

Traversierung von Graphen

Die Algorithmen für eine Breiten- und Tiefensuche auf Bäumen lassen sich auf Graphen verallgemeinern.

Tiefensuche

DFSvisit(v) besucht alle (noch nicht besuchten) Knoten, die von **v** aus erreichbar sind.

v->dfsNum codiert hinterher die Reihenfolge in der Knoten besucht wurden.

Zusätzlich speichert **v->finNum** die Reihenfolge, in der die rekursiven Aufrufe beendet wurden.

Tiefensuche

```
1 DFS()  
2   dfsCount = 1;  
3   finCount = 1;  
4   foreach v ∈ V  
5       v->state = initial;  
6       v->parent = NULL;  
7   foreach (s ∈ V)  
8       if s->state == initial  
9           DFSvisit(s)
```

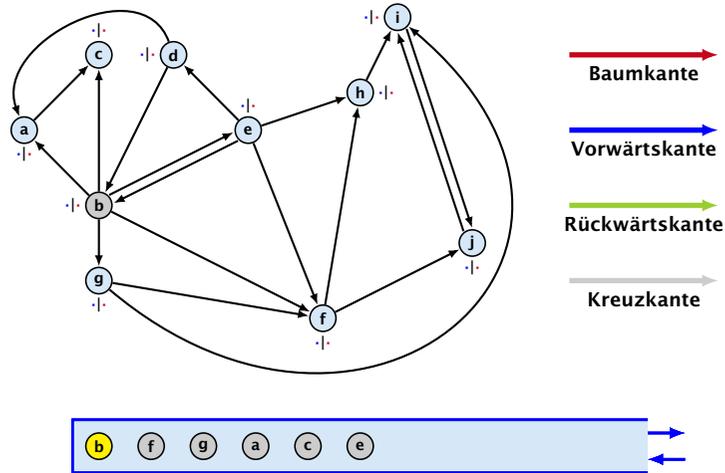
Tiefensuche – rekursiv

```
1 DFSvisit(v)  
2   v->state = active;  
3   v->dfsNum = dfsCount++;  
4   foreach x ∈ N[v] // iteriere ueber Nachbarn  
5       if (x->state == initial)  
6           x->parent = v;  
7           DFSvisit(x);  
8   v->state = finished;  
9   v->finNum = finCount++;
```

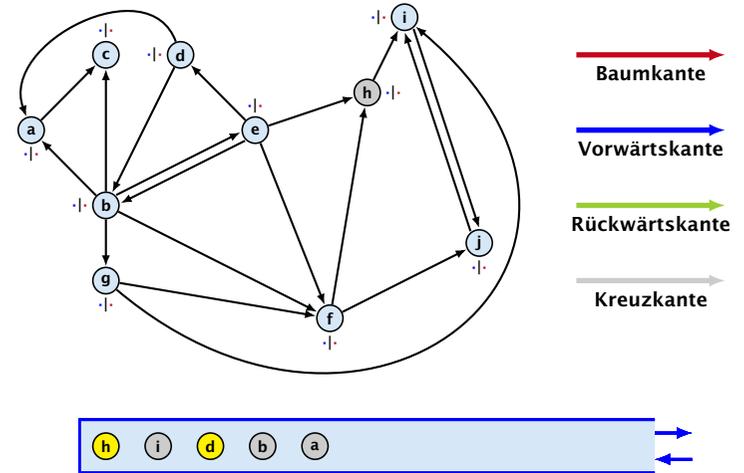
Tiefensuche – mit Stack

```
1 DFSvisit(s)  
2   Stack S; S.push(s);  
3   while (!S.empty()) {  
4       v = S.top();  
5       if (v->state == initial)  
6           v->state = active;  
7           v->dfsNum = dfsCount++;  
8           foreach x ∈ N(v)  
9               if (x->state == active) // (v,x) back edge  
10                  else if (x->state == finished) // (v,x) cross edge  
11                     else // (x->parent,x) forward edge  
12                         x->parent = v;  
13                         S.push(x);  
14           else  
15               v->state = finished;  
16               v->finNum = finCount++;  
17               S.pop();  
18   }
```

DFS



DFS



Tiefensuche

Kantentypen

▶ Baumkanten

Für jeden Nichtwurzelknoten v , speichert man den Vorgänger über den v „betreten“ wurde. Diese Kanten erzeugen einen Wald.

