Rapport Final du Projet Informatique du Groupe 7

Alexandre De Boé, Felix Charvin, Florian Da Silva, Tom Taranto 20/05/2016

Sommaire

1 Introduction

Notre agence de tourisme spatial propose plusieurs façons de visiter le système solaire. Les voyages sont organisés autour de 3 croisières qui répondent à des thèmes différents. Si le client préfère organiser son propre voyage, cela est possible. Malheureusement, le coût d'un voyage entre deux astres stellaires nous force à établir des contraintes, et le nombre de places par navette spatiale étant limité, nous devons établir des quotas. Ce concept encore unique au monde nous fait parvenir un nombre considérable de demandes. Nous devons donc attribuer à chaque personne un voyage, et tenter de satisfaire un maximum de clients. Comment répartir ces voyageurs intersidéraux? Pour répondre à ce problème, nous allons présenter le programme que nous avons développé.

2 Structures de données

Pour notre programme, nous avons besoin d'introduire les structures de données suivantes:

2.1 listePersonnes

```
1 typedef struct listePersonnes_base* listePersonnes;
```

listePersonnes contient les informations concernant les personnes : leurs choix ainsi que leur priorité. Nous avons choisi pour cette structure d'utiliser une liste triée par priorité décroissante, afin de pouvoir accéder directement à la personne avec la plus haute priorité. Les informations de la personnes sont contenues dans un tableau de 17 `char*` organisé comme : [nom, prénom, choix1, choix2, 6 planètes du choix1, 6 planètes du choix2, priorité].

2.2 dictPriorites

```
1 typedef struct dictPriorites_base* dictPriorites;
```

Les choix des personnes et leur priorité ne se trouvant pas dans le même fichier, nous avions besoin pour remplir listePersonnes d'une structure intermédiaire. dictPriorites est un dictionnaire qui à la clef [nom, prénom] renvoie la priorité de la personne en question. Nous avons choisi un dictionnaire car nous avions besoin d'une structure

adaptée aux recherches. Ce dictionnaire est implémenté par une table de hachage, utilisant pour fonction de hachage la méthode de la multiplication avec la constante $\frac{\sqrt{5}-1}{2}$.

2.3 dictContraintes

```
1 typedef struct dictContraintes_base* dictContraintes;
```

dictContraintes est la structure regroupant les informations sur les différentes contraintes. C'est un dictionnaire dont la clé est le nom d'une planète et la valeur associé un tableau de 3 `char*` : [zone associée à la planète clé, planète contrainte, zone associée à la planète contrainte] Nous avons choisi d'avoir en valeur de retour les zones des planètes car, par la façon dont nous avons contruit la structure contenant les informations sur les planètes (tableau de dictionnaires), nous avions besoin de connaître le nom d'une planète ainsi que sa zone pour savoir le nombre de places restantes. De même que pour dictPersonnes, ce dictionnaire est implémenté par une table de hachage avec la même fonction de hachage.

2.4 listeFinale

```
1 typedef struct listeFinale_base* listeFinale;
```

listeFinale est une liste contenant les informations concernant le choix attribué à chaque personne. Elle contient un tableau de 9 `char*` : [nom, prénom, choix retenu, les 6 planètes du choix retenu]. Elle sert à l'exportation vers le fichier csv. Elle est triée par ordre alphabétique sur les nom de famille des passagers.

2.5 dictDestinations

```
1 typedef struct dictDestinations_base* dictDestinations;
```

dictDestinations contient les informations concernant les croisières proposées. C'est un dictionnaire dont les clés sont les planètes et la valeur associée à chaque clé est le nombre de places restantes pour la planète en question. En pratique, nous avons choisi d'utiliser un dictionnaire par zone. Chaque croisière est représentée par un tableau

de dictionnaires de 6 cases, chacune contenant les planètes d'une zone en particulier. Nous avons choisi d'utiliser une structure de dictionnaire car nous voulions une structure adaptée aux recherches permettant d'obtenir facilement le nombre de places restantes pour une planète, en connaissant son nom et sa zone. De même que pour dictPersonnes, ce dictionnaire est implémenté par une table de hachage avec la même fonction de hachage.

