Quiz

An undirected graph has 10 vertices, each of degree 3. How many edges does it have?

Answer: 15

$$\sum_{v \in V} \deg(v) = 2\,|E|$$

$$\Rightarrow 10 \cdot 3 = 2\,|E|$$

$$\Rightarrow |E| = 15$$

A directed graph without a sink has no topological ordering.

- (True)
- False

last node of a topological sorting is always a sink

Thus: no sink $\Rightarrow$ no topological sorting

Cities $C_0, \ldots, C_n$ lie in this order along a road. Routes start at C_0 and visit cities only in increasing index order. A DP entry $D[i]$ counts routes from C_0 to C_i. What initial value should $D[0]$ have?

DP[0] = number of routes from C_0 to C_0

= 1 (only the route with length 0)

Answer: 1

A DFS of a directed acyclic graph finishes its vertices in the order $\overrightarrow{C, A, D, B}$, from first to last. Which order is guaranteed to be a topological ordering?

post-order

a. A, B, C, D

b. B, A, D, C

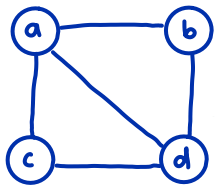
c. C, A, D, B

d. $\overleftarrow{B, D, A, C}$

umgekehrte post-order ist immer ein topological sorting (falls der Graph azyklisch ist)

A connected undirected graph has four vertices with degrees $\deg(a) = 3$, $\deg(b) = 2$, $\deg(c) = 2$, and $\deg(d) = 3$. Which choices of start and end are possible for an Eulerian walk? Select all correct answers.

Wahr	Falsch	
X		Start: a, end: d.
X		Start: d, end: a.
	X	Start: a, end: a.
	X	Start: c, end: d.



Start und Endknoten müssen die mit ungeradem Grad sein
(weil wir dort einmal mehr/weniger hineingehen statt hinausgehen)

For ordinary knapsack, $P(i, w)$ is the maximum profit using items $1, \ldots, i$ with weight limit w. Item 3 has weight 2 and profit 2. What value replaces the question mark?

Weight limit w	$i = 0$	$i = 1$	$i = 2$	$i = 3$
0	0	0	0	
1	0	0	0	
2	0	3	3	
3	0	3	4	
4	0	3	4	
5	0	3	7	?

max {7, 4+2} = 7

The table shows $P(i,w)$.

Answer: 7

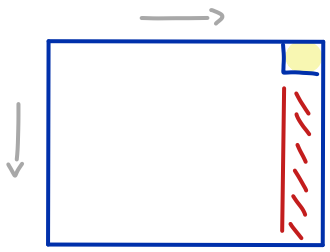
$P[i, w] = \max \{ P[i-1, w] , P[i-1, w-w_i] + p_i \}$

Entweder wir nehmen item 3 nicht:
Profit und Gewicht bleibt gleich wie bei 2 items

Oder wir nehmen item 3:
Profit erhöht sich um 2
Die ersten 2 items müssen dafür aber das Gewichtslimit $w-w_i = 5-2 = 3$ einhalten

An $n \times n$ matrix, where $n \geq 2$, is sorted in ascending order from left to right in each row and from top to bottom in each column. Its top-right entry is larger than x. When searching for x, what can we safely discard based only on this comparison? Select one answer.

a. Both the top row and the rightmost column.
b. The top row.
(c.) The rightmost column.
d. Neither a whole row nor a whole column.



A DP counts binary strings of length n with exactly k occurrences of "11" by extending shorter strings one bit at a time. Which state information lets the DP detect whether an appended bit creates a new "11"? Select one answer.

(a.) The last bit.
b. The parity of the number of ones.
c. The longest run of ones.

Fall $\ell = 0$: DP[i,j,0] = ...
Fall $\ell = 1$: DP[i,j,1] = ...
(ℓ = last bit)

Two calls to FindeZyklus return the circuits (a, b, c, a) and (b, d, e, b) in an undirected graph. Which sequence combines them into one circuit using exactly their combined edges? Select one answer.

a. (a, b, c, d, e, b, a)
b. (a, b, c, e, d, b, a)
c. (a, b, d, c, b, e, a)
(d.) (a, b, d, e, b, c, a)

(a, b, d, e, b, c, a)

A directed graph is stored using adjacency lists. It has n vertices, each with at most three outgoing edges. What is the worst-case running time of a full DFS? Select one answer.

a. $\Theta(\log n)$
b. $\Theta(n^2)$
c. $\Theta(n \log n)$
(d.) $\Theta(n)$

Worst-Case Laufzeit von DFS: $\Theta(n+m)$
$= \Theta(n+3n)$
$= \Theta(n)$
$m \leq 3n$

For an array $A[1, \ldots, n]$ of distinct entries, let $L(i)$ be the length of a longest increasing subsequence ending at position i. For each statement, decide whether it is always true.

Wahr	Falsch		
X		The length of a longest increasing subsequence of the whole array is $\max_{1 \le i \le n} L(i)$.	
	X	$L(i+1) \ge L(i)$ for every $1 \le i < n$.	
X		If $1 \le i < j \le n$ and $A[i] < A[j]$, then $L(j) \ge L(i) + 1$.	

the LAS has to end somewhere

Counterexample:

1 2 3 4

A = [1, 2, 3, 0]

L(3) = 3 > L(4) = 1

A [... , A[i] , ... , A[j] , ...]

——> L(i) ... -> L(j)

-> wir können A[j] einfach an die LAS bis A[i] anhängen

A connected undirected graph has only even degrees. Remove all edges of one circuit, keeping all vertices. For each statement about the remaining graph, decide whether it is always true.

Wahr	Falsch		
X		Every vertex still has even degree.	
	X	All remaining edges lie in a single connected component.	
X		Every remaining edge belongs to some circuit.	

because each node loses an even number of incident edges

Counterexample 

because all degrees are even, there exists an Eulerian circuit within each connected component that has some edges

Discussion of Exercise Sheet 8

Definition 1. A vertex v in a connected graph is called a *cut vertex* (or *articulation point*) if the subgraph obtained by removing v (and all its incident edges) is disconnected.

