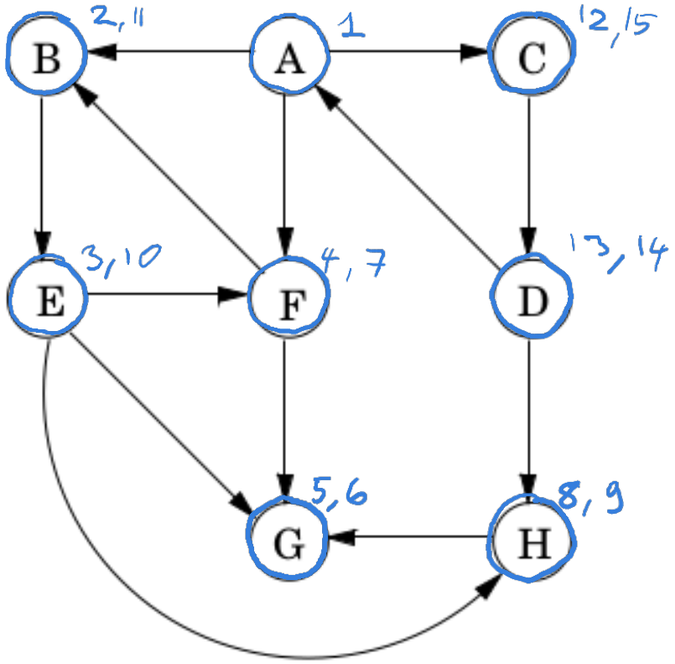
Startknoten: *A*

Iteration: 15

besuchte Knoten:

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|

Stapel:




## *Tiefensuche: Beispiel*

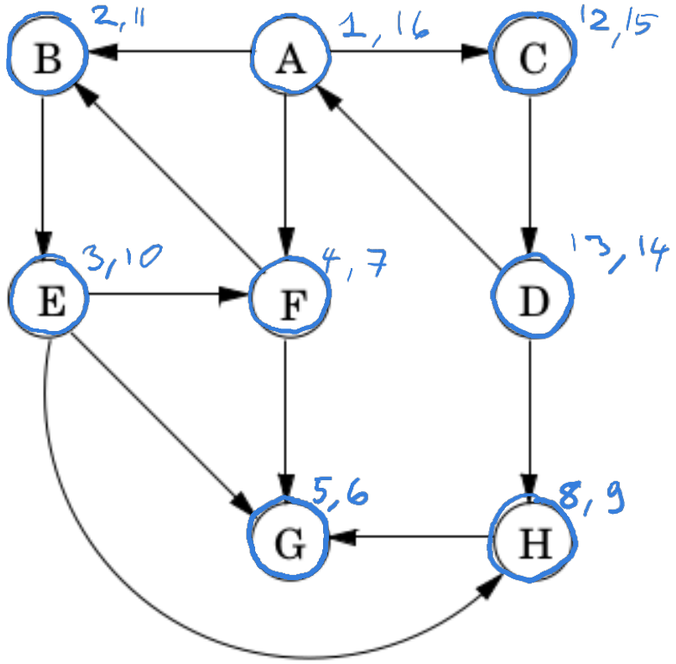
Startknoten: *A*

Iteration: 16

besuchte Knoten:

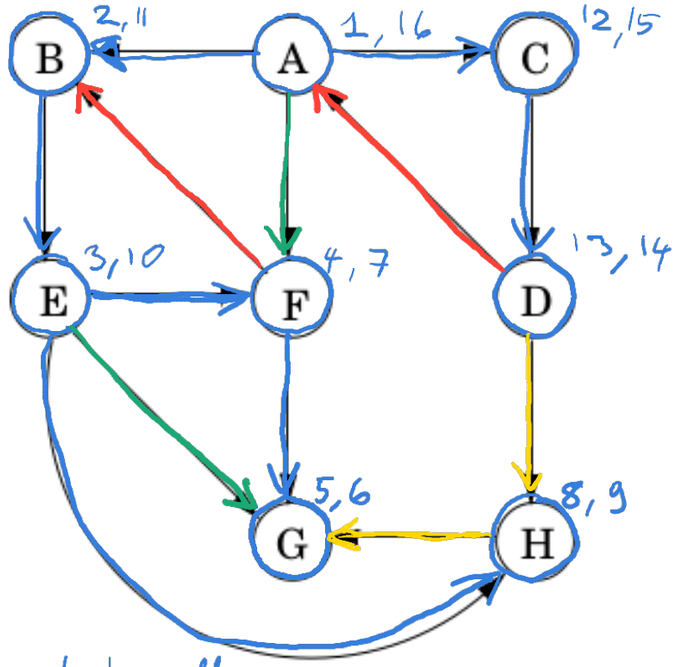
~~A~~ ~~B~~ ~~C~~ ~~D~~ ~~E~~ ~~F~~ ~~G~~ ~~H~~

