Choix Techniques

Projet Informatique 1A

Equipe :
Delrue Pierre
Louet Thibault
Lumia Yann
Person Victor

15 avril 2016

Table des matières

1 Introduction

1.1 Introduction du sujet

Une agence de tourisme spatial propose différentes croisières. Le but de ce projet est de distribuer les différentes places disponibles selon le choix des clients en fonction de leur indice de priorité.

Chaque personne a deux souhaits choisis parmi des croisières pré-organisées plus un choix de croisière libre. Si les croisières pré-organisées ne sont plus disponibles, le choix libre sera attribué par défaut. Chaque souhait est constitué de 6 planètes choisies parmi six zones.

L'attribution des croisières (organisées ou libres) se fait d'abord pour les personnes avec une croisière pré-organisée en premier souhait indépendamment de la priorité. Par exemple, une personne avec une priorité de 190 pour une croisière libre sera traitée après une personne de priorité 100 mais dont le souhait est une croisière organisée.

Le programme final prend donc en entrée un tableau contenant le nom, prénom, la priorité et les souhaits de chaque personne, un tableau contenant les différentes croisières possibles et un tableau contenant les contraintes des planètes.

1.2 Conditions/Pré-requis

1. Tous les parcours pré-conçus respectent les contraintes ;
2. Les personnes associées à un parcours ne respectant pas les contraintes perdent toute priorité ;
3. Il est impossible pour 2 planètes distinctes d'une même zone d'être liées par des contraintes ;
4. Si un client fait, comme premier choix, un choix libre, alors son deuxième choix sera nécessairement libre lui aussi.

2 Structures de données

2.1 Structure de personne et structure de planete

Les structures de personne et de planete sont des structures mixtes ayant différents attributs. Pour personne, la structure contient le nom, le prénom, la priorité, les deux id des choix, les tableaux de choix pré-organisés, un tableau de choix libre, un tableau indiquant les zones assignées définitivement au voyageur et un entier qui indique quel croisière lui a finalement été attribué (1^{er} ou 2^{nd} voeu de parcours pré-organisé ou parcours libre). Pour planete, la structure contient le nom de la planète et le nombre de places restantes.

2.2 Structure de liste

La structure de liste, de part sa taille dynamique, permet un traitement de données de même type mais dans un nombre variable. Ce qui est impossible avec un tableau qui a une taille fixe. Dans notre cas, il y aura un type liste de personnes et un type liste de planètes.

Ici, on doit traiter des personnes qui pourront avoir une même priorité. Les stocker dans une liste permettra d'optimiser l'attribution des choix. Par exemple, si la première personne n'a pas ses choix attribués et que l'on est obligé de donner des planètes au hasard, il nous sera possible de traiter d'abord les autres personnes de même priorité pour ne pas bloquer un de leur choix en attribuant aléatoirement une planète à celui d'avant. De même, pour les planètes, il y a un nombre variable de planètes entre les zones d'où la nécéssité de les stocker dans des listes.

2.3 Structure de tas

La structure de tas s'est imposée comme structure pour le traitement des personnes en fonction de leur priorité. En effet, le prédicat de base de la structure, la clef du père est plus grande que la clef de chacun de ses fils, permet immédiatement de récupérer l'objet dont la clef est la plus grande.

Dans notre cas, la clef des noeuds sera la priorité de chaque personne et le noeud contiendra le nom, le prenom, deux entiers contenant l'id de chacun des deux souhaits formulés ainsi que trois tableaux qui pourront contenir 2 choix de croisière organisée et la croisière libre par défaut.

TABLE 1 – Tableau comparatif des complexités de différentes structures envisageables

	Liste chaînée ordonnée	Arbre binaire de recherche	Tas
Insertion	$O(n^2)$	$O(nlog(n))$	$O(log(n))$
Accès et retrait de la plus grande valeur	$O(1)$	$O(n)$	$O(log(n))$

3 Algorithme d'affectation

NB : dans la suite,
- partie 1 des contraintes désigne les destinations qui en empliquent une autre ;
- partie 2 des contraintes désigne les destinations impliquées par d'autres ;

