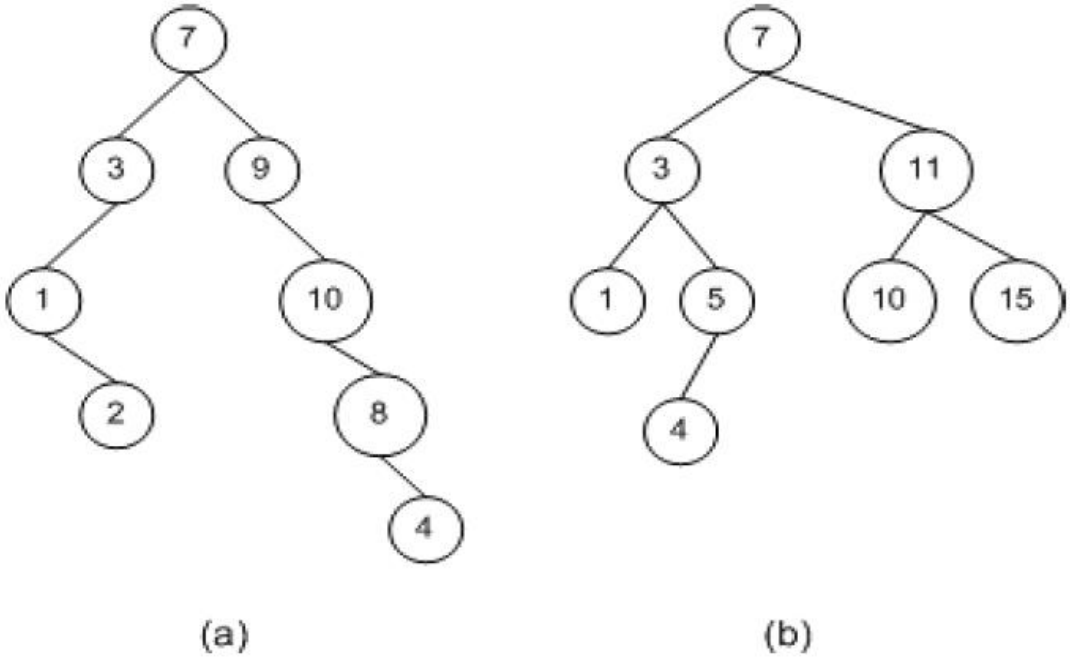
Arbori binari de căutare

Arbori binari de căutare

- sunt arbori binari în care nodurile sunt memorate ordonat în funcţie de o cheie.
- nodurile au în subarborele stâng noduri care au chei mai mici şi în subarborele drept chei mai mari.



Arbori binari de căutare

```
typedef struct tip_nod {
        tip cheie;
        informaţii utile;
  struct tip_nod *stg, *dr;
} TIP_NOD;
```

- Arborii binari de căutare sunt des folosiți pentru memorarea și regăsirea rapidă a unor informații, pe baza unei chei;
- Fiecare nod al arborelui trebuie să conțină o cheie distinctă;
- Structura arborelui de căutare depinde de ordinea de inserare a nodurilor;

Arbori binari de căutare

- Inserarea într-un arbore binar de căutare
  - recursiv, nerecursiv

- Căutarea unui nod având o cheie dată
- Traversarea unui arbore binar de căutare