Stapel:

# *Tiefensuche: Beispiel*

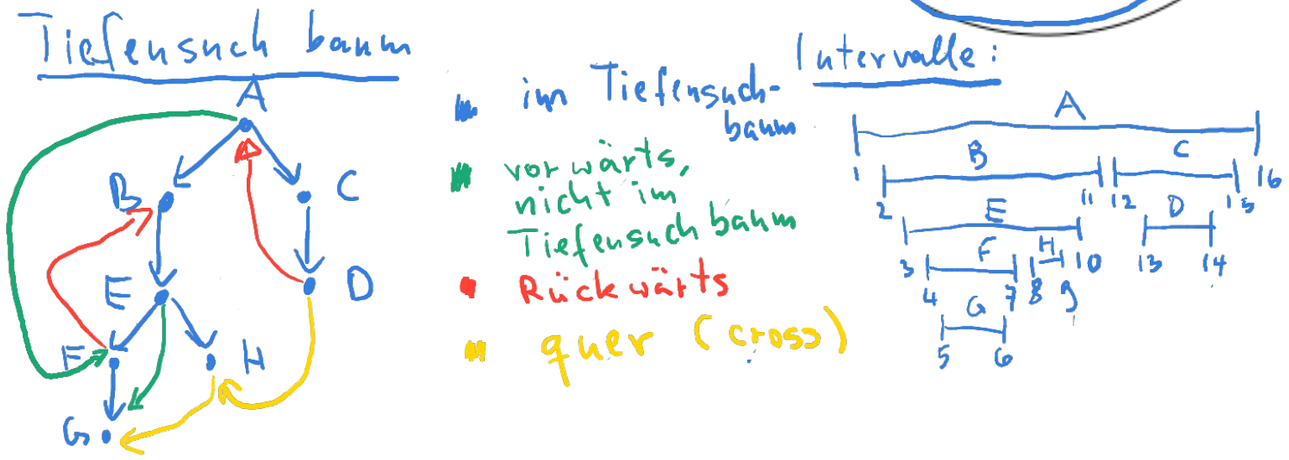
Startknoten: *A*




besuchte Knoten:

| ~~A~~ | ~~B~~ | ~~C~~ | ~~D~~ | ~~E~~ | ~~F~~ | ~~G~~ | ~~H~~ |
|---|---|---|---|---|---|---|---|

Stapel:



Graph $G = (V, E)$ und Knoten $v \in V$

**Tiefensuche$(G, v)$:** (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$

wir verwenden "pseudocode" für kürzere Beschreibung

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$

**Behauptung:** falls zu Beginn keine Knoten als besucht markiert sind, dann besucht $\text{Tiefensuche}(G, v)$ genau die Knoten, die von $v$ erreichbar sind

**Beweisidee:** da wir mit $v$ beginnen und dann nur Kanten entlang gehen, ist jeder besuchte Knoten von $v$ erreichbar

es bleibt zu zeigen, dass jeder erreichbare $u \in V$ besucht wird

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$

**Behauptung:** falls zu Beginn keine Knoten als besucht markiert sind, dann besucht Tiefensuche$(G, v)$ genau die Knoten, die von $v$ erreichbar sind

**Beweisidee:**

es bleibt zu zeigen, dass jeder erreichbare $u \in V$ besucht wird

betrachte Weg $w_0, \ldots, w_\ell$ von $w_0 = v$ nach $w_\ell = u$

$w_0 = v$ wird in der ersten Iteration besucht

falls $w_i$ besucht wird, werden auch alle Nachfolger von $w_i$ besucht, insbesondere $w_{i+1}$ (wenn $0 \leqslant i < \ell$)

*per Induktion (über $i$):* alle Knoten $w_0, \ldots, w_\ell$ werden besucht

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$

**Korollar:** falls schon eine Knotenmenge $X \subseteq V$ als besucht markiert ist, dann besucht $\text{Tiefensuche}(G, v)$ genau die Knoten, die noch von $v$ erreichbar sind mittels Wegen, die $X$ vermeiden

