TNE1 & TP5 : Tris & Recherches

Préambule important

Le travail demandé dans ce document est à commencer pendant le TNE1 du 20/11/2018 (d'une durée de 2h), et à finir pendant le TP5 de la même semaine.

Seule la 1° partie est à réaliser pour le TNE1, et donnera lieu à un dépôt à la fin de la séance. Ce travail sera poursuivi lors de la séance TP avec un deuxième dépôt.

> **Note 1 : N'oubliez qu'il y a horodatage !**
> **Note 2 :** Ce travail sera poursuivi par un autre travail lors du **TNE2** du 26/11/2018**.**

1 Introduction

L'objectif de ces travaux pratiques est la mise en œuvre et l'analyse expérimentale de quelques méthodes de recherche et de tri dans une table de données de structures. Le critère de tri (clé) sera un attribut de la structure.

Les données à traiter ici sont des structures contenant des informations relatives à la population des villes du département « Nord » où chaque ville est caractérisée par le n-uplet **(nom de la commune, code de la commune, population totale, population masculine, population féminine)** :

```
typedef struct
{
    char     nomVille[MAXNOM+1];
    int      codeCommune;
    int      populationTotale;
    int      populationMasculine;
    int      populationMasculine;
} villePop_t;
```

Ces informations proviennent d'un fichier qui est fourni, pour lequel la fonction de lecture et de chargement dans une table est également fournie.

```
villePop_t villesPop[NBMAXVILLES] = {{""}};// Table des villes
unsigned int     nbVilles;                  // Nombre de villes*/
char    fileName[] = FILENAME;              // Nom du fichier des villes

printf("Chargement du fichier \"%s\" ... ", fileName);
nbVilles = chargerVillesPop(fileName, villesPop);
if (nbVilles == 0) return EXIT_FAILURE;
printf("effectue.\n%d ville(s) chargée(s) dans la table des villes\n\n",
                  nbVilles);
```

Il conviendra de comparer les résultats expérimentaux obtenus lors de cette étude, à ceux obtenus en cours/TD. A cet effet, les comparaisons ainsi que les affectations impliquant au moins un élément de la table des villes seront comptabilisées et seulement celles-ci.

Ce comptage sera mémorisé dans la variable globale `mesures` définie comme suit :

```
struct
{
    unsigned long nbComparaisons;
    unsigned long nbAffectations;
} mesures = {0L, 0L};
```

Cette variable devra être réinitialisée à zéro avant chaque opération de recherche ou de tri, et sa valeur sera affichée après chaque recherche ou tri effectué, comme dans l'exemple ci-dessous :

```
printf("Test du tri fusion par ordre décroissant de la population féminine\n");
memset(&mesures, 0, sizeof mesures);  // RàZ de la variable mesures
triFusionParPopulationFeminine(villesPop);
// Affichage du contenu de la table des villes ... puis affichage du resultat
affichervillesPop(villesPop, nbVilles);
printf("\nNombre de comparaison(s) = %lu\nNombre d'affectation(s)  =
            %lu\n",      mesures.nbComparaisons,   mesures.nbAffectations);
```

L'affichage du contenu de la table des villes ou d'une partie de son contenu sera réalisé à l'aide de la fonction `affichervillesPop()`.

Ainsi l'appel suivant :

```
printf("\nAffichage des 3 dernieres villes\n");
afficherVillesPop(&villesPop[nbVilles - 3], 3);
```

produira un résultat comparable à celui-ci :

```
Affichage des 3 dernieres villes
        +--------------------------+--------+---------+---------+---------+
        |       NOM                |CODE COM| POP_FEM | POP_MAS | POP_TOT |
        +--------------------------+--------+---------+---------+---------+
        | ZERMEZEELE               |  59667 |     103 |      91 |     194 |
        | ZUYDCOOTE                |  59668 |     801 |     868 |    1669 |
        | ZUYTPEENE                |  59669 |     289 |     268 |     557 |
        +--------------------------+--------+---------+---------+---------+
```

2 Partie 1 : Méthodes de recherche

2.1 Recherche linéaire dans une table

Question n°1

Réaliser et tester la fonction chargée de retourner l'indice dans la table des villes où est rangée une ville dont le code commune est fourni comme paramètre :

```
int rechercheSeqParCommune (const villePop_t VP[], int n, int codeCommune)
```

Le paramètre *n* représente le nombre de villes rangées dans la table. Cette fonction retournera -1 si la ville dont « code commune » vaut *codeCommune* est absente de la table *VP* des villes.

