(ii) Zu jedem Zeitpunkt ist (R,T) ein s enthaltender Baum, denn (a) alle Knoten in R sind von s aus erreichbar (und umgekehrt)

(b) neu eingefügte Kanten verbinden die bisherige Knotenmenge R nur mit bislang nicht erreichbaren Knoten, können also keinen Kreis schließen.

Angenommen, am Ende gibt es einen Knoten $w \in V \setminus R$, der von s aus erreichbar ist.

Sei P ein s-w-Pfad, und sei $\{x,y\}$ eine Kante von P mit $x \in R$, $y \notin R$.

Da x zu R gehört, wurde x auch zu Q hinzugefügt. Der Algorithmus stoppt aber nicht, bevor x aus Q entfernt wurde. Dies wird aber in (3) nur vorgenommen, wenn es keine Kante $\{x,y\}$ mit $y \notin R$ gibt

– im Widerspruch zur Annahme. □

3.3 Tiefensuche, Breitensuche; Stapel, Warteschlange

Wie wird in (2) die Auswahl eines Knotens $v \in Q$ vorgenommen?

Beobachtung: Die Art, wie Q abgespeichert und abgearbeitet wird, hat eine unmittelbare Auswirkung auf den Ablauf des Algorithmus!

Zwei häufigste Möglichkeiten:

(B) Die zuerst aufgenommenen Knoten werden zuerst abgearbeitet, d.h. die Suche geht jeweils erst einmal in die Breite, bevor weiter entfernte Knoten erledigt werden.

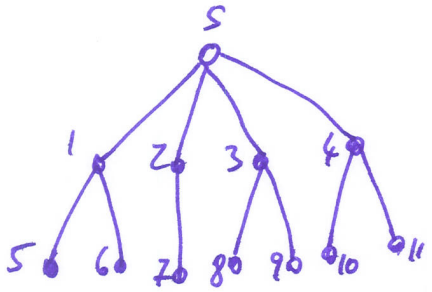
→ „Breitensuche" oder Breadth-First Search (BFS)

(D) Die zuletzt aufgenommenen Knoten werden zuerst abgearbeitet, d.h. die Suche geht jeweils erst einmal in die Tiefe, bevor weitere Knoten und deren Nachfolger erledigt werden.

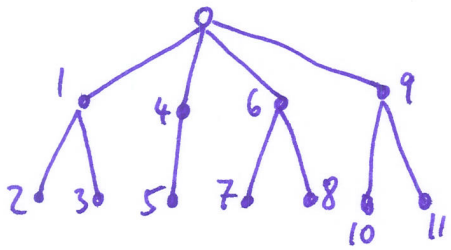
→ „Tiefensuche" oder Depth-First Search (DFS)

Bildlich: Selbst derselbe Baum wird unterschiedlich abgearbeitet:

(B)



(D)



Für Komplizierteres Graphen ist das noch deutlicher

- ein BFS-Baum ist tendenziell „breit und flach",
  ein DFS-Baum dagegen „schmal und tief".

Anwendungen:

(B) Breitensuche liefert kürzeste Wege von einer Quelle aus (→ später mehr)

(D) Tiefensuche liefert eine zusammenhängende Suche nach einem versteckten Objekt (→ z.B. Weg aus Labyrinth)

Etwas pointierter (und nicht zu ernst zu nehmen):

(B) Parallele, „kooperative" Strategie (viele Sammler bzw. Frauen)

(D) Individuelle Strategie (ein Jäger bzw. Mann)

Datenstrukturen:

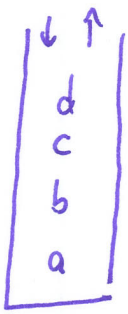
(B) Warteschlange oder „Queue", bzw.

„FIFO"-Regel: First In, First Out

Eingang → _ e d c b a, → Ausgang

(D) Stapel oder „Stack" (auch „Keller"), bzw.

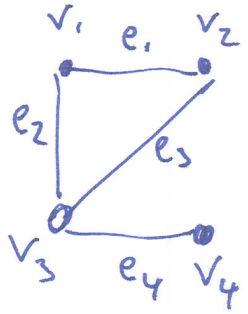
„LIFO"-Regel: Last In, First Out



3.4 Datenstrukturen für Graphen

Wie beschreibt man einen Graphen?

(1) Adjazenzmatrix



$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

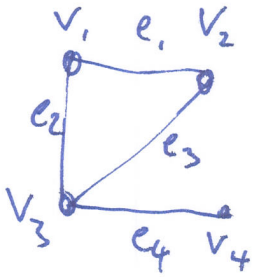
(„adjazent": verbunden)

Also: $A \in \{0,1\}^{n \times n}$

mit $a_{v,w} := \begin{cases} 1 & \text{für } \{v,w\} \in E \\ 0 & \text{sonst} \end{cases}$

Größe: n^2 für einen Graphen mit n Knoten.

(2) Inzidenzmatrix



$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

(„inzident": zusammentreffend)

Also: $A \in \{0,1\}^{n \times m}$

mit $a_{v,e} := \begin{cases} 1 & \text{für } v \in e \\ 0 & \text{sonst} \end{cases}$

Größe: nm für einen Graphen mit n Knoten, m Kanten. (i.d.R. viele Nullen!)

Wenn man tatsächlich Algorithmen laufen lassen will, braucht man Information zu einzelnen Knoten und muss diese Finden - dann sollte man noch etwas mehr Platz aufwenden!

Das tut die

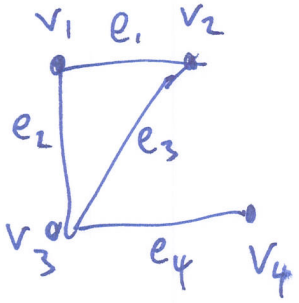
Adjazenzliste:

v_1 : v_2, v_3, v_4
v_2 : v_1, v_3, v_4
v_3 : v_1, v_2, v_4, v_5
v_4 : v_1, v_2, v_3, v_5
v_5 : v_3, v_4

Zugriff auf die einzelnen Teillisten mit n Pointern, benötigt $\mathcal{O}(n \log m)$ zusätzliche Bits.

Gesamtaufwand: $O(n \log m + m \log n)$.

(3) Kantenliste:

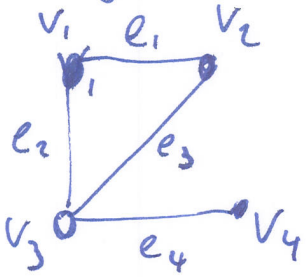


$\{v_1, v_2\}, \{v_1, v_3\}, \{v_2, v_3\}, \{v_3, v_4\}$

Benötigt wird Kantennummerierung!

Also $\log n$ für jeden Index,
insgesamt Platz in der Größenordnung $m \log n$ → später mehr!
von

(4) Adjazenzliste:



$v_1 : v_2, v_3$
$v_2 : v_1, v_3$
$v_3 : v_1, v_2, v_3$
$v_4 : v_4$

Benötigt etwas mehr Platz als die Kantenliste, ist aber u.U. praktischer im Kontext von Graphenalgorithmen (wo man z.B. Nachbarn für bestimmte Knoten sucht, die man nicht erst mühsam aus einer Liste herraussuchen will!)
Zugriff auf die einzelnen Teillisten mit n Pointern, benötigt $O(n \log m)$ zusätzliche Bits.

→ Insgesamt $O(n \log m + m \log n) = O(m \log n)$