WS 2005/06

Diskrete Strukturen

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005WS/ds/

27. Januar 2006

3. Definitionen für gerichtete Graphen

3.1 Digraph

Definition 285

Ein Digraph (aka gerichteter Graph, engl. *directed graph*) $G = (V, A)$ besteht aus einer Knotenmenge V und einer Menge $A \subseteq V \times V$ von geordneten Paaren, den gerichteten Kanten.

3.2 Grad

Definition 286

- $\mathsf{d}^-(v)$ ist der Aus-Grad von v, d. h. die Anzahl der Kanten mit Anfangsknoten v.
- $\mathsf{d}^+(v)$ ist der In-Grad von v, d. h. die Anzahl der Kanten mit Endknoten v.
- $\mathsf{d}(v) = \mathsf{d}^-(v) + \mathsf{d}^+(v)$ ist der (Gesamt-)Grad von v.

Beobachtung:

$$\sum_{v \in V} \mathsf{d}^-(v) = \sum_{v \in V} \mathsf{d}^+(v) = |A|$$

3.3 Adjazenzmatrix

Definition 287

Sei $G = (V, A)$ ein Digraph mit $V = \{v_1, \ldots, v_n\}$. Dann heißt

$$C = \left(c_{ij}\right)_{1 \leq i,j \leq n} \quad \text{mit } c_{ij} = \begin{cases} 1 & \text{falls } (v_i, v_j) \in A \\ 0 & \text{sonst} \end{cases}$$

die Adjazenzmatrix von G.

3.4 Inzidenzmatrix

Definition 288

Sei $G = (V, A)$ ein einfacher(!) Digraph mit $V = \{v_1, \ldots, v_n\}$ und $A = \{e_1, \ldots, e_m\}$. Dann heißt

$$B = \left(b_{ij}\right)_{\substack{1 \le i \le n \\ 1 \le j \le m}} \text{ mit } b_{ij} = \begin{cases} 1 & \text{falls } v_i \text{ Endknoten von } e_j \\ -1 & \text{falls } v_i \text{ Anfangsknoten von } e_j \\ 0 & \text{sonst} \end{cases}$$

die Inzidenzmatrix von G.

Beobachtung:

$$B \cdot B^T = \begin{pmatrix} \mathsf{d}(v_1) & & & 0 \\ & \mathsf{d}(v_2) & & \\ & & \ddots & \\ 0 & & & \mathsf{d}(v_n) \end{pmatrix} - A'$$

Diese Matrix heißt Laplacesche Matrix. Dabei ist, für alle i, j, der Eintrag $a'_{i,j}$ die Anzahl der im zu G gehörigen ungerichteten Graphen zwischen v_i und v_j verlaufenden Kanten. Enthält G keine antiparallelen Kanten, ist damit A' gleich der Adjazenzmatrix dieses ungerichteten Graphen.

Beobachtung: Die Laplacesche Matrix ist symmetrisch.

3.5 Gerichteter Pfad

Definition 289
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 290

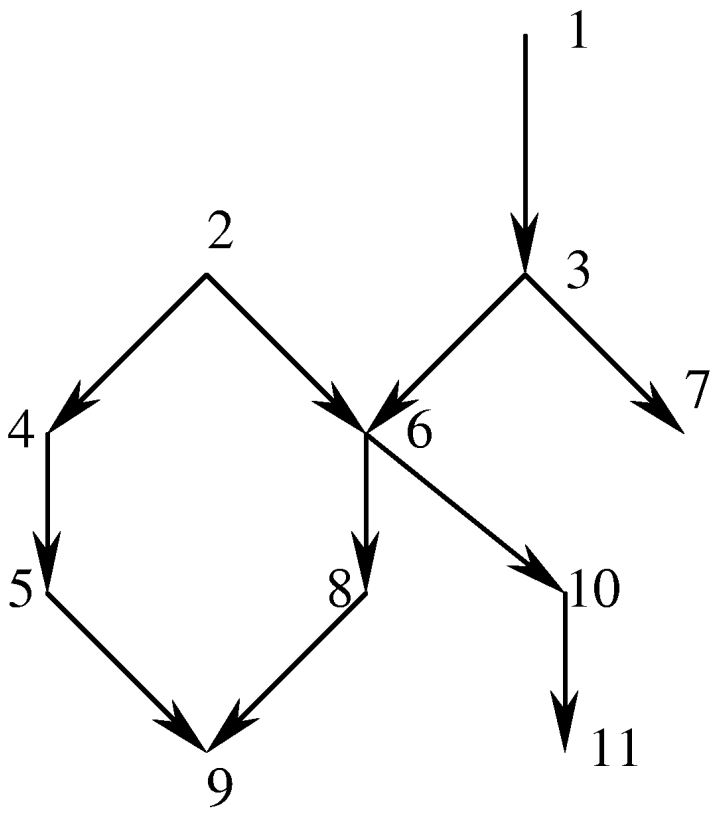
Ein gerichteter Pfad $(u_0, u_1, \ldots, u_n)$ heißt gerichteter Kreis, wenn $u_0 = u_n$.
Der gerichtete Kreis heißt einfach, falls $u_0, u_1, \ldots, u_{n-1}$ alle paarweise verschieden sind.

