

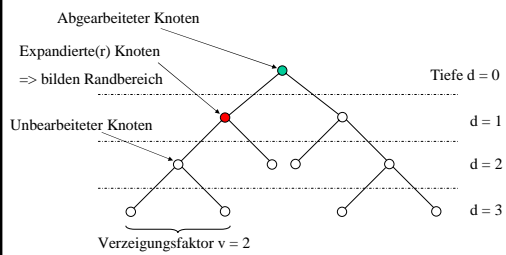
Tiefensuche

Seminar Systementwurf
Ralf Cremerius

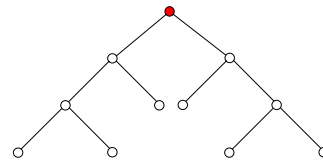
Gliederung

- Teil 1):
⇒ Tiefensuche als effizientes Suchverfahren auf Graphen
- Teil 2):
⇒ Tiefensuche zur Bestimmung der Starken Zusammenhangskomponenten in Graphen

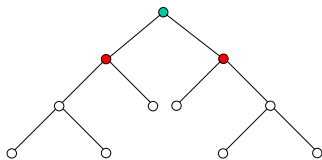
Kurz notiert



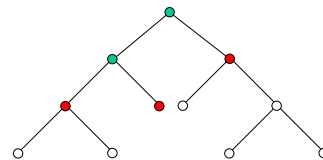
Zur Motivation: Breitensuche



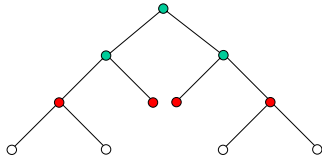
Breitensuche



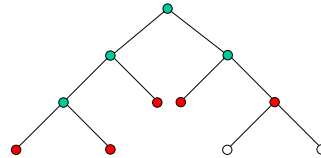
Breitensuche



Breitensuche



Breitensuche



Breitensuche

Vollständig: Für endlichen Verzweigungsgrad

Optimal: Ja

$1+b+b^2+b^3+\dots+(b^{d+1}-b)$ expandierte Knoten

⇒ Zeitkomplexität: $O(v^{d+1})$

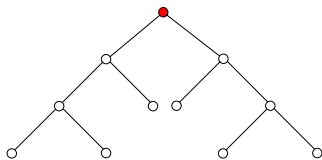
⇒ Speicherkomplexität: $O(v^{d+1})$

Tiefensuche

- Speicheranforderungen der Breitensuche sehr problematisch

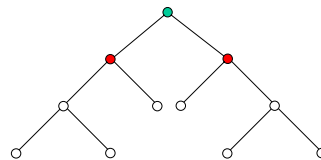
⇒ Tiefensuche verhindert Explosion des Speicherbedarfs konzeptionell

Tiefensuche



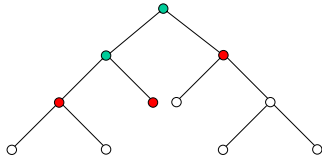
⇒ Expandiert stets tiefste Knoten des Randbereichs

Tiefensuche



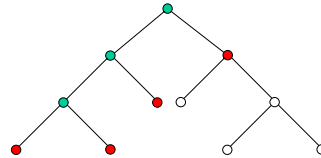
⇒ Expandiert stets tiefste Knoten des Randbereichs

Tiefensuche



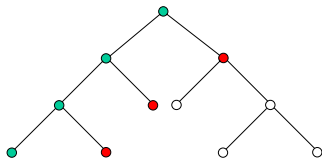
⇒ Expandiert stets tiefste Knoten des Randbereichs

Tiefensuche



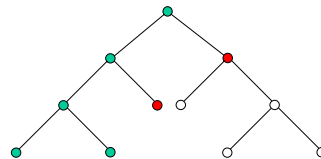
⇒ Expandiert stets tiefste Knoten des Randbereichs

Tiefensuche



⇒ Expandiert stets tiefste Knoten des Randbereichs

Tiefensuche



⇒ Expandiert stets tiefste Knoten des Randbereichs

Tiefensuche

Vollständig: Nicht in unendlichen Graphen

Optimal: Nein

Zeitkomplexität: $O(v^{d_{\max}})$

Expandierte Knoten löschar:

