SCHNEIDER Martin
PICART Damien
TP3 NF16

NF16 : Compte rendu TP 3 - Listes chaînées

Partie 1: structures et fonctions supplémentaires que nous avons choisi d'implémenter et les raisons de ces choix.

Dans cette première partie, nous allons exposer les raisons pour lesquelles nous avons choisi d'implémenter des structures et fonctions en plus que celles demandées dans le fichier PDF fourni avec le dossier TP3. Nous n'avons pas implémenté de structures supplémentaires, donc cette partie concerne seulement les fonctions supplémentaires.

Tout d'abord, nous avons choisi d'implémenter les fonctions "*affichage_Soigneurs*", "*affichage_Patients*" et "*affichage_RendezVous*" afin de les utiliser dans les possibilités 2, 3 et 4 du menu principal. Cela nous permet de clarifier notre fonction "*menuPrincipal*". Dans les options 2, 3 et 4, le programme doit afficher **les patients** et leurs rendez-vous ou afficher tous **les soigneurs** et leurs intervalles de temps disponibles ou encore afficher un **rendez-vous** en indiquant l'identifiant du patient et le soigneur correspondant. C'est ainsi que nous avons codé une fonction pour afficher les patients contenus dans une liste de patients, ainsi que les informations sur chacun de leurs rendez-vous. De même pour afficher les soigneurs et leurs intervalles. Enfin, nous avons codé une fonction pour afficher les informations de chaque rendez-vous d'une liste de rendez-vous.

De même, pour purifier notre fonction "*menuPrincipal*" nous avons implémenté les fonctions "*liberer_resource_Patients*", "*liberer_resource_RendezVous*", "*liberer_resource_Soigneurs*", "*liberer_resource_Intervalles*" et "*liberer_ressources*" afin de **libérer la mémoire allouée** lorsque nous avons créé des listes de patients, de soigneurs, de rendez-vous ou d'intervalles. Ainsi, les quatre premières fonctions citées sont utilisées dans la cinquième, qui permet d'éviter les fuites de mémoires lorsque l'utilisateur veut quitter le programme.

Nous avons implémenté la fonction "*getSoigneur*" afin de l'utiliser dans la fonction "*ordonnancer*". La fonction "*getSoigneur*" permet de retourner une variable de type pointeur sur une structure soigneur en donnant en paramètre son identifiant. Cela permet de **trouver un soigneur en particulier** dans une liste de soigneur. On utilise cette fonction directement en argument de la fonction "*affecterRDV*" (elle même appelée dans "*ordonnancer*") pour avoir le soigneur adéquat dans la liste de soigneurs de l'ordonnancement "solution".

En ce qui concerne les fonctions "*MergeSort*", "*somme_duree_rdv*", "*FrontBackSplit*" et "*SortedMerge*", nous n'avions pas vu qu'elles étaient fournies. Nous les avons donc recodées. Pour cela, nous avons implémenté une fonction supplémentaire "*ajouterFin*" utilisée dans "*SortedMerge*" afin d'ajouter un patient à la fin d'une liste de patients.

- "*FrontBackSplit*" permet de **diviser** une liste chaînée en deux sous liste chaînées.
- "*somme_duree_rdv"* permet de calculer la **somme des durées** de tous les rendez-vous d'un patient.
- "*SortedMerge"* permet de **fusionner** deux listes chaînées de patients en comparant pour chacun d'eux leur somme de durées de rendez-vous cumulées. Elle retourne alors une nouvelle liste de patients triées dans l'ordre décroissant.

Ces trois précédentes fonctions sont utilisées dans "*MergeSort"* qui, quant à elle, permet de trier une liste chaînée de patients dans l'ordre décroissant de leur somme de durées de rendez-vous cumulées.

Nous avons également choisi d'implémenter une fonction "*exist_fichier*" que l'on utilise dans la possibilité 1 du menu principal lors de l'importation du fichier. En effet, cette fonction permet de

retourner 1 si le fichier a **bien été ouvert**, et **0 sinon**. On utilisera cette valeur dans notre menu principal à chaque fois que l'utilisateur veut effectuer une action, pour s'assurer qu'il travaille bien sur un fichier que le programme a pu ouvrir.

Enfin, la fonction "*menuPrincipal*" permet de regrouper toutes les possibilités du menu principal grâce à une instruction *switch.* C'est dans cette fonction que nous allons appeler la majorité de toutes les autres fonctions implémentées. En plus de ces fonctions, nous avons ajouté des boucles while dans les différents *case* du *switch* afin de déterminer les bons attributs à passer en paramètres des fonctions appelées[1]. Ainsi, la complexité de chaque possibilité du menu principal ne dépend pas seulement des fonctions précédemment citées que nous appelons.

Partie 2: exposé succinct de la complexité de chacune des fonctions implémentées.

Dans cette deuxième partie, nous allons étudier la complexité des fonctions que nous avons recodées.

Tout d'abord, les fonctions "*creerListeSoigneur*", "*creerListePatient*", "*creerListeRendezVous*", "*affichage_RendezVous*", "*date*" et "*exist_fichier*" ne présentent que des affectations ou des tests. Leur complexité est donc en **O(1)**.

Fonction "*ajouterSoigneur*": la complexité est déterminée par la boucle while: cette boucle sert à parcourir toute la liste de soigneurs. Notons n le nombre de soigneurs dans la liste, la complexité de cette fonction est en **O(n)**

Fonction "*ajouterPatient*": cette fonction a la même structure que la fonction "ajouterSoigneur": elle présente la même boucle while qui parcourt la liste des patients. En notant n le nombre de patients dans la liste, la complexité de cette fonction est en **O(n)**.

Fonction "*ajouterPatient_avecRDV*": même fonction que "ajouterPatient" mis à part une affectation qui change. La complexité reste inchangée et est en **O(n)**.

