3. Անդրադարձում և հատում: Թվային խնդիրների լուծում

3.1. Անդրադարձում

Անդրադարձումը առաջանում է այն դեպքում, երբ տվյալների բազան պարունակում է ***անդրադարձ կանոններ***, այսինքն՝ այնպիսի կանոններ, որոնց աջ մասում հանդիպում է ձախ մասում օգտագործված պրեդիկատը: Անդրադարձումը Prolog լեզվում ունի առանձնահատուկ նշանակություն. այն փոխարինում է պրոցեդուրային ծրագրավորման լեզուներում օգտագործվող ցիկլի հրահանգը: Դիտարկենք անդրադարձման օգտագործման մի շարք օրինակներ:

3.1.1. Ֆակտորիալի հաշվում

Ֆակտորիալ ֆունկցիայի անդրադարձ սահմանումն է.

$$\mathbf{n! = \begin{cases} 1, & \text{if } \ n = 0 \\ (n-1)! \cdot n, & \text{if } \ n > 0 \end{cases}}$$

Նշանակենք **fact(N, P)** պնդումը, ըստ որի՝ **P**-ն **N** թվի ֆակտորիալն է: Հենվելով ֆակտորիալ ֆունկցիայի անդրադարձ սահմանման վրա՝ **fact(N, P)** պրեդիկատը (հարաբերությունը) ներկայացնենք հետևյալ կերպ.

```
fact(0, 1).
fact(N, P) :- N > 0, K is N - 1, fact(K, Q), P is Q*N.
```

Դիտարկենք Prolog-ի ինտերպրետատորի աշխատանքը հետևյալ հարցման դեպքում.

?- fact(3, X).

Քայլ 1.

X≡P

```
3>0
K is 3-1
fact(K, Q)
P is Q* 3
```

Քայլ 2.

X≡P

```
fact(2, Q)
P is Q* 3
```

Քայլ 3.

X≡P

Q≡P

```
2>0
K is 2-1
fact(K, Q)
P is Q* 2
```

P is Q* 3

Քայլ 4.

X≡P

Q≡P

fact(1, Q)
P is Q* 2

P is Q* 3

Քայլ 5.

X≡P

Q≡P

Q≡P

1>0
K is 1-1
fact(K, Q)
P is Q* 1

P is Q* 2

P is Q*3

Քայլ 6.

X≡P

Q≡P

Q≡P

fact(0, Q)
P is Q* 1

P is Q* 2

P is Q*3

Քայլ 7.

X≡P

Q≡P

Q≡P

Q ≡ 1

P is Q* 1

P is Q* 2

P is Q*3

Քայլ 8.

X≡P

Q≡P

Q≡P

P is 1* 1

P is Q* 2

P is Q*3

Քայլ 9.

X≡P

Q≡P

Q≡1

P is Q* 2

P is Q*3

Քայլ 10.

X≡P

Q≡P

P is 1* 2

P is Q*3

Քայլ 11.

X≡P

Q≡2

P is Q*3

Քայլ 12.

X≡P

P is 2*3

Քայլ 13.

X≡6

3.1.2. Ամենամեծ ընդհանուր բաժանարարի հաշվում

Երկու բնական թվերի ***ամենամեծ ընդհանուր բաժանարարը*** կարելի է հաշվել էվկլիդեսի ալգորիթմով, որի անդրադարձ ներկայացումն է.

$$\mathbf{gcd(x, y)} = \begin{cases} \mathbf{x}, & \mathbf{if} \;\; \mathbf{x = y} \\ \mathbf{gcd(x - y, y)}, & \mathbf{if} \;\; \mathbf{x > y} \\ \mathbf{gcd(x, y - x)}, & \mathbf{if} \;\; \mathbf{y > x} \end{cases}$$

Նշանակենք **gcd(X, Y, D)** պնդումը, ըստ որի՝ **D**-ն **X** և **Y** թվերի ամենամեծ ընդհանուր բաժանարարն է: Վերը բերված անդրադարձ նկարագրին համապատասխանում է **gcd** պրեդիկատի հետևյալ սահմանումը.

```
gcd(X, X, X).
gcd(X, Y, D):-  X > Y, X1 is X - Y, gcd(X1, Y, D).
gcd(X, Y, D):-  Y > X, Y1 is Y - X, gcd(X, Y1, D).
```