⇒ Speicherkomplexität: $O(vd_{\max})$

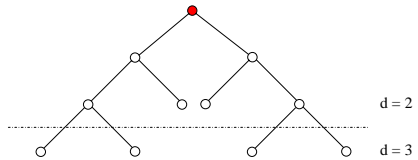
Aber wenn $d_{\max} \rightarrow \infty \dots ?$

Tiefenbeschränkte Suche

- Tiefensuche problematisch auf unendlichen Graphen

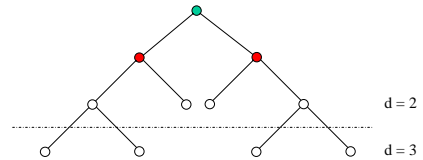
⇒ Einführung der Tiefenbeschränkung:
Tiefensuche + Knoten unterhalb Tiefe
werden nicht expandiert

Tiefenbeschränkte Suche



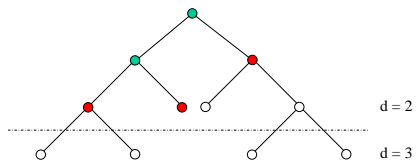
⇒ Knoten unterhalb von d=2 werden ignoriert

Tiefenbeschränkte Suche



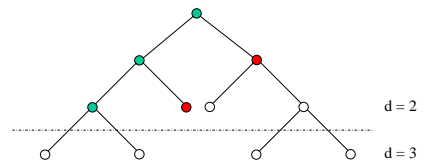
⇒ Knoten unterhalb von d=2 werden ignoriert

Tiefenbeschränkte Suche



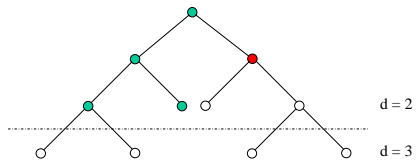
⇒ Knoten unterhalb von d=2 werden ignoriert

Tiefenbeschränkte Suche



⇒ Knoten unterhalb von d=2 werden ignoriert

Tiefenbeschränkte Suche



⇒ Knoten unterhalb von d=2 werden ignoriert

Tiefenbeschränkte Suche

Vollständig: Nein

Optimal: Nein

Zeitkomplexität: $O(v^{d_{max}})$

Speicherkomplexität: $O(v \cdot d_{max})$

⇒ Wie geeignetes d_{max} bestimmen?

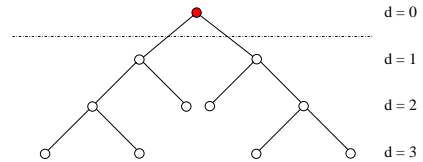
Iterativ vertiefte Tiefensuche

Werden Lösungen mit tiefenbeschränkter Tiefensuche wirklich gefunden?