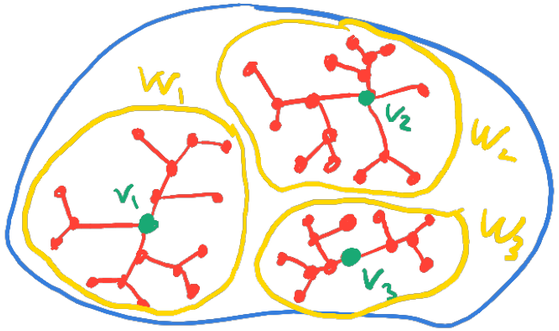
**Anwendung:** *Zusammenhangskomponenten* ($G$ ungerichtet)

führe Tiefensuche solange für noch unbesuchte Knoten aus bis alle Knoten besucht sind

jede Tiefensuche besucht dabei genau eine Z.-komponente

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$




**Anwendung:** *Zusammenhangskomponenten* ($G$ ungerichtet)

führe Tiefensuche solange für noch unbesuchte Knoten aus bis alle Knoten besucht sind

jede Tiefensuche besucht dabei genau eine Z.-komponente

Tiefensuche$(G)$:

1. markiere alle Knoten $v \in V$ als nicht besucht
2. `for` $v \in V$ `do`
   1. `if` $v$ noch nicht besucht `then` Tiefensuche$(G, v)$

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$

**Laufzeit** $\leqslant O(N_{\text{push}} + N_{\text{pop}} + N_{\text{top}} + N_{\text{markier}})$

(wenn Graph als Adjazenzliste gegeben ist)

**wobei:** $N_{\text{bla}} =$ Anzahl von bla Operationen (für push Operationen zählen wir die *Anzahl der Knoten*, die wir auf den Stapel legen)

es gilt $N_{\text{pop}} = 1 + N_{\text{push}}$ und $N_{\text{top}} = N_{\text{pop}} + N_{\text{markier}}$

**daher:** $N_{\text{push}} + N_{\text{pop}} + N_{\text{top}} + N_{\text{markier}} \leqslant O(N_{\text{markier}} + N_{\text{push}})$

sei $W =$ besuchte Knoten während Tiefensuche$(G, v)$

dann gilt $N_{\text{markier}} = |W|$ und $N_{\text{push}} \leqslant \sum_{w \in W} \deg_G^+(w)$
$\rightsquigarrow$ Laufzeit $\leqslant O\big(|W| + \sum_{w \in W} \deg_G^+(w)\big)$ für Tiefensuche$(G, v)$

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \mathrm{top}(S)$
   2. `if` $w$ schon besucht `then` $\mathrm{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\mathrm{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$

$\rightsquigarrow$ Laufzeit $\leqslant O\big(|W| + \sum_{w \in W} \deg_G^+(w)\big)$ für $\mathrm{Tiefensuche}(G, v)$

**Theorem:** $\mathrm{Tiefensuche}(G)$ hat Laufzeit $\leqslant O(|V| + |E|)$

jeder Knoten wird nur einmal besucht; $\sum_{w \in V} \deg_G^+(w) = |E|$

Angen. Tiefensuche $(G, v_1)$ besucht $W_1$

$\vdots$

Tiefensuche $(G, v_k)$ besucht $W_k$

Dann: $W_1, \dots, W_k$ ist Partition von $V$

$|W_1| + \dots + |W_k| = |V|$ und $\sum_{i=1}^{k} \deg^+(W_i) = |E|$ (ger. Graph)

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$

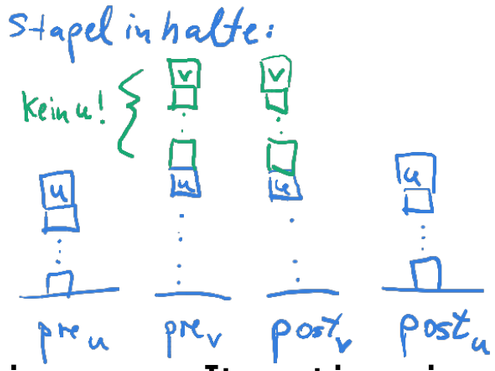
tieferes Verständnis: zu jedem Knoten $v \in V$ können wir ein Intervall $I_v \subseteq \{1, \ldots, 2n\}$ zuordnen (wobei $n = |V|$)

