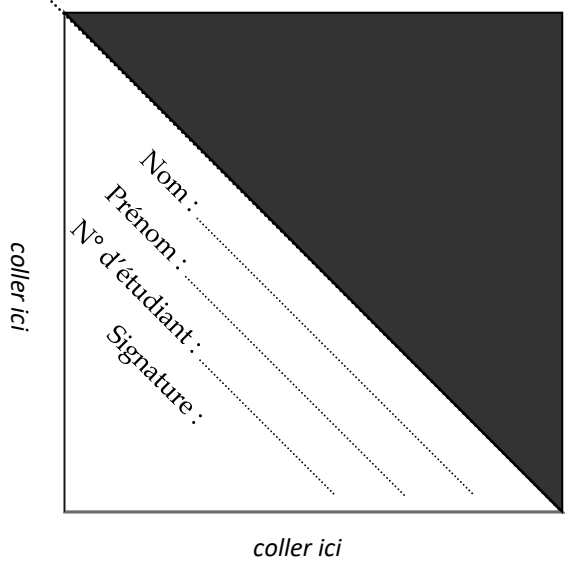
Université Claude Bernard Lyon 1

Année universitaire : 2018 / 2019

LIFAP3 : Algorithmique et programmation avancée

ECA - Epreuve Commune Anonyme
14 janvier 2019
Durée : 1h30

Note : / 20



Documents et téléphones portables interdits. Le barème est donné à titre indicatif. Répondez dans les emplacements prévus à cet effet. Travaillez au brouillon d'abord de sorte à rendre une copie propre – nous ne pouvons pas vous garantir une copie supplémentaire. **Il sera tenu compte de la présentation et de la clarté de vos réponses**.

Exercice 1 : Tri par comptage (14 points)

Le tri par comptage consiste pour chaque élément d'un tableau à trier à compter combien d'éléments sont strictement plus petits que lui. Grâce à ce chiffre, on peut connaître sa position dans le tableau trié. Plus le chiffre est grand, plus l'élément sera vers la fin du tableau. Nous supposerons que le tableau ne contient qu'une seule occurrence de chaque élément.

Exemple. Soit le tableau dynamique **`tab`** d'entiers suivant.

7	-1	0	5	2	-3	1

Le tableau de comptage **`tabComptage`** est le suivant, où pour chaque élément on compte combien d'éléments sont strictement plus petits.

6	1	2	5	4	0	3

Le tableau **`tabTrie`** trié peut-être calculé à partir de ces chiffres, où les éléments sont rangés par ordre croissant de compte (du plus petit à gauche au plus grand à droite). Le compte le plus petit (0) correspondant à la valeur -3, elle sera mise en première position dans le tableau trié. Le compte suivant (1) correspond à la valeur -1, etc.

-3	-1	0	1	2	5	7

L'intérêt de ce tri est que la valeur du compte correspond exactement à l'indice de l'élément dans le tableau trié. En effet, le compte est forcément compris entre 0 (l'élément n'ayant aucun autre élément plus petit) et la taille du tableau – 1 (l'élément ayant tous les autres éléments plus petits).

L'entête en notation algorithmique de la fonction **`triComptage`** qui trie un tableau dynamique d'entiers est la suivante.

Fonction triComptage (tab : TableauDynamique d'entiers) : pointeur sur TableauDynamique d'entiers
Post-condition : tab est inchangé
Résultat : l'adresse mémoire d'un nouveau tableau dynamique stocké sur le tas contenant les éléments de tab triés
Paramètres en mode donnée : tab

Pour rappel, la classe **`TableauDynamique`** a trois données membres : **`capacite`** et **`taille_utilisee`** de type **`unsigned int`** et **`ad`** de type pointeur sur **`ElementTD`**. Les fonctions membres de la classe sont rappelées en annexe.

Question 1.1 : Ecrire en C++ la procédure globale **`remplirTabComptage`** dont l'entête en notation algorithmique est donnée ci-dessous.

