3.5 Gerichteter Pfad

Definition 292
Eine Folge $(u_0, u_1, \ldots, u_n)$ mit $u_i \in V$ für $i = 0, \ldots, n$ heißt gerichteter Pfad, wenn

$$\big(\forall i \in \{0, \ldots, n-1\}\big)\Big[(u_i, u_{i+1}) \in A\Big].$$

Ein gerichteter Pfad heißt einfach, falls alle u_i paarweise verschieden sind.

3.6 Gerichteter Kreis

Definition 293

Ein gerichteter Pfad $(u_0, u_1, \ldots, u_n)$ heißt gerichteter Kreis, wenn $u_0 = u_n$.
Der gerichtete Kreis heißt einfach, falls $u_0, u_1, \ldots, u_{n-1}$ alle paarweise verschieden sind.

3.7 dag

Definition 294

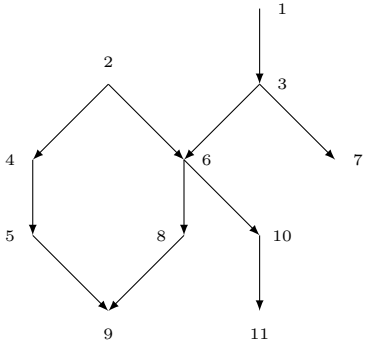
Ein Digraph, der keinen gerichteten Kreis enthält, heißt *d*irected *a*cyclic *g*raph, kurz dag.

In einem dag heißen Knoten mit In-Grad 0 Quellen, Knoten mit Aus-Grad 0 Senken.

Eine Nummerierung $i : V \to \{1, \ldots, |V|\}$ der Knoten eines *dags* heißt topologisch, falls für jede Kante $(u, v) \in A$ gilt:

$$i(u) < i(v).$$

Beispiel 295



Algorithmus zur topologischen Nummerierung:

```
while V≠∅ do
    nummeriere eine Quelle mit der nächsten Nummer
    streiche diese Quelle aus V
od
```

3.8 Zusammenhang

Definition 296
Ein Digraph heißt zusammenhängend, wenn der zugrundeliegende ungerichtete Graph zusammenhängend ist.

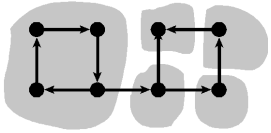
3.9 Starke Zusammenhangskomponenten

Definition 297
Sei $G = (V, A)$ ein Digraph. Man definiert eine Äquivalenzrelation $R \subseteq V \times V$ wie folgt:

$$uRv \iff \begin{cases} \text{es gibt in } G \text{ einen gerichteten Pfad von } u \text{ nach } v \\ \text{und einen gerichteten Pfad von } v \text{ nach } u. \end{cases}$$

Die von den Äquivalenzklassen dieser Relation induzierten Teilgraphen heißen die starken Zusammenhangskomponenten von G.

Beispiel 298



4. Durchsuchen von Graphen

Gesucht sind Prozeduren, die alle Knoten (eventuell auch alle Kanten) mindestens einmal besuchen und möglichst effizient sind.

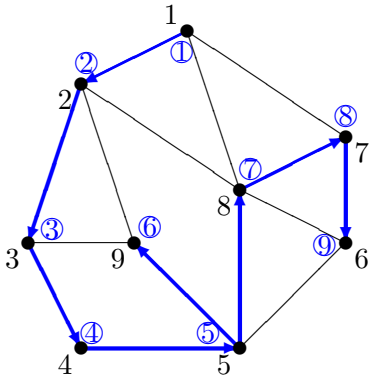
4.1 Tiefensuche, Depth-First-Search

Sei $G = (V, E)$ ein ungerichteter Graph, gegeben als Adjazenzliste.

