Aufgabe 1 – Problemlösen durch Suche

a) A*-Simulation (4 Expandierungsschritte)
Entwicklung der sortierten Liste:

A(7)

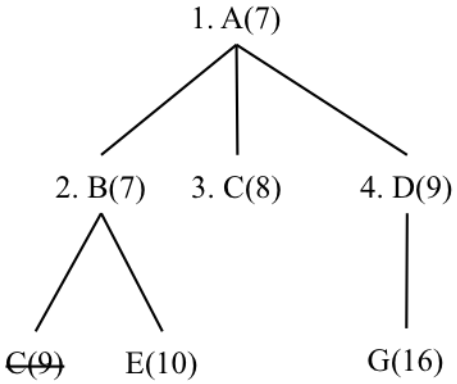
AB(7), AC(8), AD(9)

AC(8), ~~ABC(9),~~ AD(9), ABE(10) // ABC(9) wird aufgrund von dyn. Progr. entfernt

AD(9), ABE(10) // keine neuen Pfade weil AC eine Sackgasse ist

ABE(10), ADG(16)

Suchbaum:



b) Dynamische Programmierung macht grundsätzlich Sinn bei Breitensuche, weil jeder Zustand auf allen Pfaden (bis zur aktuellen Suchtiefe) erreicht wird. DP kann dann den Baum deutlich beschneiden.
Bei Tiefensuche hängt der Effekt von DP vom konkreten Problem ab:

- Bei einem hinreichend stark vermetzten Problemgraphen (z.B. einem Grid mit 4-Nachbarschaft) *ohne* Lösung muss TS ohne DP alle Suchpfade generieren. Mit DP werden alle suboptimalen Pfade erkannt und nicht weiter expandiert.
- Bei einem hinreichend stark vermetzten Problemgraphen (z.B. einem Grid mit 4-Nachbarschaft) *mit* Lösung hilft DP, eine bessere Lösung zu finden als ohne DP. Aber viele suboptimale Pfade werden nicht entdeckt, weil sie nicht direkte Nachbarzustände des aktuellen Pfades sind (TS blickt nicht weit genug über den Tellerrand).
- Bei schwächer vernetzten Problemgraphen kann DP sogar zu einer schlechteren Lösungsqualität führen, wenn nämlich ein suboptimaler Lösungspfad ohne DS durch DS entfernt wird, wodurch ein noch schlechterer Pfad beschritten wird.

Aufgabe 2 – Constraints

a) Backtrackingsuche: 1 Backtrackschritt zurück zur Wurzel
A=1, B=3, C=2, D=3
A=2, B=1, C=3, D=1, E=2

b) Forward Checking

$D_A = \{ 1 \}$
$D_B = \{ 2 \}$
$D_C = \{ 3 \}$
$D_D = \{ 2, 3 \}$
$D_E = \{ 1, 3 \}$

$D_A = \{ 1 \}$ $D_B = \{ 2 \}$
$D_C = \{ 1, 2, 3 \}$ $D_D = \{ 1, 2, 3 \}$
$D_E = \{ 1, 2, 3 \}$

c)

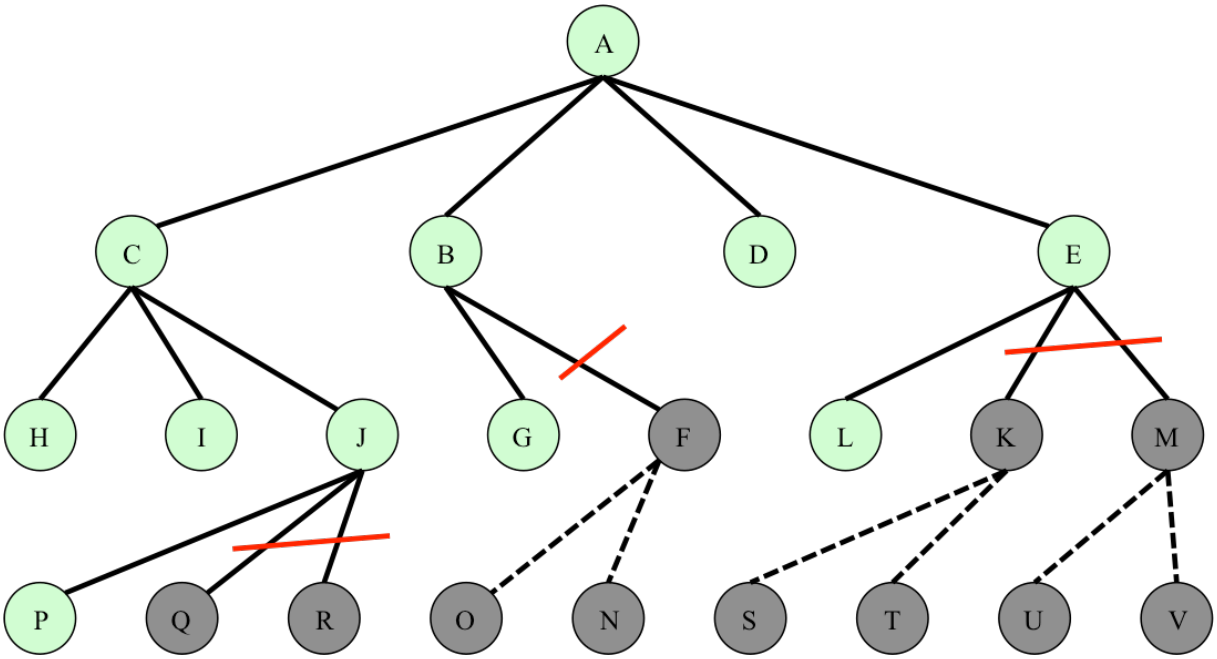
$D_A = \{ 1 \}$ $D_D = \{ 1, 2, 3, 4 \}$
$D_B = \{ 1, 2, 3, 4 \}$ $D_E = \{ 1, 2, 3, 4 \}$
$D_C = \{ 1, 2, 3, 4 \}$ $D_F = \{ 1, 2, 3, 4 \}$

