

Script generated by TTT

Title: Täubig: GAD (05.07.2012)

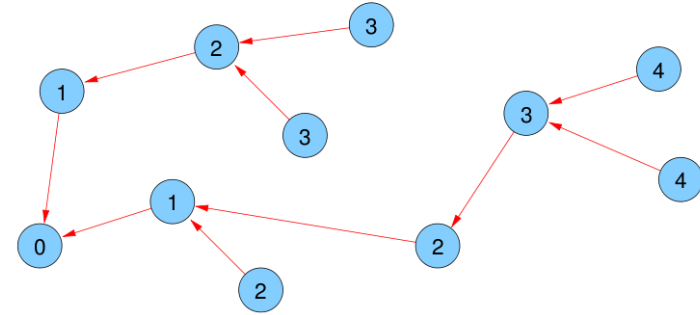
Date: Thu Jul 05 12:31:10 CEST 2012

Duration: 44:02 min

Pages: 23

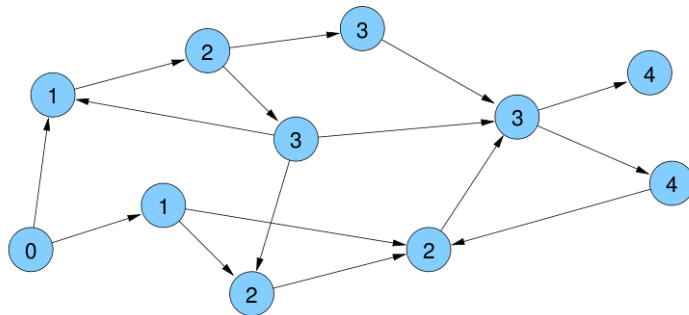
Breitensuche

- $\text{parent}(v)$: Knoten, von dem v entdeckt wurde
- parent wird beim ersten Besuch von v gesetzt (\Rightarrow eindeutig)



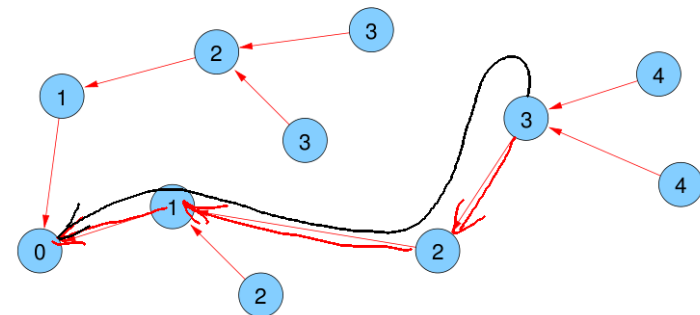
Breitensuche

- $d(v)$: Distanz von Knoten v zu s ($d(s) = 0$)



Breitensuche

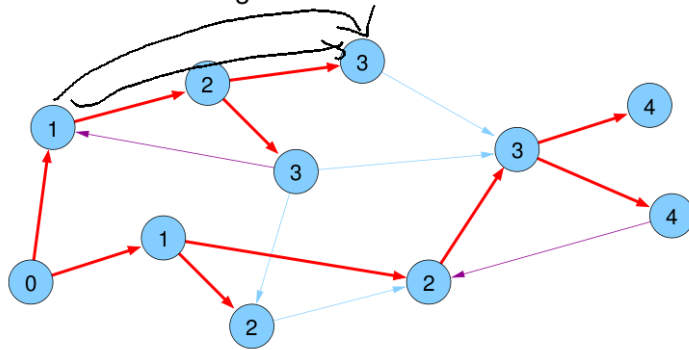
- $\text{parent}(v)$: Knoten, von dem v entdeckt wurde
- parent wird beim ersten Besuch von v gesetzt (\Rightarrow eindeutig)



Breitensuche

Kantentypen:

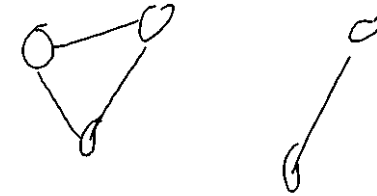
- **Baumkanten:** zum Kind
- **Rückwärtskanten:** zu einem Vorfahren
- **Kreuzkanten:** sonstige



