Algorithmique et Structures de Données 1
Projet 2 de Travaux Pratiques

Année 2019/2020

- Introduction

L'objet de ce projet est d'implémenter un algorithme de tri par fusion multiple de monotonies. Les données à trier sont d'un type précisé au début du code (`typedef … DATATYPE`), mais le reste du code doit être générique et pouvoir fonctionner pour plusieurs types (pour peu de disposer de l'opérateur de comparaison noté **<**).

Ces données seront stockées dans un chaînage qui sera ensuite partitionné en morceaux croissants puis ces morceaux seront fusionnés en une nouvelle liste triée.

Le travail est à effectuer en binôme (deux étudiant(e)s) et le code doit être compilable et exécutable sur les machines des salles de TP — vous pouvez travailler sur une machine personnelle mais **devez** vérifier le fonctionnement en salle de TP.

A - Chaînage

```
typedef struct _datum {
    DATATYPE valeur ;
    _datum * suiv ;
} data ;

typedef data * p_data ;
```

La structure de chaînage proposée est reprise du TP précédent (ci-contre)

Les fonctions et procédures suivantes doivent être définies sur ce type :

- `void aff(p_data chain)` qui affiche (sur une même ligne) les valeurs stockées dans le chaînage `chain`.
- `p_data ajoutDevant(DATATYPE uneval, p_data chain)` qui crée un nouveau maillon avec la valeur `uneval` et retourne la tête d'un chaînage commençant par ce maillon et continuant avec `chain`.
- `p_data saisieBorne(DATATYPE sentinelle)` qui lit (sur l'entrée standard) une succession de valeurs et les mémorise (dans le même ordre) dans un chaînage terminé par `nullptr`, dont la tête est renvoyée. La saisie se termine à la première occurrence d'une valeur égale à `sentinelle`.
- `p_data saisieNombre(int nb)` qui lit (sur l'entrée standard) une succession de `nb` valeurs et les mémorise (dans le même ordre) dans un chaînage terminé par `nullptr`, dont la tête est renvoyée.
- `p_data fusion(p_data prem, p_data sec)` qui fusionne deux chaînages terminés par `nullptr`, supposés triés par ordre croissant. Le résultat obtenu doit être trié par ordre croissant. Cette fonction doit effectuer au plus un nombre fixe d'allocations mémoire. Bien indiquer les PRE et POST conditions. Écrire une version itérative et une version récursive.
- `int nbCroissances(p_data chain)` qui compte le nombre de monotonies croissantes dans un chaînage terminé par `nullptr`. Par exemple si le chaînage contient les caractères de la chaîne `"copacabana"`, les monotonies croissantes sont `"cop"`, `"ac"`, `"ab"`, `"an"` et `"a"`, il y en a donc 5.
- `void extraireCroissance(p_data & chain, p_data & mono)` qui place dans le chaînage `mono` la première monotonie croissante de `chain` et l'en retire. Cette fonction doit effectuer au plus un nombre fixe d'allocations mémoire. Bien indiquer les PRE et POST conditions.

B - Monotonies

```
typedef struct _datallst
{
    int capa ;
    int nbmono ;
    p_data * monotonies ;
} datalistes ;
```

Pour trier un chaînage comme celui étudié question précédente, une solution consiste à le décomposer en monotonies croissantes, mémorisées dans un tableau (encapsulé dans le type donné ci-contre). Le champ `monotonies` est destiné à être valué par un tableau dont la dimension sera connue à l'exécution, le champ `capa` représente la taille (nombre de cases) réservée en mémoire pour le tableau, le champ `nbmono` le nombre de cases effectivement utilisées (à partir du premier indice). Les opérations suivantes doivent être définies pour ce type :