Idee: Knotenzahl/Tiefenebene wächst exponentiell

⇒ Kosten von iterativer Vertiefung haltbar

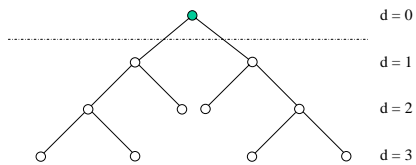
Iterativ vertiefte Tiefensuche



Durchlauf 1: $d=0$

⇒ Knoten unterhalb von $d=0$ werden ignoriert

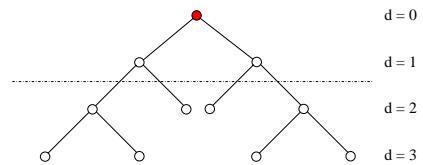
Iterativ vertiefte Tiefensuche



Durchlauf 1: $d=0$

⇒ Knoten unterhalb von $d=0$ werden ignoriert

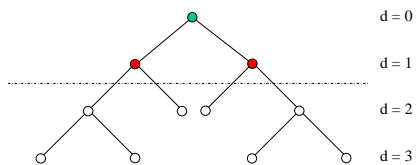
Iterativ vertiefte Tiefensuche



Durchlauf 2: $d=1$

⇒ Knoten unterhalb von $d=1$ werden ignoriert

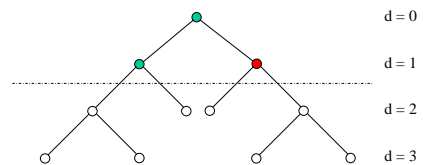
Iterativ vertiefte Tiefensuche



Durchlauf 2: $d=1$

⇒ Knoten unterhalb von $d=1$ werden ignoriert

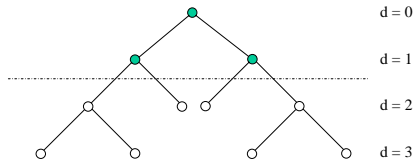
Iterativ vertiefte Tiefensuche



Durchlauf 2: $d=1$

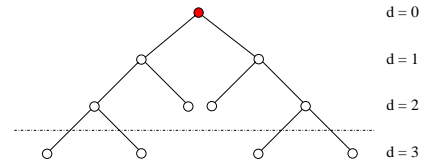
⇒ Knoten unterhalb von $d=1$ werden ignoriert

Iterativ vertiefte Tiefensuche



Durchlauf 2: $d=1$
 \Rightarrow Knoten unterhalb von $d=1$ werden ignoriert

Iterativ vertiefte Tiefensuche



Durchlauf 3: $d=2$
 \Rightarrow Knoten unterhalb von $d=2$ werden ignoriert

Iterativ vertiefte Tiefensuche

Vollständig: Für endlichen Verzweigungsgrad

Optimal: Ja

Zeitkomplexität: $O(v^{d_{\max}})$

Speicherkomplexität: $O(vd_{\max})$

\Rightarrow Komplexitäten unverändert

\Rightarrow Lösungen werden gefunden

Vergleich der Verfahren

	Zeit	Speicher	Vollständig	Optimal
Breitensuche	$O(v^{d+1})$	$O(v^{d+1})$	Ja	Ja
Tiefensuche	$O(v^d)$	$O(vd_{\max})$	Nein	Nein
Tiefenbeschränkte Tiefensuche	$O(v^d)$	$O(vd_{\max})$	Nein	Nein
Iterativ vertiefte Tiefensuche	$O(v^d)$	$O(vd_{\max})$	Ja	Ja

Teil 2

- Problemstellung der Starken Zusammenhangskomponenten in Graphen
- Problemlösung durch ein tiefensuchenbasiertes Verfahren

Starke Zusammenhangskomponenten

Ausgangssituation:

Gerichteter Graph $G=(V,E)$

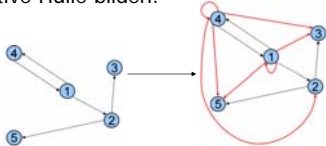
Problem:

Maximale Menge $V' \in V$ finden, so daß

$$\forall u, v \in V': (u, v) \in E^* \wedge (v, u) \in E^*$$

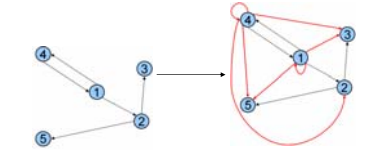
Algorithmus von Warshall - Konzept

Transitive Hülle bilden:



=> Konstruktion von Erreichbarkeitsmatrix:
 $E(i,j) = 1$ / erreichbar bzw. 0 / nicht err.

Algorithmus von Warshall - Verfahren



	1	2	3	4	5
1		X		X	
2				X	X
3					
4	X				
5					

→

	1	2	3	4	5
1		X	X	X	X
2			X		X
3					
4	X	X	X	X	X
5					

Algorithmus von Warshall - Algorithmus

Informeller Algorithmus:

<Anzahl Knoten> Durchläufe
 für jeden Knoten
 für jeden Partnerknoten
 wenn Partnerknoten erreichbar,
 dann trage seine erreichbaren
 Knoten ein

Komplexität in $O(n^3)$, es existieren jedoch
 leistungsfähigere Optimierungen

Idee des Tarjan-Algorithmus

- Tiefensuche in Graph durchführen
- ⇒ Starke Zusammenhangskomponenten sind
 Teilbäume des Tiefensuchbaums!
- Besuchte Knoten werden auf Stack abgelegt
- Mitführen von Tiefensuchindex und Zeiger
- ⇒ Entscheidung, ob ein Knoten Wurzel einer
 Zusammenhangskomponente ist
- Wenn Wurzel => Ausgabe der SZHK.