Definition 2. An edge e in a connected graph is called a *cut edge* (or *bridge*) if the subgraph obtained by removing e (but keeping all the vertices) is disconnected.

(d) If an edge e is not a cut edge, then e must be part of a cycle.

Proof: Assume $e = \{u, v\}$ is not a cut edge

$\Rightarrow G' = (V, E \setminus \{e\})$ is connected (def of cut edge)

$\Rightarrow$ there exists a walk $(u, w_1, w_2, \dots, w_i, v)$ from u to v in G' (def. connected)

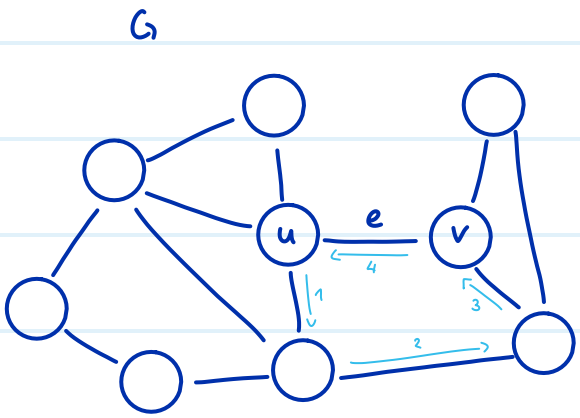
$\Rightarrow$ there exists a path $(u, p_1, p_2, \dots, p_j, v)$ from u to v in G'

because as long as a node occurs twice, we can remove the walk between its first and second occurrence and thereby construct a path.

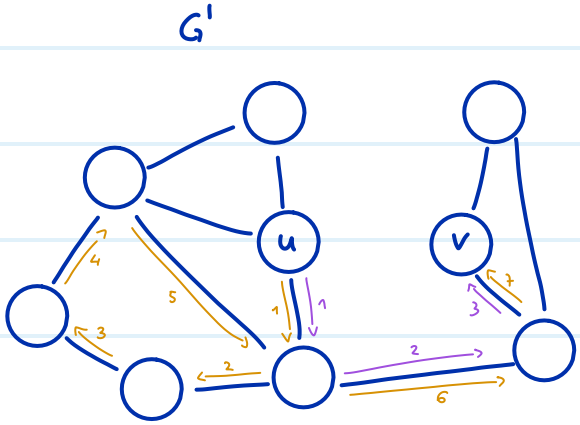
$\Rightarrow$ this path does not contain e (because the path is in G', which doesn't contain e)

$\Rightarrow (u, p_1, p_2, \dots, p_j, v, u)$ is a cycle in G (append $e = \{u, v\}$ to the path)

$\Rightarrow e = \{u, v\}$ is part of a cycle



cycle

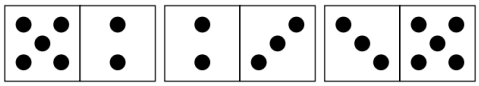


Walk Path

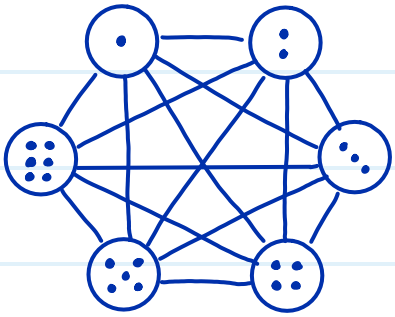
Exercise 8.4 *Domino* **(2 bonus points).** → Peergrading heute

(a) A domino set consists of all possible $\binom{6}{2} + 6 = 21$ different tiles of the form $[x|y]$, where x and y are numbers from $\{1, 2, 3, 4, 5, 6\}$. The tiles are symmetric, so $[x|y]$ and $[y|x]$ denote the same tile, which appears only once.

Show that it is impossible to form a line of all 21 tiles such that the adjacent numbers of any consecutive tiles coincide like in the example below.



We can just ignore the domino stones [x|x] because adding/removing them does not change the problem



domino sequence exists <=> Eulerian walk exists

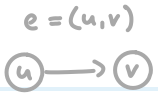
Since more than 2 degrees are odd, no Eulerian walk exists

Thus no such domino sequence exists.

(this is just an outline - your solutions need to explain in more detail)

Gerichtete Graphen (Digraphen)

Gerichtete Kanten: $E \subseteq \{(u,v) \mid u,v \in V \text{ und } u \neq v\}$



$\deg_{in}(v)$: Anzahl eingehender Kanten

$\deg_{out}(v)$: Anzahl ausgehender Kanten



$\deg_{in}(v) = 3$

$\deg_{out}(v) = 2$

$\deg(v) = \deg_{in}(v) + \deg_{out}(v)$

Quelle: falls $\deg_{in}(v) = 0$



Senke: falls $\deg_{out}(v) = 0$



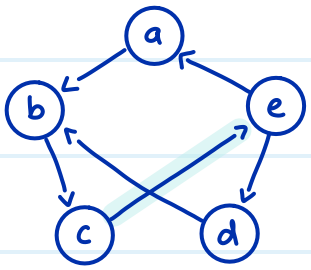
Weg, Pfad... gleich wie bei ungerichteten Graphen, aber mit der zusätzlichen Bedingung, dass man nur in Richtung der Kanten gehen darf

Handschlag-Lemma für gerichtete Graphen $\sum_{v \in V} \deg_{in}(v) = \sum_{v \in V} \deg_{out}(v) = |E|$

(weil jede Kante genau aus einem Knoten herauskommt und genau in einen Knoten hineingeht)

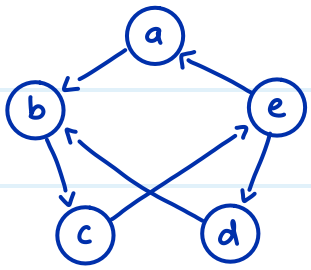
Adjazenzmatrix: $A_{uv} = \begin{cases} 1 & \text{falls } (u,v) \in E \\ 0 & \text{sonst} \end{cases}$

