1. (6 p)

Lösning:

(a) En enkel representation av en sådan databas är som en lista av listor, där varje lista är ett par `[Disease, Symptoms]`, där `Disease` är en atom som betecknar ett sjukdom, och `Symptoms` är en lista av atomer som betecknar symptom.

Ett annat sätt att representera paret `Disease`, `Symptoms` är som en term `disease(Disease, Symptoms)` istället för en lista med två element som ovan.

Det var även vanligt med lösningar som involverade att man definierade sjukdom-symptom-relation som ett predikat snarare än en lista. Sådana lösningar godkändes också (även om svaren på 1b då i strikt mening blev felaktiga eftersom exempel-databasen inte representerades som en term).

(b) `[[i1, [s1, s2, s3]], [i2, [s2, s4]], [i3, [s3, s5]]]`

(c)
```
members([], _).
members([H | T], L) :-
   member(H, L),
   members(T, L).

diagnose(_, [], []).
diagnose(Symptoms, [[Disease, Sympts] | DBT], [Disease | DT]) :-
   members(Symptoms, Sympts),
   diagnose(Symptoms, DBT, DT).
diagnose(Symptoms, [_ | DBT], DT) :-
   diagnose(Symptoms, DBT, DT).
```

(Om en annan representation valts i uppgift (a) kan lösningen bli väldigt annorlunda!)

Rättning:

(a) 1p för en textuell beskrivning som går att genomföra i Prolog.

(b) 1p för korrekt svar enligt den egna representationen från uppgift (a).

(c)
- 1p för basfallet
- 2p för första induktiva fallet
- 1p för andra induktiva fallet

Några mindre fel ger 1p avdrag, även om de är utspridda över flera av fallen.

2. (8 p)

Lösning:

`reach(a, c).`

1. skapar instans av första regeln:
   `reach(X1, Y1) :- edge(X1, Z1), reach(Z1, Y1).` unifierar: `X1 = a, Y1 = c`.
2. `edge(a, Z1).` unifierar: `Z1 = b`.
3. `reach(b, c).`
   3.1. skapar instans av första regeln:
   `reach(X2, Y2) :- edge(X2, Z2), reach(Z2, Y2).` unifierar: `X2 = b, Y2 = c`.
   3.2. `edge(b, Z2).` unifierar: `Z2 = c`.
   3.3. `reach(c, c).`
   3.3.1. skapar instans av första regeln:
   `reach(X3, Y3) :- edge(X3, Z3), reach(Z3, Y3).` unifierar: `X3 = c, Y3 = c`.
   3.3.2. `edge(c, Z3).`
   kan inte unifieras; *backtrackar*.
   3.3.3. skapar instans av andra regeln:
   `reach(X4, Y4) :- edge(X4, Y4).` unifierar: `X4 = c, Y4 = c`.
   3.3.4. `edge(c, c).` kan inte unifieras; *backtrackar*.
   3.4. `edge(b, Z2).` failar; *backtrackar*.
   3.5. skapar instans av andra regeln:
   `reach(X5, Y5) :- edge(X5, Y5).` unifierar: `X5 = b, Y5 = c`.
   3.6. `edge(b, c).`
   unifierar med andra faktumet, frågan lyckas!

Svar: true.

Rättning:

- 2p för rätt huvuddrag, att vi gör ett första rekursivt anrop till `reach(b,c)` som resulterar i ett nytt rekursivt anrop till `reach(c, c)` som failar och `reach(b,c)` sedan istället lyckas genom andra regeln, eftersom `edge(b,c)` är sant.
- 1p för rätt regelinstanser och unifieringar fram till första rekursiva anropet `reach(b, c)`.
- 2p för rekursiva anropet själv utan det inre anropet (3.*), varav 1p för kontrollflödet och 1p för (rätta) regelinstanser och unifieringar;
- 2p för inre rekursiva anropet (3.3.*), varav 1p för kontrollflödet och 1p för (rätta) regelinstanser och unifieringar;
- 1p för rätt svar.

3. (6 p)

Lösning:

(a)
```
insert(nil, leaf).
insert(leaf, branch(leaf, nil)).
```

(b) **Möjlig lösning 1:**

```
insert(branch(TL, TR), branch(TLNew, TR)) :-
   incomplete(TL), !,
   insert(TL, TLNew).
insert(branch(TL, TR), branch(TL, TRNew)) :-
   incomplete(TR), !,
   insert(TR, TRNew).
insert(branch(TL, TR), branch(TL, TRNew)) :-
   height(TL, LeftHeight),
   height(TR, RightHeight),
   RightHeight < LeftHeight, !,
   insert(TR, TRNew).
insert(branch(TL, TR), branch(TLNew, TR)) :-
   insert(TL, TLNew).
```

Möjlig lösning 2:

i. Om vänster delträd av `T1` är incomplete så ska insert göras på vänster delträd.
ii. Om höger delträd av `T1` är incomplete så ska insert göras på höger delträd.
iii. Om båda delträden till `T1` är complete och har samma höjd så ska insert göras på vänster delträd.
iv. Annars (båda delträden till `T1` complete och höger delträd mindre höjd än vänster) ska insert göras på höger delträd.

Möjlig lösning 3:

i. Om vänster delträd av `T1` är incomplete, eller om båda delträden är complete och vänster delträd har samma höjd som höger delträd, så ska insert göras på vänster delträd.
ii. Annars ska insert göras på höger delträd.

Rättning:

(a) 1p för korrekt nil, 1p för korrekt leaf.

(b) 1p för vart och ett av följande fall som hanteras korrekt. Dessa behöver inte finnas som separata fall i lösningen, de kan vara ihop-bakade i varandra (t.ex. kanske lösningen bara delar upp det i två fall, se "möjlig lösning 3" ovan)

- Induktivt fall där vänster delträd till `T1` är ofullständigt (incomplete).
- Induktivt fall där höger delträd till `T1` är ofullständigt (incomplete).
- Induktivt fall där både vänster och höger delträd är fullständigt (complete), vänster delträd har större höjd än höger delträd.
- Induktivt fall där både vänster och höger delträd är fullständigt (complete), vänster delträd har samma höjd som höger delträd.

Om många fall hanteras helt felaktigt kan mindre poäng ges. Om lösningen t.ex. alltid gör insert på vänster delträd ges inte 2 poäng trots att fall 1 och 4 ovan hanteras korrekt.

Inget avdrag om fallen getts som Prolog-kod och snitt (`!`) skulle behöva läggas till för att göra lösningen korrekt.