Algorithm 1: Algorithme d'affectation en pseudo-code - partie 1/4

```
Données: Tableau de voyageurs TdV
Resultat: Tableau traîté
Initialisation;
A ← ∅;
parcours de TdV faire
    si choix1 ≠ libre alors
        Insérer dans A à indice(priorité + 1000);
    sinon
        Insérer dans A à indice(priorité);
Corps de l'algorithme;
L ← ∅;
parcours de A, dans l'ordre décroissant d'indice faire
    C ← suivant de A;
    si L = ∅ ou prioté du dernier élément = priorité(C) alors
        Insérer C dans L;
        Supprimer C de A;
    sinon
        parcours de L faire
            C ← suivant de L;
            si choix1 de C disponible alors
                Affecter C à la liste correspondant au choix1;
                Supprimer C de L;
        parcours de L faire
            C ← suivant de L;
            si choix2 de C disponible alors
                Affecter C à la liste correspondant au choix2;
                Supprimer C de L;
            sinon
                Affecter C à la liste de choix libre;
                Supprimer C de L;
        L ← ∅;
```

Algorithm 2: Algorithme d'affectation en pseudo-code - partie 2/4

```
30 pour chaque structure des voyages pré-organisée, A faire
31     pour i ← 0 à nbZones − 1 faire
32         N ← nbPlaces disponibles dans la zone i;
33         T ← ∅;
34         L ← ∅;
35         parcours de A, dans l'ordre décroissant d'indice faire
36             C ← suivant de A;
37             si L = ∅ prioté du dernier élément = priorité(C) alors
38                 Insérer C dans L;
39                 Supprimer C de A ;
40             sinon
41                 parcours de L, dans l'ordre décroissant d'indice faire
42                     C ← suivant de A;
43                     si choix de la zone i de C possible alors
44                         Affecter son choix zone i à sa destination zone i;
45                     sinon
46                         Insérer C dans T;
47                 L ← ∅;
48             parcours de T, dans l'ordre décroissant d'indice faire
49                 C ← suivant de T;
50                 Affecter son choix zone i à la 1^re destination zone i possible;
```

Algorithm 3: Algorithme d'affectation en pseudo-code - partie 3/4

```
51 L ← ∅, T1 ← ∅, T2 ← ∅;
52 parcours de la sutructure des parcours libre de A, dans l'ordre décroissant d'indice faire
53     C ← suivant de A;
54     si C ne respecte pas les contraintes alors
55         Insérer C dans T1;
56         Supprimer C de A;
57     sinon
58         pour chaque contrainte faire
59             P ← prochaine planète de partie 2 des contraintes;
60             Q ← planète impliquant P;
61             si P ∈ choix(C) et Q ∉ choix(C) et nbPlaces(Q) > nbPlaces(P) − 1 alors
62                 Insérer C dans T2;
63                 Supprimer C de A;
           // début allocation destinations
64         si C non inséré dans T2 alors
65             pour i ← 1 à nbZones − 1 faire
66                 si destination de zone i de C vide alors
67                     P ← planète zone i choisie par C;
68                     si P est dans partie 1 des contraintes alors
69                         Q ← planète impliquée par P;
70                         si P ou Q non libre alors
71                             Insérer C dans T2;
72                             Supprimer C de A;
73                         sinon
74                             Affecter P et Q à la destination de C;
75                     sinon si P est dans partie 2 des containtes alors
76                         Q ← planète impliquant P;
77                         si Q ∈ choix(C) alors
78                             Affecter P et Q à la destination de C;
79                         sinon si nbPlaces(P) − 1 < nbPlaces(Q) alors
80                             Insérer C dans T2;
81                             Supprimer C de A;
82                         sinon
83                             Affecter P à destination de C;
84                     sinon
85                         si P est libre alors
86                             Affecter P à destination de C;
87                         sinon
88                             Insérer C dans T2;
89                             Supprimer C de A;
```

Algorithm 4: Algorithme d'affectation en pseudo-code - partie 4/4

```
T ← ∅;
Insérer T_1 et T_2 dans T;
parcours de T, dans l'ordre décroissant d'indice faire
    C ← suivant de T;
    pour i ← 1 à nbZones − 1 faire
        si zone i pas déjà affectée alors
            si P libre alors
                si P n'a aucune contrainte alors
                    Affecter P à destination de C;
                sinon
                    si P dans Partie 1 des contraintes alors
                        Q ← planète impliquée par P;
                        si Q libre alors
                            si zone de Q non affectée ou choix(C) = Q alors
                                Affecter P et Q à destination de C;
                    sinon
                        Q ← planète impliquant P;
                        si nbPlaces(P) − 1 ⩾ nbPlaces(Q) alors
                            Affecter P à destination de C;
            si zone i pas déjà affectée alors
                parcours de autres planètes faire
                    P ← planète suivante;
                    si P libre et P ∉ partie 1 des contraintes alors
                        si P dans partie 2 des contraintes alors
                            Q ← planète impliquant P;
                            si nbPlaces(P) − 1 ⩾ nbplaces(Q) alors
                                Affecter P à destination de C;
                                fin parcours(autres planètes);
                            si choix(C) = Q alors
                                Affecter P et Q à destination de C;
                                fin parcours(autres planètes);
                        sinon
                            Affecter P à C;
                            fin parcours(autres planètes);
                    sinon si P libre et P ∈ partie 1 des contraintes alors
                        Q ← planète impliquée par P;
                        si Q libre et zone de Q non affectée alors
                            Affecter Q et P à destination de C;
                            fin parcours(autres planètes);
    * // insérer code page 8
Fin de l'Algorithme
```

