Рекурсия и Списъци в Пролог

Въведение в рекурсията и нейната употреба. Ще разгледаме темите в презентацията. Рекурсията е важна концепция в програмирането. Очаквайте задълбочен анализ и примери.

Дефиниция за Рекурсия

Формална дефиниция

Рекурсията е процес, при който функцията се извиква сама.

Базов случай

Базовият случай е основен за прекратяване на рекурсията.

Рекурсивните дефиниции присъстват и извън програмирането. Например, в математическите формули и природните явления.

Писане на Рекурсивни Програми в Пролог

1 **Основни принципи**

Писане на рекурсивни правила в Пролог.

2 **Структура**

Базови случаи и рекурсивни извиквания.

3 **Важност**

Базовият случай предотвратява безкрайни цикли.

Рекурсивните предикати в Пролог имат специфичен синтаксис. Той трябва да се спазва стриктно.

Писане на смислени програми

Правилата, разгледани досега, бяха използвани за дефиниране на нови *отношения,* които се изразяват изцяло в термините на предварително въведени независими от тях понятия. Съществуват отношения, които не допускат прости дефиниции чрез правила от този тип.

Вече можем да обмислим писането на много по-сложни Пролог програми.

Важно е да обмислим внимателно какво възнамеряваме да означават предикатите, и да се уверим, че всички клаузи, които пишем, са верни.

Трябва също така да обмислим дали сме включили достатъчно клаузи, за да разрешим Пролог да направи подходящи заключения, включващи предикатите.

И накрая, трябва да обмислим как бек-чейнингът ще използва клаузите кога действително установяване на заключения.

За да обобщим, имаме нужда от:

1. истината и нищо друго;
2. „цялата“ истина (осигурени са достатъчно клаузи);
3. представени в правилната форма за обратно верижно свързване (напр. избягвайте зацикляне и отрицание на (под)цели с променливи.

Пример: Светът на Блокчетата

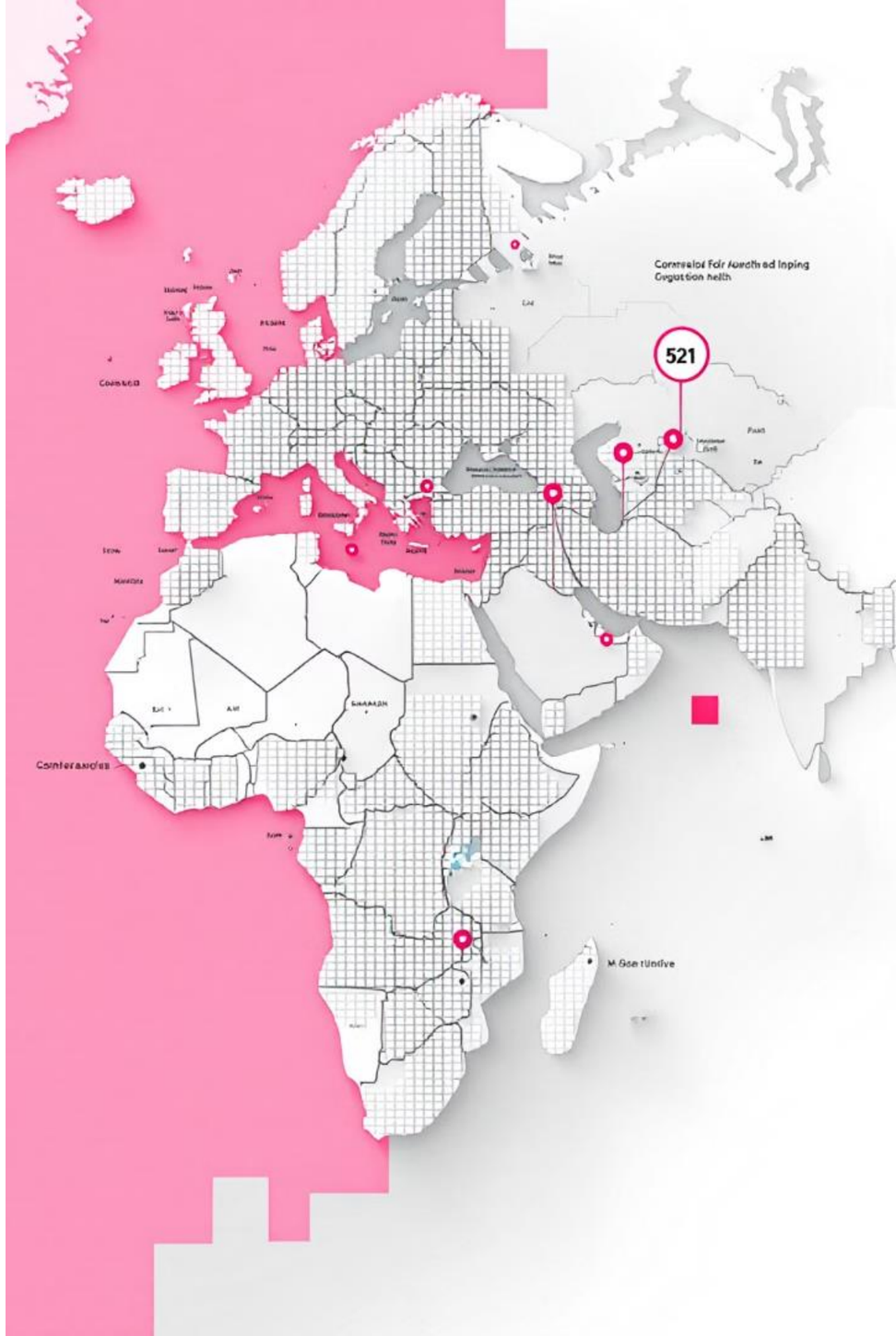
Описание

Прост свят с блокчета и релации между тях ("върху", "под").

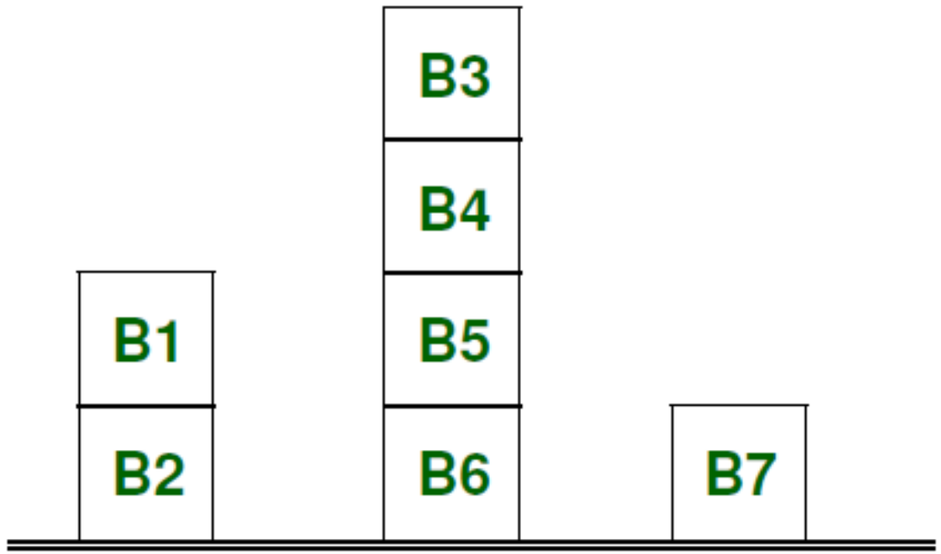
Рекурсивни правила

Определят сложни релации (напр., "поддържа").

Графичното представяне на блокчетата помага. То дава по-добро разбиране на релациите.



