Rapport TP2- PRALG

ZHANG Mofan

Ce rapport a pour but de répondre à certaines questions des exercices du TP2. Le code correspondant est ***inttree.cpp***, ***inttree.h*** et ***main.cpp***. Les questions non traitées dans ce rapport sont directement répondues par le code ou par les commentaires dedans.

Le code a été mis en open source sur Github :
https://github.com/BASARANOMO/PRALG_TP2.git

Exercice 2 : Affichage d'un arbre

2.1) A quel parcours de l'arbre correspond la suite 12 8 4 9 23 17 15 ?

Parcours en profondeur d'abord.

Exercice 3 : Gestion d'erreur

3.1) Lister tous les cas d'erreur pour les fonctions de IntTree

- **getSon()** : Erreur d'accès aux éléments de vecteurs hors de portée (out of range error)

- **setSon()** : Erreur d'accès aux éléments de vecteurs hors de portée (out of range error)

- **removeLastSon()** : Erreur d'enlever de nœud (pop() dans notre cas) d'un vecteur vide

3.2) Pour lesquelles peut-on signaler l'erreur par valeur de retour ? Auxquelles peut-on facilement ajouter un statut d'erreur ?

- Pour l'erreur d'accès aux éléments de vecteurs par **getSon()**, on peut la signaler par valeur de retour. Si la position de nœud est hors de portée, la méthode retourne **nullptr**, qui est un pointeur nul.

3.3) Pour lesquelles peut-on signaler l'erreur par exception ?

- Pour l'erreur d'enlever de nœud d'un vecteur vide par **removeLastSon()**, on peut la signaler par exception.

- Pour l'erreur d'accès aux éléments de vecteurs par par **setSon()**, on peut la signaler par exception.

Exercice 5 : Parcours d'arbre

Pour cet exercice, j'ai implémenté plusieurs méthodes de parcours d'arbre comme listées ci-après :

- DFS (profondeur d'abord)
    - Pre-order
        - De manière récursive
        - De manière itérative
    - Post-order
        - De manière récursive
        - De manière itérative

- BFS (largeur d'abord)
    - De manière itérative

Pour ce faire, la STL de pile (stack) et de file (queue) a été utilisée.

Et puis, j'ai implémenté la méthode **maxDepth()** dont la complexité en temps est *O(N)*[1] et la complexité en espace est *O(1)*, ainsi que la méthode **minDepth()** inspirée de l'idée de BFS (parcours en largeur d'abord).

[1] N est le nombre de nœuds d'arbre.