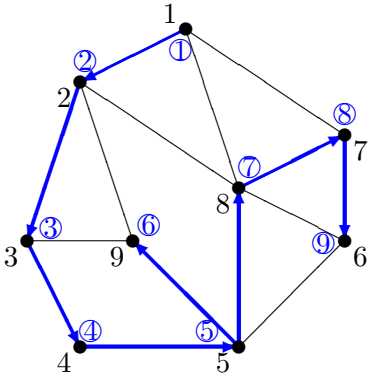
Algorithmus Tiefensuche (DFS):

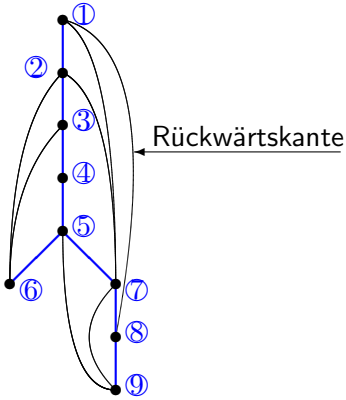
while $\exists$ unvisited v **do**
    $r :=$ pick (random) unvisited node
    push r onto stack
    while stack $\neq \emptyset$ **do**
        $v :=$ pop top element
        if v unvisited **then**
            mark v visited
            push all neighbours of v onto stack
            perform operations DFS_Ops(v)
        fi
    od
od

Beispiel 299



Beobachtung: Die markierten Kanten bilden einen Spannbaum:

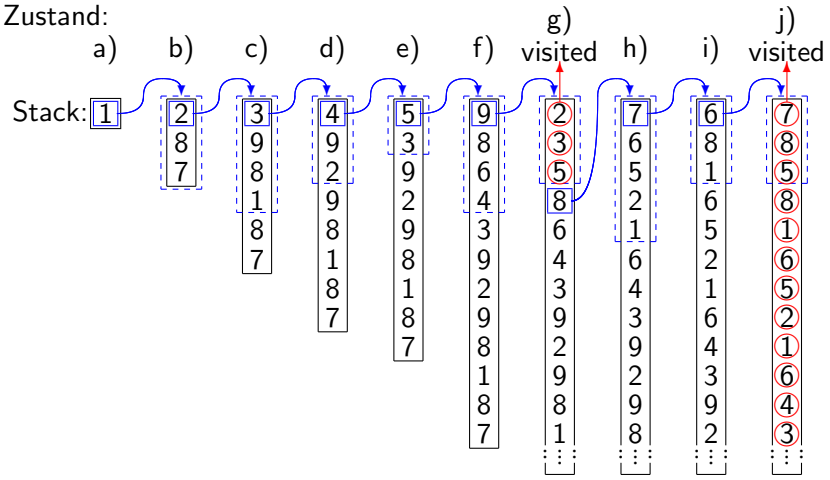




Folge der Stackzustände

□ : oberstes Stackelement ⬚ : Nachbarn

◯ : schon besuchte Knoten



Wir betrachten den Stackzustand:
Im Zustand g) sind die Elemente 2, 3 und 5 als visited markiert (siehe Zustände b), c) und e)). Deswegen werden sie aus dem Stack entfernt, und das Element 8 wird zum obersten Stackelement. Im Zustand j) sind alle Elemente markiert, so dass eins nach dem anderen aus dem Stack entfernt wird.

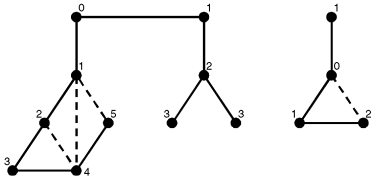
```
algorithm advanced DFS
  void proc DFSvisit(node v)
    visited[v] := true
    pre[v] := ++precount
    for all u ∈ adjacency_list[v] do
      if not visited[u] then
        type[(v,u)] := 'Baumkante'
        parent[u] := v
        DFSlevel[u] := DFSlevel[v]+1
        DFSvisit(u)
      elsif u ≠ parent[v] then
        type[(v,u)] := 'Rückwärtskante'
      fi
    od
    post[v] := ++postcount
  end proc
```

Fortsetzung

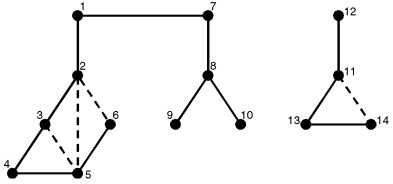
```
    co Initialisierung: oc
    for all v ∈ V do
      visited[v] := false
      pre[v] := post[v] := 0
    od
    precount := postcount := 0
    for all v ∈ V do
      if not visited[v] then
        DFSlevel[v] := 0
        parent[v] := null
        DFSvisit(v)
      fi
    od
  end
```

Beispiel 300 (gestrichelt sind Rückwärtskanten)

DFS-Level:

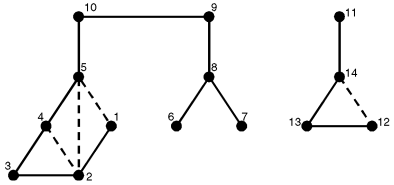


Präorder-Nummer:



Beispiel (Fortsetzung)

Postorder-Nummer:



Beobachtung: Die Tiefensuche konstruiert einen Spannwald des Graphen. Die Anzahl der Bäume entspricht der Anzahl der Zusammenhangskomponenten von G.