Aufgabe 3 –Strategische Spiele

a)

A=3	B= -5	C=3	E=2	F=4	J=9	K=5	M= -7

b) und c)

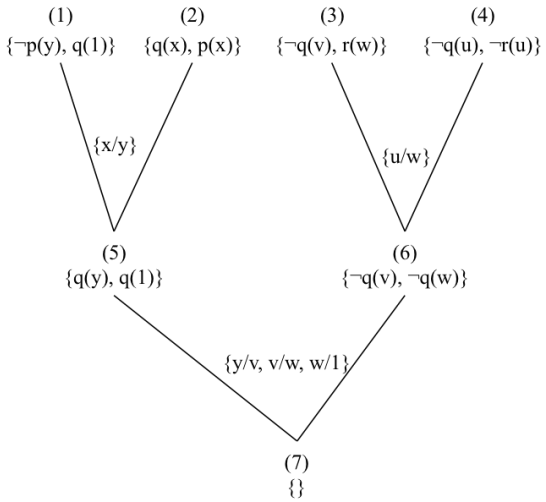


Die geforderte Rotation entspricht einer optimalen Zugsortierung. Jeder Spieler am Zug untersucht seine Züge in der Reihenfolge ihrer Qualität (absteigend). Andere Zugsortierungen führen zum selben Ergebnis: Die Knoten {H,I} und{B,D,E} können in beliebiger Reihenfolge untersucht werden. Dasselbe gilt für alle nicht untersuchte Geschwisterknoten.

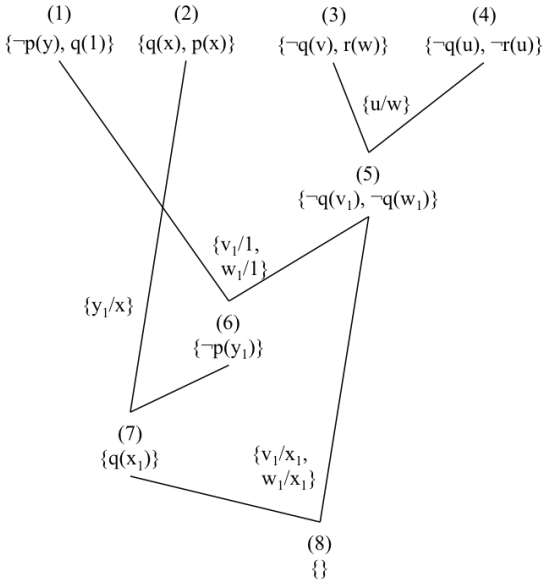
Die hell gefärbten Knoten werden untersucht, die grau gefärbten durch Cutoffs nicht generiert. Die Cutoffs sind rot eingezeichnet.

Aufgabe 4 – Maschinelles Beweisen

a) Rückwärtsverkettung ist kein geeignetes Beweisverfahren, weil (2) keine Hornregel ist.
b) Resolutionsbeweis (2 mögliche Varianten).



(7) wird durch Faktorisierung resolviert



(6) und (8) werden durch Faktorisierung resolviert

c) Faktorisierung ist notwendig, weil die Wissensbasis keine Einheitsklauseln enthält. Ohne Faktorisierung können immer nur 2er-Klauseln resolviert werden.

d) .

