6. Բինար ծառեր

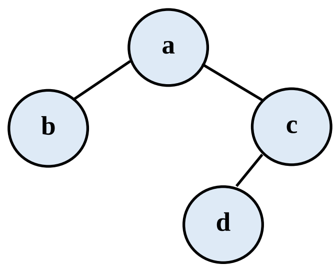
6.1. Բինար ծառեր

Բինար ծառ տվյալների տիպը սահմանենք որպես **binaryTree** հետևյալ պրեդիկատին բավարարող թերմերի բազմություն.

binaryTree(nil):- !.

binaryTree(bTree(L, _, R)):- binaryTree(L), binaryTree(R).

Օրինակ՝ Նկ. 1-ում բերված բինար ծառին



Նկ. 1

համապատասխանում է

```
bTree(
    bTree(nil, b, nil),
    a,
    bTree(
        bTree(nil, d, nil),
        c,
        nil
    )
)
```

կառուցվածքը:

Բինար ծառերի մշակման խնդիրներ

Կառուցենք բինար ծառի մի քանի բնութագրիչների որոշման ծրագրեր Prolog լեզվով:

Ծավալի հաշվում: Բինար ծառի ծավալը (գագաթների քանակը) նկարագրվում է հետևյալ անդրադարձ եղանակով.

- դատարկ ծառի ծավալը **0** է,
- ոչ դատարկ ծառի ծավալը մեկով ավելի է արմատի ձախ և աջ ենթածառերի ծավալների գումարից:

Դիցուք **size(T, N)**-ը պնդում է, ըստ որի՝ **T** բինար ծառի ծավալը **N** է: Ծավալի որոշման անդրադարձ նկարագրին համապատասխանում է **size** պրեդիկատի հետևյալ սահմանումը.

```
size(nil, 0):-  !.
size(bTree(L, _, R), N):-  size(L, N1), size(R, N2), N is N1 + N2 + 1.
```

***Բարձրության հաշվում*:** Բինար ծառի բարձրությունը սահմանվում է որպես արմատից դեպի տերև տանող ամենաերկար պարզ ճանապարհի երկարություն: Բինար ծառի բարձրությունը կարող ենք որոշել հետևյալ անդրադարձ եղանակով.

- դատարկ ծառի բարձրությունը **-1** է,
- ոչ դատարկ ծառի բարձրությունը մեկով ավելի է ձախ և աջ ենթածառերի բարձրություններից առավելագույնից:

Դիցուք **height(T, N)**-ը պնդում է, ըստ որի՝ **T** բինար ծառի բարձրությունը **N** է: Բարձրության որոշման անդրադարձ նկարագրին համապատասխանում է **height** պրեդիկատի հետևյալ սահմանումը.

```
height(nil, -1):-  !.
height(bTree(L, _, R), N):-  height(L, N1), height(R, N2),
                                              max(N1, N2, M), N is M + 1.
```

Այստեղ **max(N1, N2, N)**-ը պնդում է, ըստ որի՝ **N**-ը **N1** և **N2** թվերից առավելագույնն է: **max** պրեդիկատի սահմանումն է.

```
max(N1, N2, N1):-  N1 >= N2, !.
max(_, N2, N2).
```

Տերևների քանակի հաշվում: Բինար ծառի տերևների քանակը կարող ենք հաշվել հետևյալ անդրադարձ եղանակով.

- դատարկ ծառի տերևների քանակը **0** է,
- մեկ գագաթից բաղկացած ծառի տերևների քանակը **1** է,
- մեկից ավելի գագաթ ունեցող ծառի տերևների քանակն է ձախ և աջ ենթածառերի տերևների քանակների գումարը:

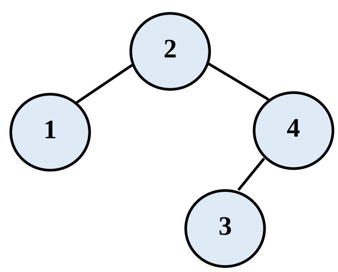
Նշանակենք **numberOfLeaves(T, N)** պնդումը, ըստ որի՝ որ **T** բինար ծառի տերևների քանակը **N** է: Տերևների քանակի որոշման անդրադարձ նկարագրին համապատասխանում է **numberOfLeaves** պրեդիկատի հետևյալ սահմանումը.

```
numberOfLeaves(nil, 0):-  ! .
numberOfLeaves(bTree(nil, _, nil), 1):-  !.
numberOfLeaves(bTree(L, _, R), N):-
            numberOfLeaves(L, N1),  numberOfLeaves(R, N2), N is N1 + N2.
```

6.2. Որոնման բինար ծառ

Բինար ծառը կոչվում է *որոնման բինար ծառ* (կամ պարզապես *որոնման ծառ*), եթե դրա գագաթները արժևորված են լրիվ կարգավորված բազմության տարրերով այնպես, որ յուրաքանչյուր գագաթի արժեքը մեծ է այդ գագաթի ձախ ենթածառի գագաթների արժեքներից և փոքր է կամ հավասար աջ ենթածառի գագաթների արժեքնե-րից:

Օրինակ՝ որոնման ծառ է Նկ. 2-ում բերված ծառը.