3.7 dag

Definition 291
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 292



Algorithmus zur topologischen Nummerierung:

`while` $V \neq \emptyset$ `do`

nummeriere eine Quelle mit der nächsten Nummer

streiche diese Quelle aus V

`od`

3.8 Zusammenhang

Definition 293

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

3.9 Starke Zusammenhangskomponenten

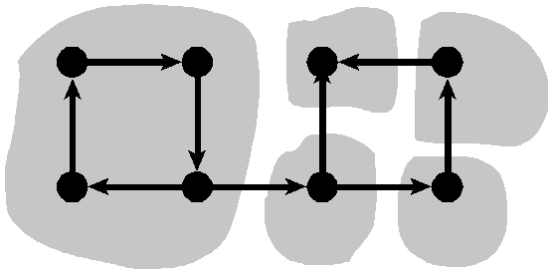
Definition 294

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 295



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.

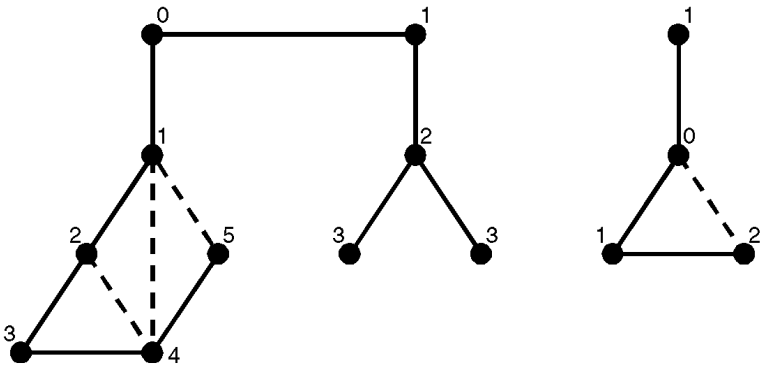
```
algorithm DFS
  void proc DFSvisit(node v)
    visited[v] := true
    pre[v] := ++precount
    for all u ∈ adjacency_list[v] do
      if not visited[u] then
        type[(v,u)] := 'Baumkante'
        parent[u] := v
        DFSlevel[u] := DFSlevel[v]+1
        DFSvisit(u)
      elsif u ≠ parent[v]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

Fortsetzung

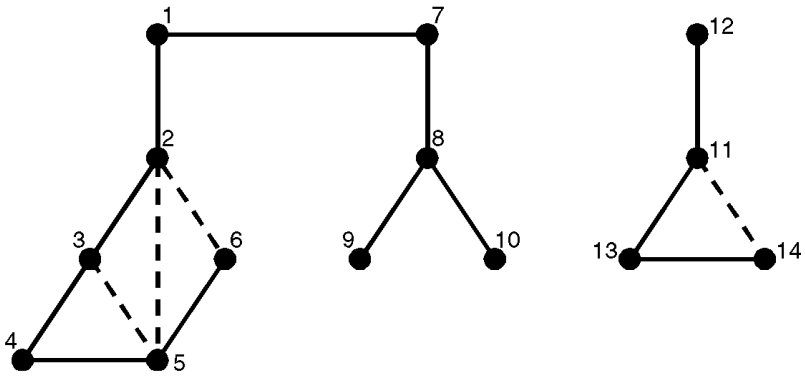
```
    co Initialisierung: oc
    for all v ∈ V do
      visited[v] := false