3.1.3. Աստիճանի հաշվում

Բնական թվի ամբողջ ոչ բացասական ***աստիճանը*** կարելի է հաշվել հետևյալ անդրադարձ եղանակով.

$$\mathbf{x^n} = \begin{cases} \mathbf{1}, & \mathbf{if} \;\; \mathbf{n = 0} \\ \mathbf{x \cdot x^{n-1}}, & \mathbf{if} \;\; \mathbf{n > 0} \end{cases}$$

Նշանակենք **power(X, N, Y)** պնդումը, ըստ որի՝ **X** բնական թվի **N**-րդ աստիճանն է **Y**-ը: Վերը բերված բանաձևին համապատասխանում է **power** պրեդիկատի հետևյալ սահմանումը.

```
power(_, 0, 1).
power(X, N, Y):-  N > 0, N1 is N - 1, power(X, N1, Y1), Y is Y1*X.
```

Նկատենք, որ աստիճանի հաշվումը ավելի արդյունավետ կլինի հետևյալ բանաձևի օգտագործման դեպքում.

$$\mathbf{x^n} = \begin{cases} \mathbf{1}, & \mathbf{if = 0} \\ \mathbf{x^{n/2} \cdot x^{n/2}}, & \mathbf{if} \;\; \mathbf{n > 0 \; and \; n \;\; is \; even} \\ \mathbf{x^{n/2} \cdot x^{n/2} \cdot x}, & \mathbf{if} \;\; \mathbf{n > 0 \; and \; n \;\; is \; odd} \end{cases}$$

Հիմք ընդունելով այս բանաձևը՝ **power** պրեդիկատը սահմանենք հետևյալ կերպ.

```
power(_, 0, 1).
power(X, N, Y):- N > 0, 0 is N mod 2, M is N//2, power(X, M, Z), Y is Z*Z.
power(X, N, Y):- N > 0, 1 is N mod 2, M is N//2, power(X, M, Z), Y is Z*Z*X.
```

3.1.4. Ֆիբոնաչիի շարքի N-րդ անդամի հաշվում

Ֆիբոնաչիի շարքի n-րդ անդամը սահմանվում է հետևյալ անդրադարձ հավասարման միջոցով.

$$f_n = \begin{cases} 1, & \text{if } n = 0 \text{ or } n = 1 \\ f_{n-2} + f_{n-1}, & \text{if } n \geq 2 \end{cases}$$

Նշանակենք **fib(N, X)** պնդումը, ըստ որի Ֆիբոնաչիի շարքի **N**-րդ անդամն է **X**-ը: **fib** պրեդիկատի սահմանումն է.

```
fib(0, 1).
fib(1, 1).
fib(N, X) :-  N >= 2, N2 is N - 2, N1 is N - 1, fib(N2, X2), fib(N1, X1), X is X2 + X1.
```

Նկատենք, որ այս ծրագրով Ֆիբոնաչիի շարքի միևնույն անդամը հաշվվում է բազմաթիվ անգամ: Այսպես, օրինակ, **fib(5, X)** հարցումը ծնում է **fib(3, X)** և **fib(4, X)** հարցումները, իսկ վերջինս՝ կրկին **fib(3, X)** հարցումը: Ավելի արդյունավետ ծրագիր ստանալու համար սահմանենք **fib1(N, X1, X)** օժանդակ պնդում, որը բավարարվում է այն և միայն այն դեպքում, երբ **N > 0**, իսկ **X1**-ը և **X**-ը համապատասխանաբար Ֆիբոնաչիի շարքի **(N - 1)**-րդ և **N**-րդ անդամներն են:

fib1 պրեդիկատի սահմանումն է.

```
fib1(1, 1, 1).
fib1(N, X1, X):-  N > 1, N1 is N - 1, fib1(N1, X2, X1), X is X2 + X1.
```

Հիմք ընդունելով **fib1** պրեդիկատը՝ **fib** պրեդիկատը ներկայացնենք հետևյալ կերպ.

```
fib(0, 1).
fib(N, X):- N > 0, fib1(N, _, X).
```

3.2. Հատում

3.2.1. Ընդհանուր դրույթներ

Prolog-ի ինտերպրետատորը այս կամ այն նպատակային պնդումն ապացուցելիս հաջորդաբար դիտարկում է բոլոր հնարավոր տարբերակները՝ կիրառելով ***վերադարձով որոնման (backtracking)*** մեթոդը: Սակայն բոլոր տարբերակների կուրորեն դիտարկումը ծրագիրը կարող է դարձնել ոչ արդյունավետ:

Դիտարկենք, օրինակ, հետևյալ կանոնը.

fatherfather(X, Y):- father(Z, Y), father(X, Z).