Les tests pourront par exemple, être effectués à l'aide d'une fonction comparable à celle-ci :

```
void rechercheSeqParCommune_test (const villePop_t VP[], int n)
{
    int indice , commune;

    printf("Entrez le(s) code(s) commune(s) de(s) ville(s) recherchees (. pour arreter)\n");
    while (1)
    {
        printf("Quel code commune ? ");
        if (scanf("%d", &commune) != 1 || commune == 0) return;

        memset(&mesures, 0, sizeof mesures);
        indice = rechercheSeqParCommune (VP, n, commune);
        if (indice == -1)
            printf("La commune n%c %05d est inconnue\n", SYMBOLE_DEGRE, commune);
        else
            affichervillesPop(&VP[indice], 1);
        printf("Nombre de comparaison(s) = %lu\n", mesures.nbComparaisons);
    }
}
```

Question n°2

Rappeler ou établir les résultats théoriques donnant :

- le nombre minimal de comparaisons,
- le nombre maximal de comparaisons,

à effectuer pour la recherche d'un élément dans une table de *n* éléments.

Question n°3

Combien de comparaisons ont été nécessaires pour rechercher la présence ou l'absence des communes pour les codes de commune suivants :

```
59016
59102
59346
59669
59095
```

Ces résultats sont-ils conformes à ceux rappelés à la question précédente.

2.2 Recherche dichotomique dans une table ordonnée

Le **fichier initial étant rangé par code de commune selon un ordre croissant**, la table des villes est donc également rangée par ordre croissant des codes de communes.

Question n°4

Réaliser la version récursive de la fonction de recherche dichotomique retournant l'indice dans la table des villes d'une commune dont le *code de la commune* est fourni en paramètre :

```
int rechercheRecParCommune (const villePop_t VP[], int debut, int fin,
                            int codeCommune);
```

Question n°5

Rappeler quels sont le nombre minimal et le nombre maximal de comparaisons nécessaires pour la recherche dichotomique d'un élément dans un ensemble de n éléments.

Question n°6

Inclure ici une capture d'écran présentant les résultats, y compris les nombres de comparaisons, des recherches des communes dont les codes sont les suivants :

```
12238
12150
12099
12010
12929
```

Ces nombres de comparaisons sont-ils conformes à ceux rappelés à la question précédente.

2.3 Recherche simultanée du minimum et du maximum

Il s'agit ici le rechercher simultanément l'indice de la ville ayant la plus petite population masculine et l'indice de la ville ayant la plus grande population masculine , en mettant en œuvre une fonction dont l'efficacité exprimée en nombre de comparaisons, soit meilleur que 2(*n*-1).

Question n°7

Développer la fonction dont le prototype est le suivant :

```
void MinMaxPopMas (const const villePop_t VP[], int n,
                   int *pMin, int *pMax);
```

Le test sera par exemple réalisé à l'aide de d'une fonction `MinMaxPopMas_test()` mise à disposition (voir fichier source fourni).

Question n°8

Rappeler ou établir le nombre de comparaisons nécessaires pour obtenir simultanément le minimum et le maximum d'un ensemble de *n* éléments.

Question n°9

Inclure ici une capture d'écran, présentant les résultats obtenus. Sont-ils conformes à ceux de la question précédente.

3 Partie 2 : Méthodes basiques de tri

3.1 Tri par sélection

La série de tests présentée ci-dessous sera réalisée à partir de l'état initial de la table des villes - **les villes sont initialement rangées selon l'ordre présent dans le fichier de départ** -.

Question n°10

Réaliser la fonction permettant d'effectuer le tri par sélection selon l'**ordre croissant de la population totale** (féminine et masculine) des villes.

```
void triSelectionParPopTotale (villePop_t VP[], int n);
```

Cette fonction doit également effectuer le comptage des comparaisons et des affectations mettant en jeu au moins un élément de la table *VP*.

Question n°11

Rappeler ou établir les coûts en nombre de comparaisons et en nombre d'affectations nécessaires pour trier un ensemble de *n* éléments, dans le cas le plus favorable et le cas le plus défavorable.

Question n°12

Effectuer le tri des 8 premières villes de la table des villes initiales, puis afficher ces 8 premières villes. Inclure une capture d'écran présentant la liste ordonnée de ces villes ainsi que les nombres de comparaisons et d'affectations effectuées. Ces nombres sont-ils conformes à ceux de la question précédente.

Question n°13

Relancer à nouveau le tri (sur la table qui est donc déjà triée) des 8 premières villes. Que deviennent alors les nombres de comparaisons et d'affectations.

Question n°14

Effectuer maintenant le tri de l'ensemble de la table des villes et indiquer quels sont les nombres de comparaisons et d'affectations obtenus.

3.2 Tri par insertion

La série de tests présentée ci-dessous sera réalisée à partir de l'état initial de la table des villes - **les villes sont initialement rangées selon l'ordre présent dans le fichier de départ** -.

