NF16 – TP4 : Les Arbres Binaires de Recherche

Anne-Soline Guilbert—Ly et Thomas Bordes

Contexte et objectifs

Le département de l'Oise souhaite mettre en place un système pour tracer son stock de vaccins pendant la période épidémique. Sachant qu'il y a plusieurs marques de vaccins, la date de livraison et la quantité sont différents pour chaque marque. C'est-à-dire que, pour un jour, le département peut recevoir plusieurs vaccins de marque différente.

Pour ce faire, on va utiliser un ABR pour enregistrer tous les vaccins reçus dans le département. Chaque nœud dans l'ABR représente la date de réception, la quantité de vaccins ainsi que les marques. Les vaccins reçus dans le même jour sont organisés par une liste chaînée. Chaque élément dans la liste chaînée représente une marque de vaccin ainsi que sa quantité.

Livrables

Les livrables attendus pour ce projet sont :

- Fichier d'en-tête tp4.h, contenant la déclaration des structures/fonctions de base,
- Fichier source tp4.c, contenant la définition de chaque fonction,
- Fichier source main.c, contenant le programme principal.

Liste des fonctions supplémentaires

- void viderBuffer() ;

Sa complexité est en O(1) .

- T_ABR* creerNode(char* date, int nb_vaccins,char* marque);

creerNode permet la création d'un ABR avec pour racine, la marque et la date en paramètre.

Sa complexité est en O(1).

- Pile* creerPile() ;

creerPile permet de créer une pile.

Sa complexité est en O(1).

- int pile_vide(Pile* P) ;

pile_vide renvoie 1 si la pile est vide, sinon 0.

Sa complexité est en O(1).

- int pile_pleine(Pile* P)

pile_pleine renvoie 1 si la pile est pleine, sinon 0.

Sa complexité est en O(1).

- void empiler(Pile* P, T_ABR* Node) ;

empiler permet d'empiler d'en une pile un nœud de l'ABR.

Sa complexité est en O(1).

- T_ABR* depiler(Pile* P, T_ABR* Node);

Depiler permet de dépiler un nœud de l'ABR d'une pile.

Sa complexité est en O(1).

- int verifdate(char* date);

verifdate permet de vérifier si une date entrée est valide. Une date valide a une année comprise entre 2021 et 2099, un mois entre 1 et 12 et une journée entre 0 et 30.

Sa complexité est en O(1).

- int compar_date(char* date1,char* date2) ;

compar_date permet de comparer deux dates et de renvoyer 2 si les dates sont identiques, 1 si la date1 > date2 et 0 si date1 < date2.

Sa complexité est en O(1).

- void afficher_date(char* date);

afficher_date permet d'afficher la date en paramètre.

Sa complexité est en O(1).

- int nb_vacc_liste(T_ListeVaccins* liste_vacc,char* marque) ;

nb_vacc_liste renvoie le nombre de vaccins d'une marque dans une liste de vaccins. On parcourt la liste jusqu'à trouver ou non la marque en paramètre dans la liste.

Sa complexité est en O(n), n étant le nombre de vaccins dans la liste vaccins.

- T_ABR* smallerDroit(T_ABR* abr) ;

  smallerDroit nous permet d'avoir le nœud le plus petit dans le SA droit de notre nœud abr.

  Sa complexité est en O(h), h étant la hauteur de l'ABR.

- T_ABR* succABR(T_ABR* noeud, T_ABR* abr) ;

succABR permet d'obtenir le successeur d'un nœud dans un ABR. On appelle pour cela la fonction recSuccABR et minABR.

Sa complexité est en O(h), h étant la hauteur de l'ABR.

- T_ABR* recSuccABR(T_ABR* abr, T_ABR* noeud, T_ABR* candidat) ;

recSuccABR renvoit le nœud de l'ABR avec la date rentré en paramètre.

Sa complexité est en O(h), h étant la hauteur de l'ABR.

- T_ABR* minABR(T_ABR* abr) ;

minABR renvoit le nœud le plus à gauche du nœud.

Sa complexité est en O(h), h étant la hauteur de l'ABR.