	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	0	0
c	0	0	0	0	1
d	0	1	0	0	0
e	1	0	0	1	0



DAG: Abkürzung für directed acyclic graph

Aufgaben:



Wie viele verschiedene gerichtete Pfade von e nach c gibt es? 2: (e, a, b, c), (e, d, b, c)

Wie viele verschiedene gerichtete Pfade von c nach e gibt es? 1: (c, e)

Wie viele verschiedene gerichtete Kreise gibt es? 2: (a, b, c, e, a), (b, c, e, d, b)

Welche möglichen Längen kann ein gerichteter Weg von d nach e haben? $\{3+4k \mid k \in \mathbb{N}_0\}$
$= \{3, 7, 11, 15, \dots\}$

Gibt es einen gerichteten Hamiltonkreis? Nein

Gibt es einen gerichteten Hamiltonpfad? Ja: (a, b, c, e, d)

Gibt es einen gerichteten Eulerzyklus? Nein

Gibt es einen gerichteten Eulerweg? Ja: (e, a, b, c, e, d, b)

Ist der Graph zusammenhängend? Ja

Gibt es einen gerichteten Zyklus, der a und d enthält? Nein

Gibt es einen gerichteten geschlossenen Weg, der a und d enthält? Ja: (a, b, c, e, d, b, c, e, a)

Topologische Sortierung eines DAG

Definition: eine Anordnung der Knoten $v_1, v_2, v_3, \ldots, v_n$ $\quad v_i \to v_j \Rightarrow i < j$

sodass für alle Kanten $(v_i, v_j) \in E$ gilt: $i < j$

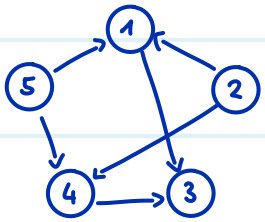
Intuitiv: wenn es eine Kante von a nach b gibt, dann muss a auch vor b in der topologischen Sortierung sein

falls $a \to b$ dann top Sortierung = $\ldots, a, \ldots, b, \ldots$

Alle Vorgänger von v kommen vor v in der topologischen Sortierung

Alle Nachfolger von v kommen nach v in der topologischen Sortierung

Aufgabe: Finde alle möglichen topologischen Sortierungen:

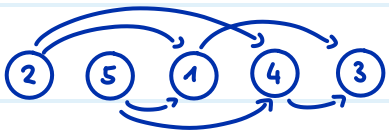


(2, 5, 1, 4, 3)

(2, 5, 4, 1, 3)

(5, 2, 1, 4, 3)

(5, 2, 4, 1, 3)

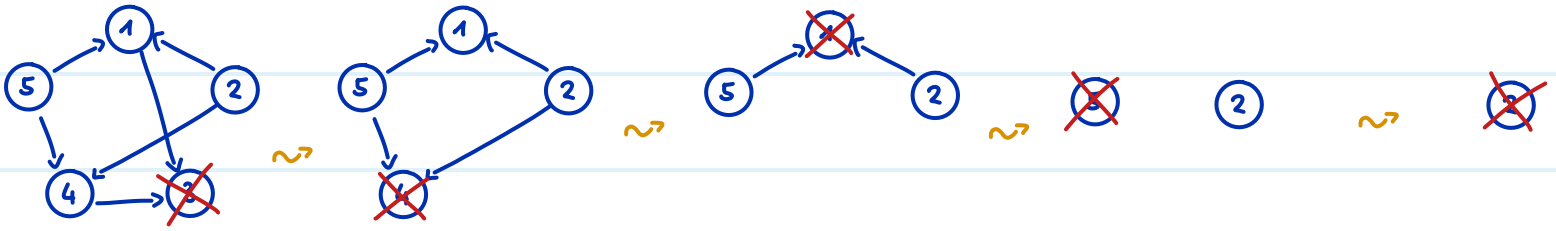


alle Kanten gehen nach rechts

Algorithmus:

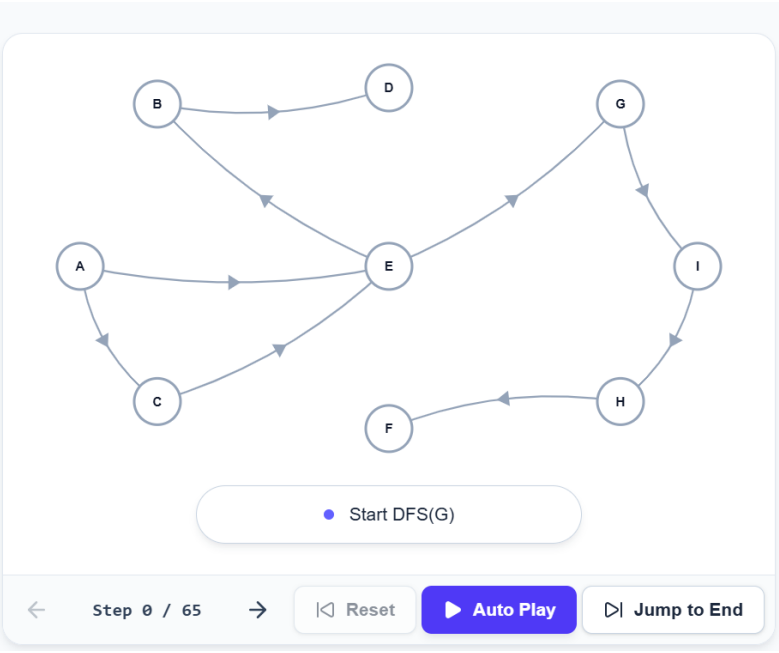
1) finde eine Senke s (irgendwo starten und so lange laufen, bis man in einer "Sackgasse" ankommt)
2) setze s an die letzte Stelle
3.) lösche s und die zu s anliegenden Kanten, und wiederhole den Algorithmus auf dem Restgraphen