Пример: Светът на блокчетата

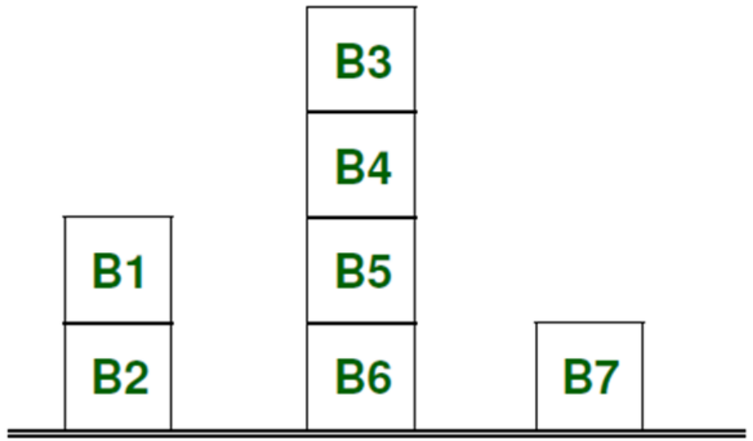


Бихме искали да опишем сцената и да накараме Prolog да определи следното:

- Блок 3 е над блок 5
- Блок 1 е отляво на блок 7
- Блок 4 е вдясно от блок 2

Програма Светът на блокчетата

```
/* on(X,Y) означава, че блок X е директно върху блок Y.*/

on(b1,b2). on(b3,b4). on(b4,b5). on(b5,b6).

/*just left(X,Y) , означава, че блокове X и Y са на масата  и че X е непосредствено отляво на Y. */

just_left(b2,b6). just_left(b6,b7).

/* above(X,Y),  означава, че блок X е някъде над блок Y в купчината, където се среща Y. */

above(X,Y) :- on(X,Y).

above(X,Y) :- on(X,Z), above(Z,Y).

/* left(X,Y),  означава, че блок X е някъде вляво от блок Y, но може би по-високо или по-ниско от Y. */

left(X,Y) :- just_left(X,Y).

left(X,Y) :- just_left(X,Z), left(Z,Y).

left(X,Y) :- on(X,Z), left(Z,Y).

left(X,Y) :- on(Y,Z), left(X,Z).

/* най-ляво е върху нещо, напр. X=B4 и Y=B7*/

/*най-вдясно е върху нещо. напр. X=B2 и Y=B5 */

/*дясно(X,Y) е обратното на ляво(X,Y).*/

right(Y,X) :- left(X,Y).
```

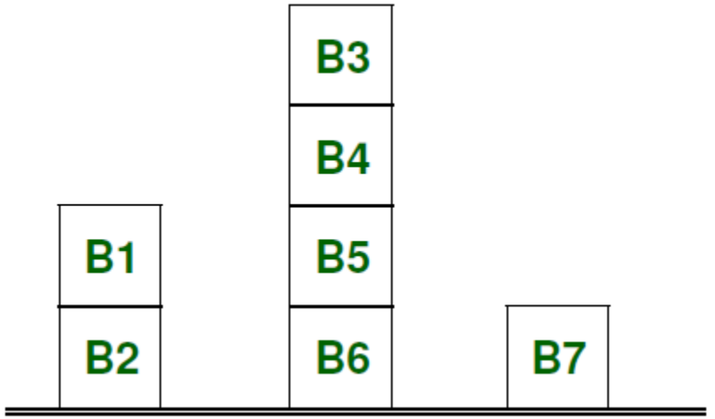


Цели/ заявки

?- above(b1,b2).
T Call: (7) above(b1, b2) % Основната заявка
T Call: (8) on(b1, b2)
%Първо опитайте (b1,b2), като използвате ред 8
T Exit: (8) on(b1, b2) % Успех поради ред 2
T Exit: (7) above(b1, b2)
% Така че основната заявка е успешна
Yes

?- above(b3,b5).
T Call: (8) above(b3, b5) % Основната заявка
T Call: (9) on(b3, b5) % - Опитайте (b3,b5)
T Fail: (9) on(b3, b5) % Това не успява
T Redo: (8) above(b3, b5)
% Преразгледайте

T Call: (9) on(b3, _L205)
% - Пробвайте с (b3,Z) от ред 9
T Exit: (9) on(b3, b4) % Това успява за Z=b4
T Call: (9) above(b4, b5)
% - Сега опитайте по-горе (Z,b5) за Z=b4
T Call: (10) on(b4, b5) % - Пробвай (b4,b5)
T Exit: (10) on(b4, b5) % Това успява
T Exit: (9) above(b4, b5) % Това успява
T Exit: (8) above(b3, b5)
% Основната заявка е успешна
Yes




```
/* on(X,Y) означава, че блок X е директно върху блок Y.*/
on(b1,b2). on(b3,b4). on(b4,b5). on(b5,b6).
/*just left(X,Y) , означава, че блокове X и Y са на масата
 и че X е непосредствено отляво на Y. */
just_left(b2,b6). just_left(b6,b7).
/* above(X,Y),  означава, че блок X е някъде над блок Y
в купчината, където се среща Y. */
above(X,Y) :- on(X,Y).
above(X,Y) :- on(X,Z), above(Z,Y).
/* left(X,Y),  означава, че блок X е някъде вляво
 от блок Y, но може би по-високо или по-ниско от Y. */
left(X,Y) :- just_left(X,Y).
left(X,Y) :- just_left(X,Z), left(Z,Y).
left(X,Y) :- on(X,Z), left(Z,Y).
left(X,Y) :- on(Y,Z), left(X,Z).
% най-ляво е върху нещо, напр. X=B4 и Y=B7
% най-вдясно е върху нещо. напр. X=B2 и Y=B5
% дясно(X,Y) е обратното на ляво(X,Y).
right(Y,X) :- left(X,Y).
```

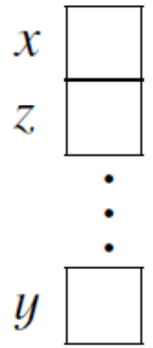
Примерът на предишните слайдове използва рекурсия.

Рекурсивна клауза е тази, в която предикатът на главата е същият като предикат, споменат в тялото.

Ред 9 в програмата:

