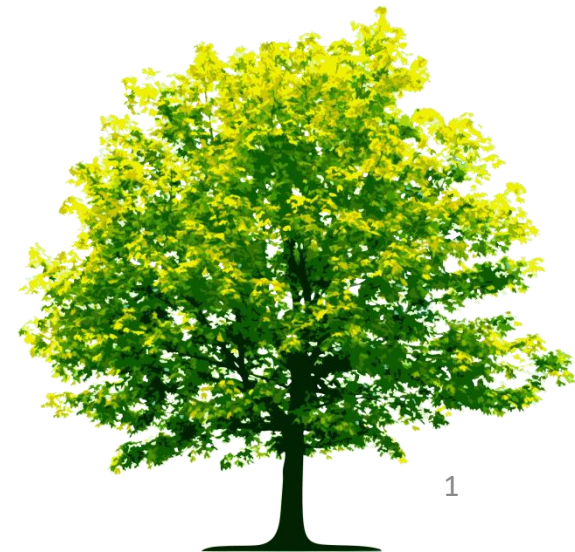


# Structuri de Date

Anul universitar 2020-2021

Prof. Adina Magda Florea



# Curs Nr. 5

---

## Arbori

- Arbori: definiții, terminologie
- Arbori binari
- Reprezentarea arborilor binari
- Adresări de baza în arbori
- Operații de baza cu arbori

# 1. Structuri ierarhice: arbori

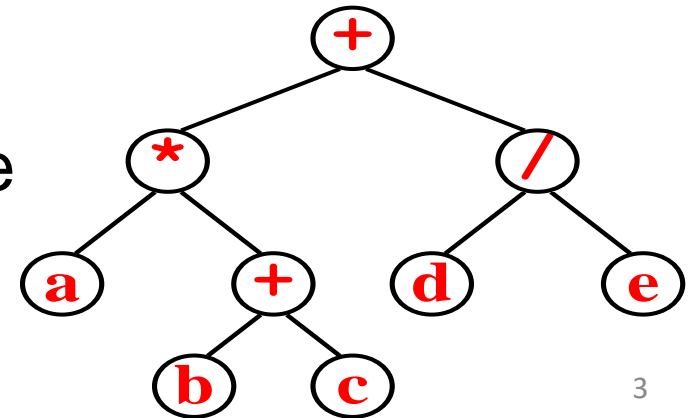
---

Liste, stive, cozi – structuri liniare

Arbori – structuri ierarhice

Multe exemple de date care pot fi structurate  
convenabil sub forma de arbore

- organizarea administrativă a societăților comerciale, ministerelor, universităților etc.
- programul meciurilor dintr-un turneu eliminatoriu
- directoare și subdirectoare
- arborele unei expresii aritmetice



# Arbori: definiții

---

Exista **mai multe tipuri de arbori**:

- Arbori binari
- Arbor multicăi sau n-ari
- Arbori sortați sau de cautare

Un **arbore** este o colecție de noduri și arce (legături între noduri) în care exista o singură cale între oricare două noduri

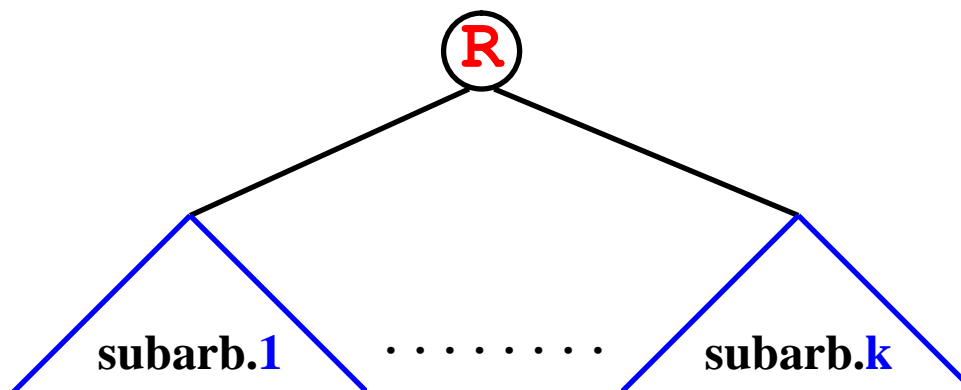
**Cale** = secvență de noduri și arce care conectează aceste noduri

