Script generated by TTT

Title: TÃ¤ubig: GAD (04.07.2013)

Date: Thu Jul 04 12:03:09 CEST 2013

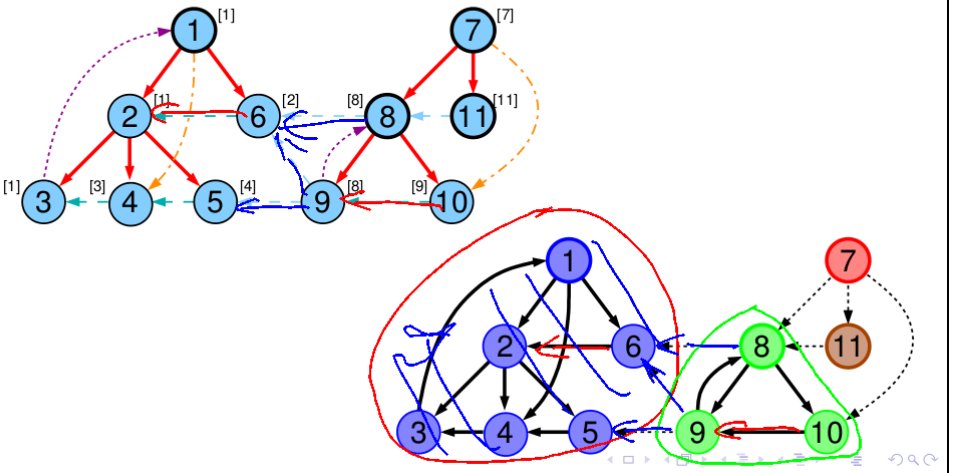
Duration: 45:38 min

Pages: 19

Starke Zhk. und DFS / Variante 1

Modifizierte DFS nach R. E. Tarjan

- Knoten v ist genau dann Wurzel einer starken Zusammenhangskomponente, wenn $\texttt{num}[v]=\texttt{low}[v]$



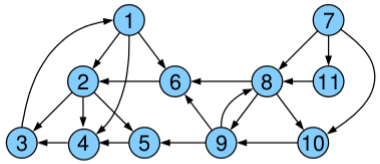
Starke Zhk. und DFS / Variante 2

Idee:

- beginne mit Graph ohne Kanten, jeder Knoten ist eigene SCC
- füge nach und nach einzelne Kanten ein

$\Rightarrow$ aktueller (current) Graph $G_c = (V, E_c)$

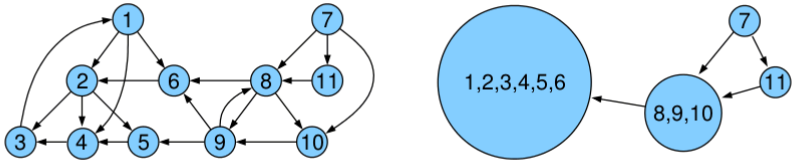
- Update der starken Zusammenhangskomponenten (SCCs)



Starke Zhk. und DFS / Variante 2

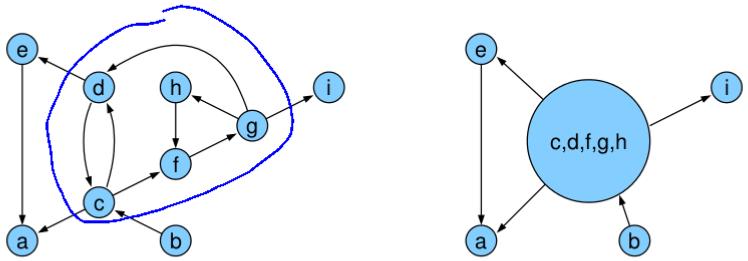
Idee:

