Algorithmique et structures de données

Colle – 1h30

mardi 11 mars 2014

Tout document est interdit. Tout appareil électronique, incluant téléphone portable et ordinateur portable, est interdit. Le barème est donné à titre indicatif

Exercice 1 : Questions de cours (5 points)

Répondez aux questions suivantes en étant très précis et concis.

Question 1.1

1. Qu'est-ce qu'un algorithme ? Qu'est-ce qu'un programme ?
2. Classer ces complexités de la plus petite à la plus grande :
   n^{10}, 10^n, $\log(10^n)$, $n \log n$, $3^{\ln n}$

Question 1.2

1. Énoncer le théorème Diviser pour Régner.
2. Énoncer le théorème Diviser pour Régner dans le cas où $f(n) = O(n)$.

Question 1.3

1. Qu'est-ce qu'une pile ?
2. Qu'est-ce qu'une file ?

Question 1.4

1. Quel est le principe du tri par insertion ? Donner, sans les justifier, les complexités en moyenne et en pire cas, en nombre de comparaisons. Dans quel cas atteint-on la complexité en pire cas ?
2. Quel est le principe du tri rapide ? Donner, sans les justifier, les complexités en moyenne et en pire cas, en nombre de comparaisons. Dans quel cas atteint-on la complexité en pire cas ?

Exercice 2 : Calcul de complexité (6 points)

On considère les fonctions suivantes :

```
int f(int n) {
  int x = 0;
  int i, j;
  for (i = 1; i < n; i++) {
    for (j = i; j < i + 5; j++) {
      x += j;
    }
  }
  return x;
}

int g(int n) {
  int y = 0;
  int i;
  for (i = 0; i < f(n); i++) {
    y += i;
  }
  return y;
}
```

Question 2.1 Quelle est l'opération fondamentale dans ces fonctions ?

Question 2.2 Quelle est la complexité de `f` ? Justifier précisément.

Question 2.3 Donner en fonction de `n` un ordre de grandeur du résultat de ce qui est calculé par `f` ?

Question 2.4 Quelle est la complexité de `g` ? Justifier précisément.

Question 2.5 Donner en fonction de `n` un ordre de grandeur du résultat de ce qui est calculé par `g` ?

Question 2.6 Réécrire la fonction `g` pour diminuer sa complexité ? Quelle est alors sa complexité ?

Exercice 3 : Vecteur creux (9 points)

Un vecteur est un élément de $\mathbb{R}^k$, c'est-à-dire un k-uplet de coordonnées. Chaque coordonnées est représentée à l'aide d'un `double`. Il y a deux manières de représenter un vecteur :

— si le vecteur a toutes ses coordonnées définies, on l'appelle un *vecteur dense*, et on le représente à l'aide d'un tableau de taille k ;

— si le vecteur a seulement quelques coordonnées non-nulles, on l'appelle un *vecteur creux*, et on ne représente alors que les coordonnées du vecteur qui sont non-nulles.

Par exemple, pour $k = 10$, le vecteur dense $[0, 0, 3.2, 0, 0, 4.5, 0, 0, 7.9, 6.1]$ peut être représenté par :

$$[(6, 4.5), (10, 6.1), (3, 3.2), (9, 7.9)]$$

Dans cet exercice, nous allons évaluer plusieurs manières de représenter un vecteur creux de $\mathbb{R}^k$ (où k est une constante très grande) avec un tableau dynamique.

Nous utiliserons les structures de données suivantes :

```
// coordonnée d'un vecteur creux
struct coord {
  int index;    // indice dans le k-uplet dans [1, k]
  double value; // valeur de la coordonnée
}

// tableau dynamique de coordonnées
struct array {
  size_t capacity;    // nombre d'éléments maximum du tableau
  size_t size;        // nombre d'éléments du tableau
  struct coord *data; // tableau
}
```

Pour que le tableau dynamique représente un vecteur creux, on a deux contraintes :

1. chaque indice n'est présent qu'en un seul exemplaire dans le tableau ;
2. toutes les valeurs de coordonnées sont non-nulles.

Dans tout cet exercice, on va s'intéresser à trois opérations :

— `access`, l'accès à une valeur suivant son indice ;

— `set`, la définition d'une nouvelle valeur à un indice donné ;

— `add`, l'addition de deux vecteurs creux.

Les prototypes de ces fonctions sont :

```
double access(const struct array *vector, size_t index);
void set(struct array *vector, size_t index, double value);
void add(struct array *result,
        const struct array *left, const struct array *right);
```

On suppose tout d'abord que le tableau dynamique n'est pas trié par ordre d'indice.

Question 3.1 Donner une implémentation en C de `access`. Quelle est sa complexité ?

Question 3.2 Quelle est la complexité de `set` si on sait qu'`index` n'existe pas dans le tableau ? On donnera une réponse très précise et justifiée.

Question 3.3 Quelle est la complexité de `set` si on ne sait pas qu'`index` existe dans le tableau ? On donnera une réponse très précise et justifiée.

Question 3.4 Quelle taille peut avoir le tableau `result->data` au maximum en fonction de `left->size` et `right->size` ? Justifier.

Question 3.5 Donner une idée de l'algorithme pour `add`. Quelle est sa complexité ?

On suppose désormais que le tableau dynamique est trié par ordre croissant d'indice.

Question 3.6 Quel algorithme utilise-t-on pour `access` ? Quelle est sa complexité ?

Question 3.7 Donner une idée de l'algorithme pour `set` ? Quelle est sa complexité ?

Question 3.8 Donner une implémentation complète en C de `add`. On prendra garde à la gestion de la mémoire.

Question 3.9 Quelle est la complexité de la fonction que vous venez d'implémenter ?

Question 3.10 Quelle amélioration peut-on proposer pour l'algorithme de la question 5 ? Quelle est alors la complexité de `add` avec cette amélioration ?