L3-INFO Algorithmique

Quick 2 – 19 novembre 2021 – durée 1h 15min

Les documents et les téléphones (incluant smartphone, tablettes,... tout ce qui contient une interface réseau) sont interdits. Les calculettes sont autorisées.
Seuls les dictionnaires pour les personnes de langue étrangère sont autorisés.
Toutes les réponses doivent être justifiées. Le barème est indicatif.

Dans tout ce devoir, on manipule les arbres binaires avec les fonctions usuelles :
$ArbreVide()$, $Noeud(g, x, d)$, $EstVide(a)$, $FilsGauche(a)$, $FilsDroit(a)$, $Cle(a)$.

---

1 Algorithme de Huffman (*11 points*)

Question 1 : *(3 points)*

À l'aide de l'algorithme de Huffman, calculer un arbre de codage optimal pour la phrase à 35 symboles :

```
SIRE RAMONS UNIS SANS NOUS CENSURER
```

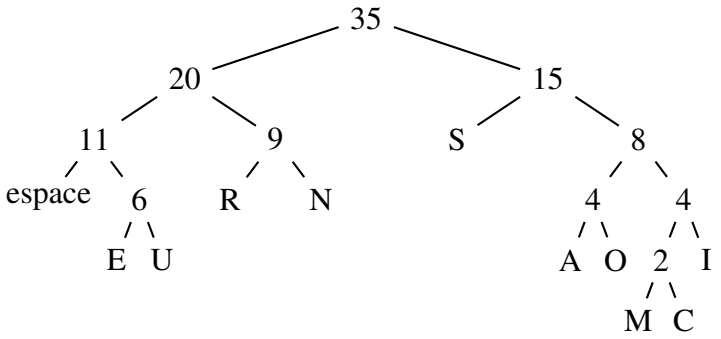
Attention à ne pas oublier le caractère espace.

Réponse 1 :

On compte les occurrences de chaque caractère :

S	I	R	E	A	M	O	N	U	C	espace
7	2	4	3	2	1	2	5	3	1	5

Plusieurs arbres de codage distincts peuvent être produits (par exemple en échangeant plusieurs nœuds de poids 4), en voici un. Les nœuds sont étiquetés avec leurs poids, sauf les feuilles qui sont étiquetées par les symboles :



En étiquetant les branches gauches par un 0 et les branches droites par un 1, on trouve la table des codes :

S	I	R	E	A	M	O	N	U	C	espace
10	1111	010	0010	1100	11100	1101	011	0011	11101	000

Remarque : quel que soit le codage produit, la longueur des codes de chaque lettre reste ici la même (5 bits pour M et C, 2 bits pour S, etc.).

Question 2 : *(1 point)*

Coder les 10 premiers caractères de cette phrase selon le code obtenu.

Réponse 2 :

Il suffit de concaténer les codes des 10 premières lettres :
101111010001000001011001110011010 11

Question 3 : *(2 points)*

Décoder le début de mot binaire suivant à l'aide de votre code :

$$01110110101110001111011101 1000\ldots$$

(Le texte obtenu n'aura pas forcément de sens, puisqu'il dépend de votre arbre de codage.)

Réponse 3 :

On suit les chemins dans l'arbre de codage et on trouve :
NSONSUOOS... (les deux derniers bits ne sont pas décodés, ils forment le début d'un code)

Question 4 : *(5 points)*

Écrire un algorithme qui prend pour arguments un arbre de codage et un mot binaire, et détermine si oui ou non ce mot binaire est le code d'un symbole de l'alphabet.