- betrachte geschrumpften (shrunken) Graph G_c^s: Knoten entsprechen SCCs von G_c, Kante (C, D) genau dann, wenn es Knoten $u \in C$ und $v \in D$ mit $(u, v) \in E_c$ gibt
- geschrumpfter Graph G_c^s ist ein DAG
- Ziel: Aktualisierung des geschrumpften Graphen beim Einfügen



Starke Zhk. und DFS / Variante 2

Geschrumpfter Graph
(Beispiel aus Mehlhorn / Sanders)



Starke Zhk. und DFS / Variante 2

Update des geschrumpften Graphen nach Einfügen einer Kante:

3 Möglichkeiten:

- beide Endpunkte gehören zu derselben SCC

$\Rightarrow$ geschrumpfter Graph unverändert

- Kante verbindet Knoten aus zwei verschiedenen SCCs, aber schließt keinen Kreis

$\Rightarrow$ SCCs im geschrumpften Graph unverändert, aber eine Kante wird im geschrumpften Graph eingefügt (falls nicht schon vorhanden)

- Kante verbindet Knoten aus zwei verschiedenen SCCs und schließt einen oder mehrere Kreise

$\Rightarrow$ alle SCCs, die auf einem der Kreise liegen, werden zu einer einzigen SCC verschmolzen

Starke Zhk. und DFS / Variante 2

Prinzip:

- Tiefensuche
  - V_c schon markierte (entdeckte) Knoten
  - E_c schon gefundene Kanten
- 3 Arten von SCC: unentdeckt, offen, geschlossen
- unentdeckte Knoten haben Ein- / Ausgangsgrad Null in G_c

$\Rightarrow$ zunächst bildet jeder Knoten eine eigene unentdeckte SCC, andere SCCs enthalten nur markierte Knoten

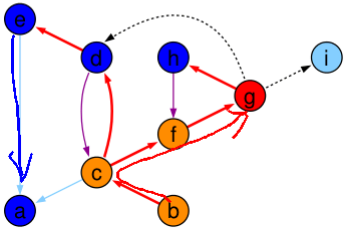
- SCCs mit mindestens einem aktiven Knoten (ohne finishNum) heißen offen
- SCC heißt geschlossen, falls sie nur fertige Knoten (mit finishNum) enthält
- Knoten in offenen / geschlossenen SCCs heißen offen / geschlossen

Starke Zhk. und DFS / Variante 2

- Knoten in geschlossenen SCCs sind immer fertig (mit finishNum)
- Knoten in offenen SCCs können fertig oder noch aktiv (ohne finishNum) sein
- Repräsentant einer SCC: Knoten mit kleinster dfsNum

Starke Zhk. und DFS / Variante 2

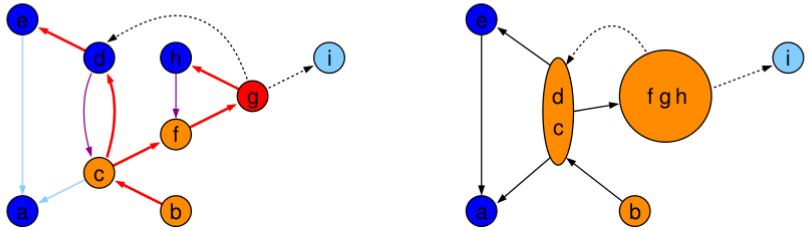
DFS-Snapshot:



- erste DFS startete bei Knoten a, zweite bei b
- aktueller Knoten ist g, auf dem Rekursionsstack liegen b, c, f, g
- (g, d) und (g, i) wurden noch nicht exploriert
- (d, c) und (h, f) sind Rückwärtskanten
- (c, a) und (e, a) sind Querkanten
- (b, c), (c, d), (d, e), (c, f), (f, g) und (g, h) sind Baumkanten

