(M3103) Algorithmique avancée

Le 4 octobre 2019 durée : 1 heure et 30 minutes

l' A. Casali alain.casali@univ-amu.fr
Aix Marseille Université
I.U.T. d'Aix en Provence - Département Informatique

Institut Universitaire de Technologie
Aix*Marseille Université
Département Informatique, Aix

Preambule

Documents autorisés : une feuille simple A4 manuscrite recto / verso.
Remarques :

- Lire attentivement tout le sujet, avant de commencer. Le barème est sur 42 points ;
- Tout au long de ce test, donnez de la sémantique à vos variables ! Toute variable n'ayant pas la sémantique adaptée sera considérée comme fausse, et par conséquent le morceau de code dans laquelle elle est présente aussi ;
- Une des réponse possible à l'avant dernière réponse est donnée en bas d'une des pages de ce test ;
- Toutes les fonctions écrites ou utilisées en cours ou en TD sont utilisables dans l'état ;
- Écriture au crayon de papier interdite.
- Pour cet examen, la classe `Clist` a été enrichie avec les accesseurs suivants :

```
const std::shared_ptr<CNode<T>> & fictionalHead() const;
const std::shared_ptr<CNode<T>> & fictionalTail() const;
```

Le but de cet exercice est de proposer des représentations de **matrices carrées creuses** (matrices pour lesquelles il y a très peu d'éléments différents de 0, *cf.* Table 1). Pour notre exercice, les données stockées seront des entiers.

1	0	0	0	0	0	0	0	0	**2**
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	**-1**	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	**9**

TABLE 1 – Un exemple

L'idée sous-jacente est de ne stocker que les informations nécessaires selon la structure de données utilisées. Autrement dit, on se demande si on peut omettre les `0`. Quel est l'impacte sur l'occupation mémoire et temps des fonctions membres ?

Nous allons comparer les structures de données suivantes :

1. ```
   typedef std::vector<int> VInt;
   typedef std::vector<VInt> matrix;
   ```

   Une représentation classique d'une matrice

2. ```
   struct triplet {
       size_t lineNum;
       size_t colNum;
       int val;
   };
   class cmatrix2
   {
   private:
       CList<triplet> matrix;
       size_t _size ();
   ```

```
public:
    cmatrix2 (const std::string & fileName);
};
```

On stocke [1], dans une liste doublement chainée, uniquement les triplets (*numéro de ligne, numéro de colonne, valeur de la matrice*) pour lesquels la valeur de la matrice est différente de `0`. Les éléments sont stockés dans l'ordre des lignes, puis celui des colonnes.

Le constructeur remplit la structure de données à partir du chemin vers un fichier (comme par exemple celui de la Table 1).

3.
```
typedef std::pair<size_t, int> pairColVal;
class cmatrix3
{
private:
    std::vector<CList<pairColVal>> matrix;
public:
    cmatrix3 (const std::string & fileName);
};
```

Chaque ligne est constituée de la liste des couples (*numéro de colonne, valeur de la matrice*) pour lesquels la valeur de la colonne est différente de `0`, *i.e.* nous avons un index hiérarchique de premier niveau (ou une table de hachage).

Le constructeur remplit la structure de données à partir du chemin vers un fichier (comme par exemple celui de la Table 1).

Exercice 1.

Question 1.1 (2 points) :

Dessiner la structure de données après l'appel au constructeur de `cmatrix2` pour le fichier exemple (*cf.* Table 1).

Question 1.2 (2 points) :

Dessiner la structure de données après l'appel au constructeur de `cmatrix3` pour le fichier exemple (*cf.* Table 1).

Pour les questions suivantes, on rappelle que l'occupation mémoire d'un `int` ou d'un `shared_ptr` est de 8 octets, et celle d'un `size_t` est de 16 octets sur des architectures 64 bits.

Question 1.3 (2 points) :

Quelle est l'occupation mémoire, pour ces trois structures de données, sur cet exemple.

Question 1.4 (1 point) :

Quelle est l'occupation mémoire, pour ces trois structures de données, si :

1. la taille de la matrice est de 100 ;
2. la Table 1 montre les 10 premières lignes (colonnes), le reste des éléments étant à 0.

Question 1.5 (1 point) :

Quelle est l'occupation mémoire, pour ces trois structures de données, si :

1. la taille de la matrice est de 1000 ;
2. la Table 1 montre les 10 premières lignes (colonnes), le reste des éléments étant à 0.

Chaque des classes ci-dessus sont munies des fonctions membres ci-dessous :

```
VInt getVectorAtCol (const unsigned & colNum) const;
VInt getVectorAtLine (const unsigned & lineNum) const;
int  getValue (const unsigned & lineNum, const unsigned & colNum) const;
```

---

1. `CList<std::pair<unsigned, CList<pairColVal>>>`.

Exercice 2.

Question 2.1 (4 points) :
Ecrire le corps de la fonction membre `getVectorAtLine ()` pour la classe `cmatrix2`.

Question 2.2 (4 points) :
Ecrire le corps de la fonction membre `getVectorAtCol ()` pour la classe `cmatrix2`.

Question 2.3 (4 points) :
Ecrire le corps de la fonction membre `getValue ()` pour la classe `cmatrix2`.

Question 2.4 (2 points) :
Pourquoi doit-on stocker la donnée membre `_size` pour la classe `cmatrix2`.

Exercice 3.

Question 3.1 (4 points) :
Ecrire le corps de la fonction membre `getVectorAtLine ()` pour la classe `cmatrix3`.

Question 3.2 (4 points) :
Ecrire le corps de la fonction membre `getVectorAtCol ()` pour la classe `cmatrix3`.

Question 3.3 (4 points) :
Ecrire le corps de la fonction membre `getValue ()` pour la classe `cmatrix3`.

Exercice 4.

Question 4.1 (1 point) :
En considérant la Table 1, quelle est la structure de données qui fait le moins d'opérations lors d'un appel à la fonction membre `getVectorAtLine ()` si la valeur du paramètre est `0` ? Même question si la valeur du paramètre est `1`

Question 4.2 (1 point) :
Justifier votre réponse.

Question 4.3 (1 point) :
En considérant la Table 1, quelle est la structure de données qui fait le moins d'opérations lors d'un appel à la fonction membre `getVectorAtCol ()` si la valeur du paramètre est `8` ? Même question si la valeur du paramètre est `9`.

Question 4.4 (1 point) :
Justifier votre réponse.

Question 4.5 (1 point) :
En considérant la Table 1, quelle est la structure de données qui fait le moins d'opérations lors d'un appel à la fonction membre `getValue ()` si les valeurs des paramètres sont `0, 0` ? Même question si les valeurs des paramètre sont `1, 1`. Même question si les valeurs des paramètre sont `9, 9`.

Question 4.6 (1 point) :
Justifier votre réponse.

Question 4.7 (1 point) :
Quelle autre structure de données aurait-on pût utiliser ?

Question 4.8 (1 point) :
Justifier votre réponse.