Bsp



(,3) (4,3) (1,4,3) (5,1,4,3) (2,5,1,4,3)

Detailliertere Version des Algorithmus:



DFS (TOPOLOGICAL SORT) DFS (TIMESTAMPS)

DFS $\mathcal{O}(n+m)$

```
1  DFS(G)
2    topologische Sortierung ← leere Liste;
3    while es gibt einen unmarkierten Knoten u₀ do
4      VISIT(u₀)

6  VISIT(u)
7    markiere u
8    while u hat einen unmarkierten Nachfolger v do VISIT(v)
9    if u ist im Restgraphen eine Senke then
10     füge u vorne an die topologische Sortierung an
11     lösche u aus G
12   else return "gerichteter Zyklus gefunden"
```

Laufzeit: $O(n+m)$ (mit Adjazenzlisten)

Theorem ∃ topologische Sortierung <=> ∄ gerichteter Zyklus

Beweis:

"=>": Wir beweisen mittels Kontraposition $(A \to B \equiv \neg B \to \neg A)$

Annahme: ∃ gerichteter Zyklus $(\neg B)$

=> Sei $(v_1, v_2, \dots, v_k, v_1)$ dieser Zyklus

=> In einer topologischen Sortierung muss v_1 vor v_k kommen und v_k vor v_1

=> Das ist unmöglich

=> ∄ topologische Sortierung $(\neg A)$

"<=": Annahme: ∄ gerichteter Zyklus

=> Alle Wege haben endliche Länge. Sei P ein längster Pfad.

=> Der Endknoten von P muss eine Senke sein. (sonst könnte man P ja noch verlängern oder es gäbe ein Zyklus)

=> ∃ Senke

=> der oben beschriebene Algorithmus funktioniert (in jedem Restgraphen gibt es eine Senke)

=> ∃ topologische Sortierung

Baum: Ein ungerichter Graph, der zusammenhängend und azyklisch ist.

Blatt: Ein Knoten mit Grad 1 in einem Baum

Wald: Ein ungerichter, azyklischer Graph (jede ZHK ist ein Baum)

DFS

DFS(G)

1 $T \leftarrow 1$; lege Arrays pre und post an
2 entferne alle Markierungen
3 **while** es gibt einen unmarkierten Knoten u_0 **do** VISIT(u_0)

"Rahmenprogramm"

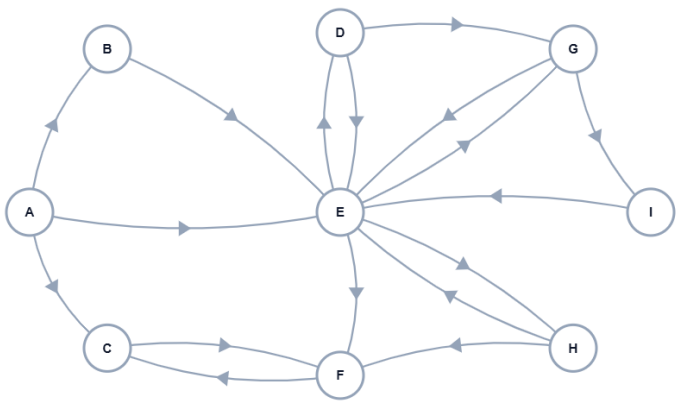
VISIT(u)

1 $\text{pre}[u] \leftarrow T$; $T \leftarrow T + 1$
2 markiere u
3 **while** u hat einen unmarkierten Nachfolger v **do** VISIT(v)
4 $\text{post}[u] \leftarrow T$; $T \leftarrow T + 1$

Laufzeit: $O(n+m)$ (mit Adjazenzlisten)

