Übung 3: Suche in Graphen / Zustandsräumen

Teil I: Grundlagen der Repräsentation und Implementation von Graphen

Graphen sind eine wichtige Datenstruktur zur Repräsentation unterschiedlicher Systeme und Prozesse. Graphen können gerichtet oder ungerichtet sein.

1 Gerichteter Graph

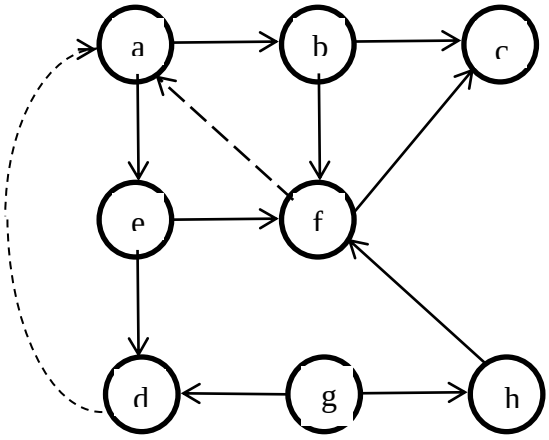

Abbildung 1: Gerichteter zyklenfreier **Graph** (mit gestrichelten Kanten = zyklischer **Graph**)

2 Ungerichteter Graph

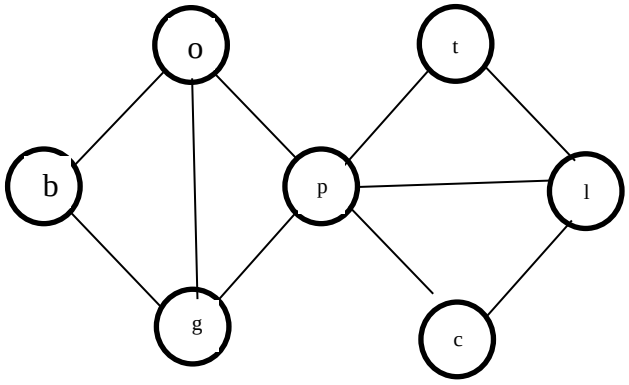

Abbildung 2: Ungerichteter Graph

Ein ungerichteter Graph kann auf einen gerichteten zurückgeführt werden, indem jede ungerichtete Kante durch zwei entgegengesetzte gerichtete Kanten ersetzt wird.

3. Repräsenation

Es gibt mehrere Möglichkeiten Graphen in PROLOG zu repräsentieren

3. 1. Adjazenzmatrix

Der Graph wird durch eine Matrix dargestellt. In dieser ist jedem Knoten eine Zeile und Spalte zugeordnet. Jedes Matrixelement entspricht somit einem Knotenpaar. Wenn diese Knoten verbunden sind, ist der Wert des Matrixelementes 1 sonst 0.

Für den gerichteten Graphen aus Abbildung 1 erhält man:

	a	b	c	d	e	f	g	h
a	0	1	0	0	1	0	0	0
b	0	0	1	0	0	1	0	0
c	0	0	0	0	0	0	0	0
d	0	0	0	0	0	0	0	0
e	0	0	0	1	0	1	0	0
f	0	0	1	0	0	0	0	0
g	0	0	0	1	0	0	0	1
h	0	0	0	0	0	1	0	0

Tabelle 1: Adjazenzmatrix des gerichteten Graphen

Für den ungerichteten Graphen aus Abbildung 2 erhält man:

	b	c	g	l	o	p	t
b	0	0	1	0	1	0	0
c	0	0	1	1	0	1	0
g	1	1	0	0	1	1	0
l	0	1	0	0	0	1	1
o	1	0	1	0	0	1	1
p	0	1	1	1	1	0	1
t	0	0	0	1	1	1	0

Tabelle 2: Adjazenzmatrix des ungerichteten Graphen

Die Matrix ist bezüglich der Hauptachse symmetrisch: Xij = Xji.
Wenn diese Symmetrie genutzt wird, brauch man nur die Hälfte des Speicherplatzes.

Eine Adjazenzmatrix zur Repräsentation von Graphen den Vorteil anschaulich zu sein hat aber zwei Nachteile:

1. Sie enthält meistens 90% Nullen, die keine Information tragen
2. Matrizen lassen sich in PROLOG schlecht darstellen, nur durch verschachtelte Listen.
3. Die Benutzung zur Suche ist aufwendig.

Eine Verbesserung stellt die Repräsentation von Graphen mittels Verbindungslisten dar.

3. 2. Verbindungsliste

Verbindungslisten geben nur die vorhandenen Verbindungen an - für jeden Knoten werden alle Folgeknoten in einer Liste angegeben. Damit wird unnötige Redundanz vermieden und die Repräsentation ist PROLOG freundlich.