Procédure remplirTabComptage (tab : TableauDynamique d'entiers, tabComptage : TableauDynamique d'entiers)
Pré-condition : tabComptage est un tableau dynamique vide
Post-condition : tabComptage contient, pour chaque élément de tab, le nombre d'éléments strictement plus petits
Paramètres en mode donnée : tab
Paramètres en mode donnée-résultat : tabComptage

```
void remplirTabComptage (const TableauDynamique & tab, TableauDynamique & tabComptage) {
   for (unsigned int i = 0 ; i < tab.taille_utilisee ; i++) {
      unsigned int cpt = 0;
      for (unsigned int j = 0; j < tab.taille_utilisee ; j++) {
         if (tab.valeurIemeElement(j) < tab.valeurIemeElement(i)) {
            cpt++;
         }
      }
      tabComptage.ajouterElement(cpt);
   }
}
```

Question 1.2 : Rappeler la complexité amortie, en notation O, des membres **`ajouterElement`**, **`valeurIemeElement`** et **`modifierIemeElement`** de la classe **`TableauDynamique`** en fonction de n la taille effective du tableau (c'est-à-dire **`taille_utilisee`**).

`ajouterElement` : $O(1)$ **`valeurIemeElement`** : $O(1)$ **`modifierIemeElement`** : $O(1)$

Question 1.3 : Calculer la complexité de la procédure **remplirTabComptage** en fonction de n la taille effective du tableau **tab**. Justifier votre réponse en calculant le nombre exact de comparaisons d'éléments effectuées.

Les comparaisons d'éléments se font dans la condition du si. Le test du si est inclus dans une boucle for (sur j) qui est faite n fois, cette boucle est elle-même incluse dans une autre boucle for (sur i) qui est faite aussi n fois.
Il y a donc exactement n^2 comparaisons d'éléments, d'où la complexité en $O(n^2)$.

Question 1.4 : Ecrire en C++ la fonction **triComptage** dont l'entête est donnée en première page. Vous pourrez utiliser la procédure **remplirTabComptage**.

```
TableauDynamique * triComptage (const TableauDynamique & tab) {
   TableauDynamique tabComptage;
   remplirTabComptage(tab,tabComptage);

   TableauDynamique * tabTrie = new TableauDynamique;
   for (unsigned int i = 0 ; i < tab.taille_utilisee ; i++) {
      tabTrie->ajouterElement(0);
   }

   for (unsigned int i = 0 ; i < tab.taille_utilisee ; i++) {
      unsigned int indice = tabComptage.valeurIemeElement(i);
      tabTrie->modifierIemeElement(tab.valeurIemeElement(i),indice);
   }

   return tabTrie;
}

// ou bien, à la place des deux boucles for :

   for (unsigned int i = 0 ; i < tab.taille_utilisee ; i++) {
      tabTrie->ajouterElement(tab.valeurIemeElement(tabComptage.rechercherElement(i)));
   }
```

Question 1.5 : Donner la complexité de la fonction **triComptage** en fonction de n, la taille effective du tableau à trier. Justifier brièvement votre réponse sans calculer exactement le nombre d'opérations effectuées.

La fonction fait appel à **remplirTabComptage** qui est $O(n^2)$. De plus, elle appelle n fois **ajouterElement** (première boucle pour) ce qui donne $O(n \times 1) = O(n)$, cf. question 1.2. Puis, elle appelle n fois (deuxième boucle pour) **valeurIemeElement** (2 fois) et **modifierIemeElement** qui donne du $O\big(n \times (2+1)\big) = O(n)$.
La fonction est donc en $O(n^2 + n + n) = O(n^2)$.

