Script **generated by TTT**

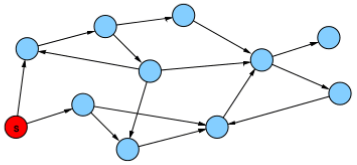
Title: Seidl: GAD (08.06.2016)

Date: Wed Jun 08 13:21:57 CEST 2016

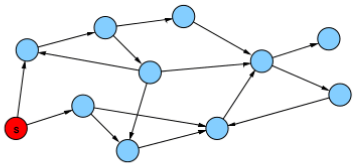
Duration: 44:41 min

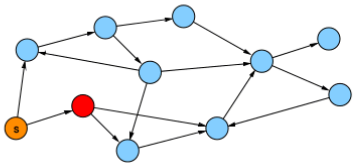
Pages: 30

Tiefensuche

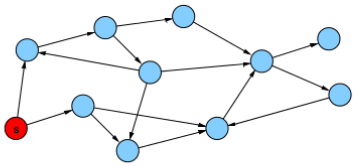


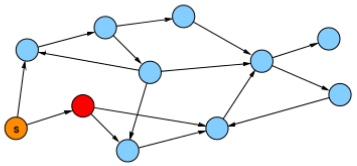
Tiefensuche

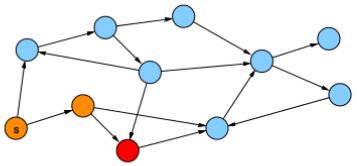




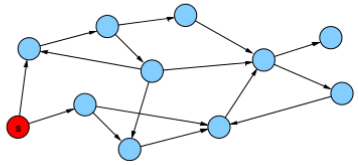
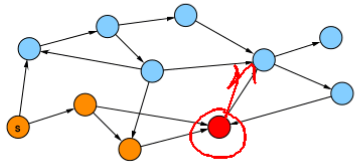
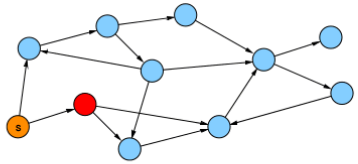
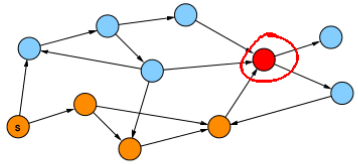
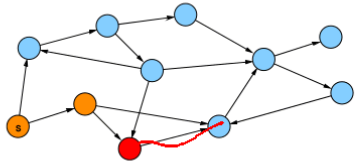
Tiefensuche



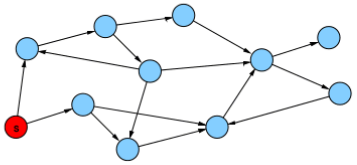
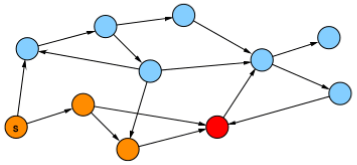
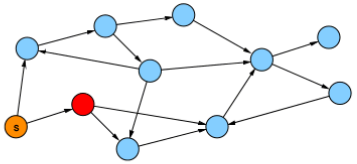
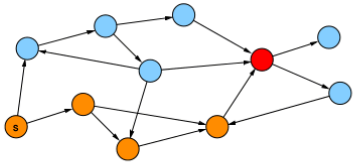
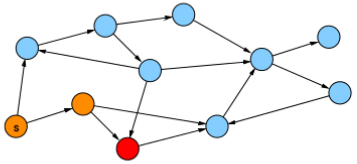
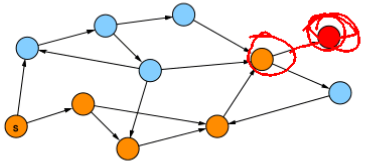




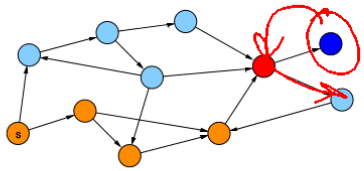
Tiefensuche

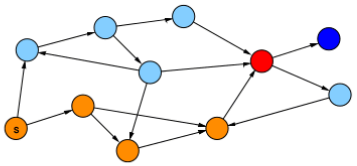
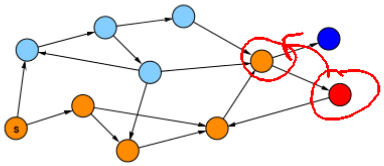
Tiefensuche