Das Rahmenprogramm wird gebraucht, damit wir auch dann alle Knoten durchgehen,

wenn G nicht zusammenhängend ist



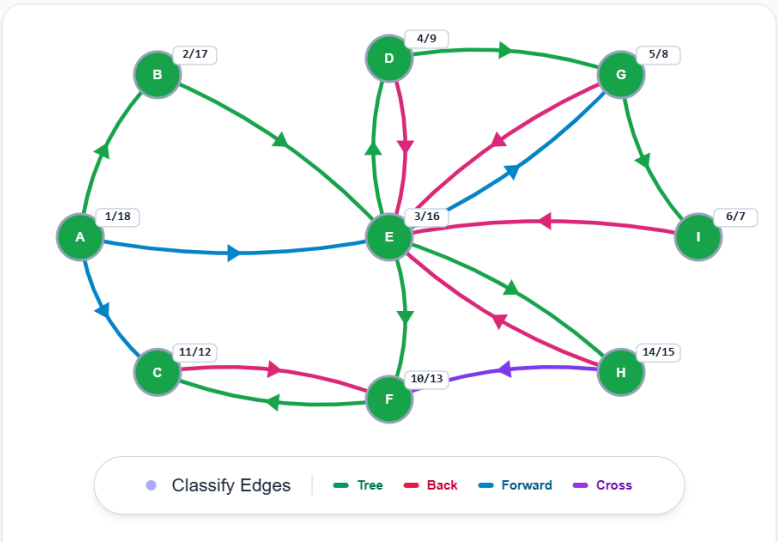
DFS $O(n+m)$

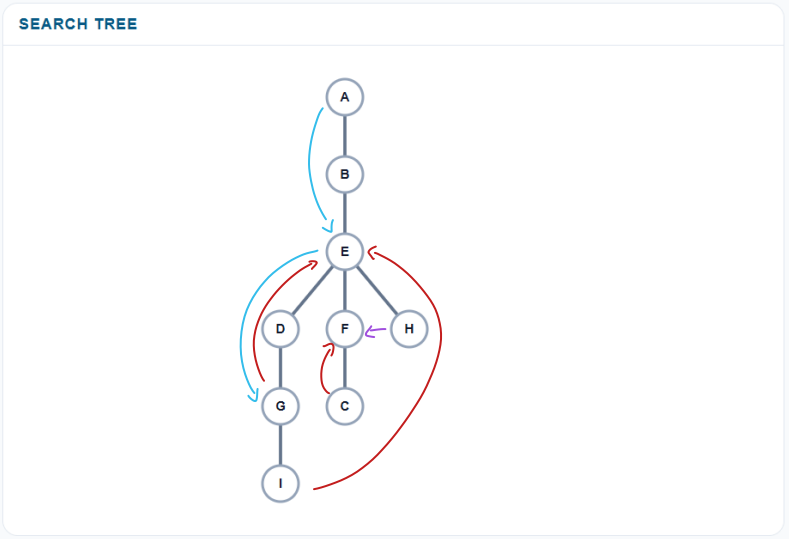
```
1  DFS(G):
2    T ← 1
3    for each v ∈ V do
4      if v is not visited then
5        DFS-VISIT(G, v)

7  DFS-VISIT(G, v):
8    Mark v as visited
9    pre[v] ← T; T ← T + 1
10   for each (v, w) ∈ E do
11     if w is not visited then
12       DFS-VISIT(G, w)
13   post[v] ← T; T ← T + 1
```

RECURSION STACK

VISITED

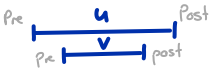




Pre numbers: Zeitpunkt wenn der Knoten angefangen wird (Start von visit(v))

Post numbers: Zeitpunkt wenn der Knoten abgeschlossen wird (Ende von visit(v))

Forward edge: Kante zeigt im DFS-Baum nach unten (= vorwärts aus der Perspektive des Alg)

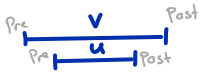


Tree edge: Der Algorithmus ist der Kante entlang gelaufen, um den nächsten Knoten zu besuchen. (Tree edges sind spezielle forward edges)

Cross edge: Kante (u,v) so dass u nicht im Teilbaum von v ist und v nicht im Teilbaum von u ist



Back edge: Kante zeigt im DFS-Baum nach oben im selben Teilbaum (= rückwärts aus der Perspektive des Alg)

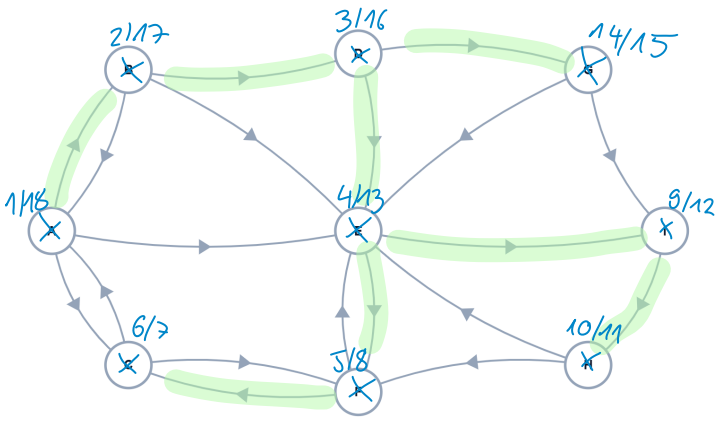


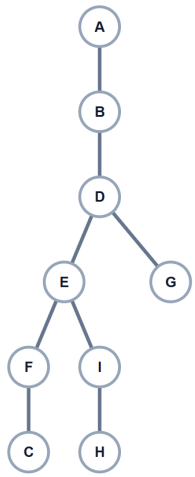
andere Fälle der Pre-Post Intervalle sind nicht möglich

Aufgabe: Führe DFS aus, um die pre/post numbers zu bestimmen, und zeichne den DFS-Baum

Falls mehrere Nachbaren möglich sind, wähle den alphabetisch kleinsten

Beginne bei Startknoten A





Klassifiziere folgende Kanten:

(E, F) → tree edge

(F, E) → back edge

(H, F) → cross edge

(A, E) → forward edge

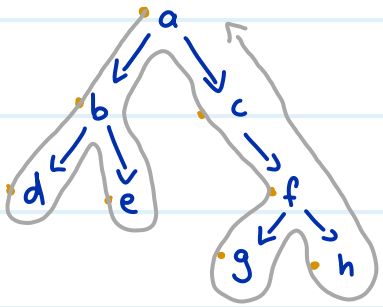
(G, E) → cross edge

(B, E) → forward edge

Trick zum bestimmen der pre/post order:

Pre = Flagge links

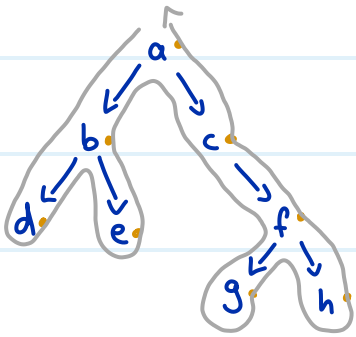




pre-order. a, b, d, e, c, f, g, h

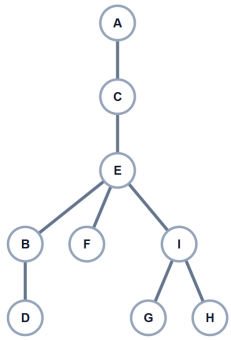
Post = Flagge rechts





post-order: d, e, b, g, h, f, c, a

Aufgabe: Bestimme die pre/post order:



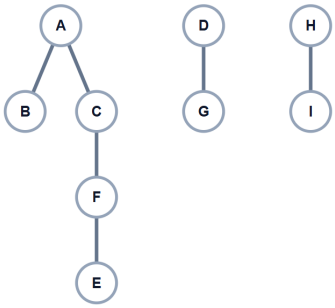
pre-order. A, C, E, B, D, F, I, G, H

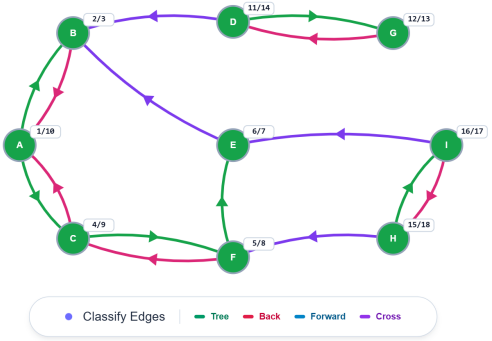
post-order: D, B, F, G, H, I, E, C, A

Theorem Falls G keinen gerichteten Zyklus hat, ist die umgekehrte post-order eine topologische Sortierung