**Rădăcină** – nodul pe care îl considerăm primul în arbore

# Arbori: definiții

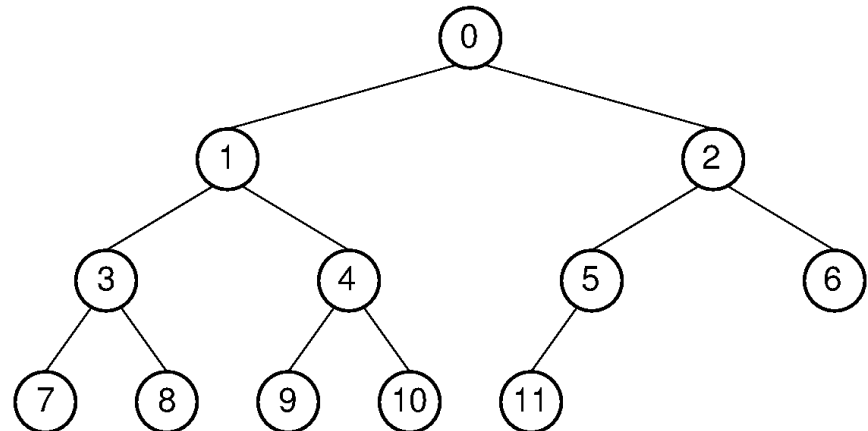
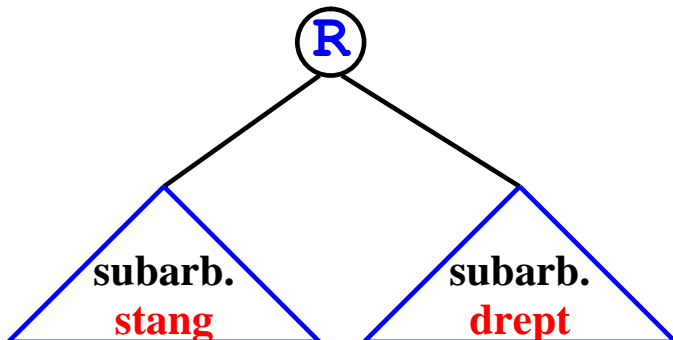
---

- Un **arbore** este format dintr-un nod **rădăcină**, căruia îi este atașat un **număr finit de arbori**.
- Pentru a evidenția relația ierarhică, aceștia sunt numiți **subarbori**.
- Orice nod dintr-un arbore poate fi privit ca rădăcină a unui (sub)arbore.



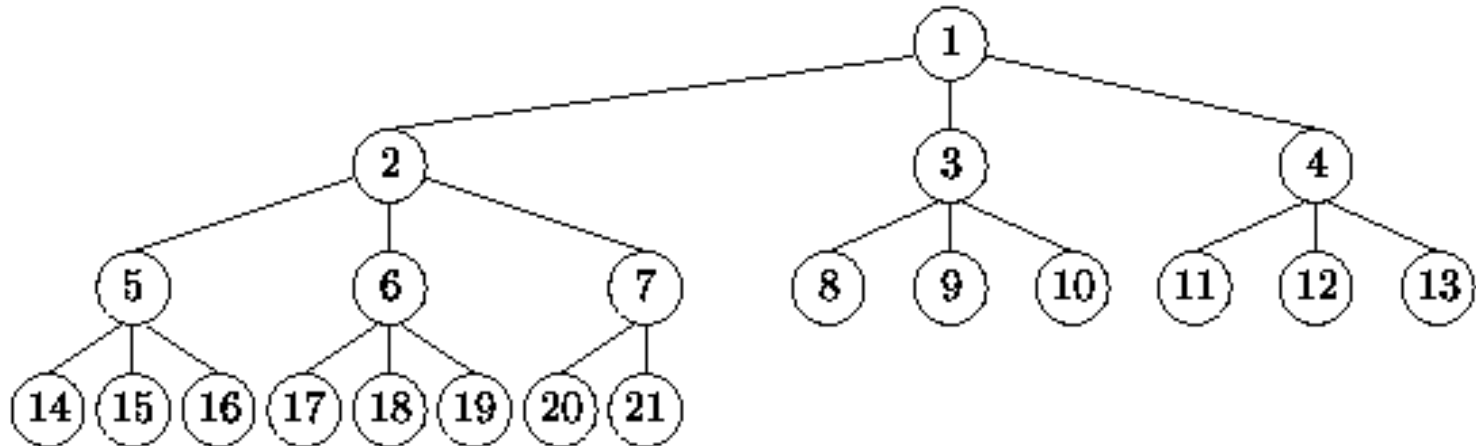
# Arbori binari

- **Arbore vid** - nu conține nici un nod
- **Arbore binar** - nodurile au **cel mult 2** subarbori
  - subarbore **stâng**
  - subarbore **drept**



# Arbori multicăi

- **Arbore multicăi** sau N-ar – arbore în care nodurile pot avea **mai mult de 2 subarbori**



# Arbori: terminologie

---

- **strămoș** al unui nod **x** – orice nod aflat pe calea (unică) de la rădăcină până la nodul **x** ; rădăcina este strămoșul tuturor celorlalte noduri
- **descendent** al unui nod **x** – orice nod aflat în arborele cu rădăcina **x**
- **tată / fiu (copil)** – noduri aflate la distanța 1 (numite și **strămoș / descendent direct**)
- **frați** (siblings) – fiii aceluiași nod



# Arbori: terminologie

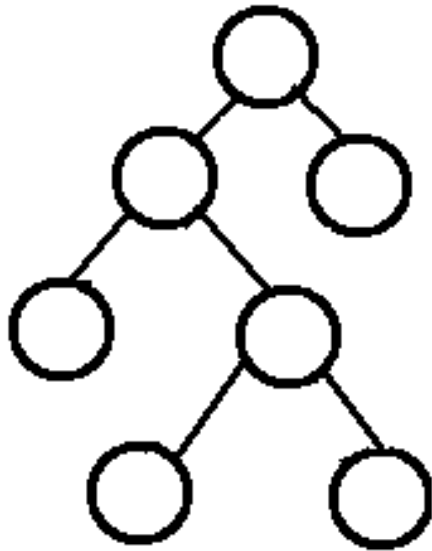
---

- **ordinul** (gradul) unui nod - **numărul de subarbori atașați** nodului
- **noduri interne** - nodurile care au subarbori
- **noduri externe** (noduri terminale sau **frunze**) – nodurile fără subarbori (cu ordin 0)
- **nivelul** sau **adâncimea** unui nod – numărul de arce de la rădăcină la nod (rădăcina are nivel 0)
- **înălțimea** unui arbore – **nivelul maxim din arbore** (adâncimea frunzei de nivel maxim)

