Algorithmus für die erweiterte Breitensuche

public void *breitensucheErweitert*(String *startKnoten*, String *zielKnoten*)

<pre>int startNummer = getKnotenNummer(startKnoten);</pre>	(bestimme die Knotennummer des Startknotens und speichere diese unter startNummer ab)
int zielNummer = getKnotenNummer(zielKnoten);	(bestimme die Knotennummer des Zielknotens und speichere diese unter zielNummer ab)
int aktuellerKnoten = startNummer;	(setze den aktuellen Knoten auf die startNummer)
anzahlWarteliste = 0;	(setze die Anzahl der Knoten in der Warteliste auf 0)
anzahlBesucht = 0;	(setze die Anzahl der besuchten Knoten auf 0)
setze die Warteliste zurück	(alle Indixe im Feld warteliste erhalten den Wert -1)
setze die Besuchtliste zurück	(alle Indixe im Feld besucht erhalten den Wert false)
setze das Distanz-Feld zurück	(alle Indixe im Feld distanz erhalten den Wert 0)
setze das kommtVon-Feld zurück	(alle Indixe im Feld kommtVon erhalten den Wert -1)
warteliste[0] = aktuellerKnoten;	(der aktuelle Knoten (= Startknoten) wird in die leere Warteliste an erster Stelle eingefügt)
kommtVon[startNummer] = startNummer;	(am Index startNummer im Feld kommtVon wird die startNummer gespeichert; daran erkennen wir, dass es
	der Startknoten ist, da er keinen Vorgängerknoten besitzt)
anzahlWarteliste++;	(die Anzahl der in der Warteliste eingefügten Knoten wird um 1 erhöht)
wiederhole solange der aktuelleKnoten ungleich der	zielNummer ist und der aktuelleKnoten ungleich -1 ist
zähle i von 0 bis anzahlKnoten - 1	
matrix[i	aktuellerKnoten][i] > 0 (es existiert eine Kante zwischen aktuellerKnoten und i)
	und der Knoten mit der Knotennummer i wurde noch nicht besucht
	und der Knoten mit der Knotennummer i ist noch nicht
	in der Warteliste
wahr	falsch
warteliste[anzahlWarteliste] = i;	
(Knoten mit Knotennummer i wird in die Wa	rteliste hinten eingefügt)
anzahlWarteliste++;	
distanz[i] = distanz[aktuellerKnoten] + matri	x[aktuellerKnoten][i];
/ do Bido do monocolo de la composición	- Martine California Martine California
aktuellerKnoten und dem Knoten i addiert)	Knotens wird das Kantengewicht zwischen
· · · · · · · · · · · · · · · · · · ·	
kommtVon[i] = aktuellerKnoten;	
(Knoten mit der Knotennummer i kon	nmt vom Knoten mit der Knotennummer
aktuellerKnoten – de Vorgängerknoten wird	
aktacherkhoten ac vorgangerkhoten whi	a appetently