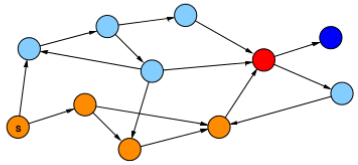
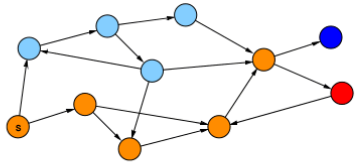
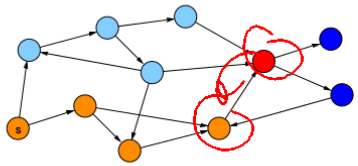
Tiefensuche



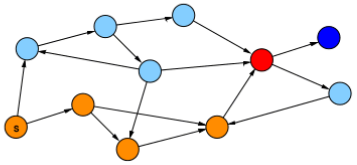
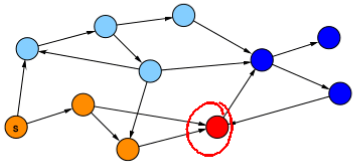
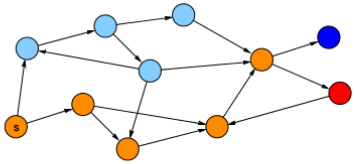
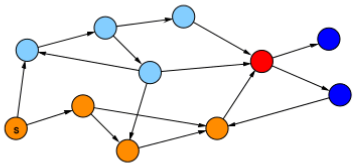
Tiefensuche

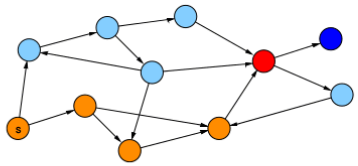
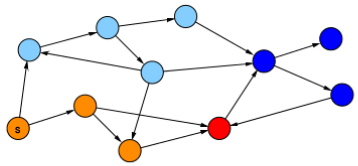
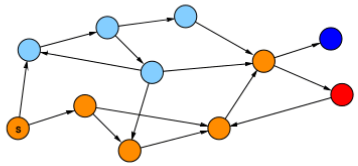
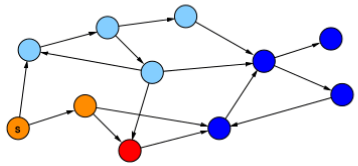
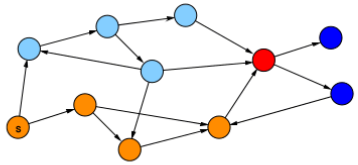
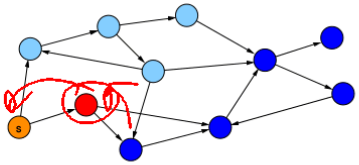
Tiefensuche

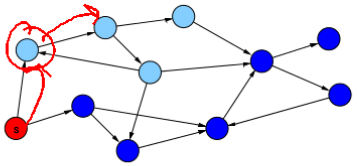
Tiefensuche

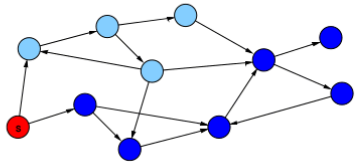
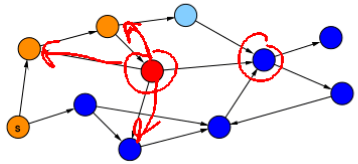
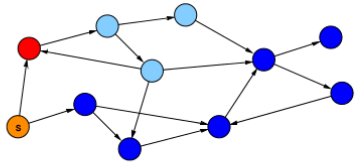
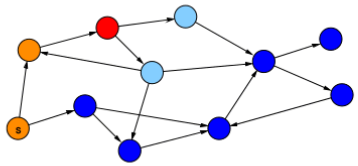
Tiefensuche

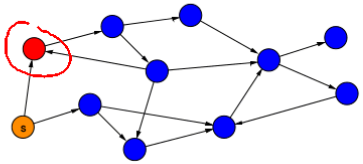
Tiefensuche



Tiefensuche

Tiefensuche



Tiefensuche

Übergeordnete Methode:

```
foreach (v ∈ V)
   Setze v auf nicht markiert;
init();
foreach (s ∈ V)
   if (s nicht markiert) {
      markiere s;
      root(s);
      DFS(s,s);
   }
```

```
DFS(Node u, Node v){
   foreach ((v, w) ∈ E)
      if (w ist markiert)
         traverseNonTreeEdge(v,w);
      else {
         traverseTreeEdge(v,w);
         markiere w;
         DFS(v,w);
      }
   backtrack(u,v);
}
```