Theorem ∃ back edge <=> ∃ gerichteter Zyklus

Falls nicht zusammenhängend nennt man es DFS - Wald





mehrere Bäume = Wald

BFS

$dist(s,v)$: die Länge eines kürzesten Weges von s nach v ($:= \infty$ falls unerreichbar)

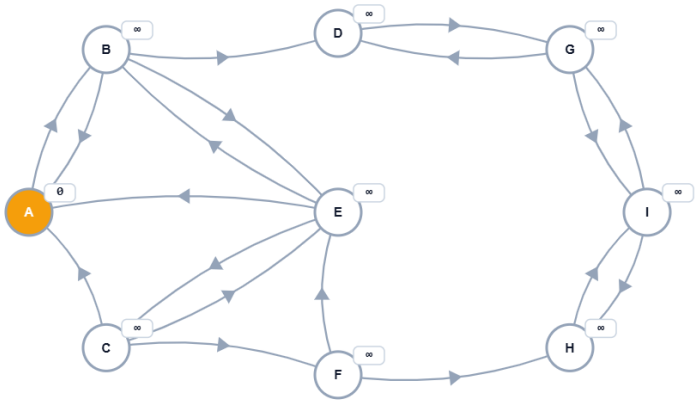
$S_k = \{ v \in V \mid dist(s,v) = k \}$: ist die Menge aller Knoten, die genau Distanz k zum Startknoten s haben

BFS berechnet die Distanzen vom Startknoten s zu allen anderen Knoten

---

BFS(s)

```
1  Q ← leere Queue;   ENQUEUE(s, Q)
2  dist[s] ← 0;   dist[v] ← ∞ für alle v ≠ s      ▷ ∞ heisst "noch nicht gefunden"
3  while Q ≠ ∅ do
4      u ← DEQUEUE(Q)
5      for alle Kanten (u, v) ∈ E do
6          if dist[v] = ∞ then
7              dist[v] ← dist[u] + 1;   ENQUEUE(v, Q)
```



BFS $\mathcal{O}(n+m)$

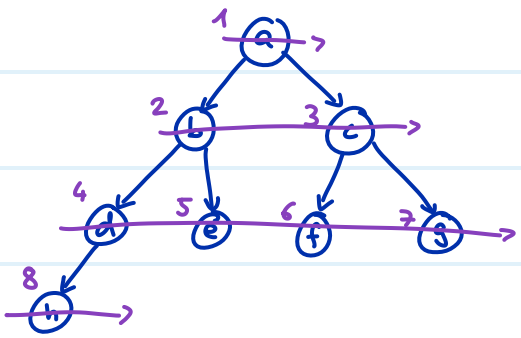
```
1   dist[s] ← 0
2   dist[v] ← ∞ for all v ≠ s
3   Q ← ∅
4   ENQUEUE(s, Q)
5   while Q ≠ ∅ do
6     u ← DEQUEUE(Q)
7     for each (u, v) ∈ E do
8       if dist[v] == ∞ then
9         dist[v] ← dist[u] + 1
10        ENQUEUE(v, Q)
```

QUEUE (Q) LEVEL SETS

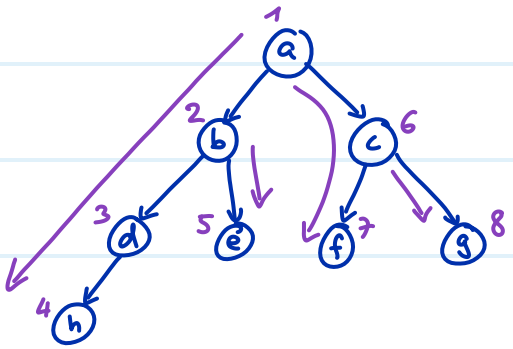
Laufzeit: $O(n+m)$ (mit Adjazenzlisten)

BFS	vs	DFS
$O(n+m)$		$O(n+m)$
funktioniert für gerichtet und ungerichtet		funktioniert für gerichtet und ungerichtet
findet genau alle von s erreichbaren Knoten		findet genau alle von s erreichbaren Knoten (ohne Rahmenprogramm)
verwendet eine Queue		verwendet einen Stack (implizit wegen der Rekursion)
–		umgekehrte post-order ist topologische Sortierung (falls G keine Zyklen hat)
berechnet Distanz zum Startknoten (falls es keine Gewichte oder uniforme (alle gleich) Gewichte gibt)		–
iteratives Programm		rekursives Programm
enter/leave numbers (wann ein Knoten in die Queue hinein/hinausgeht)		pre/post numbers

BFS enter numbers



DFS pre numbers

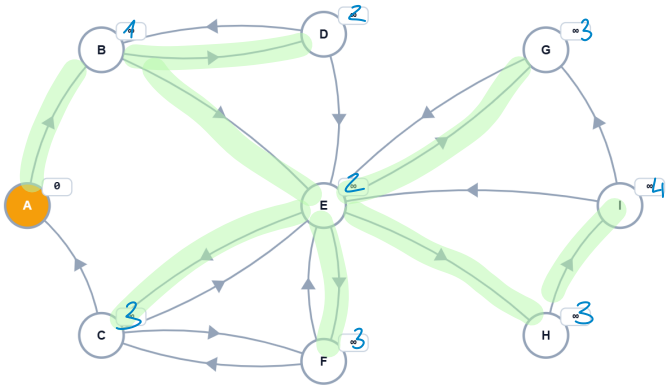


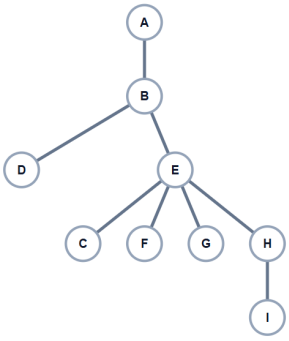
-> Ihr solltet beide manuell auf Papier ausführen und auch programmieren können