Question n°15

Réaliser la fonction permettant d'effectuer le tri par insertion selon l'**ordre alphabétique des noms de communes**.

```
void triInsertionParNom (villePop_t VP[], int n);
```

Cette fonction effectue également le comptage des comparaisons et des affectations mettant en jeu au moins un élément de la table *VP*.

Question n°16

Rappeler ou établir les coûts en nombre de comparaisons et en nombre d'affectations nécessaires pour trier un ensemble de *n* éléments, dans le cas le plus favorable et le cas le plus défavorable.

Question n°17

Effectuer le tri des 8 dernières villes de la table des villes initiales, puis afficher ces villes. Inclure une capture d'écran présentant la liste ordonnée de ces 8 villes ainsi que les nombres de comparaisons et d'affectations effectuées. Ces nombres sont-ils conformes à ceux de la question précédente.

Question n°18

Relancer à nouveau le tri de ces 8 villes. Que deviennent alors les nombres de comparaisons et d'affectations.

Question n°19

Effectuer maintenant le tri de l'ensemble de la table des villes et indiquer quels sont les nombres de comparaisons et d'affectations obtenus.

4 Partie 3 : Méthodes avancées de tri

4.1 Tri par fusion

La série de tests présentée ci-dessous sera réalisée à partir de l'état initial de la table des villes - **les villes sont initialement rangées selon l'ordre présent dans le fichier de départ** -

Question n°20

Réaliser la fonction permettant d'effectuer le tri par fusion selon l'**ordre décroissant des populations féminines**, en vous basant sur les informations ci-dessous ainsi que celles présentées lors du cours/TD :

```
Void triFusionParPopFem(villePop_t VP[], int n)
{
    triFusion_PopFem (VP, 0, n - 1);
}
```

La fonction ci-dessous correspond au nom près, à celle du poly du TD.

```
void triFusion_PopFem (villePop_t VP[], int debut, int fin)
{
    int milieu;

    if (debut < fin)
    {
        milieu = (debut + fin) /2;
        triFusion_PopFem (VP, debut, milieu);
        triFusion_PopFem (VP, milieu + 1, fin);
        fusionnner_PopFem (VP, debut, milieu, fin);
    }
}
```

La fonction `fusionnner_PopFem()` correspond à la fonction `fusionner()` du poly cité plus haut. C'est dans cette fonction que sera effectué le comptage des comparaisons et des affectations mettant en jeu au moins un élément de la table *VP*.

Question n°21

Rappeler ou établir les coûts en nombre de comparaisons et en nombre d'affectations nécessaires pour trier un ensemble de *n* éléments, dans le cas le plus favorable et le cas le plus défavorable.

Question n°22

Effectuer le tri des 10 villes de la table des villes initiales rangées à partir de l'indice 500, puis afficher ces 10 villes classées par code postal croissant. Inclure une capture d'écran présentant la liste de ces 10 villes ainsi que les nombres de comparaisons et d'affectations effectuées. Ces nombres sont-ils conformes à ceux de la question précédente.

Question n°23

Relancer à nouveau le tri des 10 villes de la question précédente. Que deviennent alors les nombres de comparaisons et d'affectations.

Question n°24

Effectuer maintenant le tri de l'ensemble de la table des villes et indiquer quels sont les nombres de comparaisons et d'affectations obtenus.

Question n°25

Peut-on qualifier le tri par fusion de :

- méthode de tri « en place » ;
- méthode de tri « stable ».

Justifiez vos réponses.

4.2 Tri rapide

La série de tests présentée ci-dessous sera réalisée à partir de l'état initial de la table des villes - **les villes sont initialement rangées selon l'ordre présent dans le fichier de départ** -

Question n°26

Réaliser la fonction permettant d'effectuer le tri rapide selon l'**ordre croissant des populations masculines**.

Question n°27

Rappeler ou établir les coûts en nombre de comparaisons et en nombre d'affectations nécessaires pour trier un ensemble de *n* éléments, dans le cas le plus favorable et le cas le plus défavorable.

Question n°28

Effectuer le tri des 12 villes de la table des villes initiales rangées à partir de l'indice 300, puis afficher ces 12 villes classées par code postal croissant. Inclure une capture d'écran présentant la liste de ces 12 villes ainsi que les nombres de comparaisons et d'affectations effectuées. Ces nombres sont-ils conformes à ceux de la question précédente.

Question n°29

Relancer à nouveau le tri des 12 villes de la question précédente. Que deviennent alors les nombres de comparaisons et d'affectations.

Question n°30

Effectuer maintenant le tri de l'ensemble de la table des villes et indiquer quels sont les nombres de comparaisons et d'affectations obtenus.