besucht[aktuellerKnoten] = true; (der Knoten mit der Knotennummer aktuellerKnoten wird als besucht gespeichert) anzahlBesucht++; (die Anzahl der besuchten Knoten wird um 1 erhöht) Wartelliste Entfernen(); (der bereits besuchte Knoten wird aus der Wartelliste vorne entfernt) Wartelliste ist nicht leer wahr aktuellerKnoten = warteliste[0]; (die erste Knoten in der Warteliste wird zum neuen akutellenKnoten) (Die Warteliste ist leer und der Zielknoten wurde nicht erreicht. Das be Graph ist nicht zusammenhängend und der Zielknoten kann nicht me werden.) aktuellerKnoten ungleich -1 wahr Ausgabe "Ziel erreicht!" String weg = zielKnoten; (in der Voriablen weg wird am Ende der komplette Pfad stehen – beginnend von hinten mit dem Zielknoten) aktuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten], getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen Weg vorangestellt)	
ausWartelisteEntfernen(); Warteliste ist nicht leer wahr aktuellerKnoten = warteliste[0]; (die erste Knoten in der Warteliste wird zum neuen akutellenKnoten) (Die Warteliste ist leer und der Zielknoten wurde nicht erreicht. Das be Graph ist nicht zusammenhängend und der Zielknoten kann nicht mwerden.) aktuellerKnoten ungleich -1 vahr usgabe "Ziel erreicht!" tring weg = zielKnoten; in der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) ktuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten] getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
Warteliste ist nicht leer wahr aktuellerKnoten = warteliste[0]; (die erste Knoten in der Warteliste wird zum neuen akutellenKnoten) (Die Warteliste ist leer und der Zielknoten wurde nicht erreicht. Das be Graph ist nicht zusammenhängend und der Zielknoten kann nicht me werden.) ahr usgabe "Ziel erreicht!" rring weg = zielKnoten; n der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten); wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
wahr aktuellerKnoten = warteliste[0]; (die erste Knoten in der Warteliste wird zum neuen akutellenKnoten) (Die Warteliste ist leer und der Zielknoten wurde nicht erreicht. Das be Graph ist nicht zusammenhängend und der Zielknoten kann nicht me werden.) aktuellerKnoten ungleich -1 ahr usgabe "Ziel erreicht!" ring weg = zielKnoten; in der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) tuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
aktuellerKnoten = warteliste[0]; (die erste Knoten in der Warteliste wird zum neuen akutellenKnoten) (Die Warteliste ist leer und der Zielknoten wurde nicht erreicht. Das be Graph ist nicht zusammenhängend und der Zielknoten kann nicht me werden.) ahr usgabe "Ziel erreicht!" rring weg = zielKnoten; In der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) ktuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
(die erste Knoten in der Warteliste wird zum neuen akutellenKnoten) (Die Warteliste ist leer und der Zielknoten wurde nicht erreicht. Das be Graph ist nicht zusammenhängend und der Zielknoten kann nicht me werden.) aktuellerKnoten ungleich -1 rahr usgabe "Ziel erreicht!" tring weg = zielKnoten; in der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) ktuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	false
aktuellerKnoten ungleich -1 ahr usgabe "Ziel erreicht!" tring weg = zielKnoten; in der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) ktuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
ahr usgabe "Ziel erreicht!" ring weg = zielKnoten; in der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) ktuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	•
isgabe "Ziel erreicht!" ring weg = zielKnoten; in der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) tuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
isgabe "Ziel erreicht!" ing weg = zielKnoten; der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) tuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
ring weg = zielKnoten; In der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von Inten mit dem Zielknoten) tuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	false
in der Variablen weg wird am Ende der komplette Pfad stehen – beginnend von inten mit dem Zielknoten) tuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	/
tuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
nten mit dem Zielknoten) ktuellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
<pre>ituellerKnoten = zielNummer; wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen)</pre>	
wiederhole solange aktuellerKnoten ungleich startNummer ist aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
aktuellerKnoten = kommtVon[aktuellerKnoten]; (der Vorgängerknoten vom momentanen aktuellen Knoten wird zum neuen aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
<pre>aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen</pre>	
<pre>aktuellen Knoten) weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen</pre>	
<pre>weg = knotenPunkte[aktuellerKnoten].getBezeichner() + "/" + weg; (der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen</pre>	
(der Bezeichner vom neuen aktuellen Knoten wird einem / und dem bisherigen	
Weg vorangestellt)	
/	
usgabe von den Variablen weg und distanz[zielNummer]	
n weg steht am Ende der komplette Pfad in richtiger Reihenfolge und in	
stanz[zielNummer] die Entfernung zwischen Start- und Zielknoten)	

Mit einer **erweiterten Breitensuche** kann man sowohl die Länge des Pfads (als Anzahl der auf dem Weg durchlaufenen Kanten) von einem Startknoten zu einem Knoten mit einer gewünschten Eigenschaft angeben als auch die Folge der Knoten auf diesem Pfad.