**Algebră booleană (Boole = matematicianul care s-a ocupat de algebra din logica matematică - o algebră definită peste mulțimea de valori (0,1))**

https://www.categories.acsl.org/wiki/index.php?title=Boolean\_Algebra

Operațiile de bază din algebra booleană:

1) Negația - care aici se notează cu o bară deasupra (NOT).

|  |  |
| --- | --- |
| a |  |
| 0 | 1 |
| 1 | 0 |

2) Conjuncția (și - AND) care aici se notează ca înmulțirea (de regulă cu ⋅ )

|  |  |  |
| --- | --- | --- |
| a | b | a⋅b |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

3) Disjuncția exclusivă (sau exclusiv - XOR) care aici se notează cu ⊕

|  |  |  |
| --- | --- | --- |
| a | b | a⊕b |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

4) Disjuncția (sau - OR) care aici se notează cu +

|  |  |  |
| --- | --- | --- |
| a | b | a+b |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Mai există și notații combinate: NOR - sau negat

NAND - și negat

XNOR sau EQUIV - ambele înseamnă . Operația XNOR se mai notează cu 🞊

**Identități în algebra booleană**

O identitate reprezintă o expresie algebrică, expresie care este adevărată indiferent de valorile pe care le au variabilele sale.

De exemplu, în algebra uzuală, formula de desfacere a lui (a+b)2 este o identitate.

În alegbra booleană, fiind alte reguli de calcul, identitățile pot avea forme diferite de cele din algebra normală.

**Iată identitățile cele mai importante:**

1) Comutativitatea: **x+y = y+x** și **x⋅y=y⋅x**

2) Asociativitatea: **(x+y)+z = x+(y+z)** și (**x⋅y)⋅z=x⋅(y⋅z)**

3) Distributivitatea: (**MARE ATENȚIE ! ÎN ALGEBRA BOOLEANĂ ESTE DISTRIBUTIVĂ ATÂT ÎNMULȚIREA FAȚĂ DE ADUNARE CÂT ȘI ADUNAREA FAȚĂ DE ÎNMULȚIRE!!**)

**x⋅(y+z)=x⋅y+x⋅z**

**x+y⋅z=(x+y)⋅(x+z)**

Identitățile în algebra booleană se demonstrează lunând valori (dat fiind că valorile posibile sunt 0 și 1 NU avem multe cazuri):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| x  (1) | y  (2) | z  (3) | y⋅z  (4)=(2)⋅(3) | x+y⋅z  (5)=(1)+(4) | x+y  (6)=(1)+(2) | x+z  (7)=(1)+(3) | (x+y)⋅(x+z)  (8)=(6)⋅(7) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

4) Idempotență: **x + x = x** **x⋅x = x**

5) Elementul neutru: **x + 0 = x** **x⋅1 = x**

6) Operațiile complementare: **x+ = 1 x⋅ = 0**

7) Operațiile anihilatoare (elementul nul) : **x + 1 = 1 x ⋅ 0 = 0**

8) Operațiile absobtive (care fac ca un termen să se piardă): ele de fapt sunt consecințe ale operațiilor de mai sus:

a) **x + x⋅y = x**  (demonstrație: scriem primul x ca și x⋅1, și apoi aplicăm inversul distrib. clasică - dăm factor comun: **x⋅1+x⋅y = x⋅(1+y)= x⋅1 = x**

b) **x + ⋅y = x+y** (demonstrație: aplicăm regula aia ciudată de distribuitivitate a lui + față de ⋅ ⇒ **x + ⋅y = (x+)⋅(x+y)= 1⋅(x+y) = x+y**

c) **x⋅(x+y) = x**  (demonstrație: aplicăm distributivitatea clasică a lui ⋅ față de + și se va obține fix prima expresie care am văzut că dă x: **x⋅(x+y)=x⋅x+x⋅y=x+x⋅y=x**

9) Factorizarea cu mai mulți termeni - ca la mate - se ia fiecare termen dintr-o paranteză cu fiecare termen din celalaltă paranteză:

**(x+y)⋅(z+t)=x⋅z+x⋅t+y⋅z+y⋅t**

10) Formulele lui DeMorgan: desfacerea negațiilor:

**= ⋅**

**= +**

11) Dubla negație păstrează valoarea originală:  **= x**

12) Scrierea lui XOR față de operațiile clasice (din acest motiv XOR este considerat operație secundară - poate fi exprimat de operațiile de bază):

**x⊕y = x⋅ + ⋅y**

13) Formule pentru XNOR:

**x🞊y = = x⊕ = ⊕y**

**Exerciții de algebră booleană**

1)

