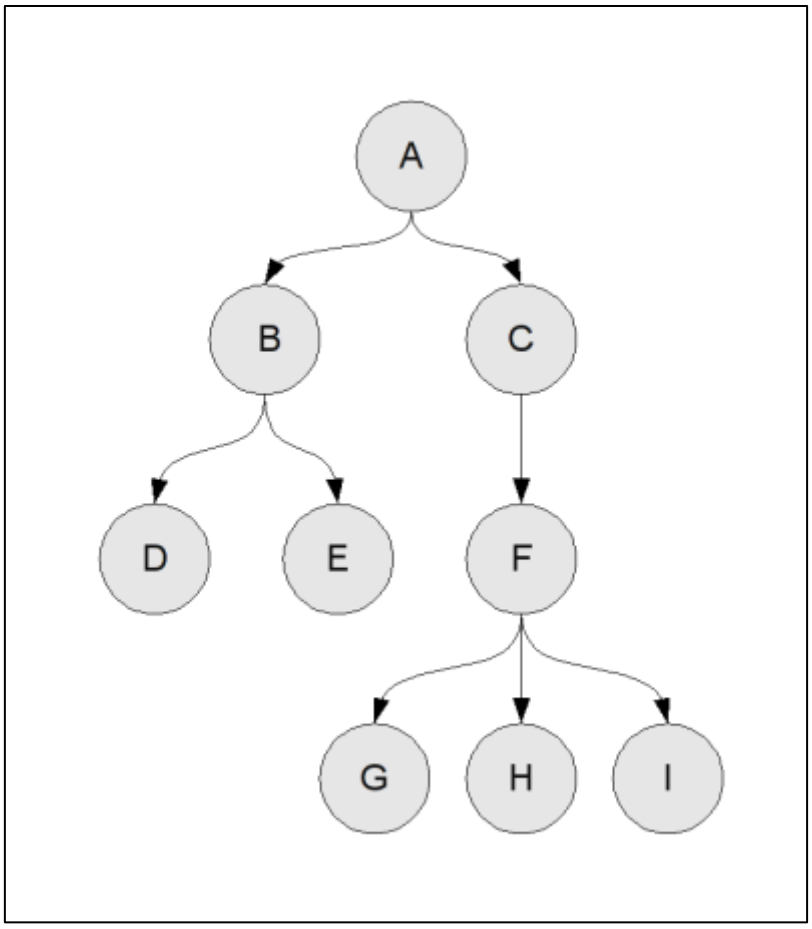
ARBRES BINAIRE ET N-AIRE

Travail de Fiona ISS, Romain SCHLOTTER, Valentin ZELLER, Kévin LI et Olivier FRANÇOIS
A Ludus-Académie, 2019-2020

I) Définition – Qu'est-ce qu'un arbre ?

En informatique, un arbre est une structure semblable à une liste. Cette structure est très utilisée aussi bien dans le jeu-vidéo (Path-Finding d'IA) ou plus simplement dans des algorithmes concernant les bases de données, ou l'arborescence de fichiers. On peut le représenter de la façon suivante :



Chaque élément de l'arbre possède donc un certain nombre de frères (ou sibling) et d'enfants (ou child). On verra plus tard que selon le type d'arbre, ce nombre est fixé.

II) Vocabulaire

Un arbre est composé de plusieurs éléments :

- Les feuilles : éléments ne possédant pas de fils dans l'arbre. Sur la représentation ci-dessus, D, E, G, H et I sont des feuilles.
- Les nœuds : éléments possédant des fils. Sur la représentation ci-dessus, B, C, et F sont des nœuds.
- Une racine : élément ne possédant pas de parent. Sur la représentation ci-dessus, A est la racine.

En plus de ces éléments, ces éléments possèdent des caractéristiques :

- La profondeur d'un nœud : se dit de la "distance" séparant le nœud de la racine de l'arbre. Par exemple, la profondeur du nœud F est 1 (ou 2 si on compte le nœud en lui-même).
- La hauteur d'un arbre : se dit de la plus grande profondeur d'une feuille de l'arbre. Par exemple, la hauteur de l'arbre illustré ci-dessus est 3 (A jusqu'à G, H ou I)
- La taille d'un arbre : se dit du nombre de nœud (éventuellement sans les feuilles) de l'arbre. Par exemple, sur l'illustration ci-dessus, l'arbre a une taille de 8.
- La longueur de cheminement : somme de la profondeur de chacune des feuilles. Sur l'illustration ci-dessus, la longueur de cheminement vaut 13.
- L'étiquette d'un nœud : désigne la donnée que stock le nœud.
- Le degré d'un nœud : désigne le nombre d'enfant que possède un nœud.

III) Les types d'arbres

On distingue plusieurs types d'arbres selon le nombre d'enfants/frères qu'on décide que chaque nœud possède.

