Algorithmus zur topologischen Nummerierung:

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

3.8 Zusammenhang

Definition 295
Ein Digraph heißt zusammenhängend, wenn der zugrundeliegende ungerichtete Graph zusammenhängend ist.

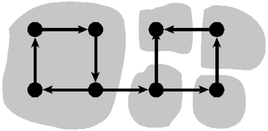
3.9 Starke Zusammenhangskomponenten

Definition 296
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 297



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.

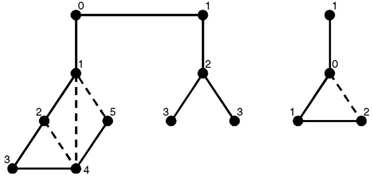
```
algorithm DFS
  void proc DFSvisit(node v)
    visited[v] := true
    pre[v] := ++precount
    for all u ∈ adjacency_list[v] do
      if not visited[u] then
        type[(v,u)] := 'Baumkante'
        parent[u] := v
        DFSlevel[u] := DFSlevel[v]+1
        DFSvisit(u)
      elsif u ≠ parent[v]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

Fortsetzung

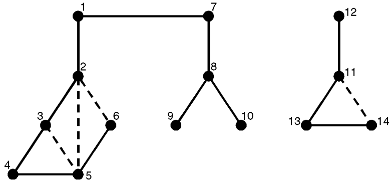
```
    co Initialisierung: oc
    for all v ∈ V do
      visited[v] := false
      pre[v] := post[v] := 0
    od
    precount := postcount := 0
    for all v ∈ V do
      if not visited[v] then
        DFSlevel[v] := 0
        parent[v] := null
        DFSvisit(v)
      fi
    od
  end
```

Beispiel 298 (gestrichelt sind Rückwärtskanten)

DFS-Level:

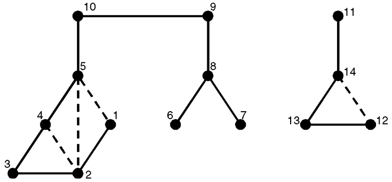


Präorder-Nummer:



Beispiel (Fortsetzung)

Postorder-Nummer:



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

Satz 299

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

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

Beweis:

Aus Algorithmus ersichtlich. $\square$

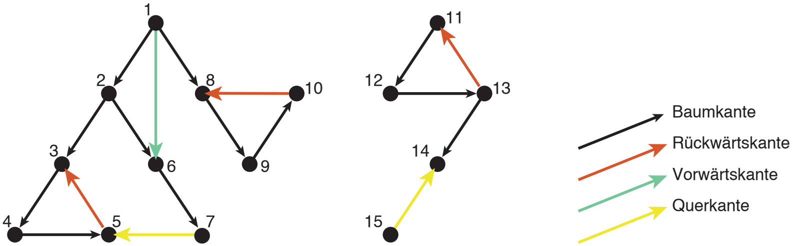
Tiefensuche im Digraphen: Für gerichtete Graphen verwendet man obigen Algorithmus, wobei man die Zeilen

```
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 300 (Präorder-Nummer)



4.2 Breitensuche, Breadth-First-Search

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

```
algorithm 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 301

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

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

Beweis:

Aus Algorithmus ersichtlich. $\square$

4.3 Matroide

Definition 302

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

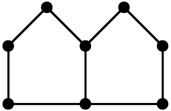
Jede bezüglich $\subseteq$ maximale Menge in U heißt Basis.
Nach 3. haben je zwei Basen gleiche Kardinalität. Diese heißt der Rang r(M) des Matroids.

Beispiel 303

Linear unabhängige Vektoren in einem Vektorraum.

Beispiel 304

G sei folgender Graph:



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

4.4 Greedy-Algorithmus

Sei $M = (S, U)$ ein Matroid, $w : S \to R$ eine Gewichtsfunktion.

algorithm greedy(S,U,w)
    B := $\emptyset$
    while ($|B| < \mathrm{r}(M)$) **do**
        sei $x \in \left\{y \in S \setminus B; B \cup \{y\} \in U\right\}$ mit
            minimalem Gewicht
        B := $B \cup \{x\}$
    od
end

Satz 305

Der Greedy-Algorithmus liefert eine Basis minimalen Gewichts.

Beweis:

Aus der Definition des Matroids (1.) folgt, dass die leere Menge $\emptyset$ eine unabhängige Menge ist.
Aus 3. folgt, dass in der `while`-Schleife wiederum nur unabhängige Mengen generiert werden.
Daher ist B am Ende des Algorithmus eine Basis (da inklusionsmaximal). Es bleibt zu zeigen, dass die gefundene Basis minimales Gewicht besitzt.
Sei also $B = \{b_1, \ldots, b_r\}$ die vom Algorithmus gelieferte Basis. Sei $b_1, \ldots, b_r$ die Reihenfolge der Elemente, in der sie der Greedy-Algorithmus ausgewählt hat. Dann gilt

$$w(b_1) \leq w(b_2) \leq \ldots \leq w(b_r).$$

Beweis (Forts.):

Sei weiter $B' = \{b'_1, \ldots, b'_r\}$ eine minimale Basis, und es gelte o. B. d. A.

$$w(b'_1) \leq w(b'_2) \leq \ldots \leq w(b'_r) \ .$$

Sei $i \in \{1, \ldots, r\}$. Gemäß Eigenschaft 3 für Matroide folgt, dass es ein $b' \in \{b'_1, \ldots, b'_i\}$ gibt, so dass $\{b_1, \ldots, b_{i-1}, b'\} \in U$.

Damit ist $w(b_i) \leq w(b'_i)$ (für alle i), und daher wegen der Minimalität von B'

$$w(b_i) = w(b'_i) \qquad \text{für alle } i \ .$$

□