ALGORITHMEN ZUR SUCHE IN GRAPHEN (V)

Minimale Spannbäume:

ALGORITHMUS	***Kruskal***	
Input:	Adjazenzliste:	TAdjazenzliste *{ ungerichteter Graph }*
Output:	Spannbaum:	Liste of TKante = (von, nach, distanz)
Lokal:	Kanten:	array[1..n^2] of TKante *{ n(n-1)/2 würde reichen }*
	Vorgänger:	array[1..n] of integer
	Knotennummer:	integer
	Kantennummer:	integer
	Kante:	TKante

{ Initialisierung }
- Kanten ← *alle Kanten der Adjazenzliste*
- *Sortiere Kanten nach aufsteigender Gewichtung { z. B. mit Quicksort }*
- Spannbaum ← *„neue Liste“* *{ hier stehen Später alle benötigten Spannbaumkanten drin }*
- Für Knotennummer ← 1 bis n *{ noch gehört kein Knoten einer Komponente an }*

  tue: • Vorgänger[Knotennummer] ← −1

{ Algorithmus }
- Für alle Kantennummer *{ d. h. durchlaufe alle Kanten mit aufsteigender Gewichtung }*

  tue: • Kante ← Kanten[Kantennummer]
  - Falls ***erster_Vorgänger***(Kante.von) ≠ ***erster_Vorgänger*** (Kante.nach)

    dann: • Vorgänger[***erster_Vorgänger***(Kante.von)] ← ***erster_Vorgänger*** (Kante.nach)
    - Spannbaum.*Einfügen*(Kante)

Hilfsfunktion zur Bestimmung der Zusammenhangskomponente:

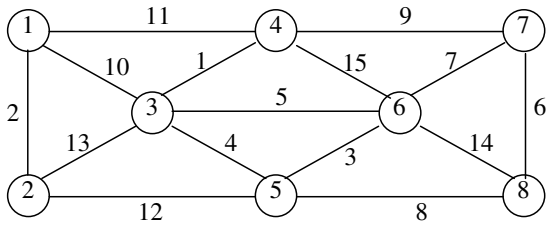
ALGORITHMUS	***erster_Vorgänger***		
Input	Knotennummer:	integer	
Output:	erster_Vorgänger:	integer	
Lokal:	MinDistanz:	integer	
	Knoten:	integer	*{ durchläuft alle mögl. Knoten }*

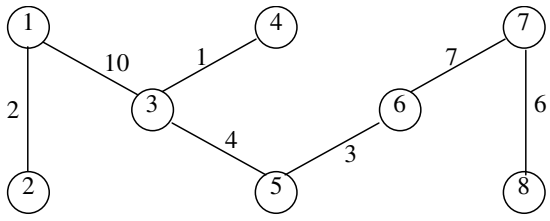
- Solange Vorgänger[Knotennummer] ≠ −1

  tue: • Knotennummer ← Vorgänger[Knotennummer]
- erster_Vorgänger ← Knotennummer

Die Entwicklung des Arrays **Vorgänger**, mit welcher die Zusammenhangskomponenten festgelegt sind, sieht bei rechts abgebildeten Graphen wie folgt aus:

Knoten / **Kante**	1	2	3	4	5	6	7	8
Initialisierung	−1	−1	−1	−1	−1	−1	−1	−1
3 → 4			4					
1 → 2	2							
5 → 6					6			
3 → 5				6				
3 → 6								
7 → 8							8	
6 → 7						8		
5 → 8								
4 → 7								
1 → 3		8						
1 → 4								
...	es tut sich allerdings nichts mehr							





ALGORITHMEN ZUR SUCHE IN GRAPHEN (IV)

Minimale Spannbäume:

ALGORITHMUS ***Prim***

Input:	Matrix:	TAdjazenzmatrix	*{ Array[1..n,1..n] of integer }*
	Startknoten:	integer	
Output:	Distanz:	array[1..n] of integer	
	Vorgaenger:	array[1..n] of integer	*{ darüber lässt sich der Spannbaum rekonstruieren }*
Lokal:	Erledigt:	array[1..n] of boolean	
	MinKnoten:	integer	*{ hat kleinste Distanz zur Zusammenhangskomponente }*
	Knoten:	integer	*{ durchläuft alle möglichen Knoten }*

{ Initialisierung }
- Für Knoten ← 1 bis n
  - tue:
    - Distanz[Knoten] ← maxInt
    - Vorgaenger[Knoten] ← −1
    - Erledigt[Knoten] ← false
- Distanz[Startknoten] ← 0

{ Algorithmus }
- Wiederhole n mal: *{ Der Startknoten ist ja bereits erledigt }*
  - MinKnoten ← ***Knoten_mit_kleinster_Distanz***
  - Falls MinKnoten ≠ −1 *{ d. h. es existiert ein Knoten mit minimaler Distanz }*
    - dann:
      - Erledigt[MinKnoten] ← true
      - Für Knoten ← 1 bis n *{ für alle möglichen Nachbarknoten }*
        - tue:
          - Falls NICHT Erledigt[Knoten] UND Matrix[MinKnoten, Knoten] < Distanz[Knoten]
            - dann:
              - Distanz[Knoten] ← Matrix[MinKnoten, Knoten]
              - Vorgaenger[Knoten] ← MinKnoten

Hilfsfunktion zur Ermittlung des Knotens mit minimaler Entfernung zur Zusammenhangskomponente:

ALGORITHMUS ***Knoten_mit_kleinster_Distanz***

Output:	MinKnoten:	integer	*{ hat kleinste Distanz zur Zusammenhangskomponente }*
Lokal:	MinDistanz:	integer	
	Knoten:	integer	*{ durchläuft alle mögl. Knoten }*

- MinDistanz ← maxInt *{ erst mal von größtmöglicher Distanz ausgehen }*
- MinKnoten ← −1
- Für Knoten ← 1 bis n
  - tue:
    - Falls NICHT Erledigt[Knoten] UND Distanz[Knoten] < MinDistanz
      - dann:
        - MinDistanz ← Distanz[Knoten]
        - MinKnoten ← Knoten

Die Entwicklung der Arrays **Vorgänger/Distanz** sieht wie folgt aus:

Knoten / Durchlauf	1	2	3	4	5	6	7
Initialisierung	**0/–1**	---	---	---	---	---	---
1: Minknoten: 1		4/1	11/1	**1/1**			**9/1**
2: Minknoten: 4			**7/4**		**2/4**	6/4	
3: Minknoten: 5		**3/5**				**5/5**	
4: Minknoten: 2							
5: Minknoten: 6							
6: Minknoten: 3							
7: Minknoten: 7