Satz 301

Der Zeitbedarf für die Tiefensuche ist (bei Verwendung von Adjazenzlisten)

$$O\big(|V| + |E|\big) \; .$$

Beweis:

Aus Algorithmus ersichtlich. □

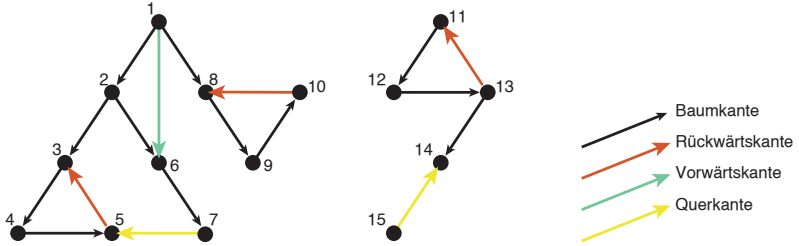
Tiefensuche im Digraphen: Für gerichtete Graphen verwendet man obigen Algorithmus, wobei man die Zeilen

```
elsif u ≠ parent[v] then
  type[(v,u)] := 'Rückwärtskante'
fi
```

ersetzt durch

```
elsif pre[u] > pre[v] then
  type[(v,u)] := 'Vorwärtskante'
elsif post[u] ≠ 0 then
  type[(v,u)] := 'Querkante'
else
  type[(v,u)] := 'Rückwärtskante'
fi
```

Beispiel 302 (Präorder-Nummer)

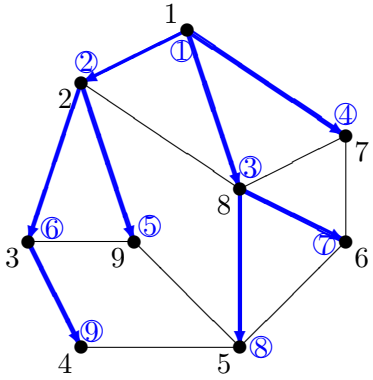


4.2 Breitensuche, Breadth-First-Search

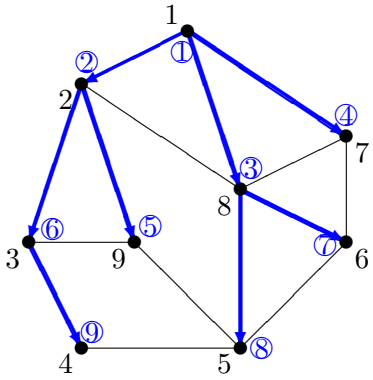
Sei $G = (V, E)$ ein ungerichteter Graph, gegeben mittels Adjazenzlisten.
BFS-Algorithmus:

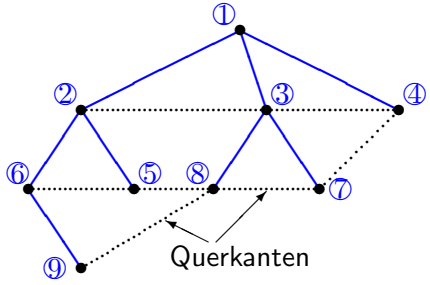
while $\exists$ unvisited v **do**
    $r :=$ pick (random) unvisited node
    push r into (end of) queue
    while queue $\neq \emptyset$ **do**
        $v :=$ remove front element of queue
        if v unvisited **then**
            mark v visited
            append all neighbours of v to end of queue
            perform operations BFS_Ops(v)
        fi
    od
od

