Arbres planaires

Structure de données

```
type  arbre = ^noeud ;
              ptr = ^elt ;
              elt = record
                    | elem : element ;
                    | suiv : ptr ;
                    end ;
```

(A)

```
noeud = record
        | val : element ;
        | fils : ptr ;
        | suiv : arbre ;
        end ;
```

Chercher un noeud

```
function appartient (a : arbre ; nd : element) : boolean ;
  var tr : boolean ;
  Begin
      tr := false ;
      while (a <> nil) and (not (tr)) do
      if a^.val = nd then tr := true else
        a := a^.suiv ;

      appartient := tr ;

  end ;
```

Trouver le père d'un noeud

```
function pere (a: arbre ; nd: element) : arbre ;
var tr: boolean ;
Begin
    tr := false ;
    if (a^.val = nd) then pere := nil
    else Begin
        while (a <> nil) and (not (tr)) do
        if appartient (a^.fils , nd) then
            tr := true else a := a^.suiv ;
        pere := a ;
    end ;
end ;
```

(2)

hauteur d'un noeud

```
function hauteur_nd (a: arbre ; nd: elt) : integer ;
var c: integer ;
Begin
    c := 0 ;
    while (pere (a, nd) <> nil) do Begin
        c := c + 1 ;
        nd := pere (a, nd)^.val ;
    end ;
    hauteur_nd := c ;
end ;
```

Tester si un noeud est une feuille

```
function feuille (a: arbre) : boolean;
  Begin
    if (a = nil) then feuille := false
    else feuille := a^.fils = nil;
  end;
```

Calculer le nombre de feuilles

```
function nb_feuille (a: arbre) : integer;
  var  n: integer;
  Begin
    n := 0;
    while (a <> nil) do
        Begin
          if (a^.fils = nil) then n := n+1;
          a := a^.suiv;
        end;
    nb_feuille := n;
  end;
```

Descendants d'un noeud

```
function descendant (a: arbre ; nd: element): ptr ;
var tete : ptr ;
    procedure des (p: ptr) ;
      var q: ptr ;
      Begin
        while (p<>nil) do
          Begin
            new(q) ;
            q^.val := p^.val ;
            q^.suiv := tete ;
            tete := q ;
            des (recherche (a, p^.val)^.fils) ;
            p := p^.suiv ;
          end ;
      end ;
Begin
    tete := nil ;
    des ( recherche (a, x)^.fils) ;
    descendant := tete ;
end.
```

(5)

```
function recherche (a: arbre ; nd: elt): ptr ;
Begin
    while (a<>nil) and (a^.val <> nd) do
        a := a^.suiv ;
    recherche := a^.fils ;
end.
```

Descendant d'un nœud

```
function descendant (a: arbre ; nd: elt): ptr ;
    function des (a: arbre) : ptr ;
      var b: arbre ;
    Begin
      if a = nil then des := nil
      else if a^.val = nd then
            Begin
              b := a^.fils ;
              while (b <> nil) do
                    Begin
                      par (b^.val) ;
                      b := b^.suiv ;
                    end ;
              des := tete ;
            end ;
          else Begin
                 b := a^.fils ;
                 while (b <> nil) and (tete = nil) do
                   Begin
                     des := des (b^.val) ;
                     b := b^.suiv ;
                   end ;
               end ;
    end ;
```

(6)

```
Begin
    descendant := des (a);
end.

Procedure par (a: arbre);
 var  p: ptr;
 Begin
     if (a<> nil) then
          Begin
               new (p);
               p^.val := a^.val;
               p^.suiv := tete;
               tete := p;
               b := a^.fils;
               while (b <> nil) do
                    Begin
                         par (b^.val);
                         b := b^.suiv;
                    end;
          end.
 end.
```

(7)

hauteur d'un arbre planaire

```
function hauteur (a: arbre): integer;
 var c: integer;
 Begin
    c:=0;
    while (a<>nil) do Begin
                        if hauteur_nd (a, a^.val)>c
                        then c:= hauteur_nd (a, a^.val);
                        a:= a^.suiv;
                      end;
    hauteur:= c;
 end;
```

(4)

Ascendants d'un noeud

```
function ancetre (a: arbre ; nd: element) : ptr;
  var   p: ptr;   b: integer;   tete: ptr;
 Begin
    tete:= nil;
    while pere (a, nd)<>nil do
                        Begin
                          b:= pere (a, nd)^.val;
                          new (p);
                          p^.val:= b;
                          p^.suiv:= tete;
                          tete:= p;
                          nd:= b;
                        end;
    ancetre:= tete;
 end;
```