(1) U(a,b) ∧ G(a,2) ∧ U(a,c) ∧ U(b,c) ⟶ Färbbar(a,b,c)
(2) U(1,2)
(3) U(2,1)
(4) U(1,3)
(5) U(3,1)
(6) U(2,3)
(7) U(3,2)
(8) G(1,1)
(9) G(2,2)
(10) G(3,3)

{ Färbbar(a,b,c) }
{ U(a,b), G(a,2), U(a,c), U(b,c) } -| (1) {}
{ G(1,2), U(1,c), U(2,c) } -| (2) {a/1, b/2}
FAIL // keine Regel für G(1,2)
{ G(2,2), U(2,c), U(1,c) } -| (3) {a/2, b/1}
{ U(2,c), U(1,c) } -| (9) {}
{ U(1,1) } -| (3) {c/1}
FAIL // keine Regel für U(1,1)
{ U(1,3) } -| (6) {c/3}
{} -| (4) {}

Aufgabe 5 – Planung

a) Definieren Sie Start- und Zielzustand in STRIPS.

S_0 = {
feld(A), ..., feld(E), // 5 Felder
r(A,B), r(B,C), ..., r(D,E), // insgesamt 4 Rechts-Von-Relationen
stein(1), ..., stein(4), // 4 Steine
farbe(1,W), ..., farbe(4,S), // Farbinformationen
ist(1,A), ist(2,B), ist(3,D), ist(4,E), // Positionsinformationen
frei(C) // freies Feld

S_Z = { ist(a,A), ist(b,B), ist(d,D), ist(e,E), farbe(a,S), farbe(b,S), farbe(d,W), farbe(E,W) }

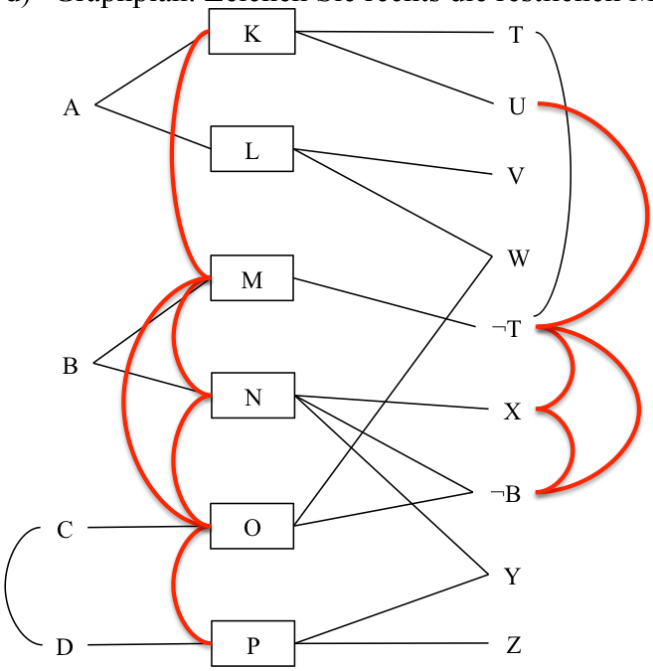
b) Definieren Sie einen Schiebe-Operator für weiße Steine (1 Feld nach rechts).

ACT schiebeW(s)
PRE stein(s), farbe(s,W), ist(s,x), n(x,y), frei(y)
EFF ist(s,y), frei(x), ¬ist(s,x), ¬frei(y)

c) Definieren Sie einen Sprung-Operator für schwarze Steine (über weiß nach links).

ACT sprungS(x)
PRE stein(s), farbe(s,S), ist(s,x), n(y,x), n(z,y), frei(z), ist(t,y), stein(t), farbe(t,W)
EFF ist(s,z), frei(x), ¬ist(s,x), ¬frei(z)

d) Graphplan: Zeichen Sie rechts die restlichen Mutexe ein.



e)

Mutex	**Ja/nein**	**Begründung, falls ja**
¬B, ¬T	ja	inkonsistenter Support (Literale werden nur über Mutex-Aktionen generiert (M,N) bzw. (M,O)
¬B, Y	nein	Y kann über P, ¬B über N erreicht werden, (N,P) sind nicht Mutex
K, L	nein	
K, M	ja	widersprüchliche Effekte (T, ¬T)
L, M	nein	
N, O	ja	Störung (B ∈ PRE(M), ¬B ∈ EFF(O))
N, P	nein	weder PRE noch EFF der Aktionen sind Mutex
O, P	ja	widersprüchliche PRE (Mutex(C,D) und C=PRE(O), D=PRE(P)
T, X	nein	T wird über K, X über N generiert
W, X	nein	W wird über L, X über N generiert