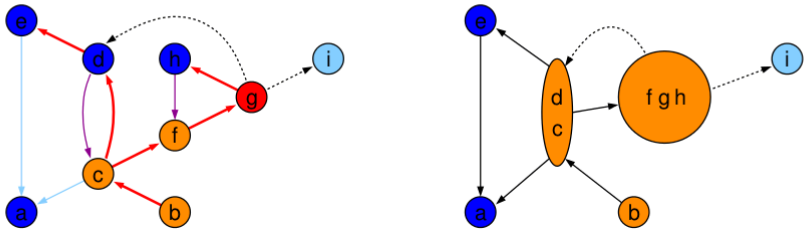
Starke Zhk. und DFS / Variante 2

DFS-Snapshot mit geschrumpftem Graph:



- unentdeckt: $\{i\}$ offen: $\{b\}$, $\{c, d\}$, $\{f, g, h\}$ geschlossen: $\{a\}$, $\{e\}$
- offene SCCs bilden Pfad im geschrumpften Graph
- aktueller Knoten gehört zur letzten SCC
- offene Knoten wurden in Reihenfolge b, c, d, f, g, h erreicht und werden von den Repräsentanten b, c und f genau in die offenen SCCs partitioniert

Starke Zhk. und DFS / Variante 2



Beobachtungen (Invarianten für G_c):

1. Pfade aus geschlossenen SCCs führen immer zu geschlossenen SCCs
2. Pfad zum aktuellen Knoten enthält die Repräsentanten aller offenen SCCs

   offene Komponenten bilden Pfad im geschrumpften Graph
3. Knoten der offenen SCCs in Reihenfolge der DFS-Nummern werden durch Repräsentanten in die offenen SCCs partitioniert

Starke Zhk. und DFS / Variante 2

Geschlossene SCCs von G_c sind auch SCCs in G:

- Sei v geschlossener Knoten und S / S_c seine SCC in G / G_c.
- zu zeigen: $S = S_c$
- G_c ist Subgraph von G, also $S_c \subseteq S$
- somit zu zeigen: $S \subseteq S_c$
- Sei w ein Knoten in S.

$\Rightarrow$ $\exists$ Kreis C durch v und w.

- Invariante 1: alle Knoten von C sind geschlossen und somit erledigt (alle ausgehenden Kanten exploriert)
- C ist in G_c enthalten, also $w \in S_c$
- damit gilt $S \subseteq S_c$, also $S = S_c$

Starke Zhk. und DFS / Variante 2

Vorgehen:

- Invarianten 2 und 3 helfen bei Verwaltung der offenen SCCs
- Knoten in offenen SCCs auf Stack oNodes
  (in Reihenfolge steigender dfsNum)
- Repräsentanten der offenen SCCs auf Stack oReps
- zu Beginn Invarianten gültig (alles leer)
- vor Markierung einer neuen Wurzel sind alle markierten Knoten erledigt, also keine offenen SCCs, beide Stacks leer
  dann: neue offene SCC für neue Wurzel s,
  s kommt auf beide Stacks

Starke Zhk. und DFS / Variante 2

Geschlossene SCCs von G_c sind auch SCCs in G:

- Sei v geschlossener Knoten und S / S_c seine SCC in G / G_c.
- zu zeigen: $S = S_c$
- G_c ist Subgraph von G, also $S_c \subseteq S$
- somit zu zeigen: $S \subseteq S_c$

- Sei w ein Knoten in S.

$\Rightarrow$ $\exists$ Kreis C durch v und w.

- Invariante 1: alle Knoten von C sind geschlossen und somit erledigt (alle ausgehenden Kanten exploriert)
- C ist in G_c enthalten, also $w \in S_c$
- damit gilt $S \subseteq S_c$, also $S = S_c$

Starke Zhk. und DFS / Variante 2

Geschlossene SCCs von G_c sind auch SCCs in G:

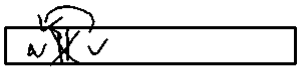
- Sei v geschlossener Knoten und S / S_c seine SCC in G / G_c.
- zu zeigen: $S = S_c$
- G_c ist Subgraph von G, also $S_c \subseteq S$
- somit zu zeigen: $S \subseteq S_c$

- Sei w ein Knoten in S.