Intervall $I_v = \{\text{pre}_v, \ldots, \text{post}_v\}$ wobei $\text{pre}_v =$ Iteration, in der $v$ als besucht markiert wird und $\text{post}_v =$ Iteration, in der $v$ zum ersten Mal vom Stapel entfernt wird

**Lemma:** $\forall u, v \in V$. $I_u$ und $I_v$ sind disjunkt oder ein Interval ist im anderen komplett enthalten

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$




Intervall $I_v = \{\text{pre}_v, \ldots, \text{post}_v\}$ wobei $\text{pre}_v =$ Iteration, in der $v$ als besucht markiert wird und $\text{post}_v =$ Iteration, in der $v$ zum ersten Mal vom Stapel entfernt wird

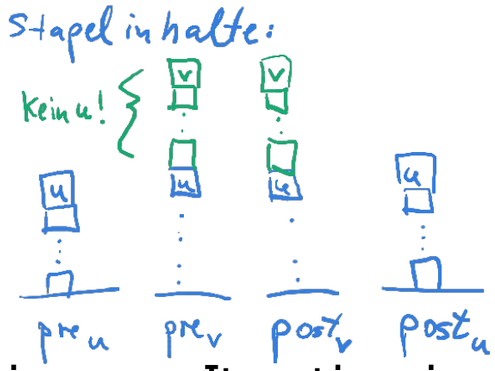
**Lemma:** $\forall u, v \in V$. $I_u$ und $I_v$ sind disjunkt oder ein Interval ist im anderen komplett enthalten

**Beweisidee:** falls $\text{pre}_u < \text{pre}_v < \text{post}_u$, befindet sich $v$ über $u$ im Stapel wenn $v$ besucht wird, so dass $\text{post}_v < \text{post}_u$

**Typ** einer Kante $(u, v) \in E$: *vorwärts* falls $I_u \supseteq I_v$, *rückwärts* falls $I_u \subseteq I_v$, *quer* falls $I_u \cap I_v = \emptyset$

Tiefensuche$(G, v)$: (en. depth-first search)

1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$




Intervall $I_v = \{\text{pre}_v, \ldots, \text{post}_v\}$ wobei $\text{pre}_v =$ Iteration, in der $v$ als besucht markiert wird und $\text{post}_v =$ Iteration, in der $v$ zum ersten Mal vom Stapel entfernt wird

**Lemma:** $\forall u, v \in V$ . $I_u$ und $I_v$ sind disjunkt oder ein Interval ist im anderen komplett enthalten

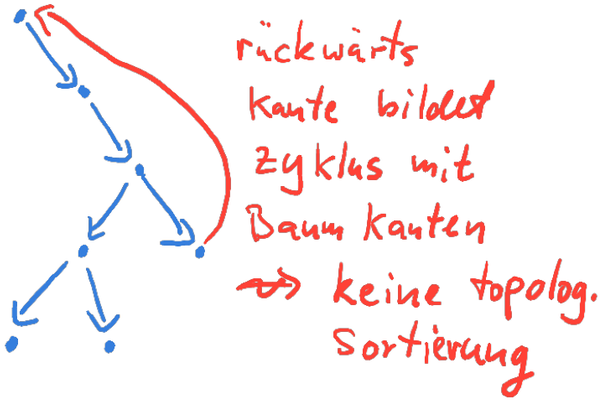
**Typ** einer Kante $(u, v) \in E$: *vorwärts* falls $I_u \supseteq I_v$, *rückwärts* falls $I_u \subseteq I_v$, *quer* falls $I_u \cap I_v = \emptyset$

**Lemma:** für jede Querkante $(u, v) \in E$ liegt $I_v$ vor $I_u$

diese Intervalle bilden einen Wald, genannt *Tiefensuchwald*

Tiefensuche$(G, v)$: (en. depth-first search)

