SAE 1.02 - Comparaison d'approche algorithmique

LABORIE Louis
LAURENT Alexis

Légende - A : Alexis a réaliser la fonction
L : Louis a réaliser la fonction

Partie 1 :

donné d'iut = Ville, Département, Nb places, Responsable (nom prénom)
Fichier binaire / Fichier texte (au choix)

ville : enregistré dans un tableau de pointeur vers des structures de type VilleIUT
Structure VilleIUT : - ville (char[30]), idDept (type ListeDept)
idDept : liste chainé des identifiants de départements rangé par ordre croissant
ListeDept : pointeur sur un maillon de type MaillonDept : -> si vide = NULL
MaillonDept : char departement[30], int nbPlaces, char nom[30] (nom + prénom), maillon *suiv (dernière = NULL)

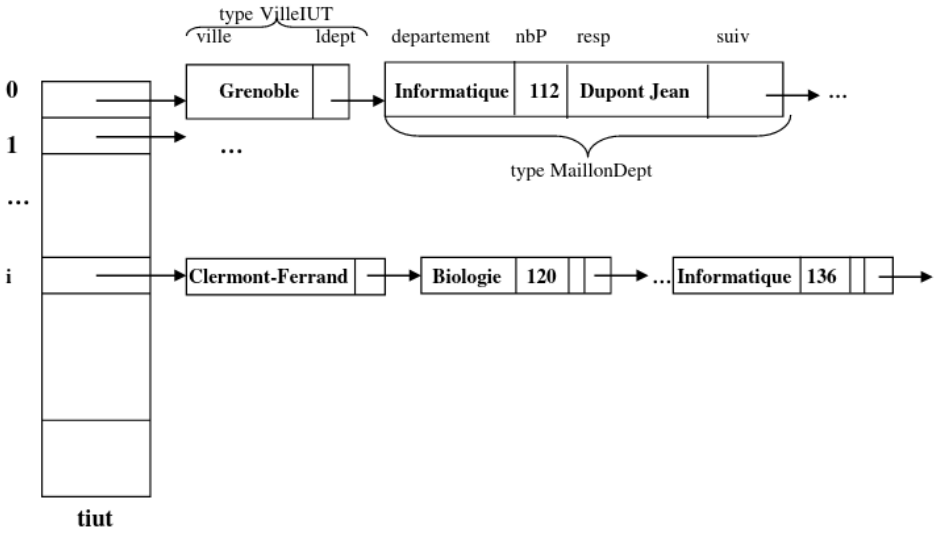

Figure 1

```c
typedef struct maillon
{
    char departement[30];
    int nbPers;
    char responsable[30];
    struct maillon *suiv;
}MaillonDept;

typedef MaillonDept* ListeDept;

typedef struct
{
    char ville[30];
    MaillonDept *idDept;
}VilleIUT;
```

Explications des fonctions :

Les données des villes sont stockées dans un fichier nommé “iut.txt”.

Les données sont stockées sous cette forme :

Clermont-Ferrand <= Ville où est située l’IUT
2 <= Nombre de départements
Informatique 130 Bouhours Cédric <= Département (nom, nombre de places, responsable)
Réseaux 5 Unmec Sympa
Lyon
2
Jeux-videos 24 Kojima Hideo
GEA 8 Marx Karl

```c
VilleIUT **chargementVille(char *nomFich, int tphys, int *tailleL) L
```

Cette fonction permet de charger les différentes villes dans un tableau de pointeurs et de retourner l’adresse de celui-ci.

```c
VilleIUT lireVille(FILE *flot) L
```

Crée un élément VilleIUT, lit les données dans le flux passé en paramètre et le retourne.

```c
ListeDept traiterDept(ListeDept l, FILE *flot) L
```

Traite les données concernant les départements du flux passé en paramètre.

```c
ListeDept insererDept(ListeDept l, char *dept, char *nom, int nbP) L
```

Crée un nouveau maillon en fonction des données passées en paramètre et l’insère dans la liste l en fonction du nom passée en paramètre.

```c
ListeDept ajouterEnTeteDept(ListeDept l, char *dept, char *nom, int
nbP) L
```

Crée un nouveau maillon en fonction des données passées en paramètre et l’insère en tête de la liste l passée en paramètre.

```c
ListeDept supprimerDept(ListeDept l, char *code) L
```

Supprime un département de la liste l en fonction du code passé en paramètre.

```
ListeDept supprimerEnTêteDept(ListeDept l) L
```

Supprime le département en tête de la liste l.

```
ListeDept rechercheDept(ListeDept l, char code[]) L
```

Recherche (en fonction de la longueur de la liste) si le département (code) est présent dans la liste l.

```
int longueurListe(ListeDept l) L
```

