TP : Structures de données en C - Arbres binaires et files FIFO

Professeur

Objectif

L'objectif de ce TP est de manipuler les structures de données en C, en particulier les arbres binaires et les files FIFO. Vous allez implémenter des fonctions pour créer et manipuler des arbres binaires, ainsi que des fonctions pour effectuer des parcours en profondeur et en largeur.

Partie 1 : Arbres binaires

Un arbre binaire est une structure de données composée de nœuds, où chaque nœud a au plus deux fils : un fils gauche et un fils droit.

Structure de l'arbre binaire

Voici la structure d'un nœud d'arbre binaire :

```c
typedef struct Node {
    int data;               // Données stockee dans le nœud
    struct Node* left;      // Pointeur vers le fils gauche
    struct Node* right;     // Pointeur vers le fils droit
} Node;
```

Fonctions à implémenter

1. `Node* createNode(int data)` : Crée un nouveau nœud avec la donnée spécifiée.
2. `int height(Node* root)` : Calcule la hauteur de l'arbre.
3. `int size(Node* root)` : Calcule la taille de l'arbre (nombre de nœuds).
4. `void inorderTraversal(Node* root)` : Parcours infixe (gauche → racine → droite).
5. `void preorderTraversal(Node* root)` : Parcours préfixe (racine → gauche → droite).
6. `void postorderTraversal(Node* root)` : Parcours suffixe (gauche → droite → racine).

7. `bool find(Node* root, int value)` : Indique si l'entier `value` est une étiquette de l'arbre.
8. `int max_label(Node* root)` : Renvoie l'étiquette maximale de l'arbre. Une assertion est levée si l'arbre est vide.

Partie 2 : Files FIFO

Une file FIFO (First-In-First-Out) est une structure de données où le premier élément ajouté est le premier à être retiré. Dans cette partie, nous allons implémenter une file FIFO capable de stocker des pointeurs vers des nœuds d'arbre binaire.

Structure de la file FIFO

Voici la structure d'une file FIFO adaptée pour stocker des pointeurs vers des nœuds d'arbre binaire :

```
typedef struct QueueNode {
struct Node* data; // Pointeur vers un nœud d'arbre binaire
struct QueueNode* next; // Pointeur vers le nœud suivant
} QueueNode;

typedef struct {
QueueNode* front; // Pointeur vers l'avant de la file
QueueNode* rear; // Pointeur vers l'arrière de la file
} Queue;
```

Fonctions à implémenter

1. `Queue* createQueue()` : Crée une file vide.
2. `int isEmpty(Queue* queue)` : Vérifie si la file est vide.
3. `void enqueue(Queue* queue, struct Node* data)` : Ajoute un pointeur vers un nœud d'arbre binaire à la file.
4. `struct Node* dequeue(Queue* queue)` : Retire et retourne un pointeur vers un nœud d'arbre binaire de la file.

Partie 3 : Parcours en largeur (BFS)

Le parcours en largeur (BFS) visite les nœuds niveau par niveau, en utilisant une file FIFO. Dans cette partie, nous allons implémenter le parcours en largeur en utilisant la file FIFO adaptée.

Fonction à implémenter

1. `void breadthFirstTraversal(Node* root)` : Parcours en largeur de l'arbre.

Travail à faire

1. Implémentez les fonctions décrites dans les parties 1 et 2.
2. Testez vos fonctions en créant un arbre binaire et en effectuant les parcours en profondeur et en largeur.
3. Affichez les résultats des parcours, ainsi que la hauteur et la taille de l'arbre.