Удобно е да представим квадрат, зададен с долен ляв ъгъл (x, y) и център (p, q) като два интервала $[x; x1]$ и $[y; y1]$, където $(x1, y1)$ е горният десен ъгъл на квадрата. Тъй като (p, q) е средата на отсечката с краища (x, y) и $(x1, y1)$, то $2p = x + x1$ и $2q = y + y1$. Така получаваме дефиницията на предиката:

$$square\_to\_intervals1([[X, Y], [P, Q]], [[X, X1], [Y, Y1]]) \iff \\ X1 = 2P - X \& Y1 = 2Q - Y,$$

където първият аргумент е представяне на квадрат по даден долен ляв ъгъл и център, а вторият – двойка от интервалите $[x; x1]$ и $[y; y1]$. Оттук получаваме и предиката на Пролог:

```
square_to_intervals1([[X,Y],[P,Q]],[[X,X1],[Y,Y1]]):-X1 is 2*P -X,
                                                     Y1 is 2*Q - Y.
```

който генерира новото представяне. Проверката дали един квадрат $S' = [x'_1; x'_2] \times [y'_1; y'_2]$ се съдържа в друг $S'' = [x''_1; x''_2] \times [y''_1; y''_2]$ е еквивалентна на проверката дали $[x'_1; x'_2] \subseteq [x''_1; x''_2]$ и $[y'_1; y'_2] \subseteq [y''_1; y''_2]$. Предикатът:

$$sub\_interval([X1, Y1], [X2, Y2]) \iff X1 \leq X2 \& Y1 \leq Y2$$

се удовлетворява точно когато $[X1; X2] \subseteq [Y1; Y2]$ и може да се дефинира на Пролог като:

```
sub_interval([X1,Y1],[X2,Y2]):-X1=<X2, Y1=<Y2.
```

Комбинирайки идеите от по-горе условието квадрат $S1$ да се съдържа в квадрат $S2$, зададени с долен ляв ъгъл и център може да се изрази така:

$$sub\_square(S1, S2) \iff \exists I_1, J_1, I_2, J_2 \quad (square\_to\_intervals1(S1, [I_1, J_1]) \& \\ square\_to\_intervals1(S2, [I_2, J_2]) \& \\ sub\_interval(I_1, I_2) \& sub\_interval(J_1, J_2)),$$

което се изразява на Пролог като:

```
sub_square(S1,S2):-square_to_intervals1(S1,[I1,J1]),
                   square_to_intervals(S2,[I2,J2]),
                   sub_interval(I1,I2),sub_interval(J1,J2).
```

Накрая един квадрат S_1 се съдържа строго в друг S_2 тогава и само тогава, когато $S_1 \subseteq S_2$, но $S_2 \not\subseteq S_1$. Така получаваме и предиката $ssub\_square(S1, S2)$ като:

```
ssub_square(S1,S2):-sub_square(S1,S2),not(sub_square(S2,S1)).
```

1. Сега може да изразим първата част от условието така. Да се генерират всички списъци S, чиито елементи са елементи на L, които **не съдържат** два последователни квадрата s_1 и s_2, така че s_1 не се съдържа строго в s_2. Така получаваме, че списък $SquaresList$ с елементи на дадения **не** бива да се генерира тогава и само тогава, когато:

$$not\_in\_squares(SquaresList) \iff$$
$$\exists A, B, S_1, S_2(SquareList = A \circ [S_1, S_2] \circ B\&$$
$$\neg ssub\_square(S_1, S_2)),$$

   което записваме на Пролог като:

```
not_in_squares(SquaresList):-append(_,[S1,S2|_],SquaresList),
                             ssub_square(S1,S2).
```

   Накрая генерираме всички списъци, които удовлетворяват условието на задачата като генерираме всички $SquareList$, които са пермутация на някой подсписък на L и от тях отсяваме тези, които не притежават свойството $not\_in\_squares$:

```
gen_in_squares(L,List_Squares):-subset(List_Squares1,L),
                             permutation(ListSquares1,ListSquares),
                             not(not_in_squares(List_Squares1)).
```