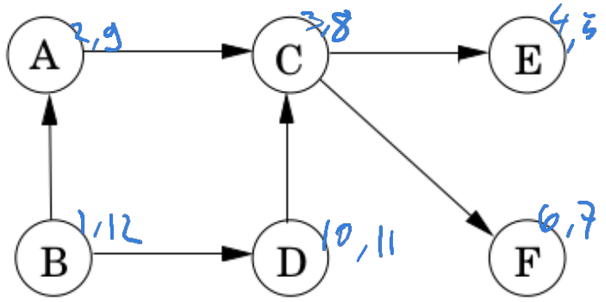
1. $S \leftarrow$ Stapel mit Knoten $v$
2. `while` $S$ nicht leer `do`
   1. $w \leftarrow \text{top}(S)$
   2. `if` $w$ schon besucht `then` $\text{pop}(S)$
   3. `if` $w$ noch nicht besucht `then` markiere $w$ als besucht und $\text{push}(S, \{u \in N_G^+(w) \mid u \text{ noch nicht besucht}\})$




*topologische Sortierung durch Tiefensuche*

**Beobachtung:** für einen azyklischen Graph kann es keine Rückwärtskante bei der Tiefensuche geben

**Theorem:** wir erhalten eine topologische Sortierung, wenn wir die Knoten nach den Zahlen $\text{post}_v$ absteigend sortieren




topologische Sortierung

| knoten | B | D | A | C | F | E |
|---|---|---|---|---|---|---|
| post | 12 | 11 | 9 | 8 | 7 | 5 |

für Vorwärts- und Querkanten $(u,v)$ gilt immer $\text{post}_u > \text{post}_v$

## *Kürzeste Wege*

oft: nicht nur irgendeinen Weg von $u$ nach $v$

sondern: einen *kürzesten Weg*

Anwendung: Routenplanung (z.B. Google Maps) und mehr

Länge eines Wegs = Anzahl der Kanten

Distanz $d(u, v)$ = Länge des kürzesten Wegs von $u$ nach $v$

($d(u, v) = \infty$ falls kein solcher Weg existiert; $d(u, u) = 0$)

später: Kanten können unterschiedliche Längen haben

## *Kürzeste Wege*

Länge eines Wegs = Anzahl der Kanten

Distanz $d(u, v)$ = Länge des kürzesten Wegs von $u$ nach $v$

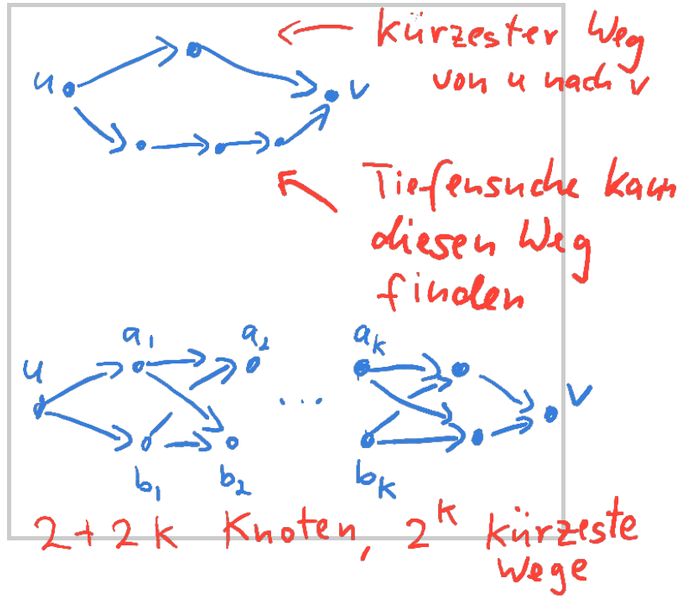
($d(u, v) = \infty$ falls kein solcher Weg existiert; $d(u, u) = 0$)

später: Kanten können unterschiedliche Längen haben

Beobachtungen: Falls $d(u, v) < \infty$, dann gilt $d(u, v) \leqslant n - 1$

Tiefensuche findet Pfad aber nicht unbedingt den kürzesten

es kann exponentiell viele Pfade von $u$ nach $v$ geben

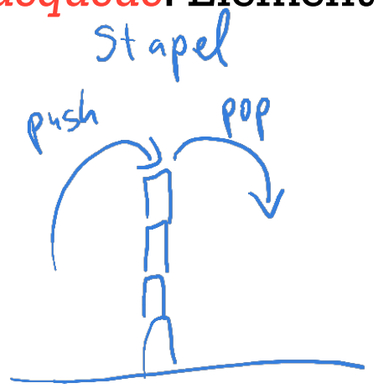


## *Datenstrukturen für Graphtraversierung*

um kürzeste Wege zu finden müssen wir nur den Stapel bei der Tiefensuche mit einer anderen Datenstruktur ersetzen