որով հավաստվում է, որ **X**–ը **Y**–ի հոր հայրն է, եթե որևէ **Z**–ը **Y** –ի հայրն է, իսկ **X** –ը՝ այդ **Z**–ի:

Պարզ է, որ առաջին կամ երկրորդ նպատակային պնդումն ապացուցելուց հետո վերադարձը դեպի ետ և նոր տարբերակների որոնումը անիմաստ է (կենսաբանական այլ հայրեր չեն կարող լինել): Նման իրավիճակներում ավելորդ տարբերակների դիտարկումը կարելի է բացառել՝ օգտագործելով ***հատում*** կոչվող **0** տեղանի նախասահմանված պրեդիկատը, որը նշանակվում է **!**-ով: **!** պրեդիկատը համադրվում է ցանկացած տվյալների բազայի հետ, այն է՝ **!** պնդումը նույնաբար ճիշտ է համարվում: Միևնույն ժամանակ **!** պնդման ապացույցը առաջացնում է կողմնակի էֆեկտ. Prolog-ի ինտերպրետատորն այլ տարբերակներ որոնելու համար նշված կետում չի կիրառում վերադարձի մեխանիզմը:

Դիտարկված օրինակում ծրագրի աշխատանքը կարող ենք դարձնել ավելի արդյունավետ, եթե վերը նշված կանոնը արտագրենք հետևյալ տեսքով.

fatherfather(X, Y):- father(Z, Y), !, father(X, Z), !.

Այսպիսով՝ հատումը հնարավորություն է տալիս.

- արագացնելու ծրագրի կատարումը՝ բացառելով անիմաստ տարբերակների դիտարկումը,
- խնայելու հիշողություն՝ նվազեցնելով վերադարձի հնարավոր կետերի քանակը:

Հատման մեխանիզմը նպատակահարմար է կիրառել անվերջ անդրադարձումը վերացնելու, ինչպես նաև միմյանց բացառող տարբերակների դիտարկումը արդյունավետ դարձնելու համար:

3.2.2. Անվերջ անդրադարձման բացառում

Դիտարկենք ֆակտորիալի հաշվման հետևյալ ծրագիրը՝

fact(0, 1).

fact(N, P):- K is N - 1, fact(K, Q), P is Q*N.

Եթե դիմենք այս ծրագրին

?- fact(0, X).

հարցումով, ապա համակարգը կտա

X=1?

պատասխանը: Սակայն, եթե մենք պահանջենք շարունակել որոնումը, ապա կանցնենք **fact(-1, X)** պնդման ապացույցին, այնուհետև՝ **fact(-2, X)** պնդման ապացույցին, և այսպես շարունակ: Նկատենք, որ եթե ծրագրի տողերը փոխանակվեն տեղերով, ապա անվերջ անդրադարձում կառաջանա ցանկացած դեպքում:

Բերված ծրագրում անվերջ անդրադարձումը վերացնելու համար բավարար է հատման պրեդիկատը աջ մասում պարունակող կանոնի վերածել առաջին փաստը.

fact(0, 1):- !.

fact(N, P):- K is N-1, fact(K, Q), P is Q*N.

3.2.3. Միմյանց բացառող պնդումների ծրագրավորում

Դիտարկենք հետևյալ ֆունկցիան՝

$$f(x) = \begin{cases} -1, & \text{if } x < -1 \\ 0, & \text{if } -1 \le x \le 1 \\ 1, & \text{if } x > 1 \end{cases}$$

Եթե այս սահմանումը տառացիորեն ներկայացնենք Prolog լեզվով, ապա կստանանք.

f(X, -1) :- X < -1.

f(X, 0) :- -1 =< X, X =< 1.

f(X, 1) :- X > 1.

Բերված ծրագիրը կդառնա ավելի արդյունավետ, եթե օգտագործվի հատման մեխանիզմը.

f(X, -1) :- X<-1, !.

f(X, 0) :- X=<1, !.

f(X, 1).

