**NF16 – TP3 : Les Listes linéaires chaînées**

*Anne-Soline Guilbert—Ly et Thomas Bordes*

**Contexte et objectifs**

Le département de l’Oise souhaite mettre en place un système pour gérer son stock de vaccins contre la Covid-19 en temps réel. Pour ce faire, on va enregistrer pour chaque commune possédant un centre de vaccination, le nombre de vaccins dont la commune dispose. En outre, le nombre de vaccins disponibles pour une commune sera réparti sur plusieurs semaines : cela correspond aux prévisions de vaccins à administrer chaque semaine.

**Livrables**

Les livrables attendus pour ce projet sont :

* Fichier d’en-tête tp3.h, contenant la déclaration des structures/fonctions de base,
* Fichier source tp3.c, contenant la définition de chaque fonction,
* Fichier source main.c, contenant le programme principal.

**Liste des fonctions supplémentaires**

* bool verifVaccSem(int semaine, int nb\_vaccins);

verifVaccSem nous permet de vérifier que l’utilisateur rentre une semaine et un nombre de vaccin acceptable (nb\_semaines compris entre 1 et 52 | un nombre de vaccins entier et positif). Le retour booléen sert à déterminer si oui ou non, les conditions sont respectées :

La complexité est en O(1).

* bool trouver\_ville(t\_ville\_elt \*liste, char\* ville);

trouver\_ville nous permet de savoir si une ville en paramètre de la fonction appartient bien à une liste de villes. En somme, on ne l’utilise pas au sein de notre programme mais pourrait servir à l’implémentation de nouvelles fonctions de service.

Sa complexité est en O(n), n étant le nombre de villes de notre liste de villes.

* t\_ville\_elt\* tri\_liste\_villes\_A(t\_ville\_elt\* liste, t\_ville\_elt\* ville, int total);

tri\_liste\_villes\_A a pour fonction de trier notre liste de villes afin qu’elle respecte les contraintes liées au nombre global de vaccins de chaque ville (par ordre croissant). Ainsi, elle gère les cas où une ville possède plus de vaccins totaux que la ville d’après. Nous l’utilisons dans AjoutVaccinV lorsqu’il est nécessaire de trier notre liste de villes.

Sa complexité est en O(n), n étant le nombre de villes de notre liste de villes.

* t\_ville\_elt\* ajout\_ville(t\_ville\_elt\* liste, t\_ville\_elt\* ville);

ajout\_ville nous permet d’ajouter une ville à une liste de villes. Elle gère les cas d’ajout en début, milieu et fin de liste. Nous l’utilisons dans AjoutVaccinV en tant que fonction de service.

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

* t\_ville\_elt\* tri\_liste\_villes\_D(t\_ville\_elt\* liste, t\_ville\_elt\* ville, int total);

tri\_liste\_villes\_D a pour fonction de trier notre liste de villes afin qu’elle respecte les contraintes liées au nombre global de vaccins de chaque ville (par ordre croissant). Ainsi, elle gère les cas où une ville possède moins de vaccins que la ville d’avant. Nous l’utilisons dans deduireVaccinV lorsqu’il est nécessaire de trier notre liste de villes.

Sa complexicité est en O(n), n étant le nombre de villes de notre liste de villes.

* t\_ville\_elt\* deduire\_Ville(t\_ville\_elt \*liste, char\* ville);

déduire\_Ville permet de supprimer une ville d’une liste de villes. Elle gère les cas de suppression en début, milieu et fin de liste. Nous l’utilisons dans deduireVaccinV en tant que fonction service.

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

* bool trouver\_semaine(t\_semaine\_elt\* liste,int semaine);

trouver\_semaine est un booléen nous permettant de savoir si une semaine appartient ou non à une liste de semaines. Elle est utilisée dans deduireVaccinV et afficherPlanification.

Sa complexité est en O(m), m étant le nombre de semaines dans notre liste de semaines.

* t\_semaine\_elt\* renvoyer\_semaine(t\_semaine\_elt\* liste,int semaine);

renvoyer\_semaine nous permet de renvoyer l’adresse de la semaine présente dans notre liste de semaines. Nous l’utilisons dans deduireVaccinV pour récupérer facilement l’adresse.

Sa complexité est en O(m), m étant le nombre de semaines de notre ville.

* t\_ville\_elt\* ajoutVaccin(t\_vaccin\_elt\* vaccinA, t\_vaccin\_elt\* vaccinB);

ajoutVaccin permet d’effectuer une copie du vaccinB dans le vaccinA. Il s’agit d’une fonction outils à fusionnerStock. Ainsi, nous allons copier toutes les villes et semaines associées du vaccinB dans le vaccinA. Pour chaque ville et chaque semaine de vaccin A, nous faisons appelle à ajouterVaccinV.

Sa complexité est en O((n + m) x (i + j)), n étant le nombre de villes de vaccinA, et m le nombre de semaines de chaque ville, i étant le nombre de villes de vaccinB, j étant le nombre de semaines de chacune de ces villes.

