Zusammenfassung Platzeffiziente Alg.

Tim Baumann, http://timbaumann.info/uni-spicker

Ziel. Algorithmen entwerfen, die wenig Speicherplatz und Speicherzugriffe benötigen, aber trotzdem schnell sind.

Problem (Erreichbarkeit)**.** Gegeben sei ein gerichteter oder ungerichteter Graph, ein Startknoten und ein Zielknoten darin. Frage: Ist der Zielknoten vom Startknoten erreichbar?

Algorithmus. Algorithmen, mit denen man Problem lösen kann, sind Breiten- und Tiefensuche.

Lem. Es sei ein Graph mit n Knoten und m Kanten gegeben. Tiefensuche benötigt $\Theta(n+m)$ Zeit und $\Theta(n \log n)$ Speicherplatz.

Algorithmus (Savitch)**.**

function $\text{sREACHABLE}(u, v, k)$
  if $u = v$ **then return** true
  if $k = 0$ **then return** false
  if $(u, v) \in E$ **then return** true
  if $k = 1$ **then return** false
  for $x \in V$ **do**
    if $\text{REACHABLE}(u, x, \lfloor \frac{k}{2} \rfloor) \wedge \text{REACHABLE}(x, v, \lceil \frac{k}{2} \rceil)$ **then**
      return true
  return false
return reachable(s,t,n-1)

Lem. Savitch's Algorithmus löst das Erreichbarkeits-Problem in $\mathcal{O}((\log n)^2)$ Speicherplatz.

Bem. Die Laufzeit von Savitch's Alg. ist allerdings sehr schlecht, im schlechtesten Fall $O(n \cdot \log n)$.

Bem. Eine Darstellung eines Graphen als Adjazenzmatrix benötigt $O(n^2)$, eine Darstellung als Adjazenzliste/-array $O(m \cdot \log n)$ Bits. Manchmal ist es nützlich, zusätzlich Rückwärtskanten oder Aus- und Ingrad von Knoten zu speichern, um diese Informationen nicht mehrmals berechnen zu müssen. Bei bestimmten Algorithmen werden sie auch als gegeben angenommen.

Konvention. Wir werden folgende Graphfunktionen benutzen:

Funktion	Ergebnis
adjfirst : $V \to P$	ersten Eintrag in der Adjazenzliste
adjhead : $P \to V$	Knoten zum Eintrag in der Adjazenzliste
adjnext : $P \to P$	nächsten Eintrag in der Adjazenzliste
deg : $V \to \mathbb{N}$	Ausgrad eines Knoten
head : $A \to V$	den k-ten Nachbar eines Knoten
tail : $B \to V$	den k-ten In-Nachbar eines Knoten
mate : $A \to A$	den „Mate" einer Kante (bei unger. Graphen)

wobei $A := \{(v, k) \in V \times \mathbb{N} \mid 1 \leq k \leq \deg(v)\}$
$B := \{(v, k) \in V \times \mathbb{N} \mid 1 \leq k \leq \text{indeg}(v)\}$

Algorithmus. Bei einer Tiefensuche in einem Graphen wird am meisten Platz für den Laufzeitstack verbraucht. Um diesen Platz zu optimieren, ist es geschickt, zunächst den Algorithmus mit explizitem Keller aufzuschreiben:

function $\text{PROCESS}(u)$
  $S \Leftarrow (u, \text{ADJFIRST}(u))$
  while $S \neq \emptyset$ **do**
    $(u, p) \Leftarrow S$
    $\mathit{color}[u] := \mathit{gray}$
    $\text{PREPROCESS}(u)$
    if $p \neq \mathit{null}$ **then**
      $S \Leftarrow (u, \text{ADJNEXT}(()p))$
      $v := \text{ADJHEAD}(()u, p)$
      $\text{PREEXPLORE}(()u, v, \mathit{color}[v])$
      if $\mathit{color}[v] = \mathit{white}$ **then**
        $S \Leftarrow (v, \text{ADJFIRST}(()v))$
      $\text{POSTEXPLORE}(()u, v)$
    else
      $\text{PREPROCESS}(u)$
      $\mathit{color}[u] := \mathit{black}$