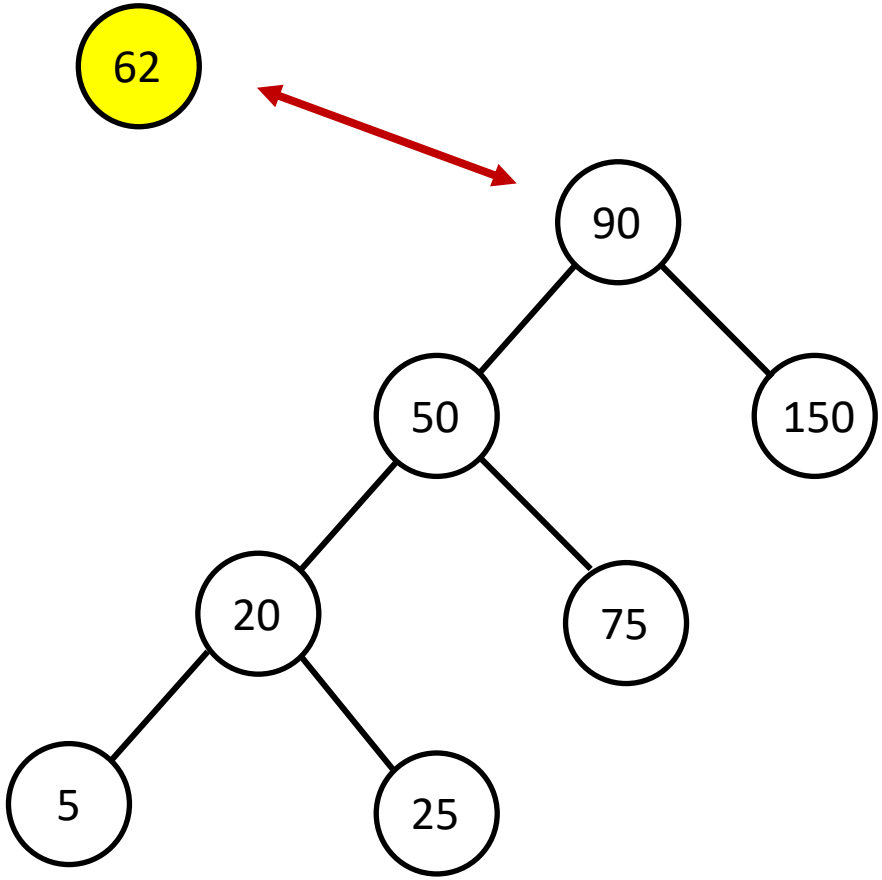
- Ștergerea unui nod de cheie dată
  - recursiv, nerecursiv
- Ștergerea unui arbore binar de căutare

Inserarea într-un arbore binar

Să considerăm cazul unui arbore binar de căutare în care dorim să inserăm o nouă cheie (nod nou) cu valoarea 62.

In final vom avea un nou nod frunză (fără copii/ descendenți) cu cheia 62.

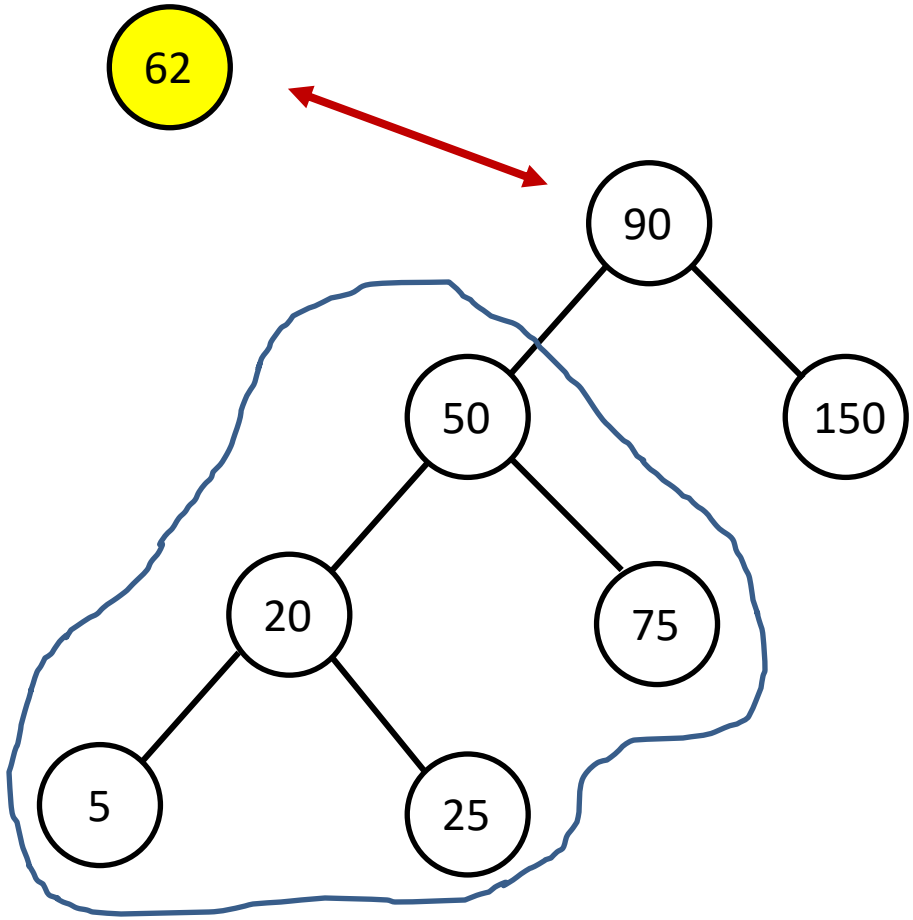
Va trebui să căutăm o poziție în arbore care respectă regula de integritate a arborilor binari de căutare.