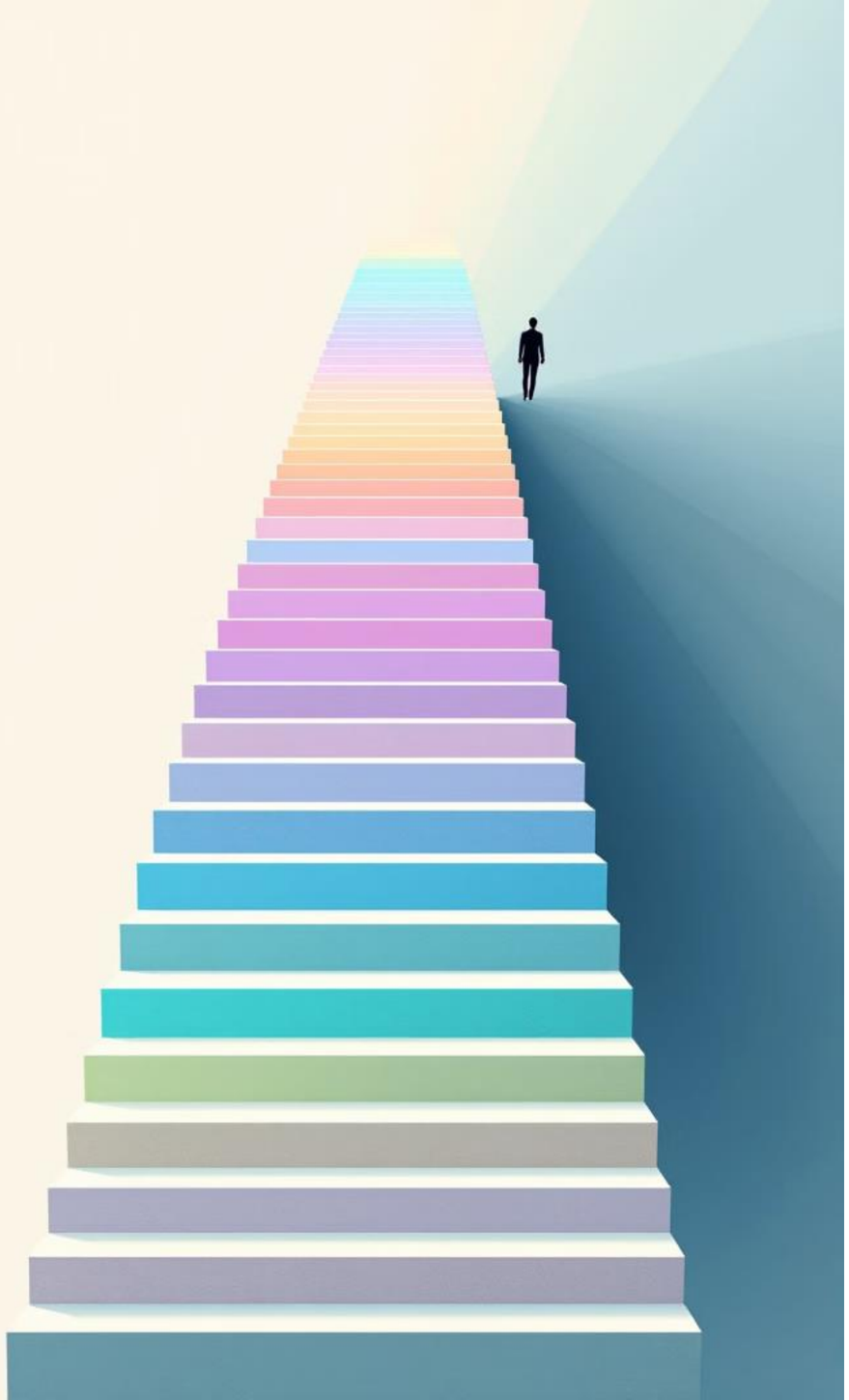
```
above(X,Y) :- on(X,Z), above(Z,Y).
```

Ако *x* е върху *z* и *z* е над *y*, тогава *x* е над *y*.



Повечето съвременни езици за програмиране (Java, Python и др.) предоставят рекурсия, която обикновено се счита за усъвършенствана техника.

Всъщност тя наистина е доста проста и е в основата на програмирането на Пролог.



Рекурсия като Математическа Индукция

1

Връзка

Между рекурсията и математическата индукция.

2

Базов случай

В рекурсията съответства на базовия случай в индукцията.

3

Рекурсивен случай

Съответства на индукционната стъпка.

Можем да докажем теореми чрез индукция. Пишем и съответни рекурсивни програми.

Рекурсия като математическа индукция: above(X,Y)

1. напишете клаузи за обработка на основния случай, където k = 0;

За предиката по-горе това означава, че x е директно върху y.

```
above(X,Y) :- on(X,Y).
```

2. приемете, че случаят за k = n вече е разгледан и напишете клаузи за обработка на случая, когато k = (n + 1);

За предиката по-горе да предположим, че числото k = (n + 1).

Тогава x трябва да е директно върху някой друг блок z, където има само n междинни блока между тези z и y.

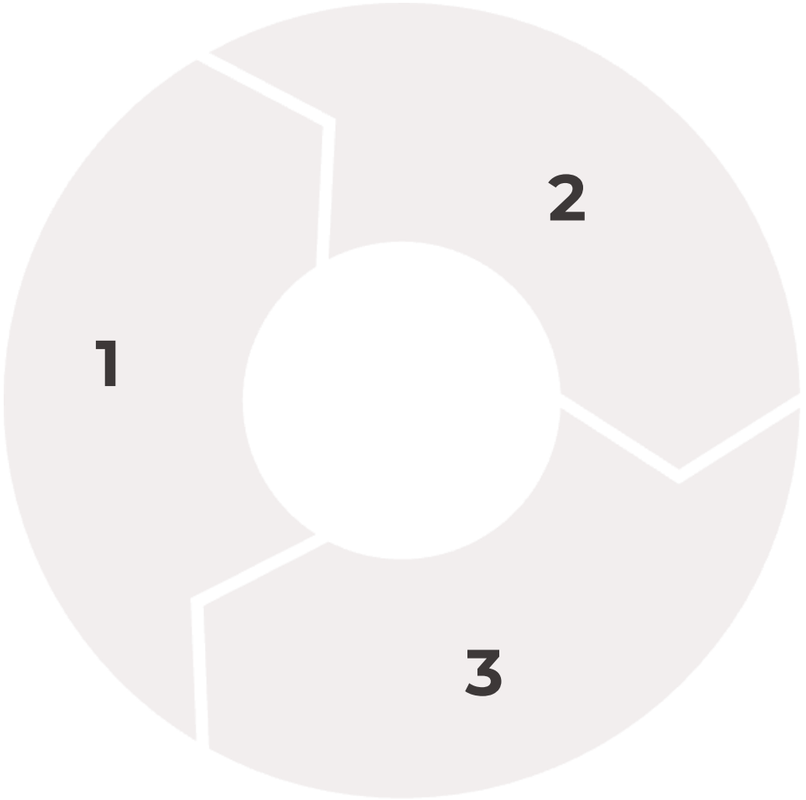
Така че приемаме, че предикатът вече работи за z и y и го използваме, за да напишем клауза за x и y.

```
above(X,Y) :- on(X,Z), above(Z,Y).
```

Същност на Рекурсивните Дефиниции

Полезност

Рекурсивните дефиниции са полезни и елегантни.



Разбиване

Решаване на сложни проблеми.

Подпроблеми

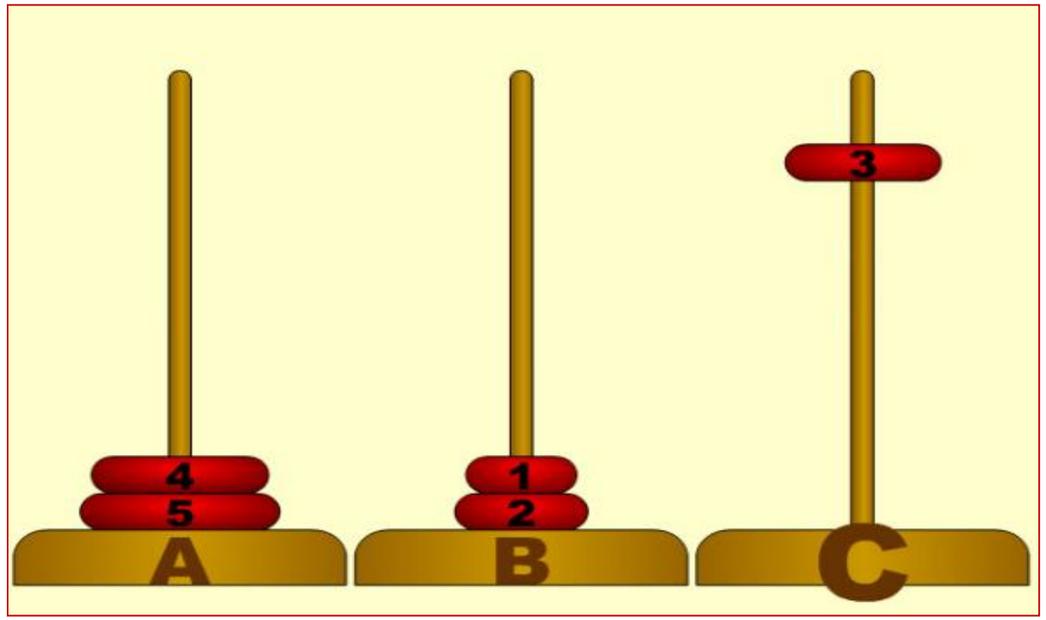
Разбиват ги на по-малки, подобни.