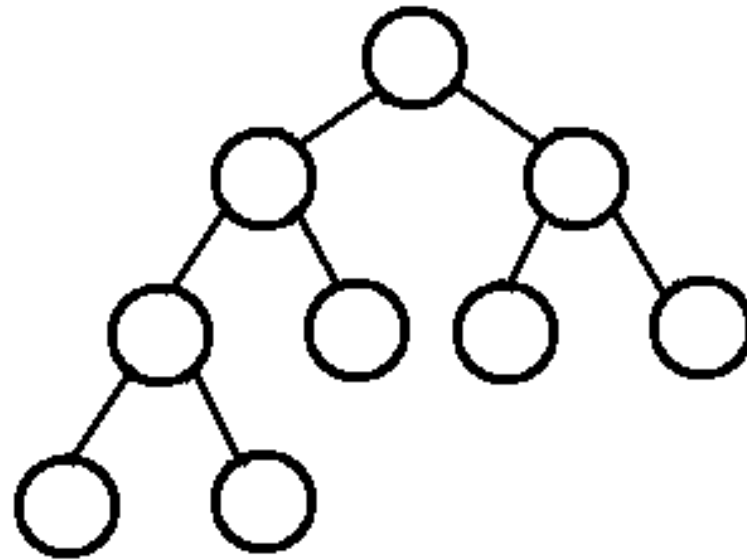
## 2. Arbori binari

---

- **Definiție recursivă:** Un **arbore binar** este (a) un nod extern sau (b) un nod intern conectat la o pereche de subarbori numiți **subarbore stâng** și **subarbore drept**
- **Arbore binar plin** (full) – fiecare nod are exact 0 sau 2 fii
- **Arbore binar complet** – un arbore binar care este complet umplut cu posibila excepție a ultimului nivel care este umplut de la stânga la dreapta



**Arbore binar plin**



**Arbore binar complet**

# Proprietăți ale arborilor binari

---

Un arbore binar plin cu **N noduri interne** are  
 **$N+1$  noduri externe**

# Proprietăți ale arborilor binari

---

Un arbore binar cu **N noduri interne** are **2N arce/legături**: N-1 legături între nodurile interne și N+1 legături la nodurile externe

# Proprietăți ale arborilor binari

---

**Inălțimea** unui arbore binar cu **N noduri interne** este **minim  $\lg N$**  și **maxim  $N-1$**

# Proprietăți ale arborilor binari

---

**Inălțimea unui arbore binar complet cu  $M$  noduri este cel mult  $O(\log M)$**

- Un arbore binar complet este un arbore special deoarece ofera raportul optim intre numarul de noduri si inaltimea arborelui.
- Inaltimea  $h$  a unui arbore binar complet cu  $M$  noduri este cel mult  $O(\log M)$

$$M = 1 + 2 + 4 + \dots + 2^{(h-1)} + 2^h = 2^{(h+1)} - 1$$

- Rezolvand pt  $h$  avem  $h = O(\log M)$

# 3. Reprezentarea arborilor binari

---

*Amintim reprezentarea **listei***

```
typedef int Item;  
typedef struct cel {  
    Item elem;  
    struct cel *next;} ListCel, *TList;
```

**sau**

```
typedef struct node *Link;  
typedef struct node {  
    Item elem;  
    Link next; } ListNode;
```

Un arbore este o generalizare a unei liste

Anumiti arbori (degenerati) devin chiar liste

*Obs: Reprezentarea arborilor multicaei – in alt curs*



# Reprezentare arbori binari (R1)

---

```
typedef int Item;  
typedef struct node {  
    Item elem;  
    struct node *lt, *rt; } TreeNode, *TTree;  
cu declaratia  
TTree root = NULL;  
si alocarea unui nod  
root = (TTree) malloc(sizeof (TreeNode));  
/* sau root = malloc(sizeof *root); */
```

# Reprezentare arbori binari (R2)

---

sau

*R1 si R2 sunt echivalente*

```
typedef struct node *TLink;
```

```
typedef struct node {  
    Item elem;  
    TLink lt, rt; } TreeNode;
```

cu declaratia

```
TLink root = NULL;
```

si alocarea unui nod

```
root = (TLink) malloc(sizeof (TreeNode));
```

```
/* sau root = malloc(sizeof *root); */
```

## 4. Adresări de baza în arbori

---

- test arborele vid: **(a == NULL)**
- Pentru arbore nevid (**a != NULL**)
  - valoarea elementului din nodul a: **a->elem**
  - pointer la subarborele stâng: **a->lt**
  - pointer la subarborele drept: **a->rt**
  - test dacă este frunză:  
**( a->lt == NULL && a->rt == NULL )**
  - test dacă este nod intern:  
**( a->lt != NULL || a->rt != NULL )**

