[CYBER1][2023-2024] Rattrapage (2h00)

Algorithmique 2

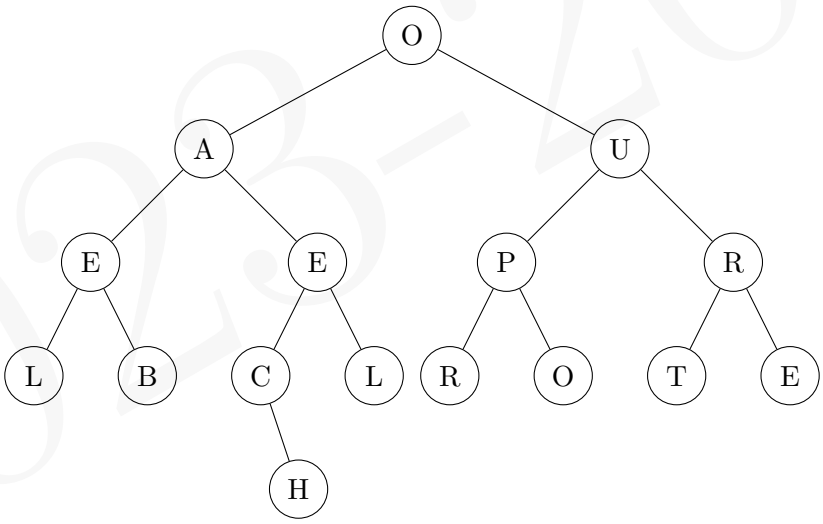
NOM : PRÉNOM :

Vous devez respecter les consignes suivantes, sous peine de 0 :

I) Lisez le sujet en entier une première fois avant de commencer à répondre
II) Lisez le sujet en entier avec attention
III) Ne détachez pas les agrafes du sujet
IV) Répondez sur le sujet (et utilisez le brouillon avant de répondre)
V) Écrivez lisiblement vos réponses (si nécessaire en majuscules)
VI) Ne trichez pas

1 Arbres Binaires (14 points)

1.1 (4 points) Indiquez toutes les propriétés que possède cet arbre, écrivez les clés lors d'un parcours profondeur main gauche de l'arbre dans les 3 ordres ainsi que lors d'un parcours largeur, puis corrigez-le pour qu'il devienne un arbre binaire strict :



Arité : Taille : Hauteur : Nb feuilles :

Parcours profondeur :

ordre préfixe : _
ordre infixe : _
ordre suffixe : _

Parcours largeur :

ordre : _

Ajoutez suffisament de nœud(s) sur le schéma pour que l'arbre binaire devienne un *arbre binaire strict*. Si l'arbre est déjà un arbre binaire strict, laissez-le tel quel sans ajouter de nœud.

1.2 (4 points) Dessinez le résultat de l'insertion dans cet ordre précis des éléments suivants dans un ABR (insertion en feuille) et dans un AVL :

Éléments insérés : 42 - 7 - 12 - 83 - 56 - 64 - 21 - 4

ABR
insertion en feuille
(2 points)

AVL
(2 points)

1.3 (2 points) Écrivez une fonction récursive « *parc_prof_rec* » effectuant un parcours profondeur main gauche dans un arbre binaire, et affichant les nœuds dans chacun des ordres :

Il faut expliciter les éventuels ordres au format : « Ordre : nœud » (exemple : « Préfixe : 42 »)

1.4 (2 points) Écrivez une fonction itérative « *parc_larg* » effectuant un parcours largeur dans un arbre binaire, et affichant chacun des nœuds dans l'ordre hiérarchique :

Vous pouvez utiliser les structures externes suivantes sans les réécrire :
stack_t (create, push, head, pop, delete) *queue_t (create, enqueue, head, dequeue, delete)*

```
void parc_larg(node *root)
```

1.5 (1 point) Écrivez une fonction récursive « *hauteur_arbre* » calculant la hauteur d'un arbre :

```
int hauteur_arbre(node *root)
```

1.6 (1 point) Écrivez une fonction « *recherche_abr* » recherchant un nœud dans un ABR et le renvoyant. Si le nœud n'est pas trouvé, il faut renvoyer `NULL` :

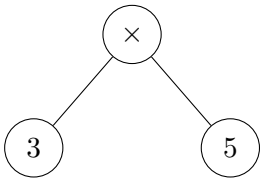
```
node *recherche_abr(node *root, int id_target)
```

2 Arbres Binaires : Usages (6 points)

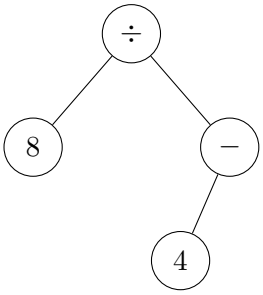
En mathématiques, vous avez appris tout au long de vos études primaires et secondaires à utiliser plusieurs opérateurs communs : « $+ - \times \div$ ». Ces opérateurs sont tous des opérateurs *binaires*, c'est-à-dire qu'ils prennent deux paramètres : « $3 + 5$ » peut également s'écrire *addition(3, 5)*.
« $+$ » et « $-$ » sont également des opérateurs *unaires* lorsqu'ils représentent le signe de la valeur absolue qui suit : « -8 » peut aussi s'écrire *négatif(8)*.

Dans cet exercice, vous allez transformer des arbres en formules mathématiques, puis l'inverse, et enfin, écrire l'algorithme de parcours et d'exécution de ces opérations.

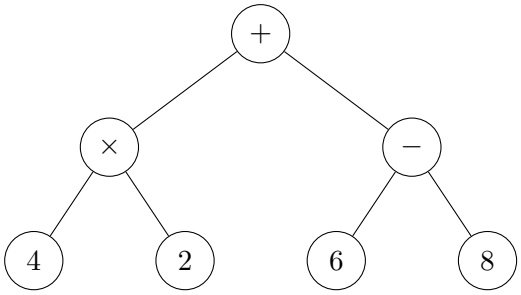
2.1 (2 points) Transformez les arbres suivants en leurs formules mathématiques associées :



(0,5 pt) Formule :



(0,5 pt) Formule :



(0.5 pt) Formule :

(0.5 point) Que remarquez-vous concernant les opérateurs et les nombres par rapport à leur placement dans les arbres ?

2.2 (2 points) Transformez chacune des formules suivantes en un arbre binaire :

(0,5 point)
$3 \times (4 - 5)$

(0,5 point)
$(-5) \div (8 + 3)$

(1 point)
$(6 \times (-4)) + ((-8) \div (5 - 3))$

2.3 (2 points) Écrivez une fonction « *exec_maths_tree* » exécutant l'expression représentée par l'arbre binaire donné en paramètre, et renvoyant le résultat :

Chaque nœud de cet arbre est une structure contenant du texte en tant que valeur, vous pouvez utiliser les fonctions suivantes pour tester ou extraire le contenu de la valeur. Les nœuds ne contiendront jamais autre chose que l'un des 4 opérateurs ou un nombre, et l'arbre ne sera jamais vide.

```
typedef struct math_node
{
  char *value;
  struct math_node lc;
  struct math_node rc;
} math_node;
```

`int UnaryOp(char *op, int val)` : fonction exécutant l'opérateur fournit en paramètre sur la valeur.

`int BinaryOp(char *op, int v1, int v2)` : fonction exécutant l'opérateur fournit en paramètre sur les 2 valeurs.

`int atoi(char *text)` : fonction transformant le texte donné en paramètre en un entier.

```
int exec_maths_tree(math_node *root)
```