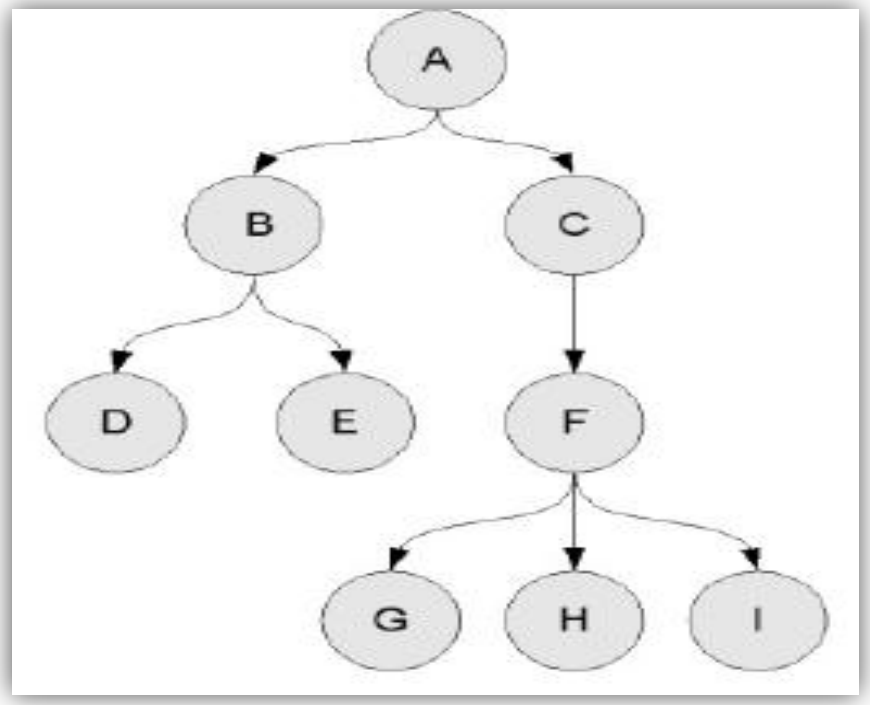
DEPTH FIRST TREE TRAVERSAL

Depth-first search (DFS) is an algorithm for traversing and searching tree or graph data structures. One starts at the root and explored as far as possible along each branch beginning from left to right before backtracking.

Example of a tree:



(+224) 666 49 47 87 / 620 25 95 13 lansanalsm@gmail.com

```
function DFS(Tree T ) : void
        while ( not isEmpty(T) then
                DFS(sonLeft(T));
                DFS(sonRight(T));
        Enf if
End function
```


The depth search of the previous tree gives: A-B-D-E-C-F-G-H-I

CHARACTERS IN STRINGS

version of N * N order	version of N order
Function find_chars (str1: String, str2 : String) : String **Begin** Variable : len1 : integer = 0 len 2 : integer = 0 result : String = null i ,J : integer = 0 *//the length of the two strings* len1 = length (str1)	Function find_chars (str1: String, str2 : String) : String **Begin** Variable : len1 : integer = 0 len 2 : integer = 0 result : String = null i ,J,k : integer = 0 already_added : boolean *//the length of the two strings*

```
        len2 = length (str2)

        //made the comparison character by character
        While (i < len1)  do
                For (j from 0 to  len2  j++) then
                        //if there is a match, it is added to result
                           If (str1[i]==str2[j]) then
                                   result .= str2[j]
                              end if
                end for
        end while

  //return the final  result
     return result ;

end function
```

```
        len1 = length (str1)
        len2 = length (str2)

        //made the comparison character by character
        While (i < len1)  do
            For (j from 0 to  len2  j++) then
                //if there is a match, it is added to result
                  If (str1[i]==str2[j]) then
                       already_added = false;

                       for (k from 0 TO lenght(result) ; k++) THEN

                             If (str2[j] == result[k] and j != i) then
                                  already_added=true;
                                    break;
                               end if

                       end for
                 //if the variable already_added is false add the caractere

                          if(already_added==false) then
                              result.= str2[j];
                            end if

                       end if
                    end if
                end for
        end while

//return the final  result
    return result ;
```


(+224) 666 49 47 87 / 620 25 95 13 lansanalsm@gmail.com

	end function

Test:

1- the version N*N order of ***find_chars*** (*"processing","processor") will return "*prroocessss*"*
2- the version N order of ***find_chars*** (*"processing","processor") will return "*process*"*

ARRAY COMPACTION

```
function  compact_array(tab1: array) : array
      begin

        Variable :
                  len1 : integer = 0
                  tab2 : Array = null   // the new array for result
                 i ,j       : integer = 0
                current : integer   //tamporally variable used to keep the current value on while……do boucle
                already_added : Boolean = false

            if(is_array(tab1)) then
```

```
                    //the length of the array
                    len1 = count(tab1);

                     while (I < len1) do
                                current = tab1[i];
                                already_added  =  false;

                                for(j from 0 to  count(tab2) j++) then
                                        if(tab2[j]==current) then
                                                already_added=true;
                                         Enf if
                                End for

                             if(already_added == false) then
                                    tab2[]= current;
                              End if

                    i++;
                     End while

                    return tab2;
          else
                     display ("you may give an array as parameter )";
          end if

end function
```

Test:

The compaction of the following array: [1, 3, 7, 7, 4, 8, 9, 9, 4, 5, 8, 9, 10]

Gives: [1, 3, 7, 4, 8, 9, 5, 10]

ARRAY ROTATING

```
function  rotate_array(tab, n)
  begin
      //  declaring the used variables
   Variable :
        Len : integer = 0
        I,j    : integer = 0
       new_tab : array

           len= length(tab);        //  the length of the array to rotate
                 if(n > len)  n = len;
                 new_tab=null;
                 //  rotating now
                 while(i < n) do
                 new_tab [] = tab[len - (i + 1)];
                          i++
                 end while
                 while (j < (len - n)) do
                          new_tab [] = tab [j];
                          j++
```

```
            end while
                        //  return the result
        return new_tab;
    end function
```


Test:

The rotation of the following array [1, 2, 3, 4, 5, 6, 7, 8 , 9,10] by N = 3

Gives: [10, 9, 8, 1, 2, 3, 4, 5, 6, 7]

LEAST COMMON MULTIPLE

```
function lcm(n1,n2)
    begin
     variable   //  the used variables
        result : integer
        I        : integer = 0
         Rest  : integer = 0
                //  calculating now
      result = n1 * n2;
      while(n1 > 1) do
```

```
            rest = n1 MOD  n2;

            if(rest == 0 ) then
                rest = result / n2;
                break;    //  get out when the result found
             end if

             n1 = n2;
            n2 = rest;
    End while

   return rest;    //  return the result
end function
```


Test:

The least common multiple of 12 AND 43 is: 516

The least common multiple of 20 AND 35 is: 140