Aufgabe: Führe BFS aus, und zeichne den BFS-Suchbaum und bestimme die Level Sets

Falls mehrere Nachbaren möglich sind, wähle den alphabetisch kleinsten

Beginne bei Startknoten A





LEVEL SETS

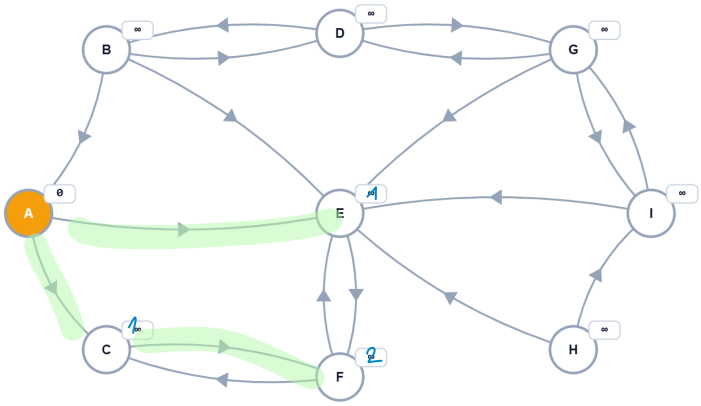
Dist 0: A

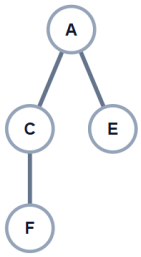
Dist 1: B

Dist 2: D E

Dist 3: C F G H

Dist 4: I





LEVEL SETS

Dist 0: A

Dist 1: C E

Dist 2: F

Aufgabe:

(b) Describe a bottom-up dynamic programming algorithm that, given a graph G with the property that $v_1, \ldots, v_n$ is a topological sorting, returns the length of the longest path in G in $O(|V| + |E|)$ time. You can assume that the graph is provided to you as a pair (n, Adj) of the integer $n = |V|$ and the adjacency lists Adj. Your algorithm can access $Adj[u]$, which is a list of vertices to which u has a direct edge, in constant time. Formally, $Adj[u] := \{v \in V \mid (u, v) \in E\}$.

In your solution, address the following aspects:

1. *Dimensions of the DP table*: What are the dimensions of the DP table?
2. *Subproblems*: What is the meaning of each entry?
3. *Recursion*: How can an entry of the table be computed from previous entries? Justify why your recurrence relation is correct. Specify the base cases of the recursion, i.e., the cases that do not depend on others.
4. *Calculation order*: In which order can entries be computed so that values needed for each entry have been determined in previous steps? Describe the calculation order in pseudocode.
5. *Extracting the solution*: How can the solution be extracted once the table has been filled?
6. *Running time*: What is the running time of your solution?

Hint: *Define the entry of the DP table as $DP[i] =$ length of longest path in G ending at vertex v_i.*

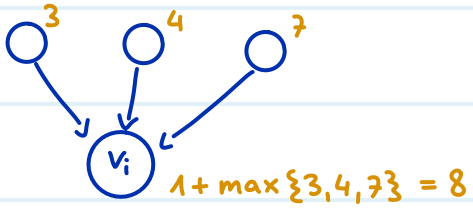
Dimension: $DP[1 \ldots n]$

Teilproblem: $DP[i]$:= length of a longest path in G ending at vertex v_i

Base Case: $DP[1] = 0$ weil v_1 eine Quelle ist, d.h. $\deg_{in}(v_1) = 0$, also kann kein Pfad in v_1 enden

Rekursion:

$$DP[i] = \begin{cases} 0 & \text{falls } v_i \text{ eine Quelle ist} \\ 1 + \max\{DP[j] \mid (v_j, v_i) \in E\} & \text{sonst } (v_i \text{ hat Vorgänger}) \end{cases}$$



Berechnungsreihenfolge: for $i = 1 .. n$ (d.h. topologische Sortierung aufsteigend)
compute DP[i]

Lösung: $\max_{1 \leq i \leq n} \{ DP[i] \}$

Laufzeit: Trick: Adjazenzliste umdrehen, weil wir immer Vorgänger brauchen (in $O(n+m)$)

Berechnung für $DP[i] = \Theta(1 + \deg_{in}(v_i))$

Berechnung für gesamte Tabelle: $\sum_{i=1}^{n} \Theta(1 + \deg_{in}(v_i))$

$= \Theta(n) + \sum_{i=1}^{n} \Theta(\deg_{in}(v_i))$ ← Handschlaglemma

$= \Theta(n) + \Theta(m)$

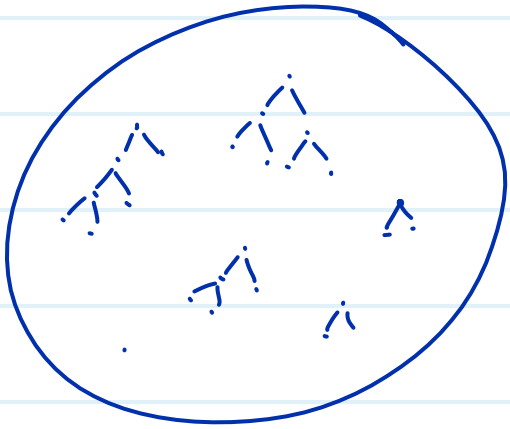
$= \Theta(n+m)$

Auslesen der Lösung in $\Theta(n)$

Total: $\underline{\Theta(n+m)}$

Aufgabe: Write a Java program that counts the number of connected components in an undirected graph represented as an adjacency list int[][] G.