Question 1.6 : Compléter le programme principal ci-dessous qui créé un tableau dynamique sur la pile, le remplit de 10 valeurs entières aléatoires comprises entre -7 et 7, effectue le tri du tableau par comptage et affiche le résultat du tri.

```
#include <stdlib.h>
#include <time.h>
#include "TableauDynamique.h"

int main () {
   srand(time(NULL));

   TableauDynamique tab;

   for (unsigned int i = 1 ; i <= 10 ; i++) {
      int valeur = rand()%15 - 7;
      tab.ajouterElement(valeur);
   }

   TableauDynamique * tabTrie = triComptage(tab);

   tabTrie->afficher();

   delete tabTrie;

   return 0;
}
```

Exercice 2 : Parcours d'arbre binaire de recherche (6 points)

Dans cet exercice, on s'intéresse aux arbres binaires de recherche implémentés par la classe **`Arbre`** vue en CM/TD/TP. Pour rappel, la classe **`Arbre`** a une seule donnée membre : **`adRacine`** de type pointeur sur **`Noeud`**. La structure **`Noeud`** a trois champs : **`info`** de type **`ElementA`**, **`fg`** et **`fd`** tous deux de type pointeur sur **`Noeud`**. Voir l'annexe pour les fonctions membres de la classe **`Arbre`**.

Question 3.1: Donner le code C++ de la procédure membre **`afficherParcoursInfixe`**. Donner également le code de toute fonction ou procédure intermédiaire nécessaire.

```
void Arbre::afficherParcoursInfixe () const {
    affichageParcoursInfixeDepuisNoeud(adRacine);
    cout << endl;
}

void Arbre::affichageParcoursInfixeDepuisNoeud (const Noeud * n) const {
    if (n != NULL) {
      affichageParcoursInfixeDepuisNoeud(n->fg);
      afficherElementA(n->info);
      cout << " ";
      affichageParcoursInfixeDepuisNoeud(n->fd);
    }
}
```

Question 3.2: Donner le code C++ de la procédure membre **`afficherParcoursEnLargeur`**.

```
void Arbre::afficherParcoursEnLargeur () const {
    File f;
    Noeud * n;

    if (estVide()) { cout << "Arbre vide" << endl; return; }

    f.enfiler(adRacine);
    while (!f.estVide()) {
        n = (Noeud *) f.premierDeLaFile();
        f.defiler();
        afficherElementA(n->info);
        cout << " ";
        if (n->fg != NULL) f.enfiler(n->fg);
        if (n->fd != NULL) f.enfiler(n->fd);
    }
    cout << endl;
}
```

Annexe : liste des fonctions membres des classes TableauDynamique, Liste, File, Pile et Arbre (constructeurs et destructeurs omis)

Classe TableauDynamique

```
void vider ();
void ajouterElement (ElementTD e);
ElementTD valeurIemeElement (unsigned int indice) const;
void modifierValeurIemeElement (ElementTD e, unsigned int indice);
void afficher () const;
void supprimerElement (unsigned int indice);
void insererElement (ElementTD e, unsigned int indice);
int rechercherElement (ElementTD e) const;
```

Classe Liste

```
void vider ();
bool estVide () const;
unsigned int nbElements () const;
ElementL iemeElement (unsigned int indice) const;
void modifierIemeElement (unsigned int indice, ElementL e);
void afficherGaucheDroite () const;
void afficherDroiteGauche () const;
void ajouterEnTete (ElementL e);
void ajouterEnQueue (ElementL e);
void supprimerTete ();
int rechercherElement (ElementL e) const;
void insererElement (ElementL e, unsigned int indice);
void supprimerElement (ElementL e);
void supprimerCellule (Cellule * c);
```

Classe File

```
void enfiler (ElementF e);
ElementF premierDeLaFile () const;
void defiler ();
bool estVide () const;
unsigned int nbElements () const;
```

Classe Pile

```
void empiler (ElementP e);
ElementP consulterSommet () const;
void depiler ();
bool estVide () const;
```

Classe Arbre

```
bool estVide () const;
void vider ();
void insererElement (ElementA e);
void supprimerElement (ElementA e);
Noeud * rechercherElement (ElementA e) const;
void afficherParcoursInfixe () const;
void afficherParcoursEnLargeur () const;
```