Inserarea într-un arbore binar

Vom începe prin compararea cheii 62 (de inserat) cu valoarea cheii din rădăcina arborelui (90).

Pentru că 62 este mai mic decât 90, noul nod va fi situat în subarborele stâng al arborelui.



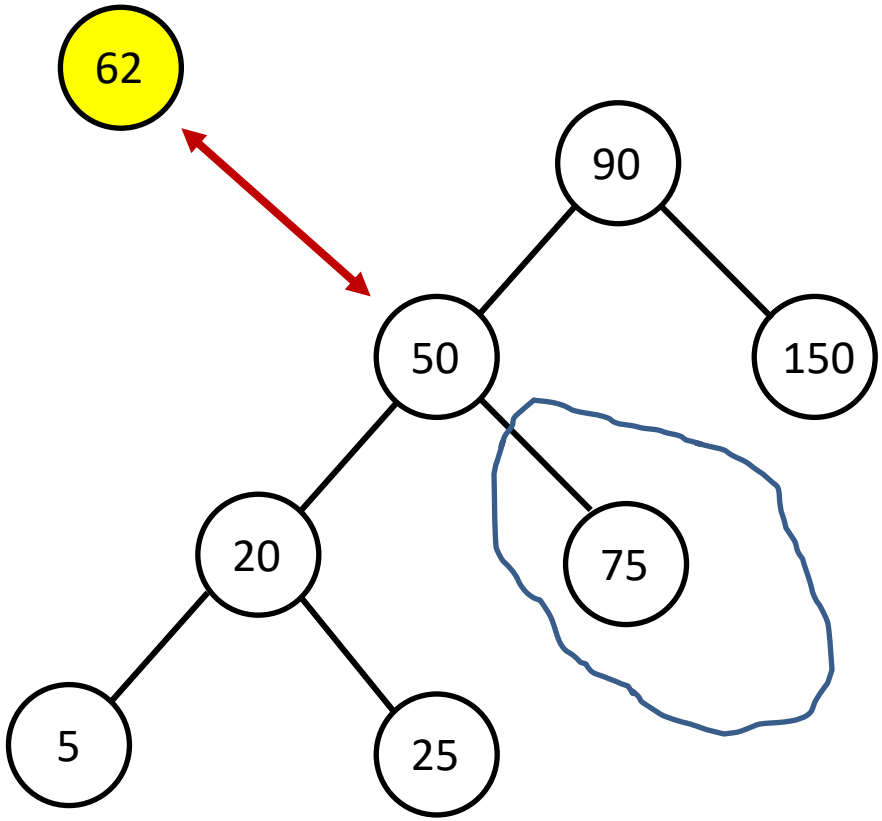
Inserarea într-un arbore binar

Vom începe prin compararea cheii 62 (de inserat) cu valoarea cheii din rădăcina arborelui (90).