Retourne la longueur de la liste l.

```
ListeDept creerListeDept(void) L
```

Crée une liste vide en renvoyant la valeur NULL.

```
void affichageListeDept(ListeDept l) L
```

Affiche les départements présents dans la liste l.

```
void SauvegarderIUT(VilleIUT **tabV, int tailleL) L
```

Sauvegarde dans un fichier “res.txt” les données sur les différents IUT.

```
void SauvegarderListe(ListeDept l, FILE *flot) L
```

Sauvegarde les données des différents départements (fonction appelée par SauvegarderIUT).

```
int plusGrand(VilleIUT **tiut, int nbEle) L
```

Retourne la position du plus grand élément du tableau tiut.

```
void echanger(VilleIUT **tiut, int i, int j) L
```

Échange les valeurs des positions i et j dans le tableau tiut.

```
void triEchange(VilleIUT **tiut, int nbEle) L
```

Tri par échange le tableau tiut.

```
void modifChefDept(VilleIUT **tiut, int nbEle) L
```

Modifie le nom du chef d’un département.

```
void creationDept(VilleIUT **tiut, int nbEle) L
```

Permet de créer un département et de l'insérer dans la liste l.

```
void MiseAJourNbPlace(VilleIUT **tiut, int nbEle) L
```

Permet de mettre à jour le nombre de places d'un département d'un IUT donné.

```
int rechVille(VilleIUT **tiut, int nbEle, char code[], int *trouve) L
```

Permet de rechercher dichotomiquement un IUT dans le tableau tiut.

Nous trouvons également plusieurs fonctions destinées à afficher les différentes données :

- `void AffichageVille(VilleIUT **tiut, int nbEle)A`
- `void AffichageGlobalDepVille(VilleIUT **tiut, int nbEle)A`
- `void AffichageNbplaceDept(VilleIUT **tiut, int nbEle)A`
- `void AffichageNbplaceDeptVille(VilleIUT **tiut, int nbEle)A`
- `void AffichageDepVille(VilleIUT **tiut, int nbEle)A`
- `void AffichageDeptVille(VilleIUT **tiut, int nbEle)A`
- `void AffichageGlobalIUT(VilleIUT **tiut, int nbEle)A`

```
bool motdepasseVerif(void) A
```

permet de rentrer un mot de passe prédéfini.

```
bool verifQuit(void)A
```

Vérifie la sortie du programme.

```
bool verifSelection(void)A
```

Vérifie si l'utilisateur veut continuer de saisir dans une fonction.

```
bool confirmationSup(void)A
```

Vérifie la confirmation de la suppression d'un élément

```
int menuUtilisateurAffiche(void)A
```

affichage du menu de l'utilisateur et renvoie le choix

```
int menuAdminAffiche(void)A
```

affichage du menu de l'administrateur et renvoie le choix

```
void gestionMenu(VilleIUT **tiut, int nbEle, Candidat **tcandid, int *nbCandid, int
*taillePCandid)A
```

Gestion du menu de l'utilisateur

```
int gestionMenuAdmin(VilleIUT **tiut, int nbEle)A
```

Gestion du menu de l'administrateur

```
void clearpage(void)A
```

Demander de presser entrer pour effacer l'écan.

Partie 2 :

```
typedef struct
{
    char departement[30];
    int decisionDept;
    int decisionCandid;
}ChoixDept;

typedef struct maillonEtu
{
    char iutCandid[30];
    int nbChoix;
    ChoixDept **tabDept;
    struct maillonEtu *suiv;
}MaillonCandid, *ListeCandid;

typedef struct
{
    int id;
    char nom[30];
    char prenom[30];
    float note[4];
    MaillonCandid *idCandIUT;
}Candidat;
```

```
ListeCandid ListeCandidvide(void)A
```

Créer une listeCandid vide.

```
ListeCandid insererEnTeteCandid(ListeCandid l, char
*ville, ChoixDept **tDept, int nbEle)A
```

insertion d'une ville dans une listeCandid "l" en première position.

```
ListeCandid insererCandid(ListeCandid l, char *ville,
ChoixDept **tDept, int nbEle)A
```

Insertion d'une ville dans une listeCandid "l".

```
int rechCandiDept(ChoixDept **tDept, int tailleL, char
dept[], int *trouve)A
```

renvoie la position du département dans le tableau tDept et utilise le pointeur de trouve, 0 si il n'y est pas sinon 1.

```
void triEchangeCandidDept(ChoixDept **tDept, int tailleL)A
```

Fonction utilisée en tant que prévention car dans le chargement, le tableau est déjà trié.

```
void videTabDept(ChoixDept **tDept, int tailleP)A
```