* void vider\_tableau\_vacc(t\_vaccin\_elt\* tabl\_vacc[10]);

vider\_tableau\_vacc, appelée à la fin du main, a pour objectif de désallouer dynamiquement le tableau de pointeurs de vaccin. Elle fait appel à vider\_vaccin.

Sa complexité est en O(n x m), n étant le nombre de villes de la liste de ville du vaccin, m étant le nombre de semaines d’une ville. Sachant que le tableau de vaccins est de taille fixe 10 maximum, cela ne rentre pas en compte dans la complexité.

* void vider\_vaccin(t\_vaccin\_elt\* tabl\_vacc);

vider\_vaccin permet de désallouer l’espace mémoire alloué dynamiquement d’un vaccin en paramètre. Il s’agit d’un fonction outils de vider\_tableau\_vacc.

Sa complexité est en O(n x m), n étant le nombre de villes de la liste de villes du vaccin, m étant le nombre de semaines d’une ville.

* void trier\_vaccins(t\_vaccin\_elt\* GESTION\_VACCINS[10], int i);

trier\_vaccins, appelée dans le case 3 du main, permet de retirer une marque de vaccin du tableau GESTION\_VACCINS lorsqu’il n’y en a plus. Elle place ensuite le dernier vaccin du tableau à l’emplacement du vaccin retiré pour éviter d’avoir une case NULL en plein milieu du tableau.

Sa complexité est en O(1), car la taille du tableau est constante et égale à 10.

**Liste des fonctions à implémenter**

* t\_semaine\_elt\* creerSemaine(int num\_semaine, int nb\_vaccins) ;

La fonction creerSemaine a une complexité en O(1) car elle ne fait que des initialisations.

* t\_ville\_elt\* creerVille(char\* ville, int num\_semaine, int nb\_vaccins) ;

La fonction creerVille appelle creerSemaine pour l’initialisation des semaines\_planifiees et a une complexité en O(1) pour la même raison.

* t\_vaccin\_elt\* creerVaccin(char\* marque) ;

La fonction creerVaccin a une complexité en O(1) pour la même raison.

* t\_semaine\_elt\* ajouterVaccinS(t\_semaine\_elt\* liste, int semaine, int nb\_vaccins) ;

La fonction ajouterVaccinS appelle la fonction creerSemaine si la liste des semaines est vide. Sinon, elle parcourt la liste des semaines et ajoute la semaine au bon endroit. Sa complexité est donc en O(m), avec m le nombre de semaines dans la liste des semaines.

* t\_semaine\_elt\* deduireVaccinS(t\_semaine\_elt \*liste, int semaine, int nb\_vaccins) ;

La fonction deduireVaccinS parcourt la liste des semaines de taille m et retire la semaine de la liste s’il ne reste plus de vaccins. Elle a une complexité en O(m).

* t\_ville\_elt\* ajouterVaccinV(t\_ville\_elt \*liste, char\* ville, int semaine, int nb\_vaccins) ;

ajouterVaccinV appelle la fonction creerVille si la liste de villes du vaccin n’existe pas. Sinon, elle parcourt la liste de villes. Si la ville n’est pas dans la liste, elle appelle creerVille et retourne la fontion supplémentaire ajout\_ville de complexité O(n). Si la ville existe, elle appelle la fonction ajouterVaccinS et éventuellement, appelle tri\_liste\_villes\_A. Sa complexité est donc en O(m+n).

* t\_ville\_elt\* deduireVaccinV(t\_ville\_elt \*liste, char\* ville, int semaine, int nb\_vaccins) ;

deduireVaccinV parcourt la liste des villes du vaccin. Elle appelle éventuellement ensuite les fonctions trouver\_semaine et renvoyer\_semaine toutes les deux de complexité en O(m). On utilise après la fonction deduireVaccinS (en O(m)) et puis la fonction deduire\_Ville s’il n’y a plus de vaccins dans cette ville. Dans un autre cas, elle appelle tri\_liste\_villes\_D pour retrier la liste des villes en fonction des totaux de vaccins de chaque ville. La fonction deduireVaccinV a donc une complexité en O(m+n).

* void afficherStock(t\_vaccin\_elt\* vaccin);

La fonction afficherStock a une complexité en O(n x m) avec n le nombre de villes de la liste de villes du vaccin et m le nombre de semaines d’une ville.

* void afficherPlanification(t\_vaccin\_elt\* vaccin,int semaine) ;

La fonction afficherPlanification a une complexité en O(n x m) avec n le nombre de villes de la liste de villes du vaccin et m le nombre de semaines d’une ville.

* t\_vaccin\_elt\* fusionnerStocks(t\_vaccin\_elt\* vaccinA, t\_vaccin\_elt\* vaccinB);

fusionnerStocks crée le nom du nouveau vaccin avec strcat() et appelle la fonction creerVaccin() puis ajoutVaccin(). Sa complexité est en O((n + m) x (i + j)) car elle appelle la fonction ajoutVaccin qui a cette complexité.