

Условие

Задача 2 (вариант 1)

Казваме, че списък X мажорира списък Y , ако всички елементи на X са елементи на Y . Да се дефинира на пролог предикат $p(L, M)$, който по даден списък от списъци L намира списък M , който съдържа всички елементи на L и в който никой елемент не се мажорира от елемент, намиращ се след него в списъка.

Задача 2 (вариант 2)

Казваме, че списък X мажорира списък $Y = [a_1, a_2, \dots, a_n]$, ако $X = [a_i, a_{i+1}, \dots, a_{i+j}]$ за някои i и j . Да се дефинира на пролог предикат $p(L, M)$, който по даден списък от списъци L намира списък M , който съдържа всички елементи на L и в който никой елемент не се мажорира от елемент, намиращ се след него в списъка.

Решения

За всеки от двата варианта се дават по две примерни решения.

общи предикати (и за двете решения на вариант 1)

```
% append(A, B, C) – C е конкатенацията на списъците A и B
append([], B, B).
append([H | T], B, [B | C]) :- append(T, B, C).
```

```
% member(L, X) – X е елемент на L
member(L, X) :- append(_, [X | _], L).
```

```
% out(L, X, M) – изважда произволен елемент X от L,
%                като M е остатък от L
out(L, X, M) :- append(A, [X | B], L), append(A, B, M).
```

```
% major(X, Y) – X мажорира Y (X е подмножество на Y)
major(X, Y) :- not(member(X, A), not(member(Y, A))).
```

решение 1 (за вариант 1)

```
% permutate(L, M) – генерира в M някоя пермутация на L
permutate([], []).
permutate(L, [X | T]) :- out(L, X, M), permutate(M, T).
```

```
% sorted(M) – проверява дали M е топологично сортиран
sorted(M) :-
    not(append(_, [X | B], M), member(B, Y), major(Y, X)).
```

```
% решение на задачата
p(L, M) :- permute(L, M), sorted(M).
```

решение 2 (за вариант 1)

```
% bottom(L, X, M) – X е елемент на L, който не се
%                               мажорира от друг елемент на L, а M е
%                               остатък от L, като извадим X
bottom(L, X, M) :-
    out(L, X, M), not(member(M, Y), major(Y, X)).
```

```
% p(L, M) – генерира в M топологична сортировка на L
p([], []).
p(L, [X | T]) :- bottom(L, X, M), p(M, T).
```

решения за вариант 2

Решенията са същите, като за вариант 1. Единствената разлика е реализацията на предиката *major(X, Y)*.

```
% major(X, Y) – X мажорира Y (X е подписък на Y)
major(X, Y) :- append(_, L, Y), append(X, _, L).
```

Критерии за оценяване

критерии за решение 1 (и за двата варианта)

предикат за мажориране (напр. *major(X, Y)*) - **1т.**
 предикат за пермутация (напр. *permute(L, M)*) - **1,5т.**
 условие за подредба на *M* (напр. *sorted(M)*) - **2т.**
 основни неща (структура и подредба на клаузите, *append* и *member*) - **0,5т.**

критерии за решение 2 (и за двата варианта)

предикат за мажориране (напр. *major(X, Y)*) - **1т.**
 избиране на най-преден елемент (напр. *bottom(L, X, M)*) - **1,5т.**
 рекурсивно сортиране на *L* (предикат *p(L, M)*) - **2т.**
 основни неща (структура и подредба на клаузите, *append* и *member*) - **0,5т.**