Tiefensuche

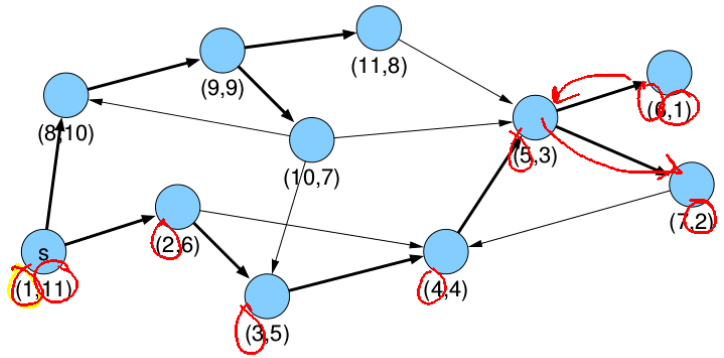
Variablen:

- int[ ] dfsNum; // Explorationsreihenfolge
- int[ ] finishNum; // Fertigstellungsreihenfolge
- int dfsCount, finishCount; // Zähler

Methoden:

- init() { dfsCount = 1; finishCount = 1; }
- root(Node s) { dfsNum[s] = dfsCount; dfsCount++; }
- traverseTreeEdge(Node v, Node w)
  { dfsNum[w] = dfsCount; dfsCount++; }
- traverseNonTreeEdge(Node v, Node w) { }
- backtrack(Node u, Node v)
  { finishNum[v] = finishCount; finishCount++; }

Tiefensuche



DFS-Nummerierung

Beobachtung:

- Knoten im DFS-Rekursionsstack (aktiven Knoten) sind bezüglich dfsNum aufsteigend sortiert

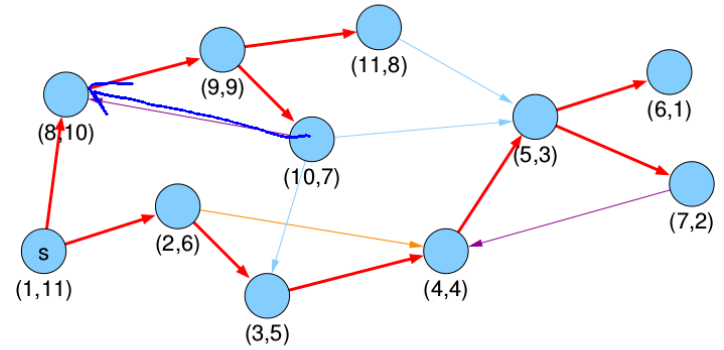
Begründung:

- dfsCount wird nach jeder Zuweisung von dfsNum inkrementiert
- neue aktive Knoten haben also immer die höchste dfsNum

DFS-Nummerierung

Kantentypen:

- Baumkanten: zum Kind
- Vorwärtskanten: zu einem Nachfahren
- Rückwärtskanten: zu einem Vorfahren
- Kreuzkanten: sonstige



DFS-Nummerierung

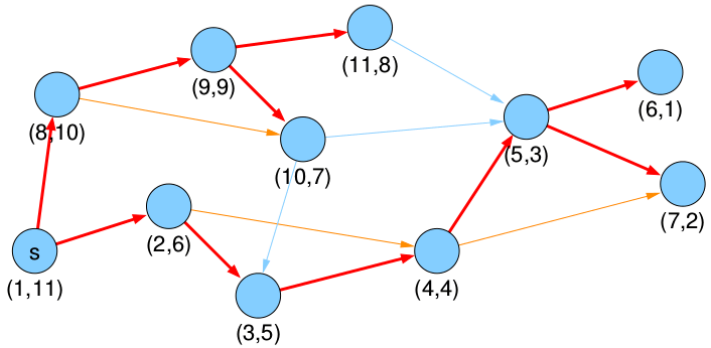
Beobachtung für Kante (v, w):

Kantentyp	dfsNum[v] < dfsNum[w]	finishNum[v] > finishNum[w]
Baum & Vorwärts	ja	ja
Rückwärts	nein	nein (umgekehrt)
Kreuz	nein	ja

DAG-Erkennung per DFS

Anwendung:

- Erkennung von azyklischen gerichteten Graphen (engl. directed acyclic graph / DAG)



- keine gerichteten Kreise

DAG-Erkennung per DFS

Lemma

Folgende Aussagen sind äquivalent:

1. *Graph G ist ein DAG.*
2. *DFS in G enthält keine Rückwärtskante.*
3. $\forall (v,w) \in E : finishNum[v] > finishNum[w]$