Vide tout le tableau d'un maillon d'une ListeCandid.

```
ListeCandid suppressionEnTeteCandid(ListeCandid l , ChoixDept **tDept, int
tailleP)A
```

Supprime le premier maillon de la ListeCandid.

```
ListeCandid suppressionCandid(ListeCandid l, char *ville, ChoixDept **tDept, int
tailleP)A
```

Supprime un maillon de la ListeCandid en indiquant sa ville.

```
void afficherCandid(Candidat **tab, int nbEle)A
```

Affiche toutes les candidatures avec toutes les informations.

```
void afficherCandidIUT(ListeCandid l)A
```

Affiche tous les information des maillons de la chaîne l.

```
void afficherCandidDept(ChoixDept **tab, int nbChoix)A
```

Affiche toutes les infos des département du tableau tab.

```
int longueur(ListeCandid l)A
```

Renvoie la longueur de la ListeCandid l.

```
Candidat **chargmentCandid(char *nomFich, int *tphys, int *tailleL)A
```

Charge le fichier dans un tableau de pointeur Candidat.

```
Candidat lireCandidat(FILE *flot)A
```

Lis les informations personnelles d'un candidat dans le fichier.

```
ListeCandid traiterCandidIUT(ListeCandid l, FILE *flot)A
ChoixDept *traiterCandidDept(int tailleL, int tailleP, FILE *flot)A
```

Lis toutes les informations des candidatures du candidat.

```
int rechCandid(Candidat **tab, int nbEle, int id, int *trouve)A
```

Renvoie la position du candidat recherché par son identifiant.

```
void triEchangecandid(Candidat **tab, int nbEle)A
```

Trie par échange le tableau de Candidats.

```
ListeCandid rechCandidIUT(ListeCandid l, char *code)A
```

Recherche dans la liste des candidatures d'IUT d'un candidat et renvoie le maillon correspondant ou NULL si pas trouvé.

```
void supprimerCandid(Candidat **tab, int *nbEle)A
```

Supprime un candidat précisé en paramètre.

```
Candidat **reallocTCandid(Candidat **tab, int *tailleP)A
```

Aggrandit le tableau si il est plein.

```
void creationCandid(VilleIUT **tiut, int nbIUT, Candidat **tcandid, int *nbCandid,
int *taillePCandid)A
```

Crée une candidature avec les informations personnels saisies, et utilise la fonction modifCandid pour faire ses vœux directement.

```
int menuCandid(void)A
```

Affiche le menu des Candidats, renvoie le choix.

```
void afficher1Candid(Candidat **tab, int nbCandid)A
```

Affiche les informations d'un Candidat choisi.

```
void gestionCandid(VilleIUT **tiut, int nbIUT, Candidat **tcandid, int *nbCandid,
int *taillePCandid)A
```

Gestion du menu des Candidats.

```
int menuGestionCandid(void)A
```

Affiche le menu pour modifier ses candidatures.

```
ListeCandid recherchecandidIUT(ListeCandid l, char code[])A
```

renvoie le maillon si le code d'une ville est dans la liste de choix du candidat sinon renvoie NULL.

```
int recherchecandidDept(ChoixDept **tab,int nbEle, char code[],int *trouve)A
```

renvoie la position si le code département est dans le tableaux de choix du candidat sinon trouve est égal à 1 si trouvé sinon 0.

```
ChoixDept RempTabCandid(char dept[],int deciCandid ,int deciDept)A
```

Remplit une candidature dans un département.

```
void ajouterDept(Candidat **tcandid, int nbCandid, VilleIUT **tiut, int nbIUT, int
iCandid)A
```

Faire des vœux d'un département dans un IUT.

```
void supprimerCandidDept(Candidat **tcandid, int nbCandid, VilleIUT **tiut, int
nbIUT, int iCandid)A
```

Supprime une candidature dans un département d'un IUT.

```
void modifCandid(Candidat **tcandid, int nbCandid, VilleIUT **tiut, int nbIUT, int
iCandid)A
```

permet au candidats de gérer ses voeux (faire un voeux ou le supprimer).

```
void decalageGauche(Candidat **tab,int pos, int nbEle) L
```

Fonction qui permet de décaler tous les éléments du tableau tab à gauche à partir de la position passée en paramètre.

```
void decalageGaucheDept(ChoixDept **tab, int pos, int nbEle) L
```

Même principe que pour la fonction précédente mais pour un tableau d'éléments différents.

```
void SauvegardeCandid(Candidat **tabC, int nbCandid)A
```

Sauvegarde le tableau Candidat dans un fichier txt.

