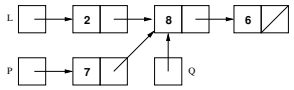
Structures de Données Listes, Piles, Files, Arbres

1. En vous aidant de la représentation graphique des listes, indiquez les éléments des trois listes P, Q, R après l'exécution de chacune des instructions suivantes :

P, Q, R : Liste ;
$P \leftarrow$ créerListe(1,créerListe(2,créerListe(3,NULL))) ;
$Q \leftarrow$ créerListe(4,P);
$(P \uparrow succ) \uparrow info \leftarrow 6$;
$R \leftarrow$ créerListe(5,$(P \uparrow succ) \uparrow succ$);
$(P \uparrow succ) \leftarrow R$;
$P \leftarrow R \uparrow succ$;
insérerFin(R,$Q \uparrow info$);
insérerAprès(Q,P,7);
supprimer($Q, Q \uparrow succ$);
$R \uparrow succ \leftarrow NULL$;
$R \leftarrow NULL$;

---

2. La figure ci-dessous donne les représentations graphiques de 3 listes simplement chaînées L, P, Q. Les séquences d'éléments de ces trois listes sont respectivement $(2, 8, 6), (7, 8, 6)$ et $(8, 6)$.



   (a) on exécute $(Q \uparrow succ) \uparrow info \leftarrow P \uparrow info$. Quelles sont les séquences d'éléments des listes L, P et Q ?

   (b) on revient à l'état décrit par la figure, puis exécute $(L \uparrow succ) \leftarrow Q \uparrow succ$. Valeurs de L, P, Q ?

   (c) on revient à l'état décrit par la figure, puis exécute $insérerAprès(P, P, 3)$. Valeurs de L, P, Q ?

3. Écrivez un algorithme vérifiant si une liste d'entiers simplement chaînée est triée.

4. Donnez un algorithme renvoyant l'adresse de la dernière cellule d'une liste simplement chaînée non vide.

5. On souhaite un algorithme qui étant donné une liste simplement chaînée `L` et un élément `e`, supprime de `L` tous les éléments de valeur `e`.
   — Donnez un algorithme utilisant l'opération (vue en cours) `supprimer(L,P)` supprimant de la liste `L` l'élément de place P (pointé par `P`). Quelle est sa complexité ?
   — Améliorez la complexité de l'algorithme précédent. Pour cela il ne faut pas utiliser l'opération de suppression. Pour simplifier, on supposera dans un premier temps que la valeur du 1er élément de la liste `L` est différente de `e`. Comment modifier l'algorithme précédent pour qu'il traite du cas où le 1er élément de `L` vaut `e` ?

6. Écrivez 2 algorithmes entrelaçant les éléments de 2 listes `L1` et `L2` de même longueur : le résultat est la liste dont le 1^{er} élément est le 1^{er} élément de `L1`, le $2^{ème}$ élément est le 1^{er} élément de `L2`, le $3^{ème}$ élément est le $2^{ème}$ élément de `L1`, le $4^{ème}$ élément est le $2^{ème}$ élément de `L2`, ... Le premier algorithme doit être récursif. Son résultat est une nouvelle liste obtenue en **dupliquant** les éléments de `L1` et ceux de `L2`. Le deuxième algorithme opère en modifiant les chaînages des listes. Aucun élément ne doit être dupliqué.
   Quelle est la complexité des 2 algorithmes ?

Algorithme 1 : entrelacer2(**dr** L1 : ListeSC, **d** L2 : ListeSC)

Données : `L1` et `L2` sont 2 listes simplement chaînées de même longueur.
Résultat : Modifie L1 de sorte qu'en fin d'algorithme L1 soit le résultat de l'entrelacement des listes initiales L1 et L2. La valeur finale de L2 est quelconque. Aucun élément n'est dupliqué. L'algorithme ne renvoie rien.

---

7. Donnez un algorithme pour la concaténation de 2 listes simplement chaînées, puis de 2 listes doublement chaînées (dans le cas des listes doublement chaînées les 2 paramètres sont des données/résultat).