Réponse 4 :

On s'inspire de l'algorithme de décodage, mais on s'arrête dès qu'on arrive à une feuille. Il y a donc deux cas d'erreur à prendre en compte :

— si le mot se termine avant d'avoir trouvé une feuille,
— ou si on arrive à une feuille avant d'avoir lu le mot entier (on ne repart pas de la racine)

On peut simplifier cet algorithme :

— les deux premiers cas de base s'écrivent sans conditionnelle, simplement avec **renvoyer** (EstVide(FilsGauche(a)) et EstVide(FilsDroit(a)))
— dans un arbre de codage de Huffman un nœud a toujours exactement 0 ou 2 fils vides, on peut donc se contenter de tester EstVide(FilsGauche(a))

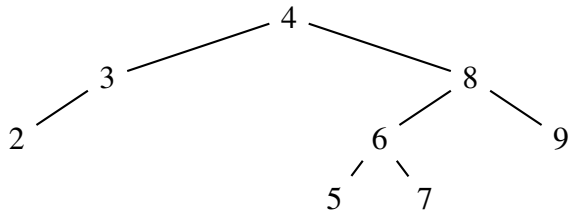
```
VERIF_CODE(a, m, i)
  Données : Un arbre de codage préfixe a, un mot binaire m[0..n−1], un indice i initialisé
            à 0 dans le premier appel de la fonction
  Résultat : Un booléen qui indique si m code un symbole de a
  si i == n alors
      si EstVide(FilsGauche(a)) et EstVide(FilsDroit(a)) alors
          renvoyer Vrai
      sinon
          renvoyer Faux
  sinon
      si EstVide(FilsGauche(a)) et EstVide(FilsDroit(a)) alors
          renvoyer Faux
      sinon
          si m[i] == 0 alors
              renvoyer VERIF_CODE(FilsGauche(a), m, i + 1)
          sinon
              renvoyer VERIF_CODE(FilsDroit(a), m, i + 1)
```

2 Partition d'arbre binaire de recherche (*9 points*)

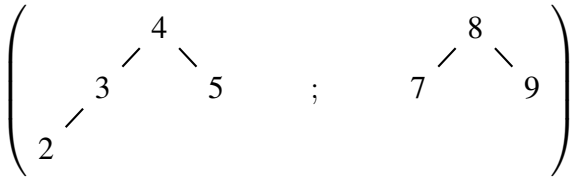
On considère un arbre binaire de recherche (ABR) dont les nœuds contiennent des éléments *tous différents*.
Le but de cet exercice est le suivant : étant donné un ABR A et un élément e **présent dans l'arbre**, créer deux ABR B et C tel que :

— B contient tous les éléments de A inférieurs à e ;
— C contient tous les éléments de A supérieurs à e.

Par exemple, si on partitionne l'arbre suivant par rapport à l'élément 6



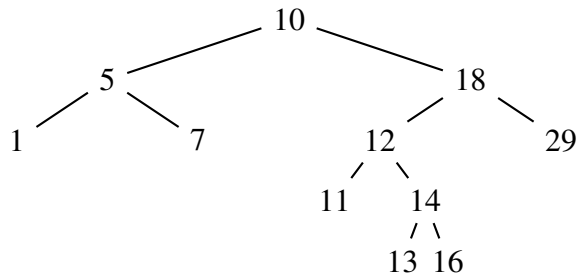
Le résultat pourrait être le couple d'arbres :



La complexité au pire de votre algorithme devra être $\mathcal{O}(h)$, la hauteur de l'arbre. Il est demandé de ne pas utiliser d'autre structure de données ni les fonctions d'insertion ou de suppression dans un ABR.

Question 5 : *(2 points)*

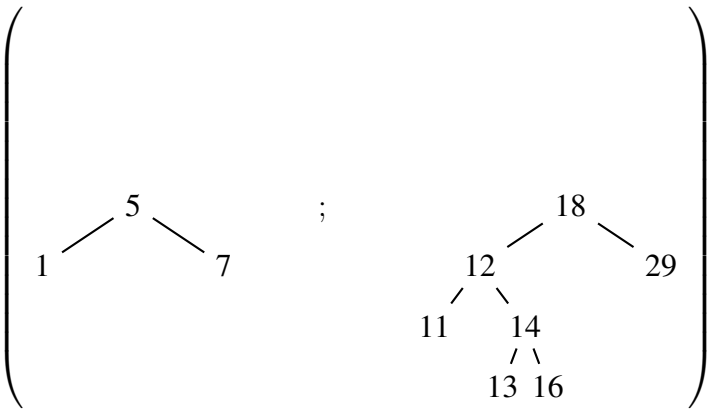
Illustrer la construction de ce couple d'arbres sur l'arbre suivant :



