1. *Solution.*
```
% graph G
edge(a,b).
edge(a,c).
edge(b,d).
edge(c,d).
edge(d,e).
edge(f,g).

% connected(X,Y) is true if X is connected to Y in G
%
connected(X,X).
connected(X,Z) :-
        edge(X,Y),
        connected(Y,Z).
```
We show that the size of the search tree is (grossly) bounded by $\mathsf{O}(n^2)$, where n is the number of vertices in the graph G. First, we observe that the number of edges in a graph with n nodes is bounded by n^2. Furthermore, in searching for a connection we need to consider each edge at most once. Hence the search tree is bounded by $\mathsf{O}(n^2)$. This argument is independent of the fact that the goal is ground or not. □

2. *Solution.*
```
duplicate(Xs,N,Ys) :-
    duplicate2(Xs,N,Ys\[]).

duplicate2([],_N,Ys\Ys).
duplicate2([X|Xs],N,Ys0\Ys2) :-
    generate(X,N,Ys0\Ys1),
    duplicate2(Xs,N,Ys1\Ys2).

generate(_X,0,Ys\Ys).
generate(X,N,Ys0\Ys1) :-
    N > 0,
    N1 is N - 1,
    generate(X,N1,Ys0\[X|Ys1]).
```
□

3. *Solution.*

```
isotree(nil,nil).
isotree(tree(X,Left1,Right1),tree(X,Left2,Right2) :-
            isotree(Left1,Left2),
            isotree(Right1,Right2).
isotree(tree(X,Left1,Right1),tree(X,Left2,Right2) :-
            isotree(Left1,Right2),
            isotree(Right1,Left2).
```

□

4. *Solution.*

```
% prop/1 --> DCG that generates well-parenthesised propositional formulas
% and stores the syntax tree
%
prop(true) --> "true".
prop(false) --> "false".
prop(not(A)) --> "not", prop(A).
prop(and(A,B)) --> "(", prop(A), "and", prop(B), ")".
prop(or(A,B)) --> "(", prop(A), "or", prop(B), ")".

% prop2/1 --> DCG that generates propositional formulas using
% the standard precedence
%
atom(p) --> "true".
atom(q) --> "false".
atom(A) --> "(", prop2(A), ")".
unary(not(A)) --> "not", unary(A).
unary(A) --> atom(A).
and(and(A,B)) --> unary(A), "and", and(B).
and(A) --> unary(A).
prop2(or(A,B)) --> and(A), "or", or(B).
prop2(A) --> and(A).
```

□

5. *Solution.*

```
jump(N,A/B,C/D) :-
    jump_dist(X,Y),
    C is A+X, C > 0, C =< N,
    D is B+Y, D > 0, D =< N.

jump_dist(1,2).
jump_dist(2,1).
jump_dist(2,-1).
```

```
jump_dist(1,-2).
jump_dist(-1,-2).
jump_dist(-2,-1).
jump_dist(-2,1).
jump_dist(-1,2).
```

□

6. *Solution.*

statement	**yes**	**no**
A rule is a universally quantified logical formula of the form $A \leftarrow B_1, B_2, \ldots, B_n$, where A is a goal and for all $i = 1, \ldots, n$: B_i is a goal.	☑	☐
An SLD-refutation is a finite SLD-derivation ending in the goal to be proven.	☐	☑
Logic programming is a declarative programming paradigm, that is, the computation of a function is made a first-class citizen.	☐	☑
The declarative semantics of a program P is the minimal model of P.	☑	☐
The order of goals is irrelevant in the computation model of logic programming, but not the order of rules.	☑	☐
A search tree is the same as an SLD tree.	☑	☐
Prolog is a language without types and the main technique to manipulate data is unification.	☑	☐
Difference lists are ineffective if the generation of different sections of a list depend on each other.	☑	☐
A meta-interpreter in Prolog interprets Prolog terms on the Warren abstract machine.	☐	☑
The predicate *bagof*(*Template*,*Goal*,*Bag*) unifies *Bag* with the alternatives of *Goal* that meet *Template*.	☐	☑

□