# Adresări de baza în arbori

---

- **Pentru arbore nevid ( $a \neq \text{NULL}$ )**

- **Ordinul unui nod**

$$((a \rightarrow \text{lt} \neq \text{NULL}) + (a \rightarrow \text{rt} \neq \text{NULL}))$$

**frunză:  $0 + 0 \Rightarrow 0$**

**numai subarbore stâng:  $1 + 0 \Rightarrow 1$**

**numai subarbore drept:  $0 + 1 \Rightarrow 1$**

**doi subarbori:  $1 + 1 \Rightarrow 2$**

# 5. Operații de bază cu arbori

---

## ADT Arbore

- Inițializare arbore
- Test arbore vid
- Construcție arbore pe bază de cheie, subarbore stâng și subarbore drept
- Test nod intern
- Test nod extern / frunză
- Parcurgere arbore (cu diferite prelucrări, de exemplu afișare)

# Operații de bază cu arbori

---

- Inserare nod la stânga unui nod
- Inserare nod la dreapta unui nod

NB: Inserarea și eliminarea depind de proprietățile arborelui

- Determinarea înălțimii unui arbore
- Determinarea nivelului unui nod
- Determinarea numărului de noduri dintr-un arbore
- Distrugerea unui arbore

## 5.1 Inițializare arbore

---

```
typedef struct node *TLink;  
typedef struct node {  
    Item elem;  
    TLink lt, rt; } TreeNode;
```

```
TLink initNode()  
/* poate intoarce un arbore cu un nod*/  
TLink initTree()  
/* poate intoarce, in reprezentarea  
cu santinela, un nod de arbore in care lt si  
rt puncteaza la nodul insusi */
```

## 5.2 Testare

---

```
typedef struct node *TLink;  
typedef struct node {  
    Item elem;  
    TLink lt, rt; } TreeNode;
```

```
int emptyTree(TLink t)  
    /* arbore vid? */  
int isleaf(TLink t) /* este frunza? */  
int isinterior(TLink t)  
    /* este nod intern? */
```



## 5.3 Creare /distrugere

---

```
TLink  BuildNode(Item a, TLink ltree,
                                   Tlink rtree);

/* construiește un nod având cheia a și subarbore
   stâng, respectiv drept pe ltree și rtree */
{TLink t;
    t = (TLink) malloc (sizeof (TreeNode));
    t->lt = ltree; t->rt = rtree; t->elem = a;
    return t;
}

void KillTree(TLink t);
/* distruge arborele */
```

## 5.4 Parcurgerea arborilor

---

Parcurgerea unui arbore implică **vizitarea tuturor nodurilor dintr-un arbore**. Deoarece arborele este o structură neliniară, nu există o unică parcurgere

Două tipuri de parcurgere:

- **Parcurgere în adancime**
- **Parcurgere pe nivel**

### **Parcurgerea în adancime pentru arbori binari**

- Parcurgere în preordine
- Parcurgere în inordine
- Parcurgere în postordine

# Parcurgere în adancime

**Algoritm Parcurge\_in\_Adancime(arbore):**

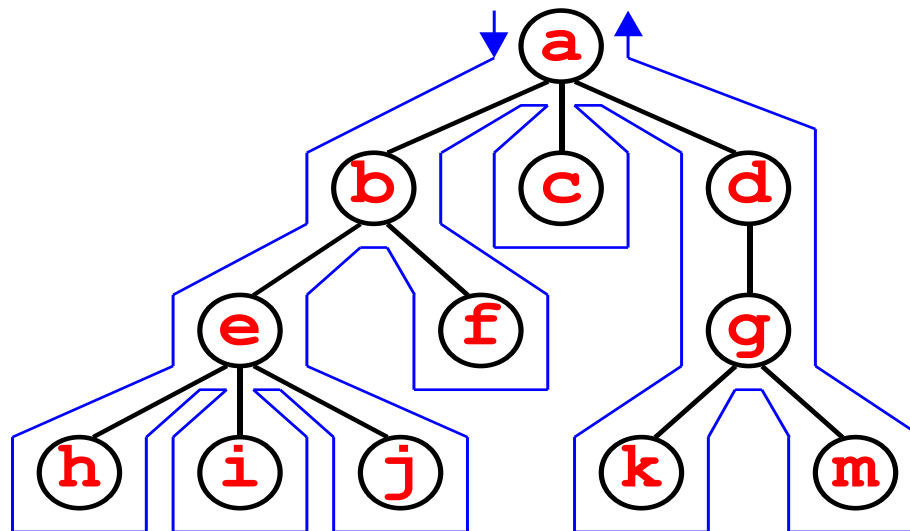
**daca** arbore vid **atunci** revenire;

prelucreaza informatia din radacina;