3 Importation/Exportation

Au début, ce module a paru plutôt simple à réaliser sachant que les données dans les fichiers .csv était toutes écrites dans un même format. Néanmoins, en réalisant les tests unitaires sur les différentes fonctions, nous nous sommes vite rendu compte que ce n'était pas si simple que ça et qu'il fallait trouver les bonnes fonctions et les bonnes options afin de bien traiter les données en brut. Cela fait, nous nous sommes confrontés au problème de l'allocation mémoire qui s'est avéré au final être plus compliqué que prévu mais que nous avons pu résoudre en raisonnant simplement en terme de données stockées et de boucle à parcourir. Le dernier problème rencontré est apparu lors du traitements des fichiers .csv contenant des "cases" vides. En effet, nous utilisions la fonction fscanf qui ne prend pas en charge les chaînes vides, ce qui nous a obligés à changer de méthode pour cette fonction. Nous sommes donc passé à une analyse caractères par caractères afin de récupérer toutes les données.

4 Algorithme d'affectation

Result: Renvoie la liste des voyageurs avec le voyage sur lequel ils sont affectés

Créer quatres listes vides;

for *i parcourant la liste des voyageurs* **do**

    if *le choix1 du voyageur i est organisé* **then**

        voyageur i va dans la liste1;

    else

        voyageur i va dans la liste2;

    end

end

for *i parcourant liste1* **do**

    if *il reste des places pour le choix1* **then**

        affecter le voyageur i a son choix1 dans la liste resultat;

    else

        if *si son choix2* == *libre* **then**

            insérer voyageur i dans liste2 avec la bonne priorité;

            son choix1 $\leftarrow$ choix2

        else

            if *il reste des places pour son choix2* **then**

                affecter le voyageur i a son choix2 dans la liste resultat;

            end

            insérer voyageur i dans liste3;

        end

    end

end

```
for i parcourant liste2 do
    if le choix1 du voyageur i respecte les contraintes then
        if Il reste des places disponibles pour son choix1 then
            affecter le voyageur i a son choix1 dans la liste resultat;
        else
            if choix2 != libre then
                if si il reste des places pour son choix2 then
                    affecter le voyageur i a son choix2 dans la liste resultat;
                else
                    insérer voyageur i dans liste3;
                end
            else
                insérer voyageur i dans liste3;
            end
        end
    else
        if choix2=!libre then
            if si il reste des place pour son choix2 then
                affecter le voyageur i a son choix2 dans la liste resultat;
            else
                insérer voyageur i dans liste3
            end
        else
            insérer voyageur i dans liste3
        end
    end
end

renverser liste3;
for i parcourant liste3 do
    trouver une croisière pour le voyageur i satisfaisante par rapport à son choix1;
    affecter le voyageur i à cette croisière dans la liste resultat;
end
retourner le fichier .csv correspondant à la liste resultat;
```

L'algorithme d'affectation en lui-même a posé de nombreux problèmes. En effet, de nombreux cas particuliers sont apparus, et il a fallu trouver une méthode pour tous les résoudre. Nous nous sommes efforcés de satisfaire le maximum de personnes possibles, nous avons donc commencé par placer les personnes de priorités les plus importantes. Ensuite, il fallait placer les personnes restantes. En suivant les choix de

chacun, un nombre non négligeable de personnes se trouvaient sans voyage possible à cause des contraintes. Il a donc fallut faire attention au contraintes lors des placements. Nous avons donc cherché à permettre aux voyageurs de trouver une place dans un voyage, le plus proche de son choix initial, bien que la plupart des planètes soient déjà remplies par d'autres voyageurs. Tâchons d'expliquer comment trouver une croisière à quelqu'un qui n'a pas ses choix. Tout d'abord, on essaye pour chacune des zones, de lui affecter la planète de son choix, lorsque c'est impossible, on lui en affecte une autre. Mais cet itinéraire doit vérifier les contraintes, donc on va le modifier pour qu'il respecte les contraintes. Si une contrainte n'est pas vérifiée, on trouve un autre astre qui permet de vérifier la contrainte (par exemple, le voyageur passe par Pluton, mais pas par Jupiter). Mais si l'astre en question n'est pas disponible (planète déjà pleine, ici Jupiter), on teste une autre planète. Pour éviter de répéter ce processus à l'infini, il faut se souvenir par quelles planètes on ne peut pas passer. On change donc de planète de départ(ici Pluton) jusqu'à trouver une planète dans la même zone qui respecte les contraintes pour le voyage. Cela peut s'avérer impossible. On ajoute donc temporairement la personne au voyage(elle passe par Jupiter), dépassant ainsi les quotas. Comment rétablir les quotas? On modifie l'itinéraire d'une personne déjà affectée et passant par cette planète(Jupiter). On choisit cette personne de la manière suivante: il faut qu'elle passe par la planète qui est liée par la contrainte, mais pas par la planète qui pose problème(On cherche un voyageur passant par Jupiter, mais pas par Pluton). On peut ainsi lui affecter une autre planète dans la zone où la contrainte posait problème. Les quotas sont donc ainsi rétablis. La personne n'ayant pas de voyage précédemment a un voyage maintenant, mais il a fallu réaffecter un autre passager.