Die Bäume sind von der Wurzel zu den Blättern gerichtet.

▶ Vorwärtskanten

Falls für Kante (x, y) ein Pfad von x nach y in einem Baum existiert (und (x, y) nicht Baumkante).

▶ Rückwärtskanten

Falls für Kante (x, y) ein Pfad von y nach x in einem Baum existiert.

▶ Kreuzkanten

Alle anderen Kanten.

Tiefensuche

Eigenschaften der DFS-Nummerierung einer Kante (x, y) :

Baumkante oder Vorwärtskante

$x \rightarrow \text{dfsNum} < y \rightarrow \text{dfsNum}$ und $x \rightarrow \text{finNum} > y \rightarrow \text{finNum}$

Rückwärtskante

$x \rightarrow \text{dfsNum} > y \rightarrow \text{dfsNum}$ und $x \rightarrow \text{finNum} < y \rightarrow \text{finNum}$

Kreuzkante

$x \rightarrow \text{dfsNum} > y \rightarrow \text{dfsNum}$ und $x \rightarrow \text{finNum} > y \rightarrow \text{finNum}$

Es kann keine Kanten mit $x \rightarrow \text{dfsNum} < y \rightarrow \text{dfsNum}$ und $x \rightarrow \text{finNum} < y \rightarrow \text{finNum}$ geben.

Komplexität der Tiefensuche

Rekursive Implementierung

- ▶ erste Schleife in DFS wird genau $|V|$ mal aufgerufen
- ▶ $\text{DFSvisit}(v)$ wird für jeden Knoten genau einmal aufgerufen
- ▶ Schleife in $\text{DFSvisit}(v)$ wird $|N[v]|$ mal aufgerufen;

$$\sum_{v \in V} |N[v]| \leq 2|E| = \Theta(|E|)$$

- ▶ Gesamtlaufzeit: $\Theta(|V| + |E|)$

Komplexität der Stackimplementierung ist asymptotisch gleich.

Tiefensuche – Anwendungen

Test auf (starken) Zusammenhang in Graphen.

- ▶ rufe $\text{DFSvisit}(v)$ nur für einen Startknoten auf
- ▶ falls danach nicht alle Knoten besucht, ist Graph nicht (stark) zusammenhängend

Test auf Zyklenfreiheit.

- ▶ Graph hat Zyklus genau dann wenn die Tiefensuche eine Rückwärtskante enthält
- ▶ \Leftarrow wenn man eine Rückwärtskante hat wir mit dieser Rückwärtskante über die Baumkanten ein Zyklus geschlossen
- ▶ \Rightarrow angenommen man hat einen Zyklus und die Tiefensuche liefert keine Rückwärtskante; dann ist finNum entlang des Zyklus steigend, da dieser nur aus Vorwärtskanten, Baumkanten, und Kreuzkanten besteht (Widerspruch)

Breitensuche

Anwendung:

Bestimmung der Entfernung zum Startknoten in einem **ungewichteten** Graphen.

