TD 5
Arbres

Polytech Marseille - IRM 3ème année
Alexandra Bac

Algorithmique et structures de données
4h TD - 4h TP

1 TD

Exercice 1 (Arbres binaires)**.** On considère un arbre binaire implémenté par la structure :
Structure C associée :

```
struct znoeud {
  Elt elt ;
  struct znoeud *fg;
  struct znoeud * fd; };
typedef struct znoeud noeud ;
typedef struct znoeud * Arbre;
```

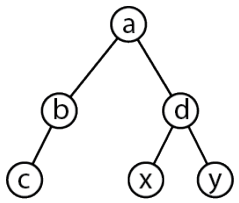


FIGURE 1 – Exemple d'arbre binaire

(i) Ecrire le fonction `creer_arbre` d'un module `arbre_binaire` pour les arbres binaires.

(ii) Ecrire une fonction type de parcours en profondeur.

(iii) Bien réfléchir aux fonctions de libération mémoire nécessaires dans ce module et proposer des fonctions pertinentes.

(iv) On est ici dans une implémentation sans effet de bord (type fonctionnel). Trouver des exemples montrant que la libération mémoire "manuelle" n'est pas évidente et pas toujours faisable (comme pour les listes).

(v) En utilisant votre module de files (construit sur les listes), écrire une fonction de parcours en largeur générique.

Exercice 2 (Fonctions sur les arbres)**.** Ecrire les fonctions récursives suivantes :

(i) Calcul de la hauteur d'un arbre

(ii) Calcul du nombre de feuilles d'un arbre

(iii) Rechercher un élément `e` dans un arbre (renvoyer un pointeur vers le noeud). Que se passe-t-il si l'on peut garantir la propriété suivante sur l'arbre : la valeur de la racine est supérieure à tous les noeuds du fils gauche et inférieure à tous ceux du fils droit.

(iv) Tester si deux arbres sont égaux

(v) En déduire une fonction déterminant si un arbre `b` est sous-arbre d'un arbre `a`

Estimer la complexité de toutes ces fonctions.

Exercice 3 (Passer une fonction en argument)**.** On veut pouvoir itérer une fonction lors du parcours en profondeur d'un arbre, c'est-à-dire si `f` est une fonction passée en argument, `e` son neutre et que l'arbre est celui présenté à la figure 1, on veut calculer `f(f(f(f(f(f(e,a),b),c),d),x),y)`. Le C permet de passer une fonction en argument, ou plus précisément un pointeur sur cette fonction. La syntaxe sera alors :

```
int iterer(Arbre a, int (*f)(int, int), int acc)
```

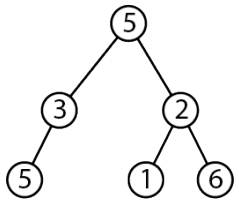
où `acc` est un accumulateur initialisé à l'élément neutre.

On écrira cette fonction itérer et on l'appliquera au calcul du minimum des valeurs de l'arbre, puis à leur somme en passant les bonnes fonctions à cette fonction `iterer`.

2 TP

(i) Compléter le module `arbre_binaire` fourni, implémentant le type abstrait des arbres binaires pour des éléments entiers.

(ii) Créer un arbre binaire :



(iii) Ecrire les fonctions de parcours en profondeur et en largeur et les tester sur cet arbre (afficher les noeuds au fur et à mesure qu'ils sont rencontrés).

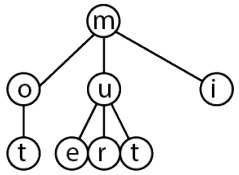
(iv) Ecrire la fonction :

```
int iterer(Arbre a, int *f(int, int), int acc)
```

on l'appliquera au calcul du minimum des valeurs de l'arbre, puis à leur somme et leur produit en passant les bonnes fonctions et les bons neutres à cette fonction `iterer`.

(v) Compléter le module `arbre_naire` fourni en matériel de TP.

(vi) L'utiliser pour créer l'arbre :



qui permet d'encoder la structure de sous-mots dans la langue.

3 Challenge

Exercice challenge. On va s'intéresser à la sérisalisation/désérialisation d'un arbre binaire. Sérialiser c'est écrire une chaîne de caractères permettant d'encoder un arbre. Modifier le type `Elt` dans votre module pour

passer à des arbres de caractères (la seule fonction qui doit être modifiée est celle d'affichage ...). On va adopter le codage suivant :

— Ecrire `1` pour un noeud interne suivi du caractère contenu puis du codage du fils gauche puis de celui du fils droit.

— Ecrire `0` pour un arbre vide

Par exemple, l'arbre 3 sera codé par `1a1b1c0001d1x001y00`.

(i) Ecrire une fonction `serialisation` effectuant ce codage.

(ii) Ecrire une fonction `deserialisation` reconstruisant l'arbre à partir de la chaîne de caractères (point fondamental : elle sera récursive ! et dans ce cas, simple ... la version itérative équivalente est très complexe).