Fonction "*ajouterRendezVous*": de même que les fonctions "ajouterSoigneur" et "ajouterPatient", la complexité de cette fonction est déterminée par la boucle while. Celle-ci permet de parcourir toute la liste des rdv. En notant n le nombre de rdv dans la liste, la complexité est alors en **O(n)**.

Fonction "*modifierRendezVous*": la complexité de cette fonction est déterminée par la boucle while. Celle-ci parcourt la liste des rdv pour trouver celui qui correspond au bon soigneur. Dans le pire des cas, on parcourt toute la liste de rdv. En notant n le nombre de rdv dans la liste, la complexité est alors en **O(n)**.

Fonction "*supprimerRendezVous*": la complexité de cette fonction est déterminée par la boucle while. Celle-ci permet de parcourir la liste des rdv pour trouver celui à supprimer. Au pire des cas, on parcourt toute la liste. Donc en notant n le nombre de rdv dans la liste, la complexité de cette fonction est en **O(n)**.

Fonction "*affichage_Soigneurs*": la complexité de cette fonction est déterminée par les deux boucles while imbriquées. La première permet de parcourir la liste des soigneurs tandis que la deuxième permet de parcourir la liste des intervalles d'un soigneur. En notant n le nombre de soigneurs dans la liste et m le nombre d'intervalles maximal affectés à un soigneur, la complexité de cette fonction est en **O(n*m)**.

[1] Par exemple, dans le *case 5,* on utilise une boucle *while* pour avoir un pointeur sur le patient dont l'utilisateur veut modifier le rendez-vous.

Fonction "*affichage_Patients*": comme pour la fonction précédente, si on note n le nombre de patients dans la liste et m le nombre de rendez-vous maximal affectés à un patient, la complexité est en **O(n*m)**.

Fonction "*creerInstance*": la complexité de cette fonction est déterminée par ses trois boucles for. La première effectue n itérations en notant n le nombre de patients contenus dans le fichier à traiter. La deuxième boucle for, qui est imbriquée dans la première, parcourt le nombre de rdv d'un patient. Si on note m le nombre maximal de rdv affectés à un patient, cette boucle effectue m itérations. Enfin, la dernière boucle for qui n'est pas imbriquée effectue k itérations en notant k le nombre de soigneurs contenus dans le fichier à traiter. Cette fonction a donc une complexité en **O(n*m+k)**.

Fonction "*affecterRdV*": la complexité est déterminée par la boucle while. Celle-ci parcourt la liste des intervalles d'un soigneur. En notant n le nombre d'intervalles pour un soigneur dans la liste d'intervalles, la complexité est en **O(n)**.

Fonction "*getSoigneur*": la complexité est déterminée par la boucle while qui parcourt la liste de soigneurs. En notant n le nombre de soigneurs dans cette liste, la complexité de cette fonction est en **O(n)**.

Fonction "*ordonnancer*": cette fonction fait appel à la "*MergeSort*" (qui est "offerte"), "*affecterRDV*" et présente deux boucles while imbriquées. On pose **p** le nombre de patients dans la liste triée, **m** le nombre maximal de rendez-vous d'un patient et **k** la complexité de la fonction "*MergeSort*". La boucle while la plus profonde parcourt la liste de rendez-vous d'un patient. Elle effectue donc au maximum m itérations. Or dans cette boucle, on appelle la fonction "*affecterRDV"* qui est en O(n). Cette boucle a donc une complexité en O(m*n). La boucle while la plus externe parcourt la liste des patients. Elle effectue donc p itérations. La complexité de cette boucle est donc en O(m*n*p). On doit ajouter à cela la complexité de "*MergeSort*". La complexité finale de la fonction "ordonnancer" est donc en **O((m*n*p)+k).**

Fonction "*somme_duree_rdv*": la complexité de cette fonction est déterminée par la boucle while. Celle-ci permet de parcourir la liste de rendez-vous d'un patient afin d'en faire la somme des durées. En notant n le nombre de rendez-vous d'un patient, la complexité est en **O(n)**.

Fonction "*ajouterFin*": la complexité est déterminée par la boucle while. Celle-ci parcourt une liste de patients. En notant n le nombre de patients dans la liste, la complexité est en **O(n)**.

Fonction "*exportSolution*": la complexité est déterminée par les deux boucles while et les deux boucles for. Les deux boucles while parcourent chacune respectivement le nombre (n) de patients dans la liste et le nombre (m) de soigneurs dans la liste. La première boucle for effectue n itérations et la deuxième, qui est imbriquée dans la première, effectue k itérations, avec k le nombre de rdv maximum pour un patient. Avec les informations, on peut déterminer la complexité qui est en **O(n+m+n*m)**.

Fonction "*liberer_ressources*": cette fonction possède deux boucles while. La première parcourt la liste des soigneurs et la deuxième parcourt la liste des patients. En notant n le nombre de soigneurs dans la liste et m le nombre de patients dans la liste, la complexité de cette fonction est en **O(m+n)**.

Fonction "*liberer_resource_Intervalles*": la boucle while parcourt une liste d'intervalles. En notant n le nombre d'intervalles dans la liste, la complexité est en **O(n)**.

Fonction "*liberer_resource_Soigneurs*": la boucle while parcourt une liste de soigneurs. En notant n le nombre de soigneurs dans la liste, la complexité est en **O(n)**.

Fonction "*liberer_resource_RendezVous*": la boucle while parcourt une liste de rendez-vous. En notant n le nombre de rendez-vous dans la liste, la complexité est en **O(n)**.

Fonction "*liberer_resource_Patients*": la boucle while parcourt une liste de patients. En notant n le nombre de patients dans la liste, la complexité est en **O(n)**.