**pentru** fiecare subarbore **repeta**

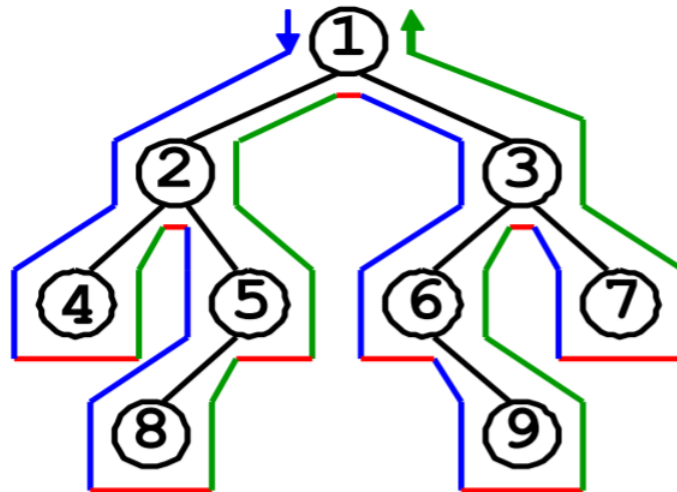
**Parcurge\_in\_Adancime**(subarbore);

**sfarsit**



# Parcurgerea în adancime arbori binari

- **Pre**ordine: Radacina, Stanga, Dreapta (RSD)
- **In**ordine: Stanga, Radacina, Dreapta (SRD)
- **Post**ordine: Stanga, Dreapta, Radacina (SDR)



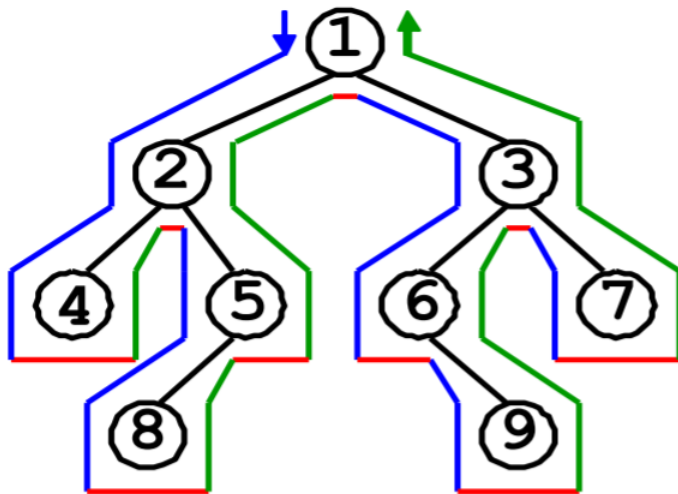
**Pre**ordine

**In**ordine

**Post**ordine

# Parcurgerea în adancime arbori binari

- **Pre**ordine (RSD)

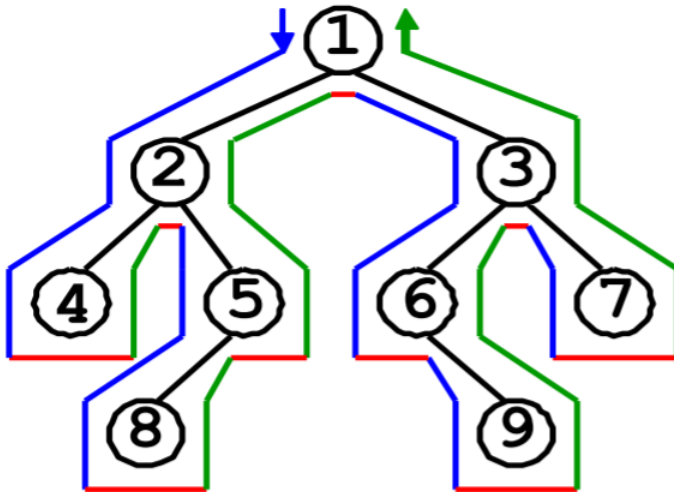


1 2 4 5 8 3 6 9 7

```
(1)
(2)
(4)
  (-) – fiu stg 4
  (-) – fiu dr 4
(5)
(8)
  (-)
  (-)
  (-)
(3)
(6)
  (-)
  (9)
    (-)
    (-)
(7)
  (-)
  (-)
```

# Parcurgerea în adancime arbori binari

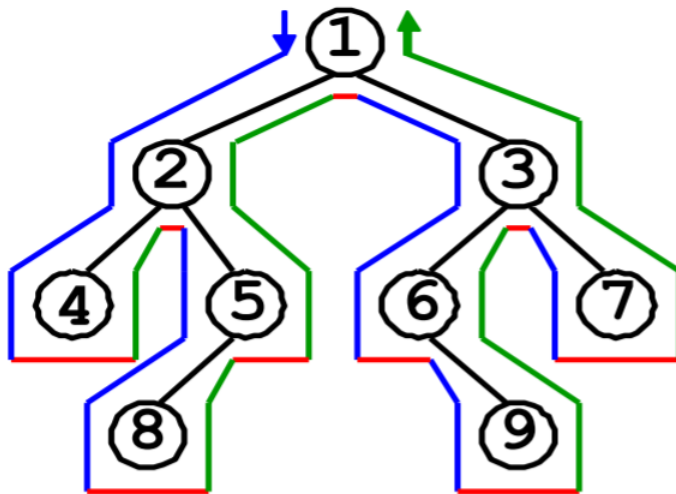
- Inordine (SRD)



4 2 8 5 1 6 9 3 7

# Parcurgerea în adancime arbori binari

- **Post**ordine (SDR)



4 8 5 2 9 6 7 3 1

# Parcurgerea în preordine (RSD)

---

**Algoritm Parcurgere\_Preordine(arbore)**

**daca** arborele este vid **atunci revenire**

prelucreaza informatia din radacina

Parcurgere\_Preordine(subarborele stang)

