Corrigé feuille d'exercices d'informatique n°5

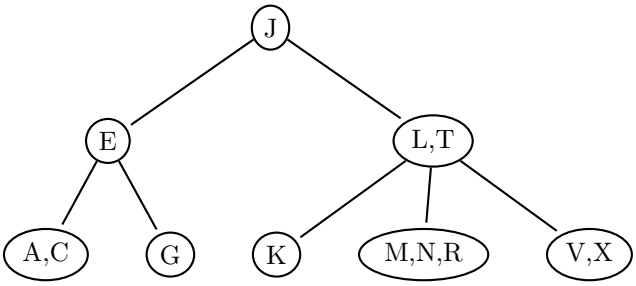
1. Déclaration du type arbre 3-2 avec des feuilles adaptées pour alléger l'écriture des arbres :

```
type 'a arbre_3_2 =
  noeudbinaire of 'a * 'a arbre_3_2 * 'a arbre_3_2
| noeudternaire of ('a * 'a) * 'a arbre_3_2 * 'a arbre_3_2 * 'a arbre_3_2
| feuille of 'a
| feuille_double of 'a*'a
| vide;;
```

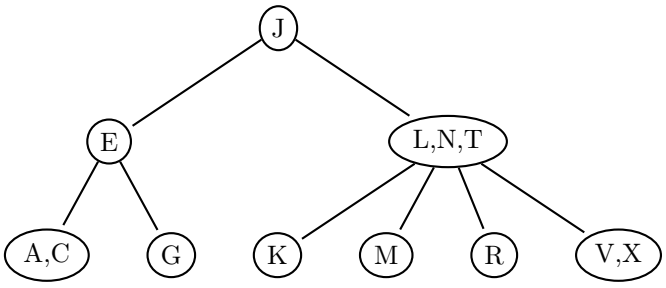
2. Fonction qui liste les clés d'un arbre 3-2 dans l'ordre :

```
(* liste : ('a -> unit) -> 'a arbre_3_2 -> unit = <fun> *)
let rec liste affiche = function
   vide -> ()
 | noeudbinaire(x,fg,fd) -> liste affiche fg; affiche x; liste affiche fd
 | noeudternaire((x,y),f1,f2,f3) ->
       liste affiche f1; affiche x; liste affiche f2;
       affiche y; liste affiche f3
 | feuille(x) -> affiche x
 | feuille_double(x,y) -> affiche x; affiche y;;
```

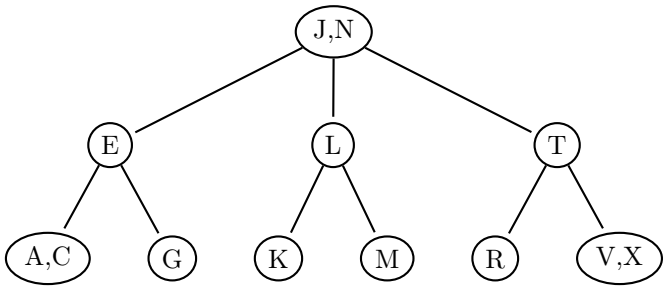
3. a) L'arbre initial, après insertion et avant transformation devient :



puis



et enfin,



b) Fonction d'insertion dans un arbre 3-2 :

```
(* ajoute : 'a -> 'a arbre_3_2 ref -> unit = <fun> *)
let ajoute x arbre =
(* le booléen indique s'il faut ou non éclater l'arbre on moment de l'insertion *)
let rec insert x = function
  vide -> (false,feuille(x))
| feuille(y) when x < y -> (false,feuille_double(x,y))
| feuille(y) when x > y -> (false,feuille_double(y,x))
| feuille(y) -> (false,feuille(y))   (* cas x=y *)
| feuille_double(y,z) when x<y -> (true,noeudbinaire(y,feuille(x),feuille(z)))
| (feuille_double(y,z) as a) when (x=y or x=z) -> (false,a)
| feuille_double(y,z) when x<z -> (true,noeudbinaire(x,feuille(y),feuille(z)))
| (feuille_double(y,z) as a) -> (true,noeudbinaire(z,feuille(y),feuille(x)))  (* cas x > z *)
| (noeudbinaire(y,fg,fd) as a)  when (x=y) -> (false,a)
| noeudbinaire(y,fg,fd) when x<y
        -> begin
      let (u,nfg) = insert x fg in
      if u then begin
        match nfg with
             noeudbinaire(z,fg',fd') -> (false,noeudternaire((z,y),fg',fd',fd))
       | _ -> failwith "cas impossible"
           end
        else (false,noeudbinaire(y,nfg,fd))
       end
| noeudbinaire(y,fg,fd)
         -> begin
       let (u,nfd) = insert x fd in
       if u then begin
         match nfd with
        noeudbinaire(z,fg',fd') -> (false,noeudternaire((y,z),fg,fg',fd'))
      | _ -> failwith "cas impossible"
         end
          else (false,noeudbinaire(y,fg,nfd))
        end
| (noeudternaire((y,z),f1,f2,f3) as a) when (x=y) or (x=z) -> (false,a)
| noeudternaire((y,z),f1,f2,f3) when x <y
      ->begin
         let (u,nf1) = insert x f1 in
         if u then begin
           match nf1 with
         noeudbinaire(t,fg',fd') -> (true ,noeudbinaire(y,nf1,noeudbinaire(z,f2,f3)))
```

```
        | _ -> failwith "cas impossible"
           end
           else (false,noeudternaire((y,z),nf1,f2,f3))
         end
| noeudternaire((y,z),f1,f2,f3) when x <z
      ->begin
         let (u,nf2) = insert x f2 in
          if u then begin
      match nf2 with
         noeudbinaire(t,fg',fd') -> (true ,noeudbinaire(t,noeudbinaire(y,f1,fg'),
                                                        noeudbinaire(z,fd',f3)))
       | _ -> failwith "cas impossible"
       end
      else (false,noeudternaire((y,z),f1,nf2,f3))
         end
| noeudternaire((y,z),f1,f2,f3)
      ->begin    (* cas x > z *)
         let (u,nf3) = insert x f3 in
         if u then begin
           match nf3 with
        noeudbinaire(t,fg',fd') -> (true ,noeudbinaire(z,noeudbinaire(y,f1,f2),nf3))
      | _ -> failwith "cas impossible"
      end
          else (false,noeudternaire((y,z),f1,f2,nf3))
         end in
arbre := snd(insert x (!arbre));;

(* exemple d'arbre 3-2 obtenu à partir des lettres de l'alphabet *)
let a = ref vide;;
for i = int_of_char 'a' to int_of_char 'z' do
    ajoute (char_of_int i) a done;;
!a;;
```

c) Voici l'arbre obtenu à partir des lettres de l'alphabet par le programme précédent :