Pentru că 62 este mai mic decât 90, noul nod va fi situat în subarborele stâng al arborelui.

Vom compara apoi 62 cu 50.

Din moment ce 62 este mai mare decat 50, nodul 62 va trebui plasatundeva în subarborele drept al lui 50.



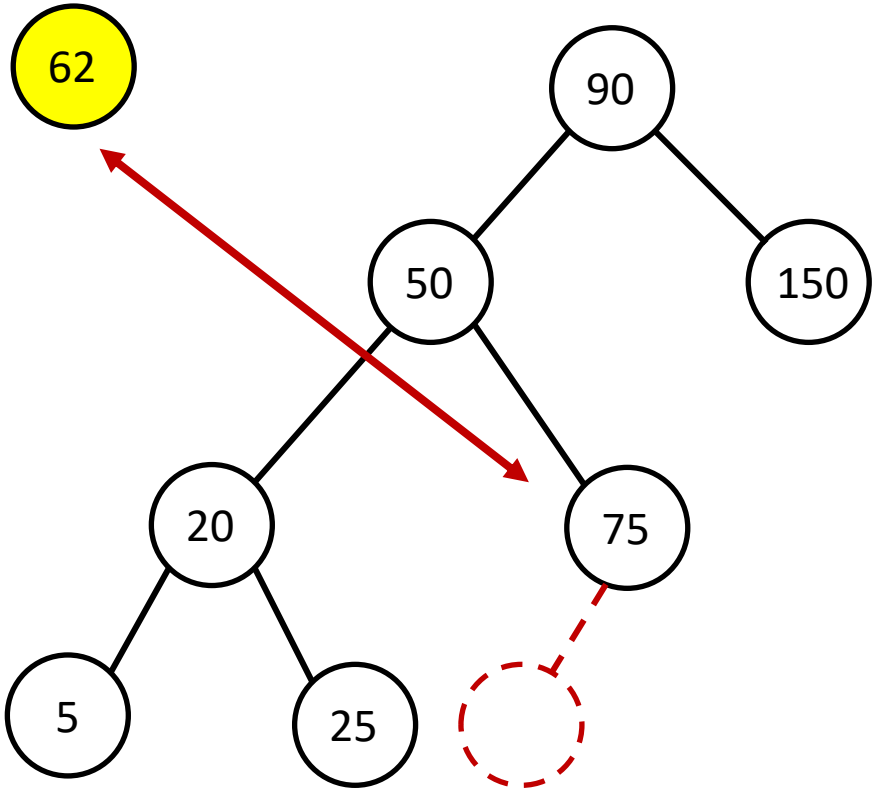
Inserarea într-un arbore binar

Se compară apoi 62 cu 75.

Deoarece 75 este mai mare decât 62, cheia 62 trebuie plasată în subarborele stâng al nodului 75.

Dar 75 nu are nici un copil (descendent) în partea stângă (subarborele stâng).

Asta înseamnă că am găsit locația pentru nodul 62: tot ceea ce mai trebuie făcut este să modificăm în nodul 75 adresa către copilul din stânga, încât să indice spre 62.



Inserare (recursivă) într-un arbore

```
SearchTree Insert( ElementType X, SearchTree T ){
      if( T == NULL ){
          /* Create and return a one-node tree */
          T = malloc( sizeof( struct TreeNode ) );
          if( T == NULL )
              FatalError( "Out of space!!!" );
          else
          {
              T->Element = X;
              T->Left = T->Right = NULL;
           }
       }
       else
       if( X < T->Element )
          T->Left = Insert( X, T->Left );
            else
             if( X > T->Element )
                 T->Right = Insert( X, T->Right );
             /* Else X is in the tree already; we'll do nothing */

      return T;  /* Do not forget this line!! */
}
```