Parcurgere\_Preordine(subarborele drept)

**sfarsit**



# Parcurgerea în preordine (RSD)

---

```
void vizit(TLink t)
{ printf("%d \n", t->elem); }

void Preorder(TLink t) /* preordine, RSD */
{ if(t==NULL) return;
  vizit(t);
  Preorder(t->lt);
  Preorder(t->rt);
}
```

# Parcurgerea în preordine (RSD)

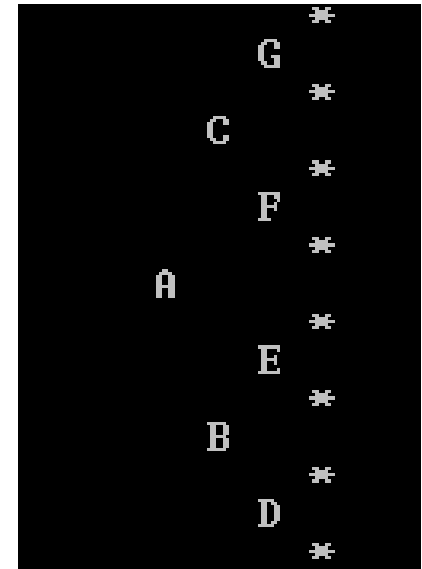
---

```
void vizit(TLink t)
{ printf("%d \n", t->elem); }
```

```
void traverse(TLink t, void (*vizit)(TLink))
{ if(t==NULL) return;
  (*vizit)(t);
  traverse(t->lt, vizit);
  traverse(t->rt, vizit);
}
```

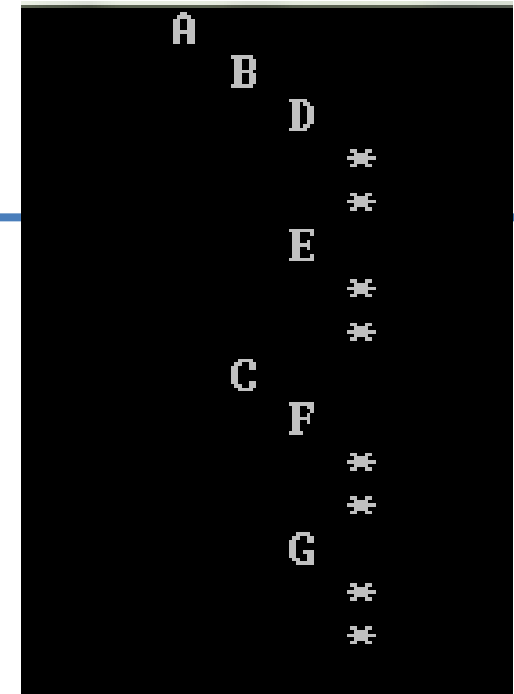
# Afişare arbore binar

```
typedef char Item;
void printnode(char a, int h)
{
    int i;
    for(i=0; i<h; i++) printf("  ");
    printf("%c\n", a);
}
void show(TLink t, int h)
{
    if(t==NULL) {printnode('*', h); return;}
    show(t->rt, h+1);
    printnode(t->elem, h);
    show(t->lt, h+1);
}
```



# Afişare arbore binar

```
typedef char Item;
void printnode(char a, int h)
{
    int i;
    for(i=0; i<h; i++) printf("  ");
    printf("%c\n", a);
}
void show(TLink t, int h)
{
    if(t==NULL) {printnode('*', h); return;}
    Ce se scrie aici pt efect din ecran ??
}
```



# Arbori binari expresie aritmetică

## Rădăcină Stânga Dreapta - în **preordine (RSD)**

generează **notatia prefixata** a unei expresii

## Stânga Rădăcină Dreapta- în **inordine (SRD)**

generează **notatia infixata** a unei expresii

## Stânga Dreapta Rădăcină - în **postordine (SDR)**

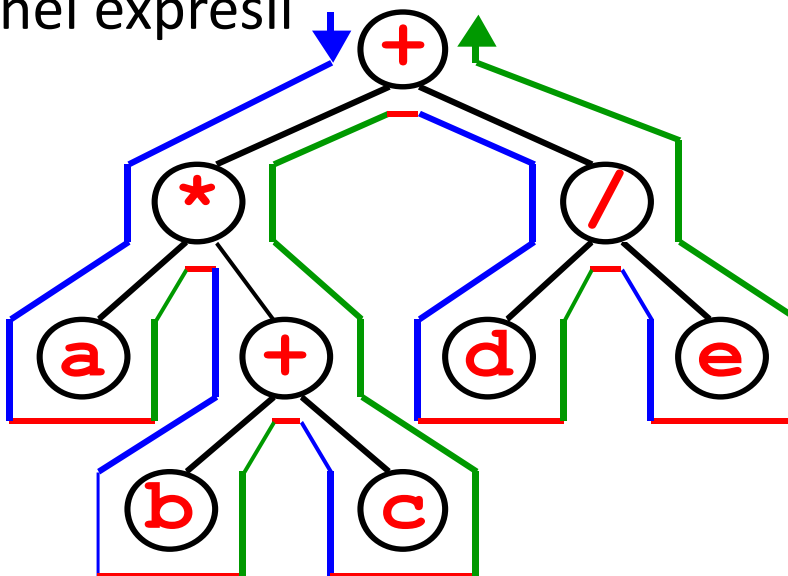
generează **notatia postfixata** a unei expresii