Breitensuche

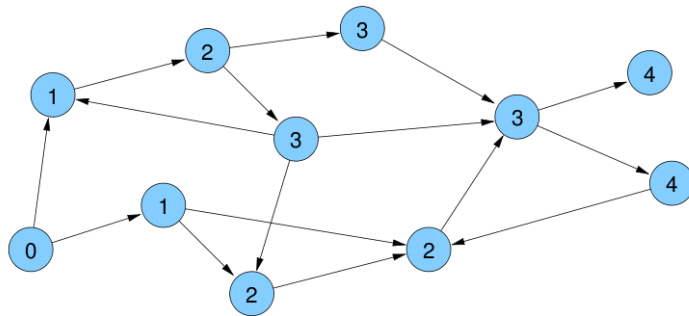
```

BFS(Node s) {
    d[s] = 0;
    parent[s] = s;
    List<Node> q = <s>;
    while (!q.empty()) {
        u = q.popFront();
        foreach ((u, v) ∈ E) {
            if (parent[v] == null) {
                q.pushBack(v);
                d[v] = d[u] + 1;
                parent[v] = u;
            }
        }
    }
}
    
```



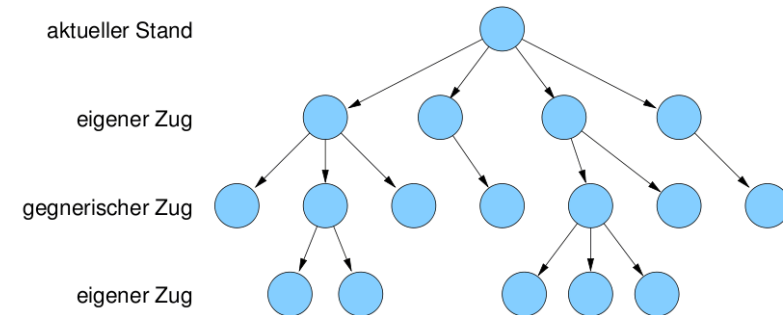
Breitensuche

Anwendung: Single Source Shortest Path (SSSP) Problem in **ungewichteten** Graphen



Breitensuche

Anwendung: Bestimmung des nächsten Zugs bei Spielen
Exploration des Spielbaums



Problem: halte Aufwand zur Suche eines guten Zuges in Grenzen

Breitensuche

Anwendung: Bestimmung des nächsten Zugs bei Spielen

- **Standard-BFS:** verwendet **FIFO-Queue**
ebenenweise Erkundung
aber: zu teuer!
- **Best-First Search:** verwendet **Priority Queue**
(z.B. realisiert durch binären Heap)
Priorität eines Knotens wird durch eine Güte-Heuristik des repräsentierten Spielzustands gegeben

Tiefensuche

Übergeordnete Methode (falls nicht alle Knoten erreicht werden)

```
foreach (v ∈ V)
  Setze v auf nicht markiert;

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

Tiefensuche



```
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

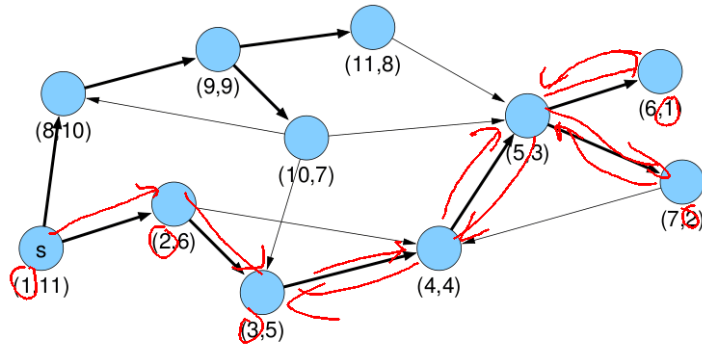
Variablen:

- 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

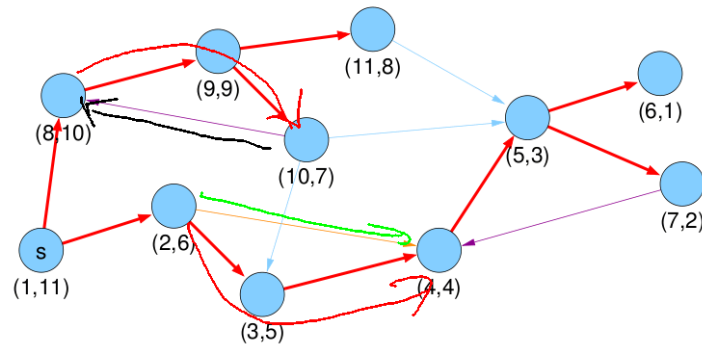
Begründung:

