NF16, Fall 2022 (A22)
Rapport Projet
Les arbres binaires de recherche

CHEN Wenlong
GAO Yuan

11/12/2022

Ideé

Nous changeons le nom en majuscule et utilisons la fonction de comparaison pour juger de la taille entre différents noms, et enfin stockons les informations du patient en fonction de la structure de l'arbre binaire de la requête

Les functions supplémentaires

```
char * MajusculeString(char *a);
// Mettre les noms en majuscule avce malloc()

int comparer(char * a, char *b);
// Si le nom a supérieur à b, return 1. Sinon return 0. Si a=b, return -1

void supprimerConsltation(Consultation *Liste);
// Librer les espaces de nom, prenom et le list de Consultation

Patient * rechercher_node_parent(Parbre * abr, char* nm);
// Rechercher le parent d'un node, pour le fonction supprimer_patient

void des(Parbre *abr);
// Librer les espaces de patient.
```

Les functions principales

```
1. Patient* CreerPatient(char * nm , char * pr);
```

Description:
On va utiliser malloc() pour générer l'espace de type Patient, ensuit on va faire l'initialisation
La complexité temporelle est O(1)

```
2. void inserer_patient(Parbre * abr, char * nm, char * pr);
Example:
    Choisissez le fonction :
    1
```

```
    vous avez choisi ajouter un patient, entrez le prenom et nom :
    wenlong chen
    OK
```

Description:
Il s'agit d'une fonction récursive qui convertit d'abord le nom en majuscule, et si le nom entré est supérieur au nom du nœud actuel, puis le transfère au fils gauche. Sinon, le transfère au fils droit.
La complexité temporelle est O(logN)

```
3. Patient * rechercher_patient(Parbre * abr, char* nm);
```

Description:
On utilise le fonction comparer, Si il egale -1, on trouve le node.

```
4. void afficher_fiche(Parbre * abr, char* nm);
Example:
    Choisissez le fonction :
    3
    vous avez choisi afficher une fiche maladicale, entrez le nom du patient:
    chen
    Informations de le/la patient:
    Nom:CHEN, Prénom:WENLONG, numbre de consultations:1
    Date:14-07-2000 | Motif:malade fievre | Niveau urgent:3
```

Description:
Utilisez d'abord la fonction de recherche pour trouver le patient cible, puis imprimez les informations du patient
La complexité temporelle est O(logN)

```
5. void afficher_patients(Parbre * abr);

Example:
    Choisissez le fonction :
    4
    vous avez choisi afficher la liste des patients
    Nom: CHEN | Prénom: WENLONG
    Nom: GAO | Prénom: YUAN
```

Description:
Voici une fonction récursive qui prend un parcours de préordre
La complexité temporelle est O(logN)

```
6. Consultation * CreerConsult(char * date, char* motif, int nivu);
```

Description:
Utiliser la fonction malloc pour générer de l'espace de type Consultation, Étant donné que le pointeur de chaîne est transmis, nous devons également utiliser malloc pour générer de l'espace et utiliser strcpy pour copier
La complexité temporelle est O(1)

```
7. void ajouter_consultation(Parbre * abr, char * nm, char * date, char* motif, int nivu);
Example:
```

```
Choisissez le fonction :
2
vous avez choisi ajouter une consultation du patient, entrez le nom du patient:
chen
Entrez le data et niveau de urgence:
14-07-2000 3
Entrez le motif:
malade fievre
```

Description:
Utilisez la fonction recher pour trouver le patient cible, puis utilisez la fonction creerConsult pour créer des informations
La complexité temporelle est O(N)

`8. void supprimer_patient(Parbre * abr, char* nm);`

Description:
Nous discuterons dans 4 cas:
(1) Le nœud à supprimer est introuvable, revenir directement
(2) Le nœud à supprimer est un nœud feuille, supprimez le nœud directement et définissez le pointeur de son nœud parent sur NULL
(3) Sous-arbre gauche/droit est null et sous-arbre droit/gauche n'est pas null. supprimez le nœud et définissez le pointeur de son nœud parent sur son fils
(4) Sous-arbre gauche et droit ne sont pas null. Nous devons d'abord trouver le plus grand nœud du sous-arbre de gauche ou le plus petit nœud du sous-arbre de droite, Ensuite, échangez les deux nœudsici et et pointez les pointeurs de ces deux nœuds vers le bon nœud, on cherche le plus grand nœud du sous-arbre de gauche, et nous devons faire attention à savoir si ces deux nœuds sont adjacents
Pour cas(3) et (4), nous devons également discuter si le nœud à supprimer est le nœud racine
La complexité temporelle est O(N)

`9. void maj(Parbre * abr, Parbre * abr2);`

Description:
Chaque fois que cette fonction est exécutée, nous allons d'abord publier l'arborescence de sauvegarde, puis créer une nouvelle arborescence basée sur l'arborescence actuelle
La complexité temporelle est O(N)