Նկ. 2

Գործողություններ որոնման ծառերի հետ

Որոնման ծառերի նկատմամբ սահմանված հիմնական գործողություններն են տարրի *որոնման*, *ավելացման* և *հեռացման* գործողությունները, որոնց բարդությունը միջինում լոգարիթմորեն է կախված տարրերի քանակից: Դիտարկենք այս գործողությունների իրականացումը Prolog լեզվով:

***Որոնում*:** Տարրի պատկանելիությունը որոնման ծառին սահմանենք հետևյալ անդրադարձ եղանակով. տարրը պատկանում է որոնման ծառին այն և միայն այն դեպքում, երբ այն.

- ծառի արմատի արժեքն է, կամ
- փոքր է ծառի արմատի արժեքից և պատկանում է արմատի ձախ ենթածառին, կամ
- մեծ է ծառի արմատի արժեքից և պատկանում է արմատի աջ ենթածառին:

Դիցուք **search(X, T)** - ն պնդում է, ըստ որի՝ **X** տարրը պատկանում է **T** որոնման ծառին: Պատկանելիության գործողության անդրադարձ նկարագրին համապատասխանում է **search** պրեդիկատի հետևյալ սահմանումը.

```
search(X, bTree(_, X, _)):-  !.
search(X, bTree(L, Y, _) ):-  X < Y, search(X, L), !.
search(X, bTree(_, _, R) ):-   search(X, R).
```

Ավելացում: Տարրի (կրկնումներով) ավելացումը որոնման ծառին սահմանենք հետևյալ անդրադարձ եղանակով.

- տարրը դատարկ ծառին ավելացնելու դեպքում ստացվում է այդ տարրով արժևորված մեկ գագաթ ունեցող ծառ,
- տարրը ոչ դատարկ ծառին ավելացնելիս այն ավելացվում է արմատի ձախ ենթածառին, եթե այն փոքր է արմատի արժեքից և աջ ենթածառին՝ հակառակ դեպքում:

Դիցուք **insert(X, T, T1)**-ը պնդում է, ըստ որի՝ **T1** որոնման ծառը ստացվում է **X** տարրը **T** որոնման ծառին ավելացնելու արդյունքում: Ավելացման գործողության անդրադարձ նկարագրին համապատասխանում է **insert** պրեդիկատի հետևյալ սահմանումը.

```
insert(X, nil, bTree(nil, X, nil)):-  !.
insert(X, bTree(L, Y, R), bTree(L1, Y, R)):-  X < Y, insert(X, L, L1), !.
insert(X, bTree(L, Y, R), bTree(L, Y, R1)):-  insert(X, R, R1).
```

Հեռացում: Տարրի առաջին հանդիպած արժեքի հեռացումը որոնման ծառից սահմանենք հետևյալ անդրադարձ եղանակով.

- տարրը դատարկ ծառից հեռացնելու դեպքում ստացվում է դատարկ ծառ,
- ոչ դատարկ ծառից արմատի արժեքից տարբերվող տարրի հեռացումը կատարվում է արմատի ձախ ենթածառից, եթե հեռացվող տարրը փոքր է արմատի արժեքից և արմատի աջ ենթածառից՝ հակառակ դեպքում:
- ոչ դատարկ ծառից արմատի արժեքի հեռացումը կարող է կատարվել հետևյալ կերպ.
  - եթե արմատի ենթածառերից որևէ մեկը դատարկ է, ապա արմատի արժեքը հեռացնելիս ստացվում է մյուս (հնարավոր է դատարկ) ենթածառը,
  - եթե արմատի ենթածառերը դատարկ չեն, ապա արմատի արժեքը հեռացնելիս արմատին վերագրվում է աջ ենթածառի փոքրագույն արժեքը, ինչից հետո այն հեռացվում է աջ ենթածառից (նշենք, որ արմատի աջ ենթածառում փոքրագույն արժեք պարունակող գագաթը չի կարող ունենալ ձախ ենթածառ):

Դիցուք **remove(X, T, T1)**-ը պնդում է, ըստ որի՝ **T1** որոնման ծառը ստացվում է **T** որոնման ծառից **X** տարրը հեռացնելու արդյունքում (ենթադրվում է, որ եթե **X**-ը չի հանդիպում է **T** ծառում, ապա **T1**-ը համընկնում է **T**-ի հետ): Հեռացման գործողության անդրադարձ նկարագրին համապատասխանում է **remove** պրեդիկատի հետևյալ սահմանումը.

```
remove(_, nil, nil):-  !.
remove(X, bTree(L, Y, R), bTree(L1, Y, R)):-  X < Y, remove(X, L, L1), !.
remove(X, bTree(L, Y, R), bTree(L, Y, R1)):-  X > Y, remove(X, R, R1), !.
remove(X, bTree(nil, X, R), R):-  !.
remove(X, bTree(L, X, nil), L):-  !.
remove(X, bTree(L, X, R), bTree(L, Y, R1)):- min(R, Y), remove(Y, R, R1).
```

Այստեղ **min(T, X)**-ը պնդում է, ըստ որի՝ **X**-ը **T** ոչ դատարկ որոնման ծառի մինիմալ տարրն է: **min** պրեդիկատի սահմանումն է.

```
min(bTree(nil, X, _), X):- !.
min(bTree(L, _, _), Y):- min(L, Y).
```