Arbore expresie

Noduri interne:

operatori binari

Frunze: operanzi



# Parcurgerea în adancime nerecursiv a.b.

---

**Algoritm Parcurge\_in\_Adancime(arbore):**

**daca** arbore vid **atunci** revenire;

initializeaza stiva de arbori (pointer la nod arbore)

Push(stiva, rad\_arbore)

**cat timp** stiva nu este vida **repeta**

{ Pop(stiva, arbore)

*prelucreaza informatia din radacina arbore*

**daca** subarbore drept nu este vid

**atunci** Push(stiva, rad\_sdr)

**daca** subarbore stang nu este vid

**atunci** Push(stiva, rad\_sstg)

}

**sfarsit**

# Parcurgerea în adancime nerecursiv

---

```
typedef char Item;  
typedef struct node *TLink;  
typedef struct node {  
    Item elem;  
    TLink lt, rt;} TreeNode;  
  
typedef struct cel {  
    TLink elem;  
    struct cel *next;} StackCel, *TStack;
```

# Parcurgere în adancime nerecursiv

---

```
Int EmptyStack(TStack s)
{ return s==NULL; }
```

Vezi cursul 4

```
TStack Push(TStack s, TLink a)
{ TStack t;
  t=(TStack) malloc(sizeof(StackCel));
  if(t==NULL)
  { printf("memorie insuficienta \n");
    return NULL; }
  t->elem = a; t->next = s;
  return t;
}
```



# Parcurgere în adancime nerecursiv

---

```
TStack Pop(TStack s, TLink *a)
{ TStack t;
  if (s == NULL)
      {printf("stiva vida \n"); return NULL;}
  *a = s->elem;
  t = s; s = s->next; free(t);
  return s;
}
```

Vezi cursul 4

# Parcurgere în adancime nerecursiv

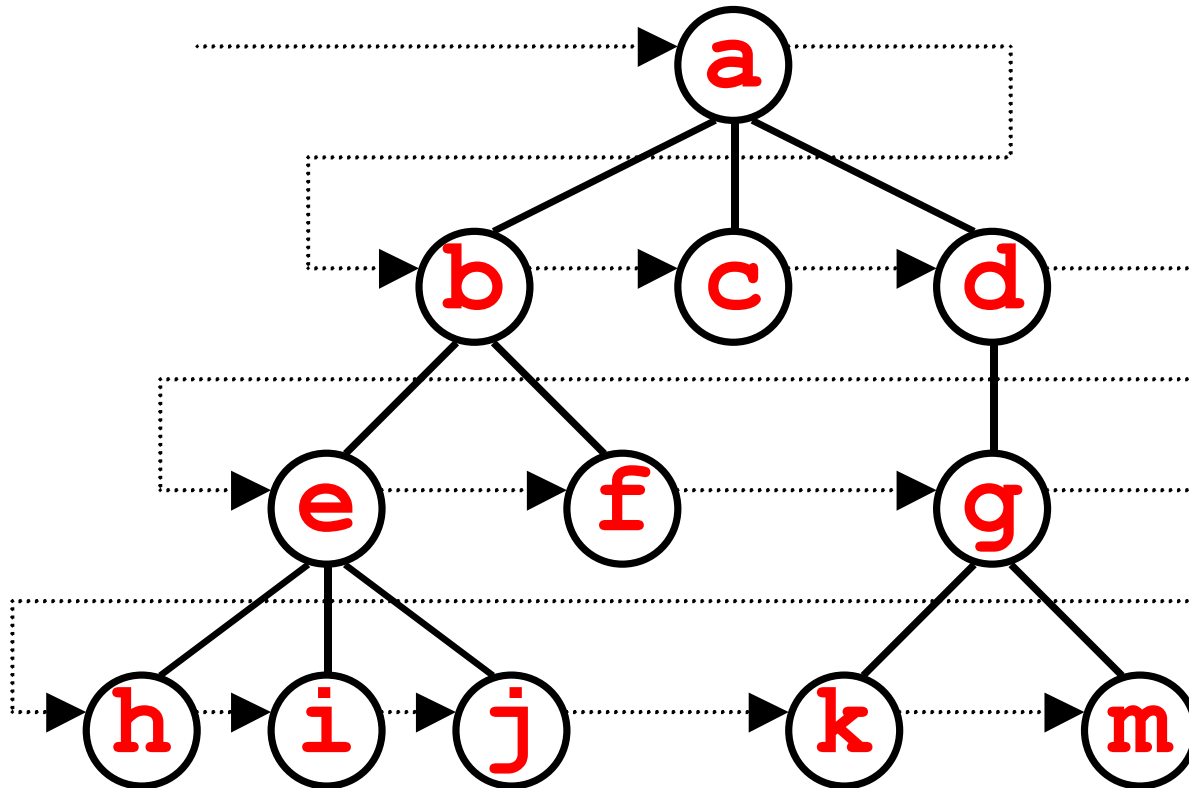
---

```
void traverse_A(TLink tree) /* adancime */
{
    TStack s; TLink t;
    if( tree == NULL) {printf("arbore vid \n");
                        return;}

    s = InitStack();
    s = Push(s,tree);
    while(!EmptyStack(s))
    {
        s = Pop(s,&t);
        printf("%c  ", t->elem);
        if(t->rt != NULL) s = Push(s,t->rt);
        if(t->lt != NULL) s = Push(s,t->lt);
    }
}
```

