TP : Sérialisation/Désérialisation

Langages OCaml et C

Exercice 1.

Sérialisation/désérialisation d'arbres binaires en C.

Dans ce sujet, les arbres binaires ont des étiquettes entières positives. Les fils d'une feuille sont deux arbres vides. L'arbre vide est modélisé en C par le pointeur `NULL`.

On sérialise un arbre binaire en écrivant les étiquettes des nœuds selon l'ordre d'un parcours en profondeur préfixe. On utilise un marqueur pour l'arbre vide : par exemple -1.

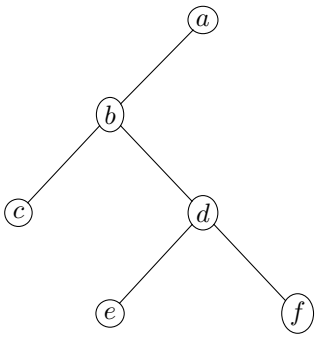

FIGURE 1 – Codage **a b c -1 -1 d e -1 -1 f -1 -1 -1**

- Télécharger l'archive pour ce sujet en entrant dans un terminal :

```
$ wget https://nussbaumcpge.be/public_html/Sup/MP2I/serialiser_btree.zip
```

L'archive contient le squelette d'un fichier C à compléter.

1. Écrire la fonction `void serialise(const char * filename, Node * root)` qui sérialise un arbre binaire dans un fichier texte dont le nom est donné. Le contenu attendu pour le fichier de sérialisation est donné en commentaire dans le `main`.
2. Écrire la fonction `Node * deserialise(const char * filename)` qui retourne l'arbre binaire résultat de la désérialisation du fichier dont le nom est donné en paramètre.

Exercice 2. Dans cet exercice on étudie une sérialisation de graphe en OCaml. On rappelle l'existence des fonctions suivantes :

```
# String.split_on_char ':' "\"0\":[0,1,4],";;
- : string list = ["\"0\""; "[0,1,4],"]
# String.sub "abcdefgh" 1 3;;
- : string = "bcd"
# int_of_string "32";;
- : int = 32
```

Les graphes sont représentés par le type

```
type graph = int list array;;
```

On veut sérialiser un graphe sous forme d'un fichier au format JSON. Par exemple le graphe `[|[0;1;4];[];[0;3]; [2];[1]|]` est sérialisé en un fichier texte dont le contenu est le suivant :

Listing 1 – sérialisation de `[|[0;1;4];[];[0;3]; [2];[1]|]`

```
{
"0":[0,1,4],
"1":[],
"2":[0,3],
"3":[2],
"4":[1]
}
```

Ainsi :

- Le contenu est entre deux accolades ;
- chaque numéro de sommet est présenté entre guillemets ;
- puis viennent deux points et une liste de voisins au formalisme `Python`. L'ordre est sans importance.
- on impose que chaque paire (clé,valeur) est séparée de la suivante par un passage à la ligne ;
- on considère qu'il n'y a pas d'espace ni de caractère de tabulation et que chaque sommet a sa liste de voisin.

1. Écrire la fonction `serialise (g:graph) (filename:string) : unit` qui sérialise le graphe g dans le fichier dont le nom est donné.
2. Écrire la fonction `deserialise (filename:string) : graph` qui retourne le graphe dont une représentation sérialisée est contenue dans le fichier appelé **filename**.

```
# let g = [|[0;1;4];[];[0;3]; [2];[1]|] in
serialise g "graphe"; deserialise "graphe";;
- : int list array = [|[0; 1; 4]; []; [0; 3]; [2]; [1]|]
```