Partie 3 :

```
int verifChefDepart(VilleIUT **tiut, int nbEle, char *dept, char
*ville, char *nom) L
```

Vérifie que le nom passé en paramètre est bien le même que celui du responsable du département passé en paramètre.

```
int menuResponsableAffiche(void) L
```

Affiche le menu avec les options disponibles pour le responsable.

```
int gestionResponsable(VilleIUT **tiut, int nbEle, Candidat **tcandid,
int tailleL) L
```

Permet d'appeler les différentes fonctions du responsable.

```
int traiterCandidIUTDept(Candidat *candid, char *dept, char *ville) L
```

Permet de vérifier qu'un candidat a postulé à un département donné. Si oui, la fonction retourne 1, 0 sinon.

```
Candidat** candidDept(Candidat** tabCandidat, char* dept, char* ville,
int tailleL, int* nb) L
```

Crée un tableau de pointeurs sur des candidats qui ont postulé à un département d'un IUT donné. L'adresse du tableau est retournée par la fonction.

```
void SauvegardeCandidAdmis(Candidat **tab, int nb, int admis) L
```

Sauvegarde les données des candidats en fonction de la valeur de la variable admis.
Si elle est égale à 0, la fonction sauvegarde dans un fichier "candidAdmis.txt" les candidats présents dans le tableau tab passé en paramètre.
Sinon, la fonction sauvegarde dans un fichier "candidMEA.txt" les candidats présents dans le tableau tab passé en paramètre.

```
void examinerCandid(Candidat **tabCandid, int nb, char* dept, char*
ville) L
```

Permet de déduire, à partir du tableau de pointeurs sur des candidats tabCandid, si les candidats sont admis, mis sur liste d'attente ou refusé.

Partie 4 :

Cette partie à été commencée mais nous n'avons pas réussi à la terminer à temps.

```
void afficherAdmiCandid(Candidat **tab,int nbCandid)A
```

Afficher en fonction d'un identifiant donné les états de ses candidatures.

```
void afficher1AdmiCandid(Candidat **tab,int nbCandid, int id, int
pos);A
```

Affiche en fonction d'un identifiant donné en paramètre les états de ses candidatures.

```
void rejeterVoeux(ListeCandid l, char ville[]);A
```

Rejette tout les voeux ou le candidat n'a pas encore traité la réponse (decisionCandid = 0) utilisé quand un candidat accepte un voeux

```
int decision(void);A
```

affiche le menu qui demande à l'utilisateur de faire un choix (accepter, refuser, ou rien).

```
void gererAdmiCandid(Candidat **tab, int nbCandid);A
```

L'utilisateur gère ses vœux en fonction du traitement de ses candidatures par les Département.

Types de structures choisis :

Partie 1 :

Pour les types de structure de la première partie, nous avons suivi le schéma présent au début du sujet. Nous avons donc utilisé un tableau de pointeurs sur des structures VilleIUT et une liste chaînée pour les départements liés à un IUT.

Partie 2 :

Pour les types de structure de la deuxième partie, nous avons décidé d'utiliser un tableau de pointeurs sur des structures Candidat afin de pouvoir trier les données (en fonction de l'identifiant) plus facilement. Nous avons choisi d'utiliser une liste chaînée pour les différentes candidatures des candidats plus précisément la ville où se trouve l'IUT et le nombre de voeux faite dans cette IUT et donc cela nous mène au Tableau de pointeur sur des structures ChoixDept qui correspond au différend Département et décision prise par le département et le candidat dans l'IUT choisie, ce tableau comme celui des Candidat sont alloué dynamiquement.

Partie 4 :

Nous envisageons d'utiliser un fils pour stocker les candidatures en attente (2 en décision de département) et les élèves acceptant leur voeux dans les quels il seront pris dans des tableau dynamique comprenant leur identifiant, nom prénom. Tout ceci dans la structure des départements.

Comparaisons algorithmiques :

Pour trier les données des tableaux, nous avons utilisé le tri échange. Cet algorithme nous a semblé le plus simple à comprendre et le plus simple à utiliser. Malgré cela, cet algorithme est extrêmement lent lorsque le nombre de données à traiter augmente (peut mettre

plusieurs heures). Pour cela il faut utiliser des méthodes de tri plus efficaces comme le tri fusion ou le quick sort.

Pour les fonctions de recherche, nous avons utilisé plusieurs méthodes comme la recherche dichotomique ou une recherche plus basique (on parcourt toutes les valeurs du tableau ou de la liste et on les compare avec les valeurs à rechercher). La recherche dichotomique s'avère bien plus efficace que la recherche basique : la recherche basique dépend du nombre d'éléments (n) stockés dans le tableau (minimum n fois) et la recherche dichotomique (complexité : log2(n)).