Insert - varianta nerecursivă

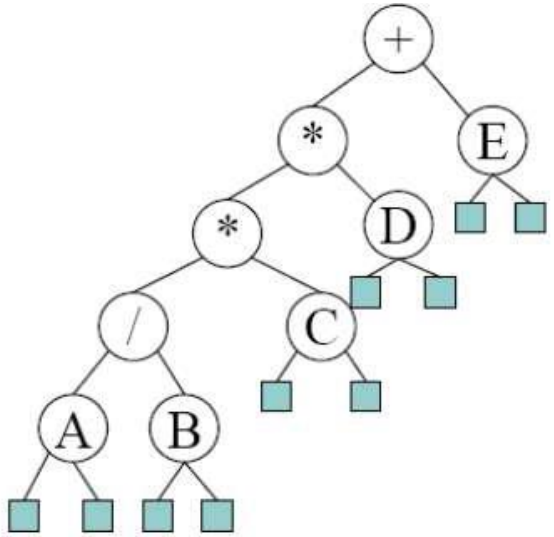
```
struct node *insertN(struct node *p,int val){
 struct node *temp1,*temp2;

 if(p == NULL){
        p = (struct node *) malloc(sizeof(struct node));
        if(p == NULL){
           printf("Cannot allocate memory\n");
           exit(0);
        }
   p->data = val;
   p->lchild=p->rchild=NULL;
 }
 else{
   temp1 = p;
   while(temp1 != NULL){
        temp2 = temp1;                    // memoreaza predecesorul (parintele)
        if( temp1 ->data > val)
              temp1 = temp1->lchild;    // deplasare in subarborele stang
        else
              temp1 = temp1->rchild;    // deplasare in subarborele drept
   }

   // adauga copil (temp2)
   // …
}

return(p);
}
```

```
struct node *insertN(struct node *p,int val){
 struct node *temp1,*temp2;
// ...
// adauga copil
 if( temp2->data > val){
   // adauga copil in stanga
   temp2->lchild = (struct node*)malloc(sizeof(struct node));
   temp2 = temp2->lchild;
   if(temp2 == NULL){
             printf("Cannot allocate memory\n");
             exit(0);
         }
   temp2->data = val;
   temp2->lchild=temp2->rchild = NULL;
 }
 else {
   // adauga copil in dreapta
   temp2->rchild = (struct node*)malloc(sizeof(struct node));
   temp2 = temp2->rchild;
   if(temp2 == NULL){
         printf("Cannot allocate memory\n");
         exit(0);
   }
         temp2->data = val;
         temp2->lchild=temp2->rchild = NULL;
  }
}
return(p);
}
```

Traversarea în inordine

- se traversează subarborele stâng
- se vizitează rădăcina
- se traversează subarborele drept

```
void inorder(tree_pointer ptr)
{
    if(ptr) {
        inorder(ptr->left_child);
        printf("%d",ptr->data);
        inorder(ptr->right_child);
    }
}
```