- T_ABR* retirerNode(T_ABR** aRetirer,T_ABR* ABR) ;

retirerNode permet de retirer un nœud de l'ABR. Il distingue les cas où l'on n'a pas de fils, 1 fils gauche, 1 fils droit ou 2 fils. Il appelle la fonction smallerDroit quand le nœud a retiré a 2 fils.

Sa complexité est en O(h), h étant la hauteur de l'arbre.

- void vider_abr(T_ABR* abr) ;

vider_abr permet de désallouer dynamiquement un ABR. On effectue un parcours postfixe. Pour chaque nœud, on supprime la liste vaccin en désallouant dynamiquement chaque élément de la liste.

Sa complexité est en O(n*m), n étant la longueur de la liste vaccin de chaque nœud et m le nombre de nœud.

- void vider_noeud(T_ListeVaccins* liste_vacc) ;

vider_noeud permet de désallouer dynamiquement une liste vaccin.

Sa complexité est en O(n), n étant la longueur de la liste vaccin.

Liste des fonctions à implémenter

- void ajouterVaccinL(T_ListeVaccins** listeVaccins, char* marque, int nb_vaccins);

ajouterVaccinL nous permet d'ajouter un nombre de vaccins associé à une marque dans une liste de Vaccin. L'ajout est effectué à la fin de la liste.

La complexité est en O(n), n étant la longueur de la liste.

- void ajouterVaccinA(T_ABR** abr,char* date, char* marque, int nb_vaccins) ;

ajouterVaccinA nous permet d'ajouter un nombre de vaccins associé à une marque dans un ABR. On va donc parcourir l'ABR jusqu'à trouver la date et ajouter le nb_vaccins

et la marque dans la liste vaccin du nœud associé. Si on ne trouve pas le nœud, on en créé un.

Sa complexité est en O(n*h), n étant la longueur de la liste vaccin du nœud et h la hauteur de l'ABR.

- void afficherStockL(T_ListeVaccins* listeVaccins) ;

afficherStockL nous permet d'afficher l'ensemble des vaccins et la marque associé d'une liste de vaccins.

Sa complexité est en O(n), n étant la longueur de la liste vaccin.

- void afficherStockA(T_ABR* abr) ;

afficherStockA affiche l'ensemble des nœuds et leurs informations d'un ABR.

Sa complexité est en O(n*m), n étant la longueur de la liste vaccins de chaque nœud et m le nombre de nœud

- int compterVaccins(T_ABR* abr,char* marque) ;

compterVaccins renvoie le nombre de vaccins d'une marque de vaccin d'un ABR. On parcourt l'ensemble des nœuds de l'ABR et pour chaque nœud, sa liste de vaccins associé.

Sa complexicité est en O(n*m), n étant la longueur de la liste vaccin de chaque nœud et m le nombre de nœud.

- void deduireVaccinL(T_ListeVaccins** listeVaccins, char* marque, int nb_vaccins) ;

deduireVaccinL permet de retirer un vaccin d'une liste de vaccins. On parcourt la liste jusqu'à trouver la marque du vaccin recherché puis, l'on supprime cet élément.

Sa complexité est en O(n), n étant la longueur de la liste vaccins.

- void deduireVaccinA(T_ABR** abr, char* marque, int nb_vaccins) ;

  deduireVaccinA permet de retirer nb_vaccins d'une marque dans un ABR. On va parcourir les m nœuds et vérifier si pour chacun des nœuds, leurs listes de vaccins

contiennent la marque en paramètre. Si oui, on retirer nb_vaccins jusqu'à ce que nb_vaccins soit égale à nulle où qu'il n'y est plus de vaccins dans l'ABR.

Pour chaque nœud, on appelle nb_vacc_liste pour savoir la quantité de vaccins disponible dans le nœud actuel. On appelle deduireVaccinL si l'on trouve la marque dans la liste vaccin du nœud actuelle.

Sa complexité est en O(m*n*n), m étant le nombre de nœuds de notre ABR et n la longueur de la liste vaccins de chaque nœud.