**Stapel:** (en. stack) "LIFO = last in, first out"
*push:* Objekt oben auf den Stapel legen
*pop:* oberstes Element vom Stapel entfernen

**Schlange:** (en. queue) "FIFO = first in, first out"
*enqueue:* Objekt am Ende der Schlange hinzufügen
*dequeue*: Element am Anfang der Schlange entfernen

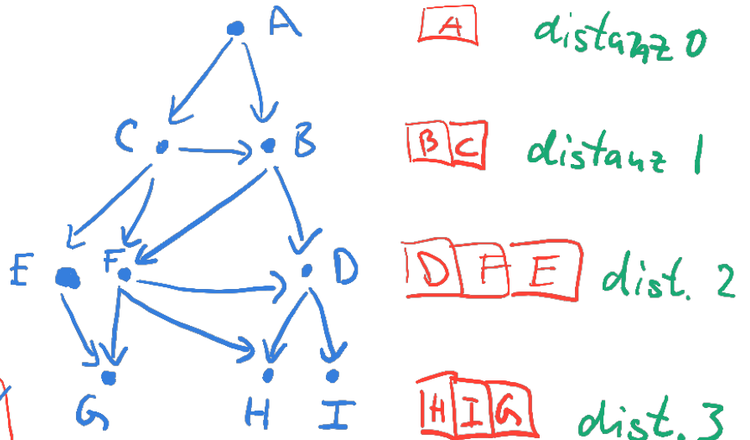



**Breitensuche$(G, s)$ :** (en. breadth-first search)

1.
2. $Q \leftarrow$ Schlange mit Knoten $s$
3. `while` $Q$ nicht leer `do`
   1. $u \leftarrow \text{dequeue}(Q)$
   2. `for` $(u, v) \in E$ `do`
      `if` $v$ nicht markiert `then`
      markiere $v$ und $\text{enqueue}(Q, v)$

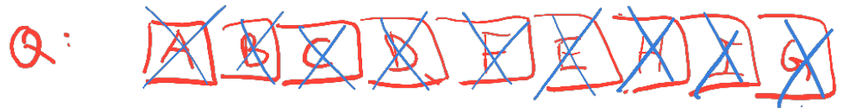
**Breitensuche**$(G, s)$ : (en. breadth-first search)

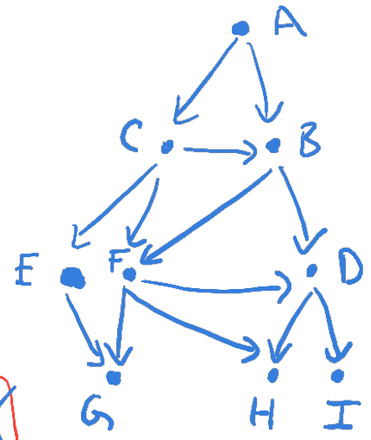
1.
2. $Q \leftarrow$ Schlange mit Knoten $s$
3. `while` $Q$ nicht leer `do`
   1. $u \leftarrow \text{dequeue}(Q)$
   2. `for` $(u, v) \in E$ `do`
      `if` $v$ nicht markiert `then`
      markiere $v$ und $\text{enqueue}(Q, v)$

**Breitensuche**$(G, s)$ : (en. breadth-first search)

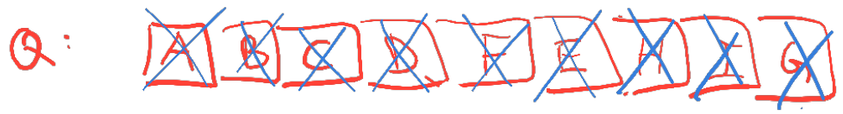
1. $\text{dist}[u] \leftarrow \infty$ für alle $u \in V \setminus \{s\}$ und $\text{dist}[s] \leftarrow 0$
2. $Q \leftarrow$ Schlange mit Knoten $s$
3. `while` $Q$ nicht leer `do`
   1. $u \leftarrow \text{dequeue}(Q)$
   2. `for` $(u, v) \in E$ `do`
      `if` $\text{dist}[v] = \infty$ `then`
      $\text{dist}[v] \leftarrow \text{dist}[u] + 1$ und $\text{enqueue}(Q, v)$