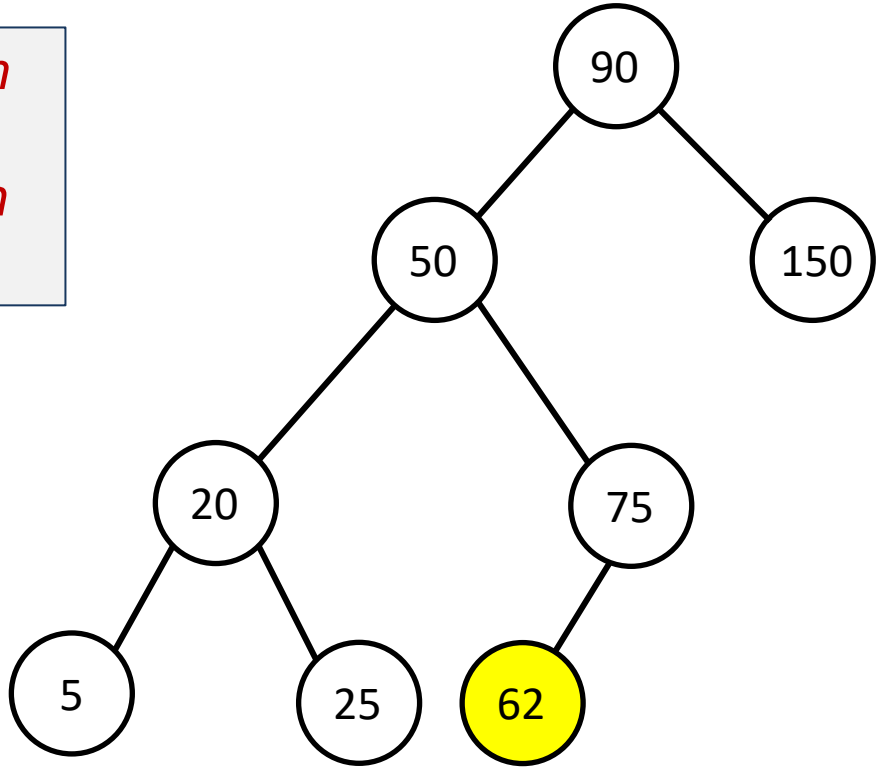


the infix form of the expression

A/B*C*D+E

Arbore binar de căutare

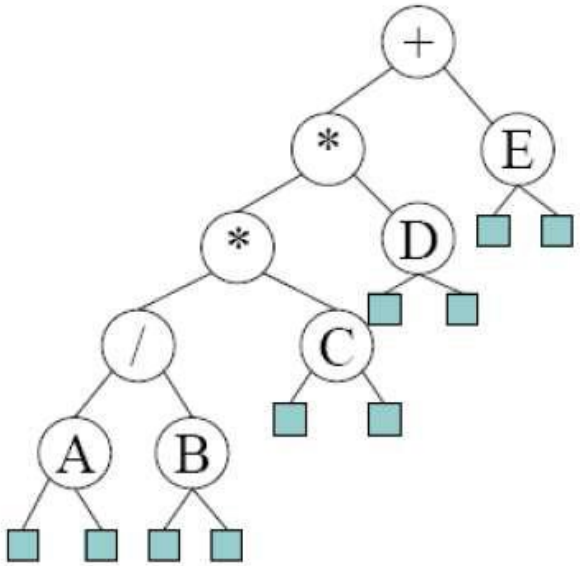
Parcurgerea unui arbore de căutare în ordinea stânga – rădăcină – dreapta conduce la obținerea listei nodurilor în ordinea crescătoare a cheilor.



Traversarea în preordine

- se vizitează rădăcina
- se traversează subarborele stâng
- se traversează subarborele drept

```
void preorder(tree_pointer ptr)
{
    if(ptr) {
        printf("%d",ptr->data);
        preorder(ptr->left_child);
        preorder(ptr->right_child);
    }
}
```



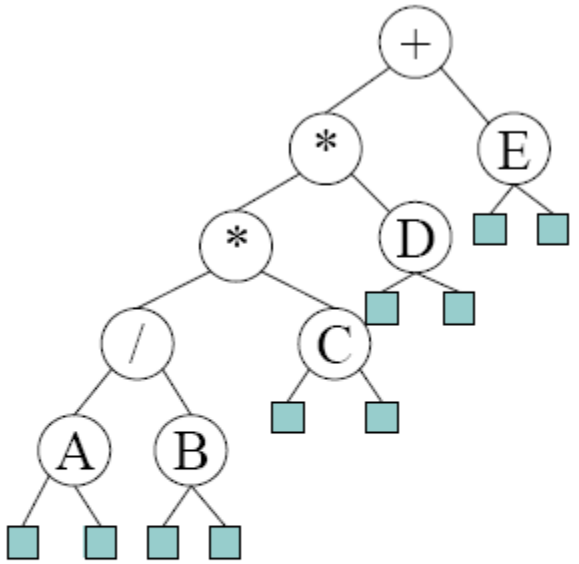
the prefix form of the expression

+**/ABCDE

Traversarea în postordine

- se traversează subarborele stâng
- se traversează subarborele drept
- se vizitează rădăcina