   Предикатите *subset* и *permutation*, които генерират съответно подсписъците на даден списък и пермутациите на даден списък са реализирани по време на упражнения. Въпреки това за признаване на решението за пълно е необходимо тези(или еквивалентни) дефиниции да бъдат експлицитно приведени. Те могат да изглеждат например така:

```
subset([],[]).
subset([A|S],[A|L]):-subset(S,L).
subset(S,[_|L]):-subset(S,L).
```

и така:

```
append([],Y,Y).
append([A|X],Y,[A|Z]):-append(X,Y,Z).

permutation([],[]).
permutation([A|L],P):-append(L1,L2,L),permutation(L1,P1),
                       permutation(L2,P2),append(P1,[A|P2],P).
```

2. За да изразим условието за максималност отново е удобно да използваме конструкция с отрицание. Наистина, един подсписък от квадрати $SquaresList$ в L няма да има максимален брой елементи според условието на задачата, ако може да намерим друг подсписък $SquaresList1$ на L, в който квадратите се съдържат един в друг, но $|SquaresList1| > |SquaresList|$. Това може да изразим по-формално така:

$$
\begin{gathered}
not\_max\_in\_squares(L, SquaresList) \iff \\
\exists SquaresList1(gen\_in\_squares(L, SquaresList1)\& \\
|SquaresList| < |SquaresList1|).
\end{gathered}
$$

Като използваме предишното подусловие може да запишем този предикат на Пролог по следния начин:

```
not_max_in_squares(L,List_Squares):-gen_in_squares(L,List_Squares1),
                        len(List_Squares,Len),
                        len(List_Squares1,Len1),Len1>Len.
```

Предикатът $len(L, Len)$ генерира в Len дължината на списъка L.

Използвайки предикатът $not\_max\_in\_squares$ и $gen\_in\_squares$ може да завършим решението по следния начин:

```
max_in_squares(L,Max_List_Squares):-gen_in_squares(L,Max_List_Squares),
                        not(not_max_in_squares(L,Max_List_squares)).
```

За пълното решение на задачата е необходимо още и експлицитната дефиниция на предиката *len*. Примерна дефиниция е показвана на упражнения и такава е следната:

```
len([],0).
len([_|L],Len):-len(L,Len1), Len is Len1 + 1.
```

За признаване на решението за пълно е достатъчно да присъства коректен код на Пролог. Математическата обосновка за неговата коректност може да е много по-оскъдна или въобще да отсъства. Едно примерно студентско решение може да изглежда и така:

```
square_to_intervals1([[X,Y],[P,Q]],[[X,X1],[Y,Y1]]):-X1 is 2*P -X,
                                                   Y1 is 2*Q - Y.

sub_interval([X1,Y1],[X2,Y2]):-X1=<X2, Y1=<Y2.
sub_square(S1,S2):-square_to_intervals1(S1,[I1,J1]),
                   square_to_intervals1(S2,[I2,J2]),
                   sub_interval(I1,I2),sub_interval(J1,J2).

subset([],[]).
subset([A|S],[A|L]):-subset(S,L).
subset(S,[_|L]):-subset(S,L).

len([],0).
len([_|L],Len):-len(L,Len1),Len is Len1 +1.

append([],Y,Y).
append([A|X],Y,[A|Z]):-append(X,Y,Z).

permutation([],[]).
permutation([A|L],P):-append(L1,L2,L),permutation(L1,P1),
```

```
                        permutation(L2,P2),append(P1,[A|P2],P).

not_in_squares(List_Squares):-append(_,[S1,S2|_],List_Squares),
                        not(ssub_square(S1,S2)).

gen_in_squares(L,List_Squares):-subset(List_Squares1,L),
                        permutation(ListSquares1,ListSquares),
                        not(not_in_squares(List_Squares1)).

not_max_in_squares(L,List_Squares):-gen_in_squares(L,List_Squares1),
                       len(List_Squares,Len),
                       len(List_Squares1,Len1),Len1>Len.

max_in_squares(L,Max_List_Squares):-gen_in_squares(L,Max_List_Squares),
                       not(not_max_in_squares(L,Max_List_Squares)).
```