Knoten	Folgeknoten
a	[b,e]
b	[c,f]
c	[]
d	[]
e	[d,f]
f	[c]
g	[d,h]
h	[f]

Tabelle 2: Verbindungslisten für den gerichteten Graphen (Abbildung 1) ohne die gestrichelten Verbindungen

Knoten	Folgeknoten
b	[g,o]
c	[g,l,p]
g	[b,c,o,p]
l	[c,p,t]
p	[c,g,l,o,t]
t	[l,o,p]

Tabelle 3: Verbindungslisten für den ungerichteten Graphen (Abbildung 2)

Verbindungslisten lassen sich gut in PROLOG implementieren.

1. Möglichkeit: Darstellung des gesamten Graphen als Liste der Verbindungslisten, die als Paare (Startknoten-Liste der Folgeknoten dargestellt werden.

Für die beiden Beispielgraphen gilt:
Gerichteter Graph:
verb_lst_ger_gr([a-[b, e], b-[c, f], c-[], d-[], e-[d, f], f-[c], g-[d, h] h-[f]]).

Ungerichteter Graph:
verb_lst_uger_gr([b-[g, o], c-[g, l, p], g-[b, c, o, p], l-[c, p, t], o-[b, g, p, t], p-[c, g, l, o, t], t-[l, o, p]]

Dann kann ein Übergang von einem Knoten zum Folgeknoten wie folgt definiert werden:

kante(X,Y):- verbindungslisten(G),member(X-L,G),member(Y,L).

Die Suchzeit ist proportional dem Quadrat der Knotenanzahl.

2. Möglichkeit: Die einzelnen Verbindungslisten werden als Fakten repräsentiert. Damit verbessert sich die Effizienz des Programms

Gerichteter Graph:

```
verblst(a, [b,e]).
verblst(b, [c, f]).
verblst(c, []).
verblst(d, []).
verblst(e, [d, f]).
verblst(f, [c]).
verblst(g, [d, h]).
verblst(h, [f]).
```

Ungerichteter Graph:

```
verblst(b,[g, o]).
verblst(c,[g, l, p]).
verblst(g,[b, c, o, p]).
verblst(l,[c, p, t]).
verblst(o,[b, g, p, t]).
verblst(p,[c, g, l, o, t]).
verblst(t,[l, o, p]).
```

Bei dieser Darstellung kann ein Übergang von einem Knoten zum Folgeknoten wie folgt definiert werden:

```
kante(X,Y):- verblst(X,L),member(Y,L).
```

Die Suchzeit ist damit proportional der Knotenanzahl.

3.3. Repräsentation der einzelnen Kanten als Fakten

Eine weitere Verbesserung erzielt man mit der unmittelbaren Darstellung der einzelnen Kanten.
Durch die Angabe der Relation <kante> wird wegen der im allgemeinen Nichtreflexivität der Relation ein gerichteter Graph implementiert.

Gerichteter Graph

```
kante(a, b).
kante(a, e).
kante(b, c).
kante(b, f).
kante(e, d).
kante(e, f).
kante(f, c).
kante(g, d).
kante(g, h).
kante(h, f).
```

Die Suchzeit ist konstant und unabhängig von der Knotenanzahl.

Beim ungerichteten Graphen können alle Übergänge dargestellt werden oder nur die Hälfte, wenn durch eine Regel die Symmetrie der Kanten ausgedrückt wird. Dabei ist darauf zu achten, dass die Kantenpaare keine Zyklen verursachen.

Lösung 1: doppelte Faktenmenge und Sicherheit

```
kante_ug(b,o). kante_ug(b,g).
```

kante_ug(c,g). kante_ug(c,l). kante_ug(c,p).
kante_ug(g,b). kante_ug(g,c). kante_ug(g,o). kante_ug(g,p).
kante_ug(l,c). kante_ug(l,p). kante_ug(l,t).
kante_ug(o,b). kante_ug(o,g). kante_ug(o,t). kante_ug(o,p).
kante_ug(p,c). kante_ug(p,g). kante_ug(p,l). kante_ug(p,0).
kante_ug(p,t).
kante_ug(t,l). kante_ug(t,o). kante_ug(t,p).

Die Suchzeit ist konstant und unabhängig von der Knotenanzahl.

Lösung 2: halbe Faktenmenge und Regel (Vorsicht!).

kante_ug1(b,o).kante_ug1(b,g).
kante_ug1(g,c).kante_ug1(g,o).kante_ug1(g,p).
kante_ug1(l,c).
kante_ug1(o,t).kante_ug1(o,p).
kante_ug1(p,c).kante_ug1(p,l).
kante_ug1(t,l).
kante_ug2(A,B):-kante_ug1(A,B);kante_ug1(B,A).

Falsch wäre eine Regel: kante_ug1(X,Y):- kante_ug1(Y,X). -> Zyklus

Die Suchzeit ist die Summe zweier konstanter Werte und ist unabhängig von der Knotenanzahl.

Teil II: Aufgaben

Aufgabe 1: Implementation des gerichteten Graphen ohne Zyklen (Abbildung 1) mit dem Prädikat kante_gg/2

Aufgabe 2: Bestimmung von Pfaden im gerichteten zyklenfreien Graphen.

Aufgabe 2.1: Definieren Sie ein Prädikat nachb3_gg/2, dass alle Nachbarn erfasst, die eine konstant begrenzte Weglänge L<=3 entfernt sind.