Примери за рекурсивни дефиниции има в математиката. Те присъстват и в компютърните науки.

Същност на рекурсивните дефиниции

Най-общо, една рекурсивна дефиниция включва в лявата страна понятието, което искаме да дефинираме (дадено в дясната страна). Смисълът на един рекурсивен проблем е в това, дали можем циклично да го опростяваме и да прилагаме същия начин за търсене на решение както за оригиналния. Освен това е необходимо да специфицираме „дъното“ на рекурсията, т.е. проблемът е решим с „единична“ (позната) операция.
Една рекурсивна дефиниция в Пролог, обикновено се задава с две клаузи - рекурсивна клауза и клауза за единичната операция (стоп клауза).
Рекурсивните дефиниции могат да бъдат:
Ляворекурсивни
Дяснорекурсивни.

Пример: Ханойски Кули



1

Описание

Проблемът "Ханойски кули".

2

Решение

Рекурсивно решение на проблема.

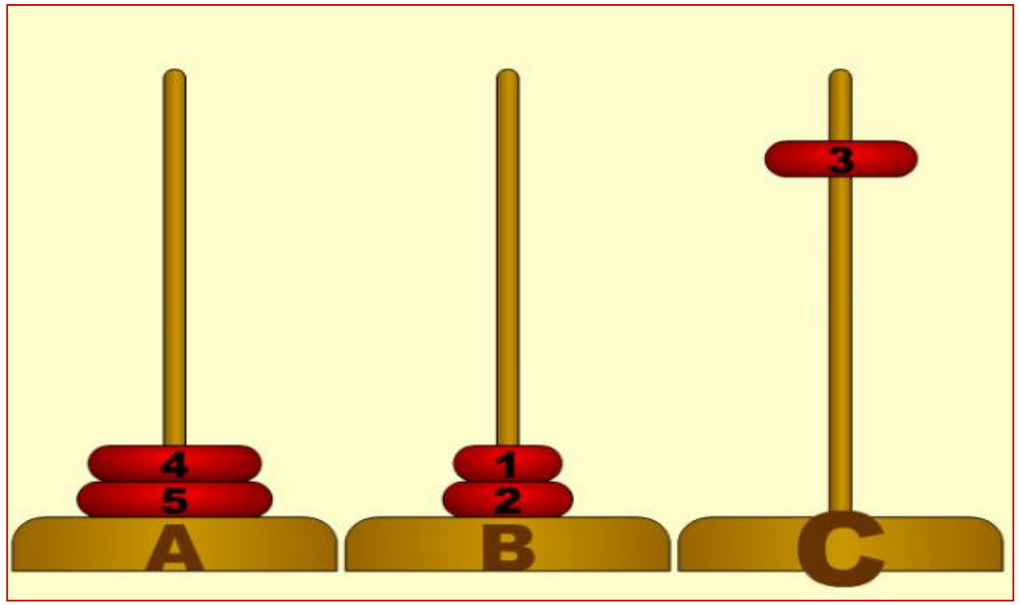
3

Код

Код на Пролог, който решава проблема.

Визуализация на решаването е важна. Тя помага за разбиране на задачата.

Пример за рекурсивен проблем: „Ханойски кули“




```
% „Дъно“ на рекурсията
move(1, X, Y, _) :-
write('Премести пул от'),
        write(X),
        write(' върху  '),
        write(Y),
         nl.
```

```
% Рекурсивна процедура
move(N, X, Y, Z) :-
          N > 1,
        M is N-1,
        move(M, X, Z, Y),
        move(1, X, Y, _),
        move(M, Z, Y, X).
```

Нека зададем следния въпрос към БД:
?- move(3, a, c, b).

Премести пул от a върху c
Премести пул от a върху b
Премести пул от c върху b
Премести пул от a върху c
Премести пул от b върху a
Премести пул от b върху c
Премести пул от a върху c
true

Съществуват различни разновидности на рекурсията. Различаваме:

Рекурсивни програми - в програмирането с **рекурсия** се обозначава случай, когато една подпрограма вика предходна своя функция. Рекурсията условно се разделя в две категории: директна (пряка) и индиректна (косвена). Рекурсията е пряка, когато в тялото на подпрограмата има референция към нея. Косвена е тази рекурсия, при която една подпрограма вика друга, а тя вика предходната. Съществуват и случаи на косвена рекурсия, при които подпрограмата извиква себе си, след поредица от обръщения към други подпрограми.

Рекурсивни структури – типичен пример за рекурсивни структури са списъците

Рекурсивни проблеми – проблемът, който искаме да решаваме, естествено, се представя рекурсивно.

Рекурсивен предикат – в дясната част на някое негово правило се среща същият предикат.

p(X):-a(X,Y), p(Y).

Използване:

Задачата се формулира рекурсивно (отношението се описва чрез себе си);

Обработва се рекурсивна структура от данни.

Рекурсията – единствен начин за реализиране на итерация (цикли).

Рекурсивната дефиниция включва:

Стоп-клауза (една или повече) за завършване на рекурсията

Рекурсивна клауза (една или повече)

Пример:

Дефиниране на потомък (наследник) чрез предикатите син и дъщеря.

```
потомък (X,иван):-син(X,иван).                  % стоп клауза
потомък(X,иван):-дъщеря(X,иван).                % стоп клауза
потомък(X,иван):-син(Y,иван), потомък(X,Y).  % рекурсивна клауза
потомък(X,иван):- потомък(Y,иван), потомък(X,Y).    % рекурсивна клауза
```

потомство(X,Y):- родител(X,Y).
потомство(X,Y):- родител (X,Z),потомство(Z,Y).

потомство(X,Y):- родител (X,Z), потомство(Z,Y).
потомство(X,Y):- родител(X,Y).

потомство(X,Y):- родител(X,Y).
потомство(X,Y):- потомство(X,Z),родител(Z,Y).

потомство(X,Y):- потомство(X,Z), родител(Z,Y).
потомство(X,Y):- родител (X,Y).

Примери: Факториел и Степен

1

Факториел

Рекурсивна дефиниция и код в Пролог.

2

Степен

(x**n, n>0): рекурсивна дефиниция и код.

Ефективността на рекурсивните решения е важна. Тя трябва да се взема под внимание.

Задача. Пресмятане на факториел n!
А) Рекурсивен вариант – съответства на написване на рекурсивна програма на процедурен език. Това е естественият начин за дефиниране на отношения.

```
% 0!=1  (стоп-клауза)          fact(0,1).

%n!=n*(n-1)! (рекурсивна-клауза)
                    fact(N,F):- N>0, N1 is N-1, fact(N1,F1), F is F1*N.
```

Примери: НОД, Фибоначи

НОД

Рекурсивно решение с алгоритъма на Евклид.

Фибоначи

Пресмятане на n-ти елемент от редицата.

Отпечатване на числа

Отпечатване на числа от 1 до зададено число.

Рекурсията предлага елегантни решения. Те са за много математически задачи.

Задачи

1. Пресмятане на степен: x^n $n>0$

2. Отпечатване на екрана на числата от 1 до зададено число.

3. Пресмятане на сумата
   където ***a*** е най-голямото четно число,
   ненадхвърлящо **n**

$$(x+2)^2+(x+4)^2+(x+6)^2+\ldots+(x+a)^2$$

4. Пресмятане на НОД на две числа

5. Пресмятане на n-ти елемент на редицата на Фибоначи

$$f_0=1;\ f_1=1;\ f_n=f_{n-1}+f_{n-2},\ n>1.$$

Списъци в Пролог

Представяне на списъците в Пролог. Ще разгледаме дефинирането, елементите и унификацията. Ще обсъдим обработката на списъци и вградените предикати. Ще завършим с примерни задачи.