Algorithme 2 : concaténation(**dr** L1 : Liste, **d** L2 : Liste)

Données : L1 et L2 2 listes
Résultat : modifie L1 de sorte que L1 soit la concaténation des 2 listes en entrée L1 et L2

---

8. Écrivez un algorithme qui étant donné une liste simplement chaînée L construit une nouvelle liste contenant les mêmes éléments que L, mais dans l'ordre inverse. Vous pouvez utiliser les algorithmes d'insertion `insérerDébut(L,x)` et `insérerFin(L,x)` vus en cours insérant dans la liste L une cellule contenant x avant ou après la cellule pointée par P.
   (*) Écrivez un deuxième algorithme qui, sans construire de nouvelle liste, inverse l'ordre des éléments d'une liste simplement chaînée.
   (**) Même question pour cette fois-ci une liste doublement chaînée.

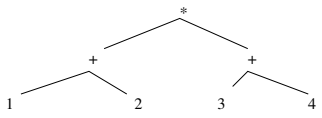
9. *Rappel* La *profondeur* de la racine d'un arbre est 0. La profondeur d'un autre noeud est celle de son père augmentée de 1. La hauteur d'un arbre non vide est la profondeur maximum de ses noeuds. Pour simplifier on supposera que la hauteur de l'arbre vide est -1.
   Donnez un algorithme qui calcule la hauteur d'un arbre binaire.
10. Écrivez un algorithme calculant le nombre de feuilles d'un arbre binaire.
11. Donnez un algorithme pour :

Algorithme 3 : nbNoeudsProf(**d** A : Arbre Binaire, **d** p : entier) : entier
Données : A un Arbre Binaire et un entier p **Résultat :** Renvoie le nombre de noeuds de A de profondeur p

12. Effeuiller un arbre binaire consiste à supprimer toutes ses feuilles. Donnez un algorithme d'effeuillage :

Algorithme 4 : effeuillage(**dr** A : Arbre Binaire)
Données : A un Arbre Binaire **Résultat :** Modifie A en l'effeuillant.

13. On considère des expressions arithmétiques simplifiées définies par : Une expression est :
    soit un nombre
    soit `( opérande1 opération opérande2)` où `opérande1` et `opérande2` sont des expressions et `opération` le nom d'une opération (+ ou *)
    On peut représenter de telles expressions par des arbres dont les étiquettes sont des nombres ou les noms + et * (voir cours). Par exemple l'expression $((1+2)*(3+4))$ est représentée par l'arbre de la figure.



    La notation postfixée d'une expression est la séquence obtenue récursivement en notant l'expression `( opérande1 opération opérande2)` par la séquence `opérande1 opérande2 opération`. Ainsi la notation postfixée de l'expression $((1+2)*(3+4))$ est la séquence 1 2 + 3 4 + *
    — Comment parcourir l'arbre d'une expression pour obtenir sa notation postfixée ? Donnez l'algorithme en codant la séquence correspondant à la notation postfixée par une liste chaînée.
    — En utilisant une Pile (dont les primitives sont `pileVide?(P), créerPile, sommetPile(P), empiler(P,e), dépiler(P)`), écrivez un algorithme calculant la valeur d'une expression représentée sous forme postfixée.
    — le parcours en largeur d'un arbre binaire est la séquence de ses étiquettes ordonnée par profondeur croissante et de ¡¡ gauche à droite ¿¿. Ainsi la séquence correspondant au parcours en largeur de l'arbre de la figure est * + + 1 2 3 4. En utilisant une File (dont les primitives sont `fileVide?(P), créerFile, têteFile(F), ajouterFile(F,e), retirerFile(F)`), écrivez un algorithme calculant la liste chaînée correspondant au parcours en largeur d'un arbre binaire.
14. On représente un dictionnaire de mots construits avec les lettres a et b par un arbre binaire (voir cours).

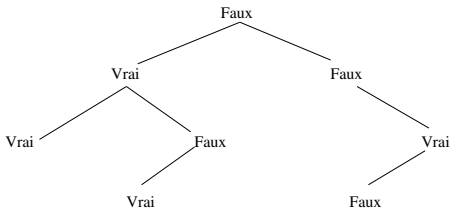


FIGURE 1 – Arbre du dictionnaire {a,aa,aba,bb}

    Donnez des algorithmes pour les opérations suivantes : Vérifier si mot appartient à un dictionnaire. Ajouter un mot à un dictionnaire. Supprimer un mot d'un dictionnaire. Un mot est représenté par la liste chaînée de ses lettres.