Այս ծրագրում չեն դիտարկվում ավելորդ տարբերակներ, և չեն կատարվում ավելորդ ստուգումներ: Նկատենք, որ **!** պրեդիկատն այստեղ էական նշանակություն ունի. եթե գրենք.

```
f(X, -1) :-  X <- 1.
f(X,  0) :-  X =< 1.
f(X, 1).
```

ապա կառաջանա Prolog համակարգի հետ գործակցության հետևյալ սցենարը.

```
?- f(-10, Y).
Y=-1?;
Y=0 ?;
Y=1
yes
```

3.2.4. Հատման հատկությունները

Հատման գործողությունը ավելի բարդ է դարձնում Prolog ծրագրի հիմքում ընկած տրամաբանությունը:

Դիտարկենք հետևյալ օրինակը.

```
P :- A, B.
P :- C.
```

որտեղ **A**-ն, **B**-ն, **C**-ն և **P**-ն տրամաբանական պնդումներ են:

Այս ծրագիրը ներկայացնում է մի պնդում **P**-ի մասին, որն արտահայտվում է հետևյալ բանաձևի միջոցով.

$$P \Leftrightarrow (A \,\&\, B) \vee C,$$

ընդ որում՝ այս բանաձևը արդարացի կլինի նաև ծրագրի տողերը տեղափոխելու դեպքում:

Այժմ դիտարկենք հետևյալ համարժեքությունները.

P:- A, !, B. **P:- C.**	համարժեք է	$P \Leftrightarrow (A \,\&\, B) \vee (\neg A \,\&\, C)$
P:- C. **P:- A, !, B.**	համարժեք է	$P \Leftrightarrow C \vee (A \,\&\, B)$

Բերված օրինակներից հետևում է, որ կրճատման գործողությունը օգտագործելու դեպքում ծրագրի տրամաբանությունը ավելի բարդ է դառնում, և ծրագիրը զգայուն է լինում փաստերի և կանոնների տեղափոխության նկատմամբ: Սա այն գինն է, որն անհրաժեշտ է վճարել բարձր արդյունավետություն ապահովելու համար:

3.2.5. fail և true նպատակային պնդումներ

Երբեմն նպատակահարմար է օգտագործել **fail** և **true 0** տեղանի ներդրված պրեդիկատները, որոնց հետ կապված է հետևյալ իմաստը. **fail** պրեդիկատի ապացույցը մշտապես անհաջողությամբ է ավարտվում, իսկ **true** պրեդիկատի ապացույցը, ընդհակառակը, մշտապես հաջողությամբ:

Օրինակ՝ **P** պնդման ժխտումը ներկայացնող **Q** պրեդիկատը կարող ենք սահմանել հետևյալ կերպ.

Q:- P, !, fail.

Q:- true.

3.2.6. Ամփոփում

Կիրառենք հատման գործողությունը՝ այս բաժնում դիտարկված ծրագրերն ավելի արդյունավետ դարձնելու համար:

Ֆակտորիալի հաշվում

```
fact(0, 1) :-  !.
fact(N, P) :-  K is N - 1, fact(K, Q), P is Q*N.
```

Ամենամեծ ընդհանուր բաժանարարի հաշվում

```
gcd(X, X, X) :-  !.
gcd(X, Y, D) :-  X > Y, X1 is X - Y, gcd(X1, Y, D), !.
gcd(X, Y, D) :-  Y1 is Y - X, gcd(X, Y1, D).
```

Աստիճանի հաշվում

```
power(_, 0, 1) :-  !.
power(X, N, Y) :-  0 is N mod 2, M is N//2, power(X, M, Z), Y is Z*Z, !.
power(X, N, Y) :-  M is N//2, power(X, M, Z), Y is Z*Z*X.
```

Ֆիբոնաչիի շարքի N-րդ անդամի հաշվում

```
fib1(1, 1, 1) :-  !.
fib1(N, X1, X) :-  N1 is N - 1, fib1(N1, X2, X1), X is X2 + X1.
fib(0, 1) :- !.
fib(N, X) :-  fib1(N, _, X).
```

Խնդիրներ

1. Որոշել տրված բնական թվի տասական թվանշանների գումարը:
2. Որոշել տրված բնական թիվը պալինդրո՞մ է, թե՞ ոչ (այն է՝ ճի՞շտ է արդյոք, որ համընկնում են այդ թվի առաջին ու վերջին, երկրորդ ու նախավերջին և այդպես հաջորդաբար թվանշանները):