# Parcurgere în lățime / pe nivel

---



# Parcurgerea pe nivel a.b.

---

**Algoritm Parcurge\_pe\_Nivel(arbore):**

**daca** arbore vid **atunci** revenire;

initializeaza coada de arbori (pointer la nod arbore)

Enqueue(coada, rad\_arbore)

**cat timp** coada nu este vida **repeta**

{ Dequeue(coada, arbore)

prelucreaza informatia din radacina arbore

**daca** subarbore stang nu este vid

**atunci** Enqueue(coada, rad\_sstg)

**daca** subarbore drept nu este vid

**atunci** Enqueue(coada, rad\_sdr)

}

**sfarsit**

# Parcurgere pe nivel

---

```
typedef char Item;
typedef struct node *TLink;
typedef struct node {
    Item elem;
    TLink lt, rt;} TreeNode;

typedef struct cel {
    TLink elem;
    struct cel *next;} QueueCel, *TQueue;

TQueue front, rear;
```

# Parcurgere pe nivel

---

```
void InitQueue()
{ front = rear = NULL;}
Int EmptyQueue()
{ return front == NULL;}
void Enqueue(TLink a)
{TQueue q;
  q = (TQueue)malloc(sizeof(QueueCel));
  if(q==NULL){printf("not enough memory\n"); return;}
  q->elem = a; q->next = NULL;
  if(front == NULL && rear == NULL)
    { front = rear = q; return; }
  rear->next = q;
  rear = q;
}
```

Vezi si cursul 4

# Parcurgere pe nivel

---

TLink Dequeue()

```
{TQueue q = front; TLink t;
```

```
    if(front == NULL)
```

Vezi si cursul 4

```
        { printf("empty queue\n"); return NULL; }
```

```
    t = front->elem;
```

```
    if(front == rear) front = rear = NULL;
```

```
    else front = front->next;
```

```
    free(q) ;
```

```
    return t;
```

```
}
```

# Parcurgere pe nivel

---

```
void traverse_N(TLink tree) /* nivel*/
{
    TLink t;
    if (tree == NULL) {printf("arbore vid \n");
                        return;}

    InitQueue();
    Enqueue(tree);
    while (!EmptyQueue())
    {
        t=Dequeue();
        printf("%c  ", t->elem);
        if (t->lt != NULL) Enqueue(t->lt);
        if (t->rt != NULL) Enqueue(t->rt);
    }
}
```



# Parcurgerea pe nivel pt arbori N-ari

---

**Algoritm Parcurge\_pe\_Nivel(arbore):**

**daca** arbore vid **atunci** revenire;

initializeaza coada de arbori (pointer la nod arbore)

Enqueue(coada, rad\_arbore)

**cat timp** coada nu este vida **repeta**

{ Dequeue(coada, arbore)

prelucreaza informatia din radacina arbore

**pentru** fiecare subarbore **repeta**

**daca** subarbore nu este vid **atunci**

Enqueue(coada, rad\_subarbore)

}

**sfarsit**

# Parcurgerea în adâncime pt arbori N-ari

---

**Algoritm Parcurge\_in\_Adancime(arbore):**

**daca** arbore vid **atunci** revenire;

initializeaza stiva de arbori (pointer la nod arbore)

Push(stiva, rad\_arbore)

**cat timp** stiva nu este vida **repeta**

{ Pop(stiva, arbore)

prelucreaza informatia din radacina arbore

**pentru** fiecare subarbore **repeta**

**daca** subarbore nu este vid **atunci**

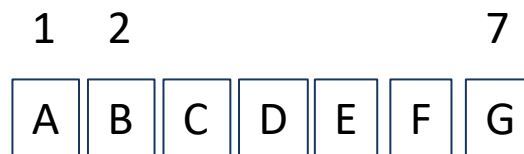
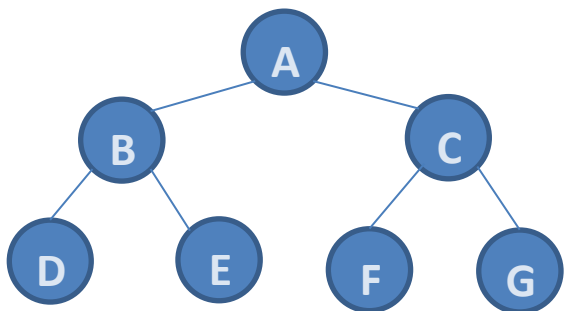
Push(stiva,rad\_subarbore)

}

**sfarsit**

## 6. Reprezentarea arborilor binari prin vectori

- Radacina – indice  $i = 1$
- Nodul cu indice  $i$  are subarborii la pozitiile  $2*i$  si  $2*i+1$
- Arbore de inaltimea  $h$  are  $2^{(h+1)} - 1$  celule
- Daca cunosc indexul unui nod  $i$ , obtin indexul tatalui  $i \div 2$



Reprezentare potrivita in cazul in care arborele este construit de la inceput echilibrat si nu se mai modifica