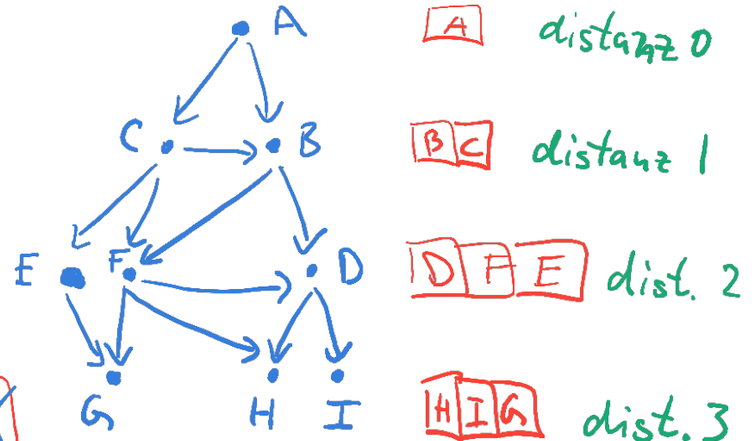



**Breitensuche**$(G, s)$ : (en. breadth-first search)

1. $\mathrm{dist}[u] \leftarrow \infty$ für alle $u \in V \setminus \{s\}$ und $\mathrm{dist}[s] \leftarrow 0$
2. $Q \leftarrow$ Schlange mit Knoten $s$
3. `while` $Q$ nicht leer `do`
   1. $u \leftarrow \mathrm{dequeue}(Q)$
   2. `for` $(u, v) \in E$ `do`
      `if` $\mathrm{dist}[v] = \infty$ `then`
      $\mathrm{dist}[v] \leftarrow \mathrm{dist}[u] + 1$ und $\mathrm{enqueue}(Q, v)$



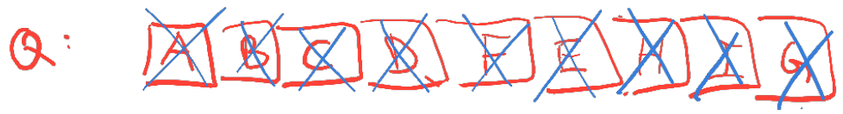

---

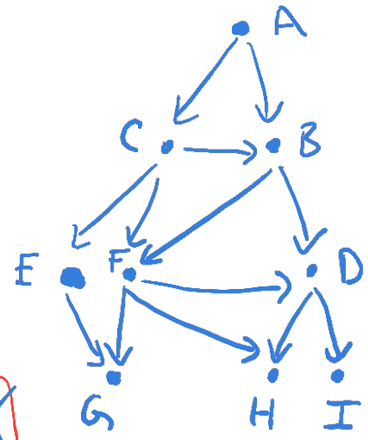
**Behauptung:** Für alle $t \in \mathbb{N} \cup \{0\}$, gibt es einen Moment, so dass die folgenden Aussagen gelten

1. für alle Knoten $v \in V$ mit $d(s, v) \leqslant t$ gilt $\mathrm{dist}[v] = d(s, v)$
2. für alle anderen Knoten $v \in V$ gilt $\mathrm{dist}[v] = \infty$
3. die Schlange enthält genau die Knoten mit $d(s, v) = t$

**Breitensuche**$(G, s)$ : (en. breadth-first search)

1. $\text{dist}[u] \leftarrow \infty$ für alle $u \in V \setminus \{s\}$ und $\text{dist}[s] \leftarrow 0$
2. $Q \leftarrow$ Schlange mit Knoten $s$
3. `while` $Q$ nicht leer `do`
   1. $u \leftarrow \text{dequeue}(Q)$
   2. `for` $(u, v) \in E$ `do`
      `if` $\text{dist}[v] = \infty$ `then`
      $\text{dist}[v] \leftarrow \text{dist}[u] + 1$ und $\text{enqueue}(Q, v)$





---

**Behauptung:** Für alle $t \in \mathbb{N} \cup \{0\}$, gibt es einen Moment, so dass die folgenden Aussagen gelten

1. für alle Knoten $v \in V$ mit $d(s, v) \leqslant t$ gilt $\text{dist}[v] = d(s, v)$
2. für alle anderen Knoten $v \in V$ gilt $\text{dist}[v] = \infty$
3. die Schlange enthält genau die Knoten mit $d(s, v) = t$

**Beweis:** per Induktion über $t$

*Ende*