⋅ = A(A+B)⋅(+) = A(A+B)⋅(+) = A⋅(A+) = A

(ne reamintim că A(A+B)=A formulă pe care am aplicat-o de două ori)

Răspunsul este A

2) 

Metoda 1: luăm toate valorile posibile pentru A și B fără să facem vreun calcul de tip identitate și vedem la care dintre (A,B) expresia dă 1:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A  (1) | B  (2) | A+B  (3)=(1)+(2) | (4)= | (5)= | B  (6)=(5)⋅(2) | B  (7)=(4)+(6) | (8)= |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

Răspuns: expresia finală dă 1 pentru perechile (1,0) și (1,1)

De fapt, rezultatul expresiei coincide cu prima coloană, deci cu siguranță, dacă facem calculele corect, expresia asta complicată are valoarea A

Metoda2: calculăm expresia și vedem dacă ne prindem pentru cea simplificată, care sunt regulile:

= ⋅ = (A+B)(A+) = A⋅A+A⋅ =

A+ A⋅+A⋅B = A+A(B+) = A+A⋅1 = A+A = A

Se vede că cea simplificată este A, care dă 1 doar când A este 1 (de fapt nu mai depinde de B)

**STRUCTURI DE DATE**

Sunt structuri complexe, pe care atunci când le programăm putem folosi diverse elemente ale limbajului de programare (tablouri, structuri dinamice).

Ele sunt formate dintr-o colecție de elemente (valori) aranjate într-un mod particular.

Aceste structuri sunt de regulă la pachet cu metodele **push** și **pop** care se referă la introducerea unui element într structură respectiv la scoaterea unui element din structură.

Cele strudiate la ACSL sunt coada (queue), stiva (stack), arbori de căutare binară (binary search trees), cozi de priorități (priority queue sau heap)

**Coada**

Este o structură cu două capete, pe la un capăt se bagă elemente (**PUSH(element)**) iar pe la celălalt capăt se scot elemente (**POP(X)** - are semnificația că variabila X primește valoarea care se scoate din coadă). Se mai numește structură de tip FIFO (First In First Out cu sensul de primul care a intrat, în sensul de primul care s-a pus la coadă, adică cel mai vechi, este primul care iese)

Ea de fapt este modelarea unei cozi civilizate de persoane, în care operația PUSH(element) înseamnă că vine o persoană nouă la coadă iar POP(x) că persoana căreia i-a venit rândul (adică cea care stă de cel mai mult timp în coadă) va fi "deservită" și pleacă de la coadă.



Exemplu: Fie următoarele operații pe care le facem asupra unei cozi inițial vide. După ce se fac operațiile, care este valoarea lui Z?

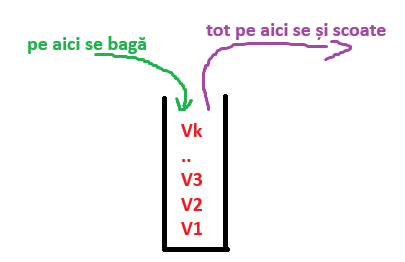
PUSH(3) PUSH(6) PUSH(8) POP(Y) POP(X) PUSH(X-Y) POP(Z)

|  |  |
| --- | --- |
| Operația | Coada |
| PUSH(3) | |  | | --- | | 3 | |
| PUSH(6) | |  |  | | --- | --- | | 3 | 6 | |
| PUSH(8) | |  |  |  | | --- | --- | --- | | 3 | 6 | 8 | |
| POP(Y) | |  |  | | --- | --- | | 6 | 8 |   Y=3 |
| POP(X) | |  | | --- | | 8 |   X=6 |
| PUSH(X-Y) | |  |  | | --- | --- | | 8 | 6 | |
| POP(Z) | |  | | --- | | 6 |   Z=8 |

**Stiva**

Este o structură cu un singur capăt, pe la care atât se bagă elemente (**PUSH(element)**) cât și se scot elemente (**POP(X)** - are semnificația că variabila X primește valoarea care se scoate din stivă). Se mai numește structură de tip LIFO (Last In First Out cu sensul de ultimul = cel mai recent care a intrat este primul care iese)

Ea de fapt este modelarea unei stive de obiecte (să zicem că unele foarte heavy, ca să nu putem lua obiecte de dedesubt) în care operația PUSH(element) înseamnă că deasupra celor existente mai vine unul iar POP(x) că se ia din stivă elementul cel mai de deasupra.



Exemplu: Fie aceleași operații pe care le facem asupra unei stiva inițial vide. După ce se fac operațiile, care este valoarea lui Z?