Testfälle:
*nachb3*_gg(a,c).->yes
nachb3_gg(a,h).->no
nachb3_gg(a,X).->X = b; X = e; X = c; X = f; X = d; X = f; X = c; X = c
nachb3_gg(A,c).->A = b; A = f; A = a; A = b; A = e; A = h; A = a; A = a; A = g

Aufgabe 2.2: Pfade mit unbegrenzter Weglänge lassen sich nur durch ein rekursives Prädikat ausdrücken. Definieren Sie ein rekursives Prädikat pfad(X,Y), das den unbegrenzten Pfad vom Knoten X nach Knoten Y präsentiert. Das Prädikat stellt die transitive Hülle der Kantenrelation dar.

Testfälle:
pfad_gg(g,c). Ist der Knoten c vom Knoten g erreichbar: true
pfad_gg(c,g). Ist der Knoten g vom Knoten c nicht erreichbar: false
pfad_gg(b,Z). Welche Knoten sind vom Knoten b erreichbar: Z = c ;Z = f ;Z = c

Aufgabe 3: Erzeugung des Gesamtpfades für Graphen ohne Zyklen

Aufgabe 3.1: Definieren Sie ein rekursives Prädikat pfad(X,Y,P) ohne Akkumulator, das die Relation zwischen dem Anfangsknoten X, dem Endknoten Y und dem dazugehörigen Pfad P repräsentiert.

Testfälle:
pfad_gg(g,c,P). P = [g, h, f, c]
pfad_gg(a,c,P). P = [a, b, c] ;P = [a, b, f, c] ;P = [a, e, f, c]
pfad_gg(a,Y,P). Y=b,P=[a,b]; Y=e,P=[a,e]; Y=c,P=[a,b,c]; Y=f,P=[a,b,f];Y=c,P=[a,b,f,c]; Y=d,P=[a,e,d];
Y=f,P=[a,e,f]; Y=c,P=[a,e,f,c]

Aufgabe 3.2: Definieren Sie ein rekursives Prädikat pfad(X,Y,A,P) mit einem Akkumulator, das den Pfad in inverser Reihenfolge liefert. Achten Sie darauf, dass der Akkumulator bei der Anfrage mit dem richtigen Wert initialisiert wird: pfad(X,Y,[X],P) und dass der Pfad im Ergebnis in umgekehrter Reihenfolge vorliegt, da er mit Akku gebildet wird.

Testfälle:

pfad_gg(g,c,[g],P). P = [c, f, h, g]
pfad_gg(a,c,[a],P). P = [c, b, a] ; P = [c, f, b, a] ;P = [c, f, e, a] ;

Aufgabe 4: Pfaddefinition für einen Graphen mit Zyklen
Die vorhergehende Pfaddefinition eignet sich nur für Graphen ohne Zyklen. Bei einem Graphen mit Zyklus muss in jedem Schritt getestet werden ob der Folgeknoten schon besucht worden ist. Das ist nur möglich, wenn der bereits durchlaufene Pfad in jedem Schritt vollständig vorliegt. Das ist aber nur bei Benutzung eines Akkumulators möglich, da der Pfad sonst erst beim rekursiven Aufstieg zusammengesetzt wird.

Aufgabe 4.1: Fügen Sie zur Graphenrepäsentation die zwei Fakten (gestrichelte Pfeile) hinzu, die den Graph in Abbildung 1 zu einem Graphen mit Zyklen machen. Definieren Sie ein rekursives Prädikat pfadz(X,Y,A,P) das die Relation zwischen dem Anfangsknoten X, dem Endknoten Y und dem dazugehörigen Pfad P in einem Graphen mit Zyklen repräsentiert.

Testfälle:

pfadz(g,c,[g],P). P=[c,b,a,d,g];P=[c,f,b,a,d,g];P=[c,f,e,a,d,g];P=[c,f,h,g];
P=[c,b,a,f,h,g];
pfadz(a,a,[a],P). P=[a,f,b,a];P=[a,d,e,a];P=[a,f,e,a]

Aufgabe 4.2: Definieren Sie ein Prädikat zyklus/0, das überprüft, ob ein Graph Zyklen enthält.

Der Einfachheit halber bezieht sich das Prädikat zyklus/0 auf den im Programm definierten Graphen, so dass es kein Argument benötigt.
Eine Anfrage mit zyklus/0 scheitert wenn der im Programm definierte Graph keine Zyklen enthält. Beim Vorhandensein von Zyklen ist die Anfrage erfolgreich.

Aufgabe 5: Implementation eines ungerichteten Graphen.

Aufgabe 5.1: Implementieren Sie einen ungerichteten Graphen, der die Nachbarschaftsbeziehung zwischen den Bundesländern der BRD repräsentiert.

Aufgabe 5.2: Definieren Sie ein Prädikat test(X,Y), das überprüft, ob es zu jeder Kante die dazugehörige Umkehrkante gibt und die - wenn eine Umkehrkante fehlt - die zwei Knoten X,Y ohne Umkehrkante zurückgibt. Testen Sie damit, ob die Fakten aus Aufgabe 5.1 vollständig sind.