Algorithm 5: (*)

```
129 pour chaque planète dans partie 1 des contraintes faire
130     P ← planète suivante;
131     Q ← planète impliquée par P;
132     si zone de Q non affectée à C alors
133         parcours de autres planètes de la zone de Q faire
134             R ← planète suivante;
135             si R libre alors
136                 Affecter R à destination de C;
```

Remarque : (*) permet de régler le cas où il n'y a plus assez de place autre part que dans une planète qui en implique une autre qui n'a pas été choisie par C (pas besoin de tester les contraintes sur R grâce à la condion 3 (cf 1.2. Conditions/Pré-requis)

4 Découpage en modules

4.1 conversion.h

```c
#ifndef CONVERSION_H_INCLUDED
#define CONVERSION_H_INCLUDED

#include "listePersonne.h"
#include "listePlanete.h"
#include "tas.h"

/* @requires : f1 et f2 deux flux sur les fichiers des personnes
   (f1 = flux sur le tableau de priorites et f2 = flux sur le tableau de
      souhaits)
   @assigns : nothing
   @ensures : retourne le tas ou sont ranges toutes les personnes par
      ordre de priorite */
tas convertir_t_personne(FILE * f1, FILE * f2);

/* @requires : fd un flux sur le fichier des zones
   @assigns : nothing
   @ensures : retourne le tableau de listes de planete */
liste_planete* convertir_t_planete(FILE * fd);

/* @requires : f un flux sur le fichier des contraintes, sizetab le
      nombre de lignes du fichier
   @assigns : nothing
   @ensures : retourne le tableau des contraintes */
char*** convertir_contrainte(FILE * f, int sizetab) ;

/* @requires : une personne et un flux sur un fichier
   @assigns : rajoute une ligne au fichier
   @ensures : la ligne a ete ajoutee */
void ajout_ligne(personne, FILE * ) ;

/* @requires : un flux de fichier
   @assigns : nothing
   @ensures : retourne le nombre de lignes du fichier */
int nombreligne(FILE * f) ;

#endif /* CONVERSION_H_INCLUDED */
```

4.2 listPlanete.h

```c
#ifndef LISTPLA_H
#define LISTPLA_H

typedef struct planete_base {
   char* nom; /* nom de la planete */
   int nbPlaces; /* nombre de places restantes */
} * planete;

typedef struct liste_planete_base {
   planete val;
   struct liste_planete_base * next;
} * liste_planete;

/* @requires : nothing
   @assigns : nothing
   @ensures : retourne une planete */
planete creer_planete();

/* @requires : nothing
   @assigns : nothing
   @ensures : retourne une liste de planete vide */
liste_planete creer_liste_planete();

/* @requires : une liste de planete valide
   @assigns : nothing
   @ensures : retourne 1 si la liste est vide et 0 sinon */
int liste_planete_vide(liste_planete);

/* @requires : un pointeur sur liste de planete valide et une planete
   valide, non presente dans la liste
   @assigns : la liste de planete
   @ensures : la planete a ete inseree en tete de liste */
void inserer_liste_planete(liste_planete*, planete);

/* @requires : tableau de liste_planete tabListPla valide, verifiant
      len(tabListPla) == tabSize
   @assigns : nothing
   @ensures : si namePla est le nom d'une planete contenue dans
      tabListPla, retourne le nombre de places restantes de cette planete
      ; sinon, retourne -1 */
int nb_places_planete(char* namePla, liste_planete* tabListPla, int
   tabSize);

#endif /* LISTPLA_H */
```