- `datalistes initT(int nb)` qui crée et renvoie une structure (de type `datalistes`) initialisée avec un tableau de `nb` chaînages (non valués).
- `void ajouterFin(p_data chain,datalistes & mono)` qui ajoute dans le tableau `mono` le chaînage `chain` (dans la prochaine case non utilisée). Cette fonction ne doit pas effectuer plus qu'un nombre fixe d'allocations mémoire. Bien indiquer les PRE et POST conditions.
- `void affT(datalistes mono)` qui affiche (une ligne par case) les valeurs stockées dans le tableau `mono`.
- `p_data suppressionFin(datalistes & mono)` qui supprime du tableau `mono` et renvoie le dernier chaînage encore stocké (celui de plus grand indice). Bien indiquer les PRE et POST conditions.
- `p_data suppressionTotale(datalistes & mono)` qui renvoie un chaînage formé en mettant bout à bout (dans l'ordre) tous les chaînages stockés dans le tableau `mono`, en les y enlevant. Précisez PRE et POST conditions.

C - Test

Pendant le développement des fonctions précédentes, et avant de continuer, il est impératif de vérifier que le code déjà produit est correct : écrivez au fur et à mesure (dans le même fichier) un programme principal permettant de tester les procédures et fonctions déjà écrites. Il peut être intéressant de réutiliser le travail du TP précédent (création et affichage d'un chaînage en particulier). Bien présenter les jeux d'essai.

Ces tests seront ensuite placés en commentaires mais devront rester dans le code rendu, il doit être possible de les réactiver.

D - Fusion multiple

Écrivez une fonction `datalistes separation(p_data & chain)` retournant un tableau dont les cases contiennent (dans l'ordre) les monotonies croissantes du chaînage contenu dans `chain`. Après exécution, `chain` doit contenir le pointeur null.

Écrivez ensuite une procédure `void trier(datalistes & tabmono)` effectuant une fusion multiple des monotonies stockées dans le tableau, pour ne conserver à la fin qu'un seul chaînage trié dans la première case. La fusion multiple consiste remplacer le contenu d'une case du tableau par la fusion de ce contenu avec le contenu d'une autre case (le nombre de cases occupées diminue) et à recommencer. Discutez dans votre rapport de l'ordre dans lequel vous allez programmer cette suite de fusions.

En déduire une procédure `void trier(p_data & chain)` permettant de trier un chaînage en utilisant une fusion multiple.

Compléter le programme principal pour tester ces fonctions et procédures.

E - Étude

Effectuez une rapide étude pratique de coût temporel de l'algorithme de tri, en fonction du nombre de valeurs à trier. Un échantillonnage aléatoire n'est pas exigé, une méthode pourrait être de prendre (pour `DATATYPE` fixé à `string`) un nombre croissant de mots au début d'un fichier texte ; des fichiers vous seront fournis pour comparer entre vous, mais vous pouvez bien sûr en proposer d'autres correspondant par exemple à des cas particuliers.

Justifiez les résultats obtenus (tracez des courbes) par un rapide calcul théorique (une majoration).

F - Travail à rendre

Le travail d'implémentation sera réalisé dans un unique fichier source, clairement présenté et documenté, comportant l'ensemble des procédures et fonctions réalisées ainsi qu'un programme principal démontrant leur utilisation.

Vous rédigerez un document de travail au préalable et en parallèle, précisant les ambiguïtés du sujet, les difficultés rencontrées. Il servira de base à votre rapport.

Ce document sera complété au fur et à mesure avec les choix effectués (constantes, comportement des procédures et fonctions non précisé dans l'énoncé, …) et les limites d'utilisation.

Il devra être bien rédigé (français correct, attention à l'orthographe !) et agréable à lire, comporter introduction et conclusion et éventuellement des annexes pour les détails techniques (programmes, tests réalisés, tableaux de valeurs, ...).

Le rapport (au format PDF) et le code C++ (fichier(s) source .cpp) sont à déposer
sur MADOC dans un répertoire compacté (zip ou tar.gz)

au plus tard le **vendredi 20 décembre 2019 à 19h00**.

Un seul rendu par binôme (sur l'un ou l'autre des deux comptes).