Един списък е последователност от елементи, затворени в скоби [ ]. Например нека е даден списъкът [витоша, мусала, шипка]. Първият елемент от списъка се нарича глава или заглавна част (в примера витоша). Останалите елементи на списъка се наричат остатък или опашка (в примера [мусала,шипка]). Опашката се интерпретира отново като списък.

Дефиниране на списъци

Синтаксис

Списъците се дефинират с [Елемент1, Елемент2, ...].

Празният списък е [].

Рекурсивна структура

Представят се рекурсивно като [Глава|Опашка].

Главата е първият елемент, опашката – останалите.

Един списък е последователност от елементи, затворени в скоби **[ ]**.
Например нека е даден списъкът [витоша, мусала, шипка]. Първият елемент от списъка се нарича глава или заглавна част (в примера витоша). Останалите елементи на списъка се наричат остатък или опашка (в примера [мусала,шипка]). Опашката се интерпретира отново като списък.

Елементите на списъците могат да бъдат произволни терми, включително и списъци.
Така например, списъкът [a,[b,c],d] се състои от три елемента - a,[b,c] и d. Един списък може да не съдържа елементи – нарича се празен списък и се означава [ ].
С помощта на оператора | един непразен списък може да бъде разложен на глава и опашка [глава|опашка]. Синтаксисът на Пролог позволява преди | да бъдат зададени повече от един елемент. Следващият | терм трябва винаги да бъде списък, вкл. Празен. Така например за горния пример съществуват следните алтернативни представяния:
[витоша, мусала, шипка] =
[витоша |[мусала, шипка]] =
[витоша, мусала |[шипка]] =
[витоша, мусала, шипка|[ ]].

Списък (определение) – наредена последователност от 0,1 или повече елементи. Елементите могат да бъдат произволни терми, включително други списъци. (Списъците също са терми – специален вид структури.)

Примери:

[] – празен списък

[1] – списък с един елемент

[a,b] – списък с два елемента

[p(maria,ivan), 3] – списък с два елемента

[a,b,f(N,k),1,[a,1]] – списък с пет елемента

- [X | L] – списък
- [1, 2, 3, 4, 5]

Глава опашка

Обработка на списъци – чрез отделяне на глава и опашка.

Списък	Глава	Опашка
[a,b]	a	[b]
[1]	1	[]
[]	няма	няма
[X,f(Y),1]	X	[f(Y),1]
[[1,2],[3,4]]	[1,2]	[[3,4]]
[X+Y,p(sofia)]	X+Y	[p(sofia)]

Елементи на списъците

Разнообразни типове

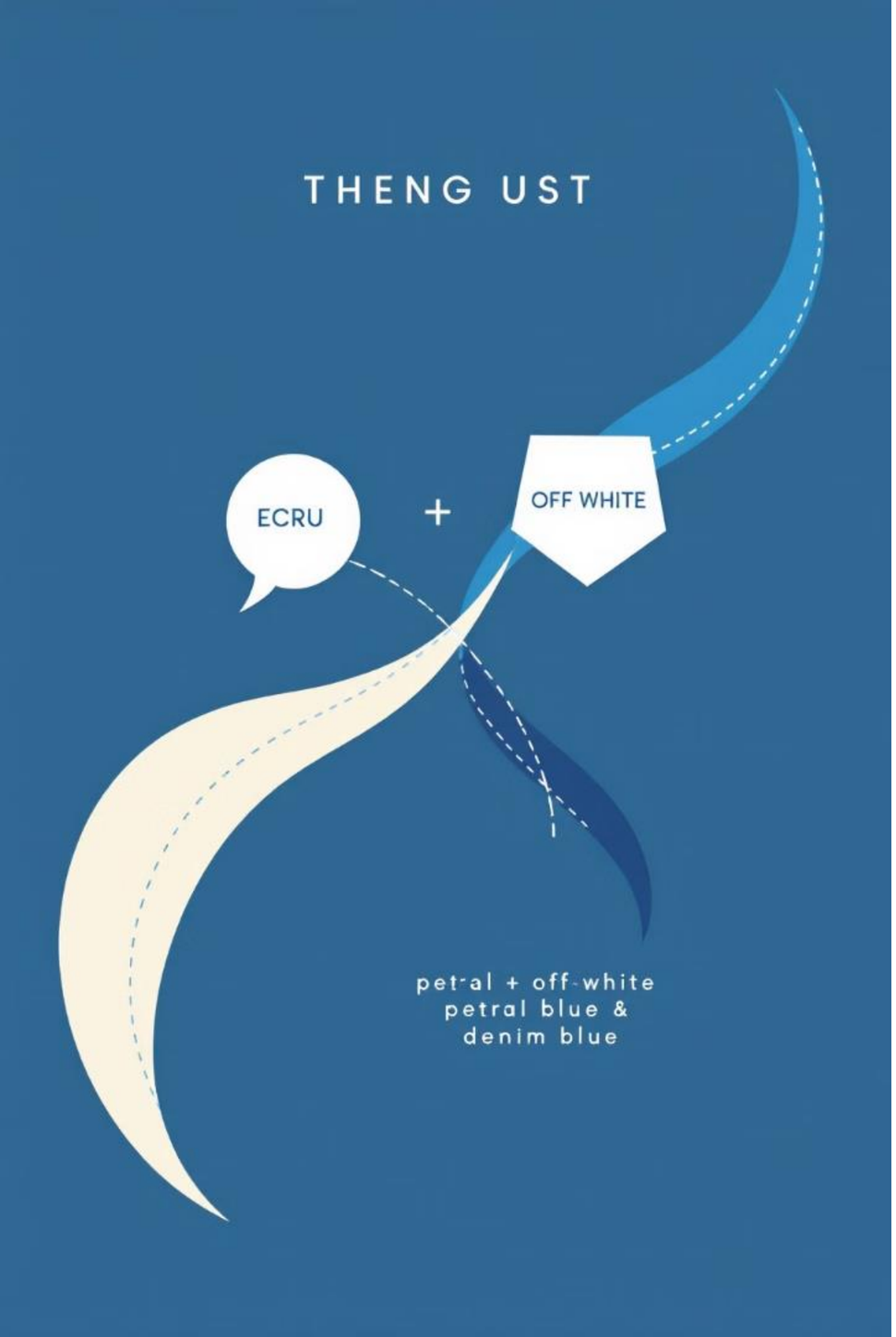
Елементите могат да бъдат атоми, числа или променливи.

Вложени списъци

Списъците могат да съдържат други списъци.

Пример

[a, 1, X, [b, c], f(Y)] демонстрира различни типове елементи.



Унификация на списъци

1

Съпоставяне

Унификацията съпоставя структурите на списъците.

2

Пример 1

[X, Y] = [1, 2] води до X = 1, Y = 2.

3

Пример 2

[A|B] = [1, 2, 3] води до A = 1, B = [2, 3].

Унификация на списъци – примери.

?- [X|Y]=[a,b,c].

X=a, Y=[b,c]

?- [X,Y|Z]=[a,b].

X=a, Y=b, Z=[]

?- [X,Y|Z]=[a].

no

?-[X,_,Y|Z]=[a,b,c,d,e].

X=a, Y=c, Z=[d,e]

?-[X,Y]=[a,b,c.d].

no

?-[X,Y]=[a,b].

X=a, Y=b

?-[X,a,X,f(X,a)|Y]=[Z,Z|L].

X=Z=a, Y=_, L=[a,f(a,a)|Y]

?-
[_,X,7,Y,_,f(1,abc)|Z]=[99,abc,ABC,X,f(s,s,s),C,3,abc|_].