5 Découpage en module

Il a été décidé de découper le programme en 3 modules. Le premier module s'occupe de l'importation et de l'exportation des données .csv en les mettant dans les bonnes structures de données. Alexandre De Boé s'en est occupé.
Le second module va implémenter les listes et toutes les structures nécessaires au bon fonctionnement du programme. C'est l'oeuvre de Florian Da Silva.
Le dernier module permet l'affectation de personnes à des croisières, Félix et Tom s'en sont chargé.
Une fonction main et un makefile sont aussi disponibles.

6 Choix du langage

Le langage PHP/MySQL semblait adapté, en effet, il s'agit de traiter une base de donnée (ici les voyageurs), le langage nous aurait aussi permis de coder le site web correspondant à l'inscription. De plus, la partie langage objet de PHP aurait pu être utile pour ce projet. Mais cela ne nous semblait pas adapté, nous ne possédions pas alors les connaissances nécessaires pour être tout à fait à l'aise. Nous avons choisi le langage C pour résoudre ce problème. En effet, nous ne nous intéressons pas à l'aspect "objet" de ce projet. De plus, de nombreuses structures de données sont utilisées et le langage C nous permet de toutes les définir et de les utiliser facilement. Par ailleurs, le langage C est le langage que nous maîtrisons le mieux.

7 Problèmes rencontrés

Nous avons rencontré de nombreuses difficultés lors de ce projet. Lors de l'importation, il y a besoin d'allouer beaucoup de mémoire, qu'il faut libérer au bon moment. Ce ne fut pas un travail aisé mais nous en sommes venu à bout. L'affectation des voyages est la partie qui nous a posé le plus de problèmes. De nombreux cas particuliers se sont présentés, il a fallu tous les prendre en compte avec l'algorithme. A chaque fois que l'algorithme était terminé, un nouveau cas se présentait, ce qui nous obligeait à repenser et réécrire, parfois dans sa globalité l'algorithme. Un des problèmes majeurs n'est apparu qu'à la fin du projet. Pour des cas particuliers, nous devions libérer des places pour permettre à tout les clients d'avoir une place. Il fallait alors réaffecter des personnes possédant un voyage dans un autre, et ainsi permettre à une personne supplémentaire d'aller visiter l'espace. Nous avons aussi passé du temps à essayer d'optimiser la complexité du code. Nous avons choisi les structures qui nous semblaient les plus adaptées.

8 Conclusion

Afin de permettre à un maximum de nos clients d'aller visiter l'espace, nous avons créé un programme fonctionnel capable d'affecter l'ensemble des voyageurs à un voyage, tant qu'il reste des places disponibles. Pour répondre à notre problématique, nous avons séparé le projet en 3 secteurs. Nous avons choisit le secteur qui nous correspondait le mieux afin de travailler dans des conditions optimales. Nous nous sommes mis d'accord pour travailler dans le langage qui correspondait le mieux au groupe, et bien que nous ayons rencontré de nombreux problèmes, et ce à tous les stades du projet, nous les avons tous surmontés. Nous avons donc un programme permettant

d’affecter des personnes voulant visiter l’espace, selon certains souhaits et contre certaines contraintes, à des voyages spatiaux correspondant au mieux à leurs désirs.