PUSH(3) PUSH(6) PUSH(8) POP(Y) POP(X) PUSH(X-Y) POP(Z)

|  |  |
| --- | --- |
| Operația | Stiva |
| PUSH(3) | |  | | --- | | 3 | |
| PUSH(6) | |  | | --- | | 6 | | 3 | |
| PUSH(8) | |  | | --- | | 8 | | 6 | | 3 | |
| POP(Y) | |  | | --- | | 6 | | 3 |   Y=8 |
| POP(X) | |  | | --- | | 3 |   X=6 |
| PUSH(X-Y) | |  | | --- | | -2 | | 3 | |
| POP(Z) | |  | | --- | | 3 |   Z=-2 |

**Arbori binari de căutare**

Sunt arbori în care fiecare nod are proprietatea că valoarea oricărui nod din subarborele stâng este mai mic sau egal cu nodul respectiv și orice nod din subarborele drept este strict mai mare.

Construcția unui arbore binar de căutare se face progresiv, adică:

- primul element al construcției reprezintă rădăcina arborelui

- orice element nou se dorește a fi băgat în arbore, se compară cu rădăcina și, dacă este mai mic sau egal se merge pe stânga, dacă nu pe dreapta, până ajungem într-un loc liber în care punem nodul.

Operația de băgat nodul în arbore se numește "PUSH(valoare)".

Să vedem cum se construiește un arbore cu "FAMERICANO"

|  |  |
| --- | --- |
| Operația | Arborele |
| PUSH(F) |  |
| PUSH(A) |  |
| PUSH(M) |  |
| PUSH(E) |  |
| PUSH(R) |  |
| PUSH(I) | **Ce am făcut**: Am plecat de la arborele precedent, din rădăcină, și am comparat I-ul cu aceasta (la noi F). I fiidn mai mare, am mers pe dreapta. Dreapta e ocupată (cu M) și prin urmare acum comparăm I cu M. Fiind mai mic, mergem pe stânga. Stânga fiind liberă, punem acolo I-ul. |
| PUSH(C) |  |
| PUSH(A) |  |
| PUSH(N) |  |
| PUSH(O) |  |

Operația de ștergere a unui nod dintr-un arbore binar nu se mai notează cu PUSH, pentru că ștergerea, de data asta, este țintită - adică vrem să ștergem un anumit nod (știm exact pe care) din arbore.

Avem următoarele situații simple:

- dacă nodul de șters este frunză (orice nod fără fii), pur și simplu se șterge

- dacă nodul de șters are un singur fiu (chiar dacă sub fiul ăla e o întreagă structură) atunci pur și simplu se șterge nodul și se aduce în locul său fiul respectiv. Iată niște ilustrații:

|  |  |  |
| --- | --- | --- |
| Arborele inițial | Nodul pe care-l ștergem | Arborele final |
|  | O |  |
|  | R |  |

- dacă nodul de șters are doi fii, facem astfel:

• fie x = nodul de șters

• notăm cu st = subarborele stâng al său, cu dr = subarborele drept

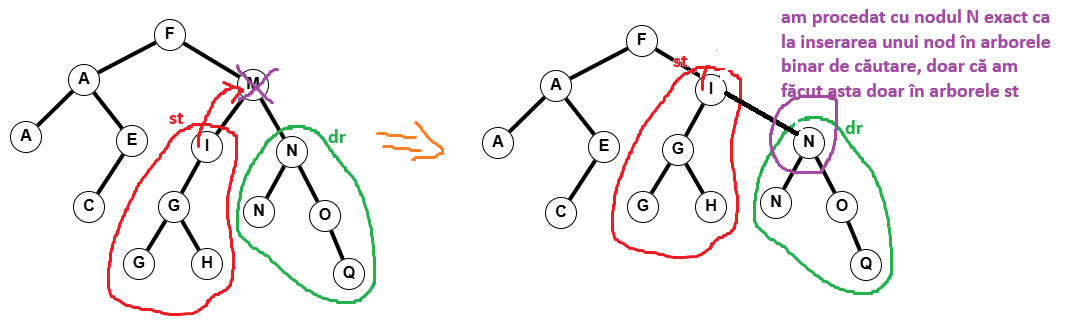
• se înlocuiește nodul x cu st

• în st se procedează exact ca la inserarea unui nod nou în arborele binar de căutare, luând ca referință valoarea din rădăcina lui dr, și nu atașăm DOAR rădăcina ci ÎNTREG dr.