X=Y=abc, Z=[3,abc|_], ABC=7, C=f(1,abc)

?-
[X,7,[Y|L],34|Z]=[[a,b,c],ABC,X,34,a(1,2)|[x,y,z]].

X=[a,b,c], Y=a, L=[b,c], Z=[a(1,2),x,y,z],ABC=7

Обработка на списъци

1 Рекурсивни предикати

Използват се рекурсивни предикати за обработка.

2 Базов случай

Празният списък е базовият случай на рекурсията.

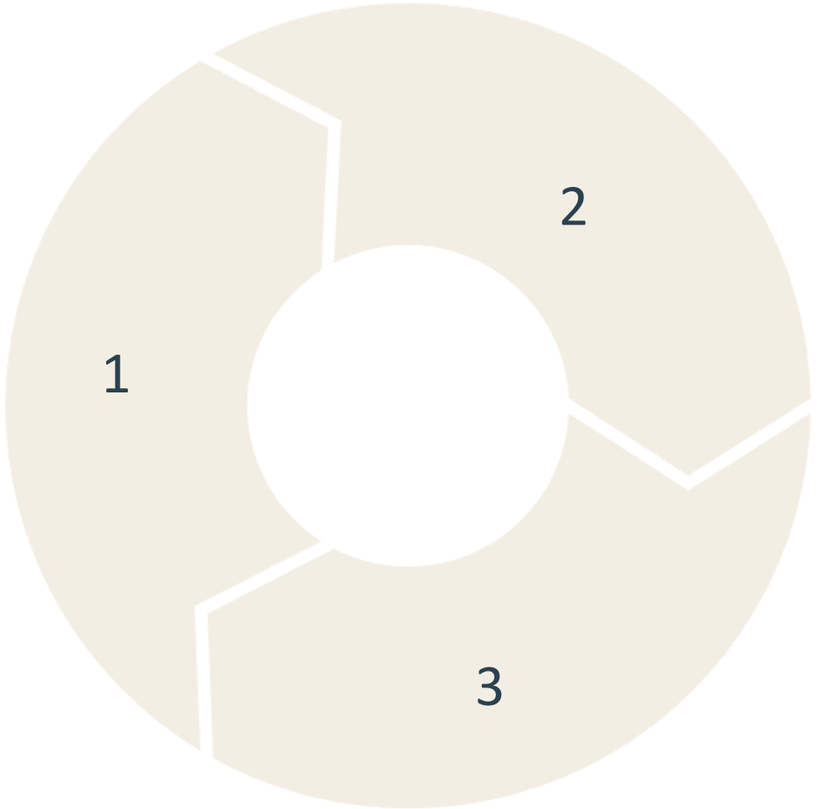
3 Рекурсивен случай

Обработка на главата и рекурсия върху опашката.

Търсене на елемент в списък

Предикат

Използва се предикатът member/2.



member(X, [X|_])

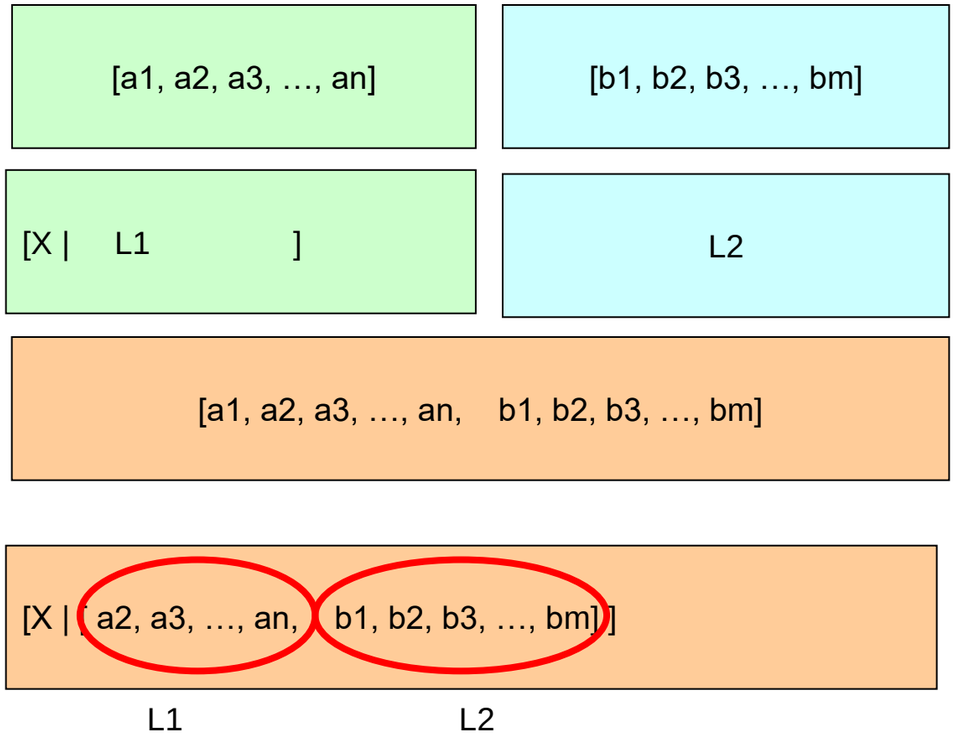
member(X, [_|T])

Рекурсивно търсене в опашката.

Проверка за принадлежност на даден елемент на списък

```
% вграден
member(X,[X|_]).
member(X,[_|L]):-
member(X,L).
```

?- member(1,[1,2,3]).
yes
?- member(1,[4,1,2]).
yes
?- member(X,[1,2,3]).
X = 1 ? ;
X = 2 ? ;
X = 3 ? ;
No



Свързване на списъци

1 Предикат

Използва се append/3.

2 append([], L, L)

3 append([H|T], L, [H|R])

Рекурсивно свързване на опашките.

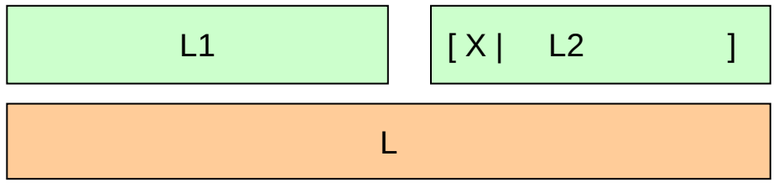
Слепване на списъци

```
% вграден
append([],L2,L2).
append([X|L1],L2,[X|L3]):-
append(L1,L2,L3).
 ?- append([1,2],[3,4],X).
X = [1,2,3,4] ? ;
no
 ?- append(X,[2,3],[1,2,3]).
X = [1] ? ;
no
 ?- append([1,2],X,[1,2,3]).
X = [3] ? ;
no
```

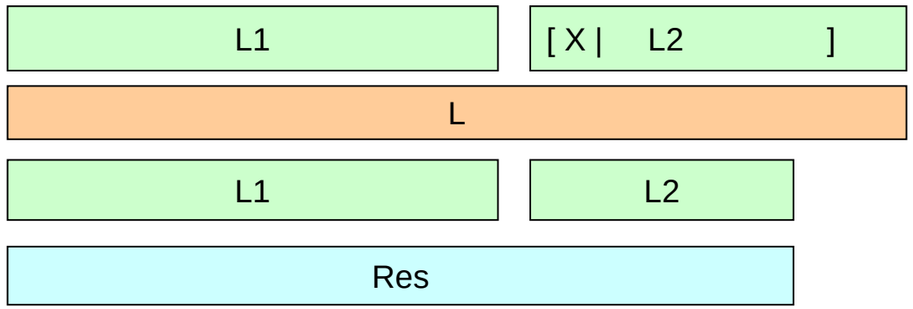
```
 ?- append([1,2],[3,4],X).
X = [1,2,3,4] ? ;
no
 ?- append(X,[2,3],[1,2,3]).
X = [1] ? ;
no
 ?- append([1,2],X,[1,2,3]).
X = [3] ? ;
no
 ?- append(X,Y,[1,2,3]).
X = [],
Y = [1,2,3] ? ;
X = [1],
Y = [2,3] ? ;
X = [1,2],
Y = [3] ? ;
X = [1,2,3],
Y = [] ? ;
no
```

Дефиниции чрез append

member(X,L):- append(_,[X|_],L).



del(X,L,Res):- append(L1,[X|L2],L), append(L1,L2,Res).