Vorbereitung

- Stack von Knoten initialisieren
- Zählvariable „i“ initialisieren
- Führen zweier Arrays für die Knoten:
 „Index“ -> Tiefensuchindex
 jeweils = -1 für unbesuchte Knoten
 „Zeiger“ -> Verweis auf höherliegende
 Knoten im Tiefensuchbaum

Vorbereitung

```

Initialisiere()
{
    Stack = {}
    i = 0
    für alle Knoten
        Index[Knoten] = -1
}
    
```

Hauptschleife

- Initialisierung aufrufen
Initialisiere());
- Alle unbesuchten Knoten abarbeiten
für alle Knoten
wenn (Index[Knoten] == -1)
Tarjan(Knoten)

Rekursive Prozedur für Knoten

```
Tarjan( Knoten ) {  
  Zeiger[Knoten] = Index[Knoten] = i = i+1  
  Stack = Stack + { Knoten }  
  für jeden Nachfolger „j“ von „Knoten“:  
    if ( Index[Knoten] == -1 ) {  
      Tarjan(„j“)  
      Zeiger[Knoten] = min(Zeiger[Knoten], Zeiger[j]) }  
    else if ( Stack.enthalten(j) UND Index[Knoten] > Index[j] )  
      Zeiger[Knoten] = min(Zeiger[Knoten], Zeiger[j])  
  if ( Index[Knoten] == Zeiger[Knoten] )  
    new SZhK;  
    solange ( Stack.enthalten(Knoten) == wahr )  
      SZhK = SZhK + { Stack.pop() };  
}
```

Fazit

- Besondere Eigenschaft:
Komplexität in $O(|V| + |E|)$
⇒ Effizientestes Verfahren zur
Bestimmung von SZK
⇒ Heute ein Standard, damals Sensation
⇒ Tarjan später zusammen mit Hopcroft
mit dem Turing-Preis ausgezeichnet

Quellen

- Russell/Norvig – „Künstliche Intelligenz, ein moderner Ansatz“
(2. Auflage)
- Karsten Schmidt - Vorlesungsskript "Computergestützte
Verifikation"
<http://www2.informatik.hu-berlin.de/top/mitarbeiter/schmidt/lehre/cgv/cgv030506b.pdf>
- Ekkart Kindler, Universität Paderborn
- Vorlesungsskript „Model Checking“
<http://wwwcs.uni-paderborn.de/cs/kindler/Lehre/WS04/MC/PDF/Tarjan.pdf>
- Martin Reyher – Ausarbeitung Proseminar
„Algorithmen auf Graphen“:
Finden von (starken) Zusammenhangskomponenten und
Erkennen von Brücken
<http://fuzzy.cs.uni-magdeburg.de/studium/graph/txt/reyher.pdf>

Quellen – Teil 2

- Dokument „TI-Vorbereitung“ von Oliver Schwarz
<http://www-user.tu-chemnitz.de/~olivs/gs/TI/VorbereitungTI.pdf>
- Seminarvortrag über Robert Tarjan von Matthias Bender an
Universität des Saarlandes:
<http://www.virtosphere.de/schillo/teaching/WS2001/Vortraege/Tarjan.pdf>
- Bachelorarbeit „Algorithmen für endliche Automaten mit
Ausgabe“ von Michael Thomas an der Universität Hannover:
<http://www.thi.uni-hannover.de/forschung/arbeiten/daten/thomas-ba.pdf>
- Vorlesung „Praktische Informatik: Datenstrukturen“ von Martin
Rammerstorfer an der Johannes Kepler Universität Linz:
<http://www.ssw.uni-linz.ac.at/Teaching/Lectures/PID/2005/Download/Folien08.pdf>
- Volker Turau - „Algorithmische Graphentheorie“, Addison-
Wesley, 1996, S.95-100