Beispiel 303

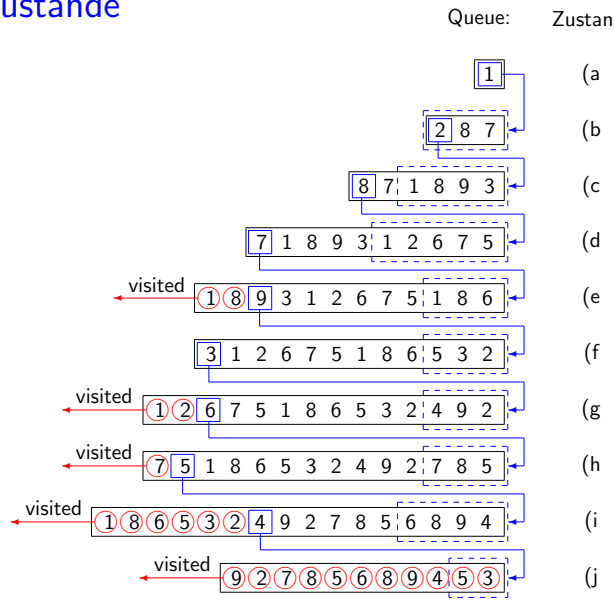


Beobachtung: Die markierten Kanten bilden einen Spannbaum:





Folge der Queuezustände



Wir betrachten die Folge der Queuezustände. Wiederum bedeutet die Notation:

: vorderstes Queue-Element : Nachbarn

: schon besuchte Knoten

Im Zustand e) sind die Elemente 1 und 8 als visited markiert (siehe Zustände a) und c)). Deswegen werden sie aus der Queue entfernt, und so wird das Element 9 das vorderste Queueelement. Das gleiche passiert in den Zuständen g), h) und i). Im Zustand j) sind alle Elemente markiert, so dass sie eins nach dem anderen aus der Queue entfernt werden.

```
algorithm advanced BFS
  for all v ∈ V do
    touched[v] := false
    bfsNum[v] := 0
  od
  count := 0
  queue := ∅
  for all v ∈ V do
    if not touched[v] then
      bfsLevel[v] := 0
      parent[v] := null
      queue.append(v)
      touched[v] := true
      while not empty(queue) do
        u := remove_first(queue)
        bfsNum[u] := ++count
```

Fortsetzung

```
            for all w ∈ adjacency_list[u] do
              if not touched[w] then
                type[(u,w)] := 'Baumkante'
                parent[w] := u
                bfsLevel[w] := bfsLevel[u]+1
                queue.append(w)
                touched[w] := true
              elsif not w = parent[u] then
                type[(u,w)] := 'Querkante'
              fi
            od
          od
        fi
      od
    end
```

Beobachtungen:

1. Die Breitensuche konstruiert einen Spannwald.
2. Der Spannwald besteht genau aus den Baumkanten im Algorithmus.
3. (u, v) ist Querkante $\Rightarrow \left|\mathrm{bfsLevel}(u) - \mathrm{bfsLevel}(v)\right| \leq 1$

Satz 304

Der Zeitbedarf für die Breitensuche ist (bei Verwendung von Adjazenzlisten)

$$O\big(|V| + |E|\big) \; .$$

Beweis:

Aus Algorithmus ersichtlich. □

4.3 Matroide

Definition 305

Sei S eine endliche Menge, $U \subseteq 2^S$ eine Teilmenge der Potenzmenge von S. Dann heißt $M = (S, U)$ ein Matroid und jedes $A \in U$ heißt unabhängige Menge, falls gilt:

1. $\emptyset \in U$
2. $A \in U,\ B \subseteq A \quad \Longrightarrow \quad B \in U$
3. $$A, B \in U,\ |B| = |A| + 1 \quad \Longrightarrow \quad (\exists x \in B \setminus A)\Big[\big(A \cup \{x\}\big) \in U\Big]$$

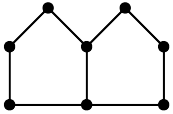
Jede bezüglich $\subseteq$ maximale Menge in U heißt Basis.
Nach 3. haben je zwei Basen gleiche Kardinalität. Diese heißt der Rang $\mathsf{r}(M)$ des Matroids.

Beispiel 306

Linear unabhängige Vektoren in einem Vektorraum.

Beispiel 307

G sei folgender Graph:



$S =$ Menge der Kanten von G
$U =$ Menge der kreisfreien Teilmengen von S