Feststellung: Anzahl ZHKs = Anzahl Bäume im DFS Wald



Lösung mit DFS:

```
public static int countConnectedComponents(int[][] G) {
  int n = G.length;
  boolean visited[] = new boolean[n]; // all initialized to false by default

  int ZHKcount = 0;
  for (int i = 0; i<n; i++) {
    if (visited[i] == false) {
      dfs(i, G, visited);    // visits ZHK of node i
      ZHKcount++;
    }
  }

  return ZHKcount;
}
```

pass by reference

```
public static void dfs(int x, int[][] G, boolean[] visited) {
  visited[x] = true;                    // mark the current node as visited

  for (int j = 0; j<G[x].length; j++) {  // iterate over all neighbors of x
    int y = G[x][j];                    // y is the j-th neighbor
    if (visited[y] == false) {          // if y is not visited yet
      dfs(y, G, visited);               // then visit y
    }
  }
}
```

Lösung mit BFS:

```
public static int countConnectedComponents(int[][] G) {
  int n = G.length;
  boolean visited[] = new boolean[n]; // all initialized to false by default

  int ZHKcount = 0;
  for (int i = 0; i<n; i++) {
    if (visited[i] == false) {
      bfs(i, G, visited);    // visits ZHK of node i
      ZHKcount++;
    }
  }

  return ZHKcount;
}
```

→ Einziger Unterschied im Rahmenprogramm

```
public static void bfs(int s, int[][] G, boolean[] visited) {
  visited[s] = true;                     // mark the current node as visited
  Queue<Integer> Q = new LinkedList<>();  // creates an empty queue Q
  Q.add(s);

  while(!Q.isEmpty()) {
    int x = Q.poll();                    // deuqeues the next node
    for (int j = 0; j<G[x].length; j++) {// iterate over all neighbors of x
      int y = G[x][j];                   // y is the j-th neighbor
      if (visited[y] == false) {         // if y is not visited yet
        visited[y] = true;               // then visit y
        Q.add(y);
      }
    }
  }
}
```

Prove or disprove: Es gibt einen zusammenhängenden Graphen mit $n \geq 2$ Knoten, in dem alle Knoten paarweise verschiedene Knotengrade haben.

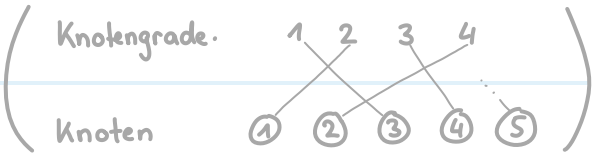
Falsch. Sei G ein zusammenhängender Graph mit n ≥ 2 Knoten

=> Mögliche Knotengrade sind $\{1, 2, \ldots, n-1\}$ (0 ist ausgeschlossen weil G zusammenhängend ist)

=> Es gibt also nur n-1 verschiedene mögliche Knotengrade, aber n Knoten

=> Nach dem Pigeon hole Prinzip folgt, dass es zwei Knoten u und v gibt, die denselben Grad haben

=> Dies widerlegt die Aussage



Prove or disprove: Es gibt einen Graphen mit $n \geq 2$ Knoten, in dem alle Knoten paarweise verschiedene Knotengrade haben.

Falsch. Sei G ein Graph mit n ≥ 2 Knoten

=> Mögliche Knotengrade sind $\{0, 1, \ldots, n-1\}$

=> Es kann nicht gleichzeitig ein Knoten mit Grad 0 und ein Knoten mit Grad n-1 geben (weil ein Knoten mit Grad n-1 zu allen anderen Knoten eine Kante hat, d.h alle anderen Knoten haben Grad ≥ 1)

=> Es gibt also nur n-1 verschiedene mögliche Knotengrade, aber n Knoten

↳ entweder $\{1, 2, \ldots, n-1\}$ oder $\{0, 1, \ldots, n-2\}$

=> Nach dem Pigeon hole Prinzip folgt, dass es zwei Knoten u und v gibt, die denselben Grad haben

=> Dies widerlegt die Aussage

Decide if the following statements are true or false:

- [x] In a DAG, every edge (u,v) satisfies post[u] > post[v].

because there are no back edges

- [ ] In a DAG, every edge (u,v) satisfies pre[u] < post[v].

Counterexample: cross edges

- [x] Every DAG has at least one source.

DAG has top sorting, and the first node of the top sorting is a source

- [x] Every DAG has at least one sink.

DAG has top sorting, and the last node of the top sorting is a sink

- [x] Reverse all edges of a DAG. Is it still a DAG.

If there was a cycle in the reversed graph, we could read the edges backwards and construct a cycle in the original graph

- [ ] In any directed graph, the vertex with the smallest post number is a sink.

Counterexample: 

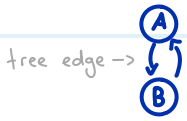
B has smallest post number but is not a sink

- [x] If a graph has k sources, the DFS forest has at least k trees.

Sources are not reachable from other nodes, thus DFS starts a new tree

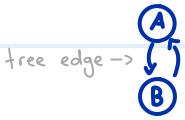
☒ A tree edge can never be part of a directed cycle.

Counterexample:



☒ Let $e = (u, v)$ be an edge. The classification of the edge as forward/backward/cross edge is always the same, independent of where we start.

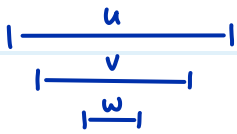
Counterexample:



(A,B) is a tree edge if we start at A

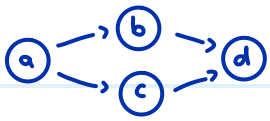
but a back edge if we start at B

☑ If $e_1 = (u, v)$ is a forward edge and $e_2 = (v, w)$ is a forward edge, then $e_3 = (u, w)$ is also a forward edge.



☒ If a DAG has exactly one source and exactly one sink, its topological ordering is unique.

Counterexample:



a, b, c, d and a, c, b, d are two different topological sortings

☑ Given the pre/post numbers of all n nodes (but we don't know the original Graph), we can determine a topological sorting in $O(n)$.

(assuming the original graph was a DAG)

We know that the largest post number is 2n.

Thus we can use Bucket Sort which runs in O(n)