ALGORITHMEN ZUR SUCHE IN GRAPHEN (II)

Zusammenhang:

ALGORITHMUS ***Warshall***

Input:	Matrix:	TAdjazenzmatrix	*{ Array[1..n,1..n] of integer }*
Output:	Zusammenhang:	TAdjazenzmatrix	
Lokal:	Start, Ziel, Mitte:	integer	

{ Initialisierung }

- **Zusammenhang** ← **Matrix**
- Bringe **Zusammenhang** (falls Entfernungsmatrix) eventuell auf die gewünschte Form:
  Zusammenhang[x,y] = 1 ⇔ Es existiert Kante von Knoten x nach Knoten y
  Zusammenhang[x,y] = 0 ⇔ Es existiert keine Kante von Knoten x nach Knoten y

{ Algorithmus }

- Für **Mitte** von 1 bis n *{ d. h. durchlaufe mit Mitte alle Knoten }*
  tue: • Für **Start** von 1 bis n *{ d. h. durchlaufe mit Start alle Knoten }*
  tue: • Falls **Zusammenhang**[**Start**, **Mitte**] = 1 *{ d. h. es existiert Kante S→M }*
  dann: • Für **Ziel** von 1 bis n *{ d. h. durchlaufe mit Ziel alle Knoten }*
  tue: • Falls **Zusammenhang**[**Mitte**, **Ziel**] = 1 *{ d. h. es ex. M→Z }*
  UND **Start** ≠ **Ziel**
  dann: • **Zusammenhang**[**Start**, **Ziel**] ← 1

kürzeste Wege:

ALGORITHMUS ***Floyd***

Input:	Matrix:	TAdjazenzmatrix	*{ Array[1..n,1..n] of integer }*
Output:	Entfernung:	TAdjazenzmatrix	
Lokal:	Start, Ziel, Mitte:	integer	

{ Initialisierung }

- **Entfernung** ← **Matrix**
  Entfernung[x,y] = maxInt ⇔ Es existiert keine Kante von Knoten x nach Knoten y

{ Algorithmus }

- Für **Mitte** von 1 bis n *{ d. h. durchlaufe mit Mitte alle Knoten }*
  tue: • Für **Start** von 1 bis n *{ d. h. durchlaufe mit Start alle Knoten }*
  tue: • Für **Ziel** von 1 bis n *{ d. h. durchlaufe mit Ziel alle Knoten }*
  tue: • Falls **Entfernung**[**Start**, **Mitte**]
  + **Entfernung**[**Mitte**, **Ziel**] < **Entfernung**[**Start**, **Ziel**]
  UND **Start** ≠ **Ziel**
  dann: • **Entfernung**[**Start**, **Ziel**] ←
  Entfernung[**Start**, **Mitte**] + **Entfernung**[**Mitte**, **Ziel**]

Hinweis: Speichert man sich in der Matrix **Entfernung** noch jeweils den *Zwischenpunkt* (**Mitte**), über den man die Verkürzung des Weges von **Start** nach **Ziel** erreicht hat, so ist auch eine Rekonstruktion des kürzesten Weges im Nachhinein möglich:

ALGORITHMUS ***Rekonstruktion***

Input:	Entfernung:	TAdjazenzmatrix	*{ mit Zwischenpunkten }*
	Start, Ziel:	integer	*{ kürzester Weg S→Z ? }*
In-/Output:	Ergebniskanten:	TLinList	*{ über TKantenelement }*
Lokal:	Mitte:	integer	

- **Mitte** ← **Entfernung**[**Start**, **Ziel**].*Zwischenpunkt*
- Wenn **Mitte** = −1 *{ d. h. es gibt keinen Zwischenpunkt }*
  dann: • Ergebniskanten.*Einfuegen*(**Start** − **Ziel**)
  sonst: • ***Rekonstruktion***(**Entfernung**, **Start**, **Mitte**, **Ergebniskanten**)
  • ***Rekonstruktion***(**Entfernung**, **Mitte**, **Ziel**, **Ergebniskanten**)