Вградени предикати

length/2

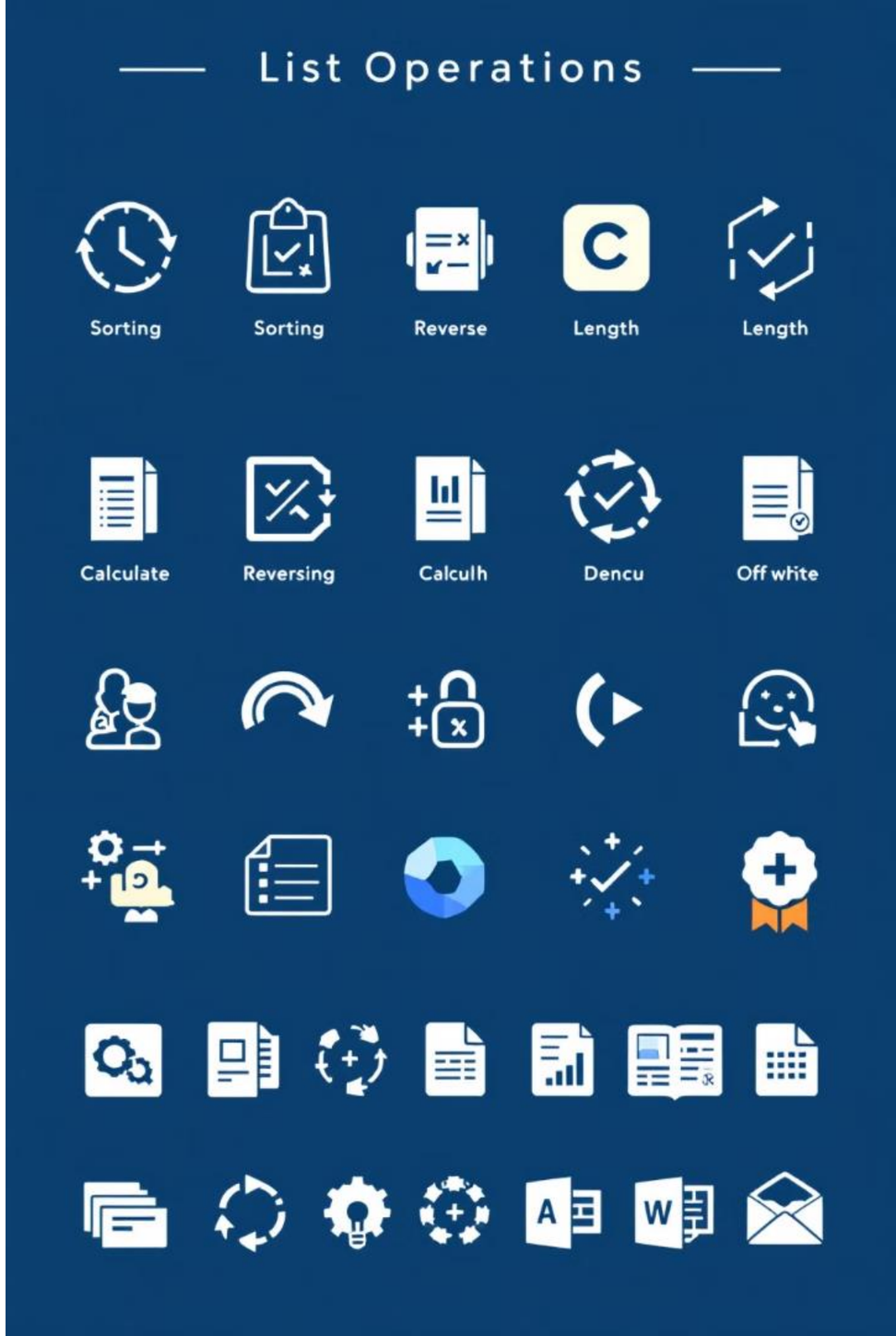
Дължина на списък.

reverse/2

Обръщане на списък.

sort/2

Сортиране на списък.



Дължина на списък

```
% вграден
length([ ], 0).
length([X|L],N):-
        length(L,M),
        N is M+1.
```

Изтриване на елемент

1

delete(X, [X|T], T)

2

delete(X, [H|T], [H|R])

Рекурсивно изтриване.

Изтриване на първото срещане на елемент в списък

```
del(X,[X|L],L).
del(X,[Y|L],[Y|Res]):-
    del(X,L,Res).

?- del(1,[1,2,3],L).
L = [2,3] ? ;
no
?- del(1,[1,2,1,3],L).
L = [2,1,3] ? ;
L = [1,2,3] ? ;
no
```

Изтриване на всички срещания на елемент в списък

```
del_all(X,[],[]).
del_all(X,[X|L],Res):- del_all(X,L,Res).
del_all(X,[Y|L],[Y|Res]):- del_all(X,L,Res).

?- del_all(1,[1,2,1,3],L).
L = [2,3] ? ;
L = [2,1,3] ? ;
L = [1,2,3] ? ;
L = [1,2,1,3] ? ;
no
```

Забележете, че del и del_all не правят това което очакваме

Изтриване на всички срещания на елемент в списък

```
del_all(X,[],[]).
del_all(X,[X|L],Res):-
                        del_all(X,L,Res).
del_all(X,[Y|L],[Y|Res]):-
                        X \=Y, del_all(X,L,Res).

 ?- del_all(1,[1,2,1,3],L).
L = [2,3] ? ;
no
```

Примерни задачи

Последен елемент

Намиране на последния елемент в списъка.

Премахване на дубликати

Премахване на повтарящи се елементи.

Разделяне на списък

Разделяне на две части.

Задачи:

1.Напишете предикат, който има два аргумента елемент и списък и е верен, ако елементът е последен елемент на списъка.

```
last(X,[X]).
last(X,[_|Z]):-last(X,Z).
```

2.Напишете предикат, който има два аргумента елемент и списък и е верен, ако елементът принадлежи на списъка, но не само на първо, а на произволно ниво.

```
member_1(Y,[Y|_]).
member_1(Y,[_|Z]):-member_1(Y,Z).
member_1(Y,[X|_]):-member_1(Y,X).
```

Особености:

При преудовлетворяване нещата стоят по същият начин, както при обикновения предикат member.

3.Напишете предикат, който има три аргумента първите два са елементите Ел1 и Ел2,а третият е списък и е верен, ако елементите Ел1 и Ел2 принадлежат на списъка в този ред и са последователни.

```
sl(X,Y,[X,Y|_]).
sl(X,Y,[_|Z]):-sl(X,Y,Z).
```

4.Напишете предикат, който има два аргумента- списъци и е верен, ако първият аргумент е подсписък на втория.

```
match([],_).
match([X|L],[X|M]):-match(L,M).
sub([X|L],[X|M]):-match(L,M).
sub(L,[_|M]):-sub(L,M).
sub([],_).
```

Особености:

Предикатът match проверява дали един списък е начало на друг.