4.3 listPersonne.h

```c
#ifndef LISTPER_H
#define LISTPER_H

typedef struct personne_base {
   int priorite;
   char* nom;
   char* prenom;
   int id1; /* identifiant choix 1 */
   int id2; /* identifiant choix 2 */
   char* tabChxOrg1[6]; /* tableau correspondant au 1er choix de voyage
      pre-orgarnise de la personne */
   char* tabChxOrg2[6]; /* tableau correspondant a son 2nd voeu de voyage
       de type pre-organise */
   char* tabChxLib[6]; /* tableau contenant les choix de la personne dans
       le cas d'un parcours libre */
   int assigned[6]; /* tableau qui indique les zones assignees de maniere
       definitive au voyageur */
   int chxFin; /* entier qui indique le choix attribue a la fin a l'
      individu (1 : chxOrg1 ; 2 : chxOrg2 ; 3 : chxLib) */
} * personne;

typedef struct liste_personne_base {
   personne val;
   struct liste_personne_base * next;
} * liste_personne;

/* @requires : nothing
   @assigns : nothing
   @ensures : retourne une personne */
personne creer_personne();

/* @requires : nothing
   @assigns : nothing
   @ensures : retourne une liste de personne vide */
liste_personne creer_liste_personne();

/* @requires : une liste de personne valide
   @assigns : nothing
   @ensures : retourne 1 si la liste est vide et 0 sinon */
int liste_personne_vide(liste_personne);

/* @requires : pointeur sur liste de personne valide et une personne
   valide, non presente dans la liste
   @assigns : la liste de personne
   @ensures : la personne a ete inseree en tete de liste */
void inserer_liste_personne(liste_personne*, personne);

/* @requires : pointeur sur liste de personne et personne valides
   @assigns : la liste de personne *listPer
   @ensures : si per est presente dans *listPer, retire per de *listPer
      et retourne 0; sinon, retourne -1; */
int retirer_liste_personne(liste_personne*, personne);

#endif /* LISTPER_H */
```

4.4 tas.h

```
#ifndef TAS_H
#define TAS_H

#include "listePersonne.h"

typedef struct tas_base {
   personne* tableau;
   int prochain_vide;
} * tas;

/* @requires : nothing
   @assigns : nothing
   @ensures : retourne un tas vide */
tas creer_tas();

/* @requires : tas valide
   @assigns : nothing
   @ensures : retourne 1 si tas vide, 0 sinon */
int tas_vide(tas);

/* @requires : pointeur sur tas valide et persone valide, non presente
   dans le tas
   @assigns : le tas
   @ensures : personne inseree a la bonne place dans le tas */
void inserer_tas(tas*, personne);

/* @requires : pointeur sur tas valide
   @assigns : le tas
   @ensures : retirer personne en conservant nature de tas */
personne retirer_tas(tas*);

/* @requires : pointeur sur tas valide et indice valide
   @assigns : nothing
   @ensures : renvoi de la priorite de la personne concernee */
int personne_priorite(tas*, int);

/* @requires : pointeur sur tas valide
   @assigns : le tas
   @ensures : echange la racine avec le fils droit */
void echanger_racine_fd(tas*, int i);

/* @requires : pointeur sur tas valide
   @assigns : le tas
   @ensures : echange la racine avec le fils gauche */
void echanger_racine_fg(tas* t, int i);

/* @requires : pointeur sur tas valide
   @assigns : le tas
   @ensures : reequilibrer le tas apres retrait d'un element et conserver
       sa structure */
void reorganiser_tas_retirer(tas*);

/* @requires : indice valide
   @assigns : nothing
   @ensures :  la parite de ind (retourne 1 si impair, 0 sinon) */
int est_pair(int ind);
```

```
55
56 /* @requires : pointeur sur tas valide
57    @assigns : le tas
58    @ensures : reequilibrer le tas apres ajout d'un element et conserver
       sa structure */
59 void reorganiser_tas_inserer(tas*);
60
61 /* @requires : pointeur sur tas valide
62    @assigns : le tas
63    @ensures : liberer la zone memoire associee au tas */
64 void liberer_tas(tas*);
65
66 #endif /* TAS_H */
```

5 Choix du langage de programmation

Le langage utilisé pour le projet sera le C. Le projet sera découpé selon les différents modules évoqués plus haut. Néanmoins, il serait possible selon l'avancement du projet de créer une interface utilisateur dans une application web s'appuyant sur une base de données qui permettra à un utilisateur de modifier, supprimer, ajouter des choix valides à l'aide de la structure de base de données. Le programme en lui-même pourrait quant à lui récupérer les classes de cette base de données pour les traiter.