Breitensuche

```
1 BFS()
2   bfsCount = 1;
3   finCount = 1;
4   foreach s ∈ V
5       v->state = initial;
6       v->parent = NULL;
7   foreach (s ∈ V)
8       if s->state == initial
9           BFSvisit(s)
```

Breitensuche - mit Queue

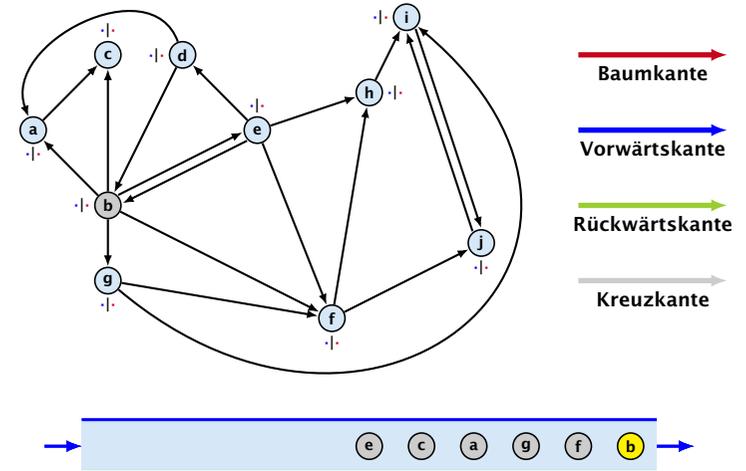
```

1 BFSvisit(s)
2   Queue Q; Q.enqueue(s);
3   while (!Q.empty()) {
4     v = Q.front();
5     if (v->state == initial)
6       v->state = active;
7       v->bfsNum = bfsCount++;
8       foreach x ∈ N(v)
9         // v->parent = x; removed statement
10        Q.enqueue(x);
11   else
12     v->state = finished;
13     v->finNum = finCount++;
14     Q.dequeue();
15 }

```

Wir haben im wesentlichen nur den Stack durch eine Queue ersetzt!

BFS



Beobachtungen

- ▶ **finNum** ist immer gleich **bfsNum** (also überflüssig)
- ▶ keine Vorwärtskanten
- ▶ Klassifizierung von Kreuzkanten und Rückwärtskanten ist nicht so einfach möglich; wird normalerweise bei BFS auch nicht gemacht
- ▶ Komplexität die gleiche wie DFS, da nur ein Stack gegen eine Queue getauscht wurde.

Breitensuche - mit Queue

```

1 BFSvisit(s)
2   Queue Q; Q.enqueue(s); s->dist = 0;
3   while (!Q.empty()) {
4     v = Q.front();
5     if (v->parent) v->dist = v->parent->dist+1;
6     if (v->state == initial)
7       v->state = active;
8       v->bfsNum = bfsCount++;
9       foreach x ∈ N(v)
10        Q.enqueue(x);
11   else
12     v->state = finished;
13     v->finNum = finCount++;
14     Q.dequeue();
15 }

```

Berechnet die Distanz zu **s** im Baum. Für einen BFS-Baum ist dies auch die Distanz zu **s** im **ungewichteten** Graphen.

Beweis:

- ▶ Sei $\text{dist}(x)$ der berechnete Wert, und $\text{rdist}(x)$ die wirkliche Distanz zu s .
- ▶ $\text{rdist}(x) \leq \text{dist}(x)$ folgt, da der Baum einen Pfad zu s der Länge dist codiert.
- ▶ Angenommen es gibt einen Knoten v mit $\text{rdist}(v) < \text{dist}(v)$; sei v ein solcher Knoten mit kleinstem Wert von $\text{dist}(v)$

Die Knoten werden gemäß der Größe ihrer dist -Werte besucht. (warum?)

Sei p_v , der vom Algorithmus gefundene Vorgänger von v ($v \rightarrow \text{parent}$) und sei p'_v der Vorgänger von v auf einem kürzesten s - v Pfad.

- ▶ $\text{dist}(p_v) = \text{rdist}(p_v) = \text{dist}(v) - 1$
- ▶ $\text{dist}(p'_v) = \text{rdist}(p'_v) - 1 < \text{dist}(v) - 1$
- ▶ dann wird aber p'_v zuerst besucht; und dabei würden wir $v \rightarrow \text{parent}$ auf p'_v setzen. Widerspruch.