      pre[v] := post[v] := 0
    od
    precount := postcount := 0
    for all v ∈ V do
      if not visited[v] then
        DFSlevel[v] := 0
        parent[v] := null
        DFSvisit(v)
      fi
    od
  end
```

Beispiel 296 (gestrichelt sind Rückwärtskanten)

DFS-Level:

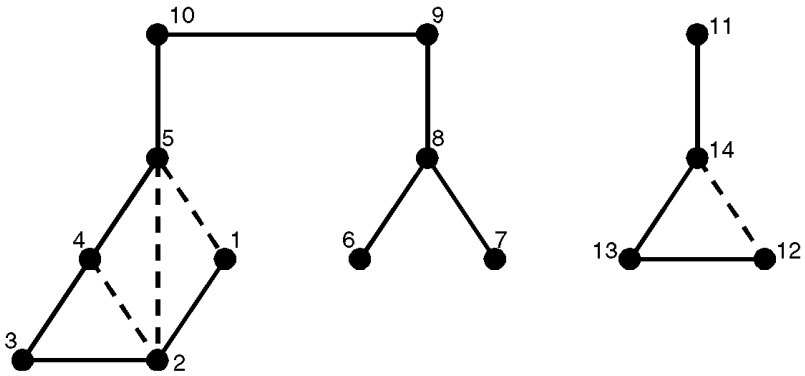


Präorder-Nummer:



Beispiel (Fortsetzung)

Postorder-Nummer:



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

Satz 297

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

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

Beweis:

Aus Algorithmus ersichtlich. □

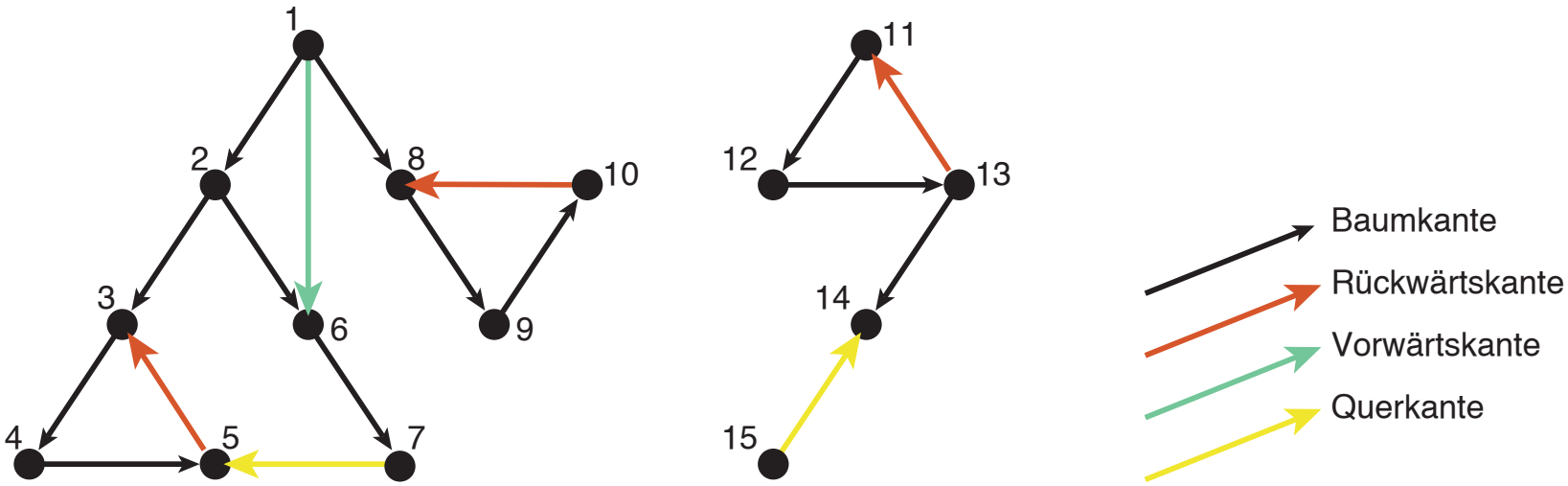
Tiefensuche im Digraphen: Für gerichtete Graphen verwendet man obigen Algorithmus, wobei man die Zeilen

```
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 298 (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 $\Rightarrow \left|\text{bfsLevel}(u) - \text{bfsLevel}(v)\right| \leq 1$

Satz 299

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

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

Beweis:

Aus Algorithmus ersichtlich. □