Ex:

|  |  |  |
| --- | --- | --- |
| Arborele inițial | Nodul pe care-l ștergem | Arborele final |
|  | M |  |

Detalierea operației:



|  |  |  |
| --- | --- | --- |
| Arborele inițial | Nodul pe care-l ștergem | Arborele final |
|  | F |  |

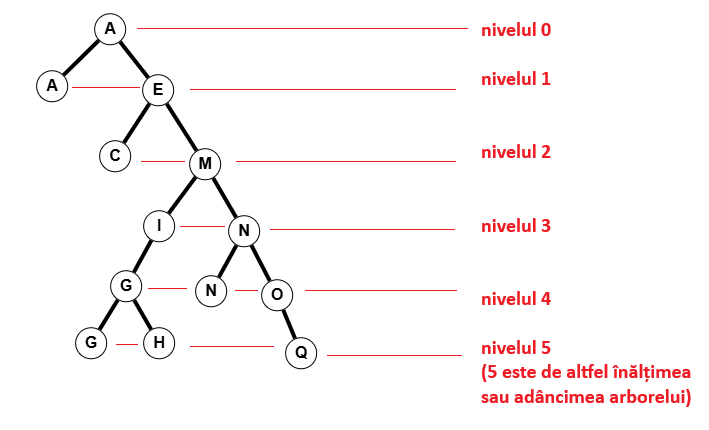
La exercițiile cu arbori din acestea vi se poate cere, în urma construirii, să răspundeți la diverse întrebări.

De exemplu, câte frunze are. Pe ultimul exemplu, în urma ștergerii lui F, arborele obținut are 6 frunze (A, C, G, H, N, Q).

O altă noțiune: înălțimea (sau adâncimea) arborelui: este dată de numărul de "crăci" (linii care unesc noduri) dintre rădăcină și cel mai de jos nod.

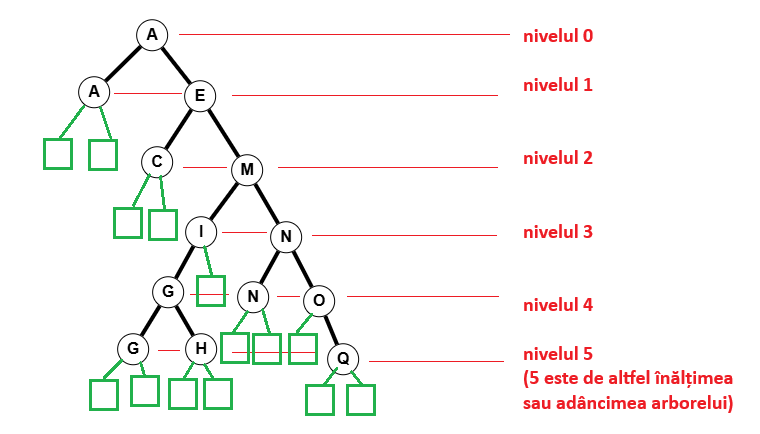
În cazul nostru, pe același arbore de mai sus din dreapta, înălțimea este 5.

De regulă, nivelurile grafului se numerotează de la 0 la numărul care reprezintă înălțimea:



Mai apare noțiunea de nod extern = un nod virtual, care nu există în arbore, și care reprezintă posibilele locuri în care mai putem băga fii la nodurile existente în arbore.

Le desenăm mai jos cu verde, și de formă pătrată

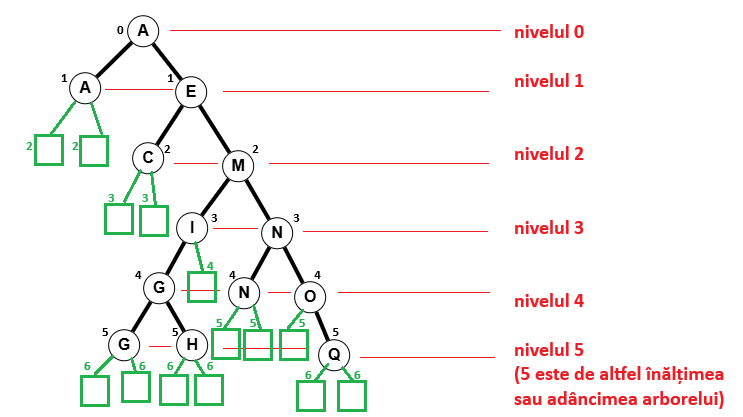


Pe baza lor, în arbore mai putem avea noțiunile de

Internal Length Path = suma adâncimilor (înălțimilor) tuturor nodurilor din arbore (ălea negre)

External Length Path = suma adâncimilor (înălțimilor) tuturor nodurilor externe din arbore (ălea verzi).

În desenul de mai jos am scris pe fiecare nod care este adâncimea (înălțimea sa) deci tre' făcute sumele:



Internal length path: 0+1+1+2+2+3+3+4+4+4+5+5+5 = 2+4+6+12+15 = 38

External length path: 2+2+3+3+4+5+5+5+6+6+6+6+6+6 = 4+6+4+15+36 = 65