Exercise 4.1:
Show with the help of the Prolog tree how the cut is used in the following program,

\[
\begin{align*}
r(a) & . \\
r(b) & . \\
q(a) & \leftarrow r(X), !, p(a) . \\
q(f(X)) & \leftarrow r(X) . \\
p(X) & \leftarrow r(X) . \\
p(f(X)) & \leftarrow q(X), !, r(X) . \\
p(g(X)) & \leftarrow r(X) .
\end{align*}
\]

and where the query ?- p(X). is taken. What would happen without the cut?

Exercise 4.2:
Take the following program \( P \):

\[
\begin{align*}
p & \leftarrow . \\
p & \leftarrow p . \\
q & \leftarrow r . \\
q & \leftarrow \neg r , p . \\
r & \leftarrow \neg p . \\
t & \leftarrow q . \\
t & \leftarrow r , \neg q .
\end{align*}
\]

a) Construct the dependency graph \( D_P \) of \( P \).
b) Is \( P \) \textit{stratified} and/or \textit{hierarchical}? 
c) Give a stratification of \( P \).
d) Using your stratification to show how to compute the standard model \( M_P \) of \( P \).
Exercise 4.3:
The built-in predicate `fail/0`, fails when Prolog encounters it as a goal. Thus, it can be viewed as an instruction for backtracking. On the other hand, the cut predicate `!`, blocks backtracking.

Define the predicate `neg/1` which allows you to express *negation as failure*. 