$\Rightarrow$ $\exists$ Kreis C durch v und w.

- Invariante 1: alle Knoten von C sind geschlossen und somit erledigt (alle ausgehenden Kanten exploriert)
- C ist in G_c enthalten, also $w \in S_c$
- damit gilt $S \subseteq S_c$, also $S = S_c$

Starke Zhk. und DFS / Variante 2



Prinzip: betrachte Kante $e = (v, w)$

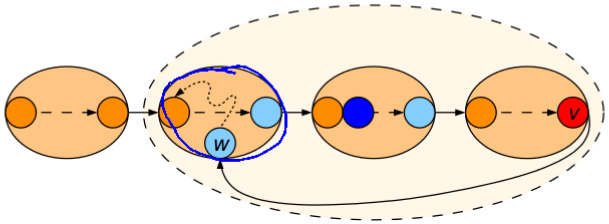
- Kante zu unbekanntem Knoten w (Baumkante):
  neue eigene offene SCC für w (w kommt auf oNodes und oReps)
- Kante zu Knoten w in geschlossener SCC (Nicht-Baumkante):
  von w gibt es keinen Weg zu v, sonst wäre die SCC von w noch nicht geschlossen (geschlossene SCCs sind bereits komplett), also SCCs unverändert
- Kante zu Knoten w in offener SCC (Nicht-Baumkante):
  falls v und w in unterschiedlichen SCCs liegen, müssen diese mit allen SCCs dazwischen zu einer einzigen SCC verschmolzen werden (durch Löschen der Repräsentanten)

Wenn Knoten keine ausgehenden Kanten mehr hat:

- Knoten fertig
- wenn Knoten Repräsentant seiner SCC ist, dann SCC schließen

Starke Zhk. und DFS / Variante 2

Vereinigung offener SCCs im Kreisfall:



- offene SCC entsprechen Ovalen, Knoten sortiert nach dfsNum
- alle Repräsentanten offener SCCs liegen auf Baumpfad zum aktuellen Knoten v in SCC S_k
- Nicht-Baumkante (v, w) endet an Knoten w in offener SCC S_i mit Repräsentant r_i
- Pfad von w nach r_i muss existieren (innerhalb SCC S_i)

$\Rightarrow$ Kante (v, w) vereinigt $S_i, \ldots, S_k$

Starke Zhk. und DFS / Variante 2

- init() {
  component = new int[n];
  oReps = ⟨ ⟩;
  oNodes = ⟨ ⟩;
  dfsCount = 1;
  }

- root(Node w) / traverseTreeEdge(Node v, Node w) {
  oReps.push(w); // Repräsentant einer neuen ZHK
  oNodes.push(w); // neuer offener Knoten
  dfsNum[w] = dfsCount;
  dfsCount++;
  }

Starke Zhk. und DFS / Variante 2

- traverseNonTreeEdge(Node v, Node w) {
  if (w ∈ oNodes) // verschmelze SCCs
  while (dfsNum[w] < dfsNum[oReps.top()])
  oReps.pop();
  }

- backtrack(Node u, Node v) {
  if (v == oReps.top()) { // v Repräsentant?
  oReps.pop(); // ja: entferne v
  do { // und offene Knoten bis v
  w = oNodes.pop();
  component[w] = v;
  } while (w!=v);
  }
  }

Starke Zhk. und DFS / Variante 2

Zeit: $O(n+m)$

Begründung:

- init, root: $O(1)$
- traverseTreeEdge: $(n-1) \times O(1)$
- backtrack, traverseNonTreeEdge:
  da jeder Knoten höchstens einmal in oReps und oNodes landet,
  insgesamt $O(n+m)$
- DFS-Gerüst: $O(n+m)$
- gesamt: $O(n+m)$