A) Les arbres binaires

Les arbres binaires sont des arbres où chaque élément possède au plus 2 enfants. On peut ainsi les modéliser de la façon suivante :

```
typedef struct BinaryTree
{
    int value;
    struct BinaryTree *left;
    struct BinaryTree *right;
    struct BinaryTree *parent;
}BinaryTree;
```

Il est à noter que certains modèles ne prennent pas la peine de garder en mémoire pour chaque nœud son parent. Cependant, il est essentiel de modéliser les deux enfants, que l'on appelle souvent enfant gauche et enfant droit.
La structure contient donc des pointeurs vers ces éléments ainsi qu'un entier : c'est l'étiquette du nœud. Cette étiquette pourrait cependant être de n'importe quel type : chaine, structure…

Nous allons donc désormais nous intéresser aux méthodes permettant de manipuler ces arbres.

1) Création
2) Insertion, suppression
3) Taille, profondeur
4) Recherche, tri
5) Équilibrage d'un arbre binaire

B) Les arbres n-aires

Les arbres n-aires sont une généralisation des arbres binaires. Dans ces arbres-ci, chaque nœud peut avoir au plus n enfants. Puisque n peut être aussi grand que l'on veut, il n'est pas possible pour un nœud, comme pour les binaires, de le faire pointer vers ses n-enfants. Dans ce modèle-ci, on préfère donc faire pointer un nœud vers

- Son premier enfant (child)
- Son premier frère (sibling)

Cela peut ainsi donner quelque chose comme ça :

```
typedef struct Tree
{
    int value;
    struct Tree *sibling;
    struct Tree *child;
}Tree;
```

Tout comme pour les arbres binaires, voici quelques méthodes pour manipuler ces arbres.

1) Création

```
Tree *CreateTree(char value, Tree *child, Tree *sibling)
//Creer un arbre a  partir des parametres passes.
{
    Tree *tempTree = malloc(sizeof(*tempTree));

    if (tempTree == NULL)
    {
        printf("ERREUR DANS L'ALLOCATION : l'arbre n'a pas pu etre cree.\n");
        return NULL;
    }

    tempTree->value = value;
    tempTree->child = child;
    tempTree->sibling = sibling;

    return tempTree;
}
```

2) Insertion, suppression

```
void InsertFirstChild(Tree *parent, Tree *newChild)
{
    if (parent == NULL)
        return NULL;

    //Si le parent a déjà un child, il devient le sibling du nouveau child
    if (parent->child != NULL)
        newChild->sibling = parent->child;

    parent->child = newChild;
}
```

Pas grand-chose à dire de plus sur l'insertion.

La suppression est un peu plus délicate, il y a deux plus deux types de suppressions : celle où l'on supprime un enfant et celle où l'on supprime un sibling. Les deux étant néanmoins assez proche, nous n'allons ici expliciter que la suppression d'un enfant. Il faut penser à tous les cas ; pour cela prenons l'arbre suivant :

Imaginons que l'on souhaite supprimer l'enfant de A : le nœud B.

- Il faut commencer par déclarer D, l'enfant de B, comme étant l'enfant de A
- Puis déclarer C, le sibling de B, comme étant le sibling de E.

Ainsi, pour chaque suppression il faut donc penser à vérifier :

- Est-ce que le nœud à supprimer à un child ?
    - Si oui, cet enfant devient le nouveau child du parent.
- Est-ce que le nœud à supprimer à un sibling ?
    - Si oui, ce sibling devient le sibling du dernier child du nœud à supprimer.

Le code étant plutôt long, le lecteur est invité à le consulter dans le cbp associé à cette documentation.

3) Taille, hauteur

Les fonctions de taille et de hauteur étant semblables, nous ne parlerons ici que de la fonction calculant la taille.

```
int GetTreeSize(Tree *node)
//Renvoie la taille (nombre de noeuds) d'un arbre a  partir du noeud donne en comptant la racine.
{
    if (node == NULL)
    {
        ChangeColor(color_Red, color_Black);
        printf("ERREUR ARBRE VIDE\n");
        ChangeColor(color_White, color_Black);

        return -1;
    }

    int count = 1;

    Tree *currentNode = node->child;
    while (currentNode != NULL)
    {
        count += GetTreeSize(currentNode);
        currentNode = currentNode->sibling;
    }

    return count;
}
```

La fonction se base sur le principe de récursivité. On commence par partir de la racine de l'arbre puis l'on regarde son premier child. Si ce child possède lui-même un child, on le regarde (jusqu'à arrivé à

un nœud sans child) tout en incrémentant un compteur. Puis on fait pareil sur le sibling et lorsque l'on atteint un nœud n'ayant ni sibling ni child, on renvoie le compteur et on remonte au premier parent ayant un sibling (grâce à la récursivité) puis on recommence.

4) Binarisation d'un arbre n-aire