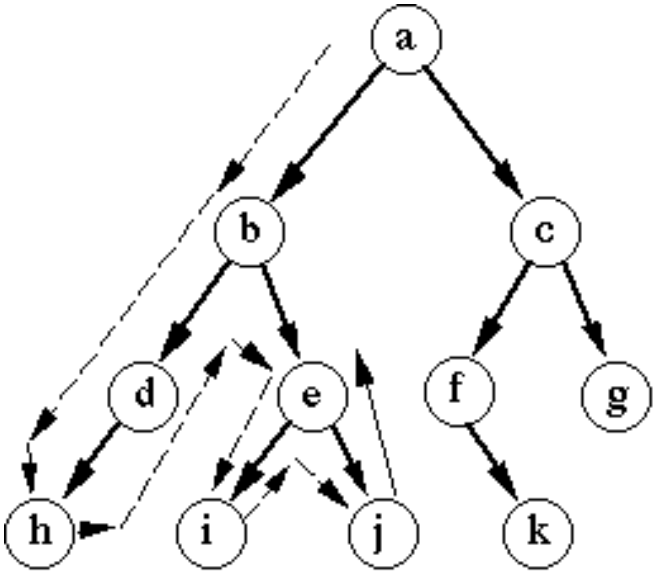
```
void postorder(tree_pointer ptr)
{
    if(ptr) {
        postorder(ptr->left_child);
        postorder(ptr->right_child);
        printf("%d",ptr->data);
    }
}
```



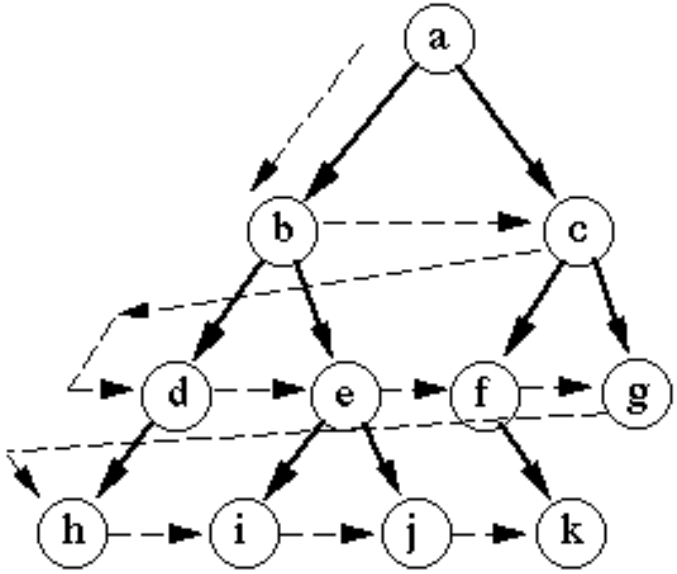
the postfix form of the expression

AB/C*D*E+

Traversarea în lățime vs. adâncime



Depth-first search



Breadth-first search

Căutarea unei chei

```
TIP_NOD *cautare (TIP_NOD *rad, int key)
{
     TIP_NOD *p;
     if (!rad) return NULL;                  /* arbore vid */

     p = rad;

     while (p != 0) {
        if (p->cheie == key) return p;            /* this is it */

        else

        if (key < p->cheie) p = p->stg;       /* căutare în subarborele stâng */
        else p = p->dr;                       /* căutare în subarborele drept */
     }

     return NULL;
}
```

Căutarea unei chei

```
Position FindTreeNode( ElementType X,   SearchTree T ){

            if( T == NULL )
               return NULL;

            if( X < T->Element )
               return Find( X, T->Left );
            else
               if( X > T->Element )
                        return Find( X, T->Right );
               else
                        return T;
}
```

Next: ABC(2): BFS,DFS, DELETE