5.Напишете предикат, който има два аргумента- списъци и е верен, ако първият аргумент е списък, в който елементите на втория са изброени в обратен ред.

```
rev(L1,L2):-rv(L1,[], L2).
rv([], L,L).
rv([X|L], L1,L2):-rv(L,[X|L1], L2).
```

Особености

Предикатът използва помощен предикат, при който имаме една променлива в повече, която е работна, и в нея натрупваме текущия резултат.

6.Реализирайте предикатите за принадлежност и последен елемент чрез append.

```
member1(X,Y):-append(_,[X|_], Y).
last1(X,Y):-append(_,[X], Y).
```

Задачи

Задача 1: Пет деца – таланти

В семейство има пет деца (братя и сестри), съответно на 4,5,6,7 и 8 години, които имат различни таланти. Едното от тях се казва Невена, а друго свири на пиано. Иванка е на 4 години и не разбира от математика. Детето, което е програмист е с една година по-голямо от Иван. Детето, което свири на китара е на 7 години. Йоана не е на 8 години. Станко е на 5 години и е по-малък от този, който разбира от литература. Кой на колко години е и в каква насока е талантът му?

ПОДРОБНО ОПИСАНИЕ НА РЕШЕНИЕТО
deca(Ds)
Ds е списък от деца

В задачата е дадено:
1.В семейство има 5 деца, съответно на 4, 5, 6, 7 и 8 години, които имат различни таланти.
2.Едно от тях се казва Невена,
3.а друго свири на пиано, т.е Невена не свири на пиано
4.Иванка е на 4 години и не разбира от математика.
5.Детето, което е програмист е с една година по-голямо от Иван
6. Детето, което свири на китара е на 7 години.
7. Йоана не е на 8 години.
8. Станко е на 5 години и е по-малък от този, който разбира от литература.
Кой на колко години е и в каква насока е талантът му?

```
% Създаваме таблица за децата
:- use_rendering(table,[header(d('Име', 'Талант', 'Години'))]).

deca(Ds) :-
        % всяко дете в списъка съдържа данни за:
        %      d(Name, Talent, Age)

length(Ds, 5), % дължина на списъка с деца Ds
%Проверка за принадлежност на даден елемент на списъка
    member(d(ivanka,_,4), Ds),      % Иванка е на 4 години
    member(d(_,kitara,7), Ds),
% Детето, което свири на китара е на 7 години
    member(d(ivan,_,_),Ds),         % Дете Иван
    member(d(nevena,_,_), Ds),      % Дете Невена
    member(d(stanko,_,5), Ds),      % Станко е на 5 години
```

```
member(d(_,programist,_), Ds),    % Дете - програмист
    member(d(yoana,_,_),Ds),      % Дете Йоана
    member(d(_,literatura,6),Ds),
% Детето, което разбита от литература е по-голямо от Станко и е на
% 6 години
    member(d(_,piano,_),Ds),      %Дете, което свири на пиано
    member(d(_,matematika,_),Ds),% Дете - математик
    member(d(_,_,8),Ds),          % Дете, което е на 8 години
    ((member(d(ivan,_,6),Ds), member(d(_,programist,7),Ds));
% ако Иван е на 6 години, програмиста е на 7 години
    (member(d(ivan,_,7),Ds), member(d(_,programist,8),Ds))),
% ако Иван е на 7 години, програмиста е на 8 години
```

```
    not(member(d(ivanka,matematika,_),Ds)),
% Иванка не е математик

    not(member(d(nevena,piano,_),Ds)),
% Невена не свири на пиано

    not(member(d(yoana,_,8),Ds)),
% Йоана не е на 8 години


/* искаме списъка да бъде подреден по възрастта на
децата */
Ds= [d(_,_,4),d(_,_,5),d(_,_,6),d(_,_,7),d(_,_,8)].

 /* ?-deca(Deca). */
```

Присъствие
12.05.2025 г.

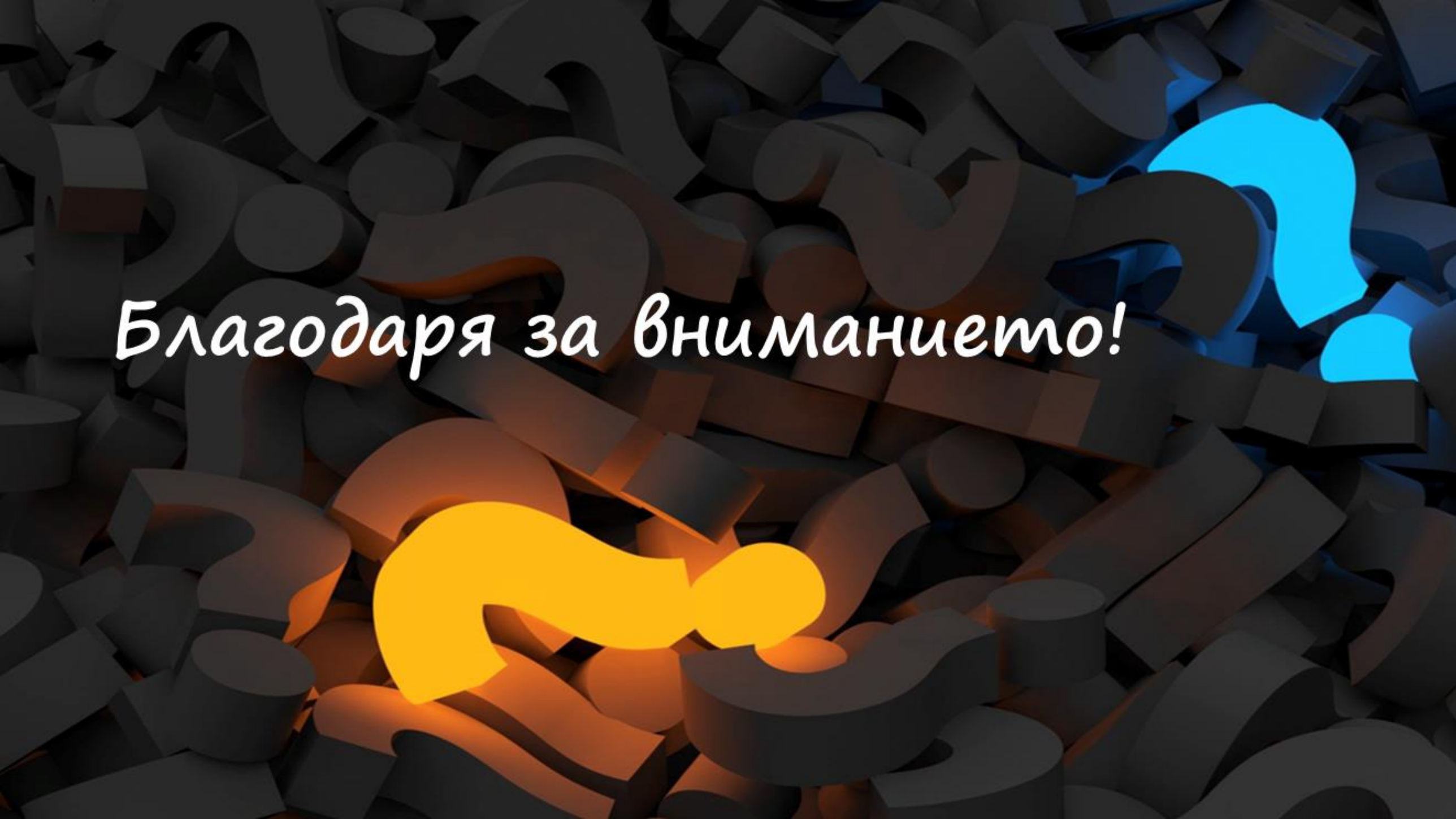
Регистрация

https://tinyurl.com/24xwqljp

Присъствие
13.05.2025 г.

Регистрация

https://tinyurl.com/23v7xjle


Благодаря за вниманието!