DFS-Nummerierung

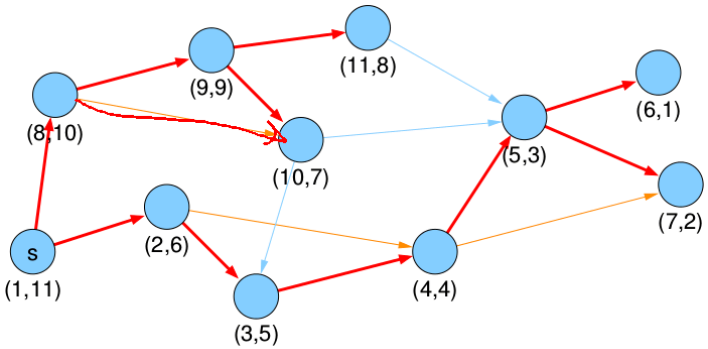
Beobachtung für Kante (v, w):

Kantentyp	dfsNum[v] < dfsNum[w]	finishNum[v] > finishNum[w]
Baum & Vorwärts	ja	ja
Rückwärts	nein	nein (umgekehrt)
Kreuz	nein	ja

DAG-Erkennung per DFS

Anwendung:

- Erkennung von azyklischen gerichteten Graphen (engl. directed acyclic graph / DAG)



- keine gerichteten Kreise

DAG-Erkennung per DFS

Lemma

Folgende Aussagen sind äquivalent:

1. *Graph G ist ein DAG.*
2. *DFS in G enthält keine Rückwärtskante.*
3. $\forall (v,w) \in E : finishNum[v] > finishNum[w]$

Beweis.

- (2)⇒(3): Wenn (2), dann gibt es nur Baum-, Vorwärts- und Kreuz-Kanten. Für alle gilt (3).
- (3)⇒(2): Für Rückwärtskanten gilt sogar die umgekehrte Relation finishNum[v]<finishNum[w].
  Wenn (3), dann kann es also keine Rückwärtskanten geben (2).

...

DAG-Erkennung per DFS

Lemma

Folgende Aussagen sind äquivalent:

1. *Graph G ist ein DAG.*
2. *DFS in G enthält keine Rückwärtskante.*
3. $\forall (v,w) \in E : finishNum[v] > finishNum[w]$

Beweis.

- ¬(2)⇒¬(1): Wenn Rückwärtskante (v,w) existiert, gibt es einen gerichteten Kreis ab Knoten w (und G ist kein DAG).
- ¬(1)⇒¬(2): Wenn es einen gerichteten Kreis gibt, ist mindestens eine von der DFS besuchte Kante dieses Kreises eine Rückwärtskante (Kante zu einem schon besuchten Knoten, dieser muss Vorfahr sein).

□

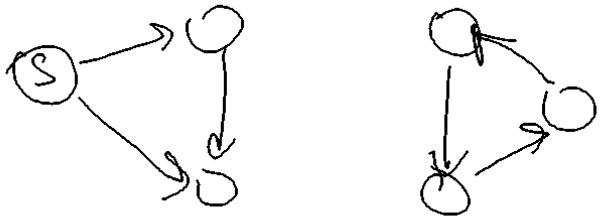
Zusammenhang in Graphen

Definition

Ein ungerichteter Graph heißt zusammenhängend, wenn es von jedem Knoten einen Pfad zu jedem anderen Knoten gibt.

Ein maximaler zusammenhängender induzierter Teilgraph wird als Zusammenhangskomponente bezeichnet.

Die Zusammenhangskomponenten eines ungerichteten Graphen können mit DFS oder BFS in $O(n+m)$ bestimmt werden.



Zusammenhang in Graphen

Definition

Ein ungerichteter Graph heißt zusammenhängend, wenn es von jedem Knoten einen Pfad zu jedem anderen Knoten gibt.

Ein maximaler zusammenhängender induzierter Teilgraph wird als Zusammenhangskomponente bezeichnet.

Die Zusammenhangskomponenten eines ungerichteten Graphen können mit DFS oder BFS in $O(n+m)$ bestimmt werden.

Knoten-Zusammenhang

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt k-fach zusammenhängend (oder genauer gesagt k-*knoten*zusammenhängend), falls

- $|V| > k$ und
- für jede echte Knotenteilmenge $X \subset V$ mit $|X| < k$ der Graph $G - X$ zusammenhängend ist.

Knoten-Zusammenhang

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt k-fach zusammenhängend (oder genauer gesagt k-*knoten*zusammenhängend), falls

- $|V| > k$ und
- für jede echte Knotenteilmenge $X \subset V$ mit $|X| < k$ der Graph $G - X$ zusammenhängend ist.

Bemerkung:

- "zusammenhängend" ist im wesentlichen gleichbedeutend mit "1-knotenzusammenhängend"

  Ausnahme: Graph mit nur einem Knoten ist zusammenhängend, aber nicht 1-zusammenhängend