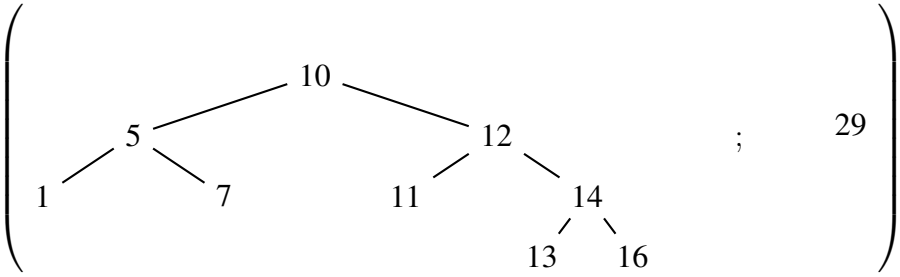
— lorsqu'on partionne par rapport à 10 ;
— lorsqu'on partionne par rapport à 18 ;
— et enfin lorsqu'on partionne par rapport à 14.

Réponse 5 :

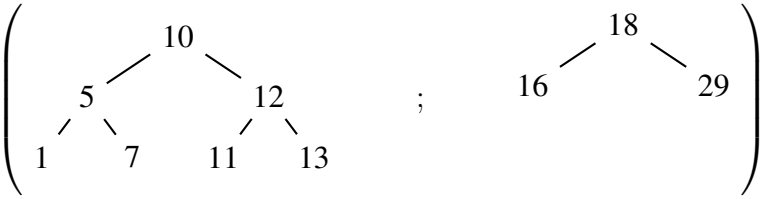
— Lorsqu'on partionne par rapport à 10 : c'est immédiat, ce sera le cas de base de notre algorithme.



— Lorsqu'on partionne par rapport à 18 : il faut « raccrocher » le sous-arbre de racine 12, il trouve sa place comme fils droit de 10.



— Lorsqu'on partionne par rapport à 14 : on remonte du nœud 14 vers la racine. On commence par raccrocher 13 comme fils droit de 12, alors que 16 reste seul nœud supérieur à 14. Ensuite, on peut raccrocher 16 comme fils gauche de 18, et garder l'arbre 16-18-29 comme arbre des éléments supérieurs à 14. Enfin il faut raccrocher 12 comme fils droit de 10.



Question 6 : *(7 points)*

Écrire un algorithme *récursif* qui prend pour argument un arbre binaire de recherche A et un élément e présent dans A, et qui renvoie un *couple* d'ABR contenant les éléments respectivement inférieurs et supérieurs à e.

Il est recommandé d'illustrer le fonctionnement de votre algorithme à l'aide de schémas.

Réponse 6 :

Le cas où a est vide n'a pas besoin d'être pris en compte, puisqu'on a supposé e présent dans a ; on prend soin que tous les appels récursifs s'arrêtent sur le cas de base $cle(a) == e$.

```
PARTITIONNER(a, e)
    Données : Un arbre binaire de recherche a, un élément e présent dans a
    Résultat : Un couple d'ABR contenant les éléments de a respectivement inférieurs et
               supérieurs à e

    si cle(a) == e alors
        renvoyer (FilsGauche(a) ; FilsDroit(a))
    si cle(a) > e alors
        (GG, GD) = PARTITIONNER(FilsGauche(a), e)
        renvoyer (GG ; Noeud(GD, clé(a), FilsDroit(a)))
    si cle(a) < e alors
        (DG, DD) = PARTITIONNER(FilsDroit(a), e)
        renvoyer (Noeud(FilsGauche(a), clé(a), DG) ; DD)
```