- 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

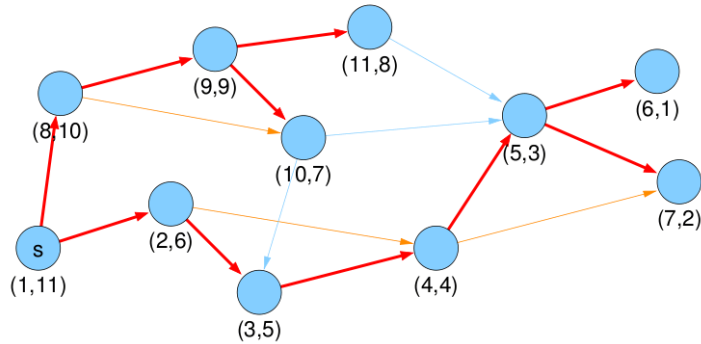
Beobachtung für Kante (v, w) :

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:

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

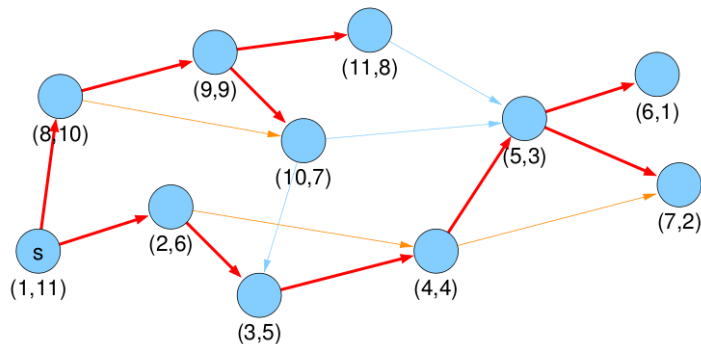
Beweis.

- (2) \Rightarrow (3): wenn (2), dann gibt es nur Baum-, Vorwärts- und Kreuzkanten
Für alle gilt (3)
- (3) \Rightarrow (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

Anwendung:

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

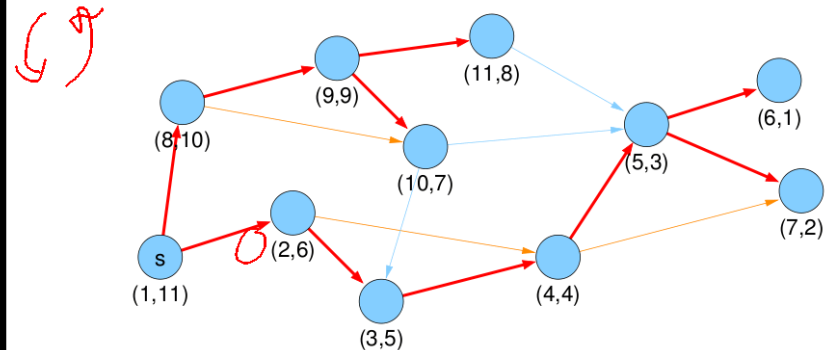


- keine gerichteten Kreise

DAG-Erkennung per DFS

Anwendung:

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



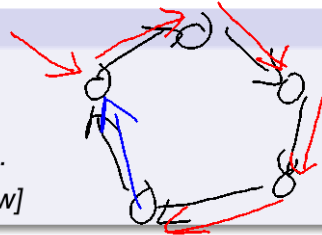
- keine gerichteten Kreise

DAG-Erkennung per DFS

Lemma

Folgende Aussagen sind äquivalent:

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

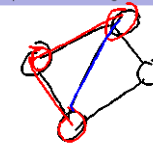


Beweis.

- $\neg(2) \Rightarrow \neg(1)$: wenn Rückwärtskante (v, w) existiert, gibt es einen gerichteten Kreis ab Knoten w (und G ist kein DAG)
- $\neg(1) \Rightarrow \neg(2)$: 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.

Knoten-Zusammenhang

Definition

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

Artikulationsknoten und Blöcke

Definition

Ein Knoten v eines Graphen G heißt **Artikulationsknoten** (engl. *cut-vertex*), wenn sich die Anzahl der Zusammenhangskomponenten von G durch das Entfernen von v erhöht.

Definition

Die **Zweifachzusammenhangskomponenten** eines Graphen sind die maximalen Teilgraphen, die 2-fach zusammenhängend sind.

Ein **Block** ist ein maximaler zusammenhängender Teilgraph, der keinen Artikulationsknoten enthält. D.h. die Menge der Blöcke besteht aus den Zweifachzusammenhangskomponenten, den Brücken (engl. *cut edges*), sowie den isolierten Knoten.