Algorithmique et langage C

Devoir maison

Votre travail sera à rendre dans un fichier sous la forme **NOM_prenom_AP3_C_DM.zip**.
Par exemple : GOSSWILLER_robin_AP3_C_DM.zip .
Cette archive devra contenir **le fichier source** (.c) de votre code, **une version compilée** de votre travail, ainsi que tout autre fichier qui vous semble nécessaire à une bonne compréhention de votre travail. Merci de bien faire paraître dans votre travail la méthode utilisée pour compiler (makefile ou au minimum l'instruction console utilisée).

Introduction

Ce devoir maison tourne autour des différences entre tableau, liste chaînée et arbre binaire de recherche dans le cadre de l'algorithmie de recherche et de tri. Au cours de votre travail, vous serez amenés à implémenter ces trois méthodes et à constater leur avantages et défauts.

Les tests se feront sur un ensemble de 100 000 valeurs sous format 'double' compris entre 0 et 1 000 000.

Partie 1 - Tableau

1. — Réserver un tableau de 100 000 doubles.
   — A l'aide d'un processus aléatoire, générer un entier entre 0 et 1 000 000, et l'ajouter au tableau.
   — Répéter ce processus jusqu'à remplir le tableau en évitant les doublons.
   — Afin d'examiner la performance du code, on ajoute une variable $(int)\ lectures$ et une autre $(int)\ ecritures$, initialisées à 0 et qui s'incrémentent lorsqu'un élément du tableau est lu (en ajoutant lectures++ ; après chaque opération de lecture) ou écrit (même principe). Quelle est la valeur de ces variables une fois le tableau terminé ? (effectuer quelques éxecutions du code pour obtenir une moyenne)
2. — Créer un nouveau tableau qui sera un tri du tableau précédent. On utilisera la méthode de tri nommée 'tri séléction' de manière peu optimisée :
   — on initialise notre 'precédente sélection' à 0 (car toutes les valeurs sont plus grandes que 0) et notre variable d'incrément i à 0.

- on parcours le tableau pour en trouver la plus petite valeur plus grande que notre 'précédente séléction'
- on la met dans la i-ieme case du tableau
- on incrémente i de 1 et on enregistre la valeur nouvellement ajoutée comme la 'précédente séléction'
- on recommence le procédé de parcours de tableau jusqu'à avoir selectionné toutes les valeurs (100 000 fois en tout).

— Combien de lectures et d'ecritures ce procédé a-t-il demandé ?

3. A présent, on s'intéresse à la lecture. Prendre un nombre aléatoire entre 0 et 1 000 000 et chercher sa présence dans le tableau non trié et dans le tableau trié. Combien de lectures cela demande-t-il ?

Partie 2 - Liste chaînée

1. — Créer une structure de chaîne disposant d'une valeur au format double et d'un pointeur vers le prochain élément de la chaîne. On prendra la valeur 500 000 pour le premier élément.
   — A l'aide d'un processus aléatoire, générer un entier entre 0 et 1 000 000, et l'ajouter à la chaîne.
   — Répéter ce processus pour la chaîne en évitant d'avoir des doublons jusqu'à avoir 100 000 valeurs différentes.
   — Afin d'examiner la performance du code, on ajoute une variable $(int)\ lectures$ et une autre $(int)\ ecritures$, initialisées à 0 et qui s'incrémentent lorsqu'un élément de la chaîne est lu (nécessaire pour parcourir la chaîne !) ou écrit (création de maillon ou modification d'une donnée). Quelle est la valeur de ces variables une fois la chaîne étendue ? (effectuer quelques executions du code pour obtenir une moyenne)
   — Quelle taille en mémoire fait cette structure ? Est-ce significatif (au moins 10 fois plus ou 10 fois moins) par rapport à la taille en mémoire du tableau ?
2. — Créer une nouvelle chaîne qui sera un tri de la chaîne précédente. On utilisera une méthode de tri différente :
   - on prends le premier élément de la chaîne initiale et on le met comme premier élément de la seconde chaîne
   - pour chaque élément suivant de la chaîne initiale, on progresse dans la chaîne triée jusqu'a tomber sur une valeur plus grande, et on range l'élément courant juste avant (attention à la gestion du premier élément)

— Utiliser une chaîne et cet algorithme change-t-il significativement (au moins 10 fois plus ou 10 fois moins) le nombre d'opérations nécessaires pour trier les valeurs ?

3. A présent, on s'intéresse à la lecture. Prendre un nombre aléatoire entre 0 et 1 000 000 et chercher sa présence dans chacune des chaînes. Le nombre de lectures est-il significativement différent à utiliser un tableau ?

Partie 3 - Arbre binaire de recherche

1. — Créer une structure d'arbre disposant d'une valeur au format double et de deux pointeurs : gauche et droite.
   — A l'aide d'un processus aléatoire, générer un entier entre 0 et 1 000 000, et l'ajouter à l'arbre avec le procédé suivant :
     (a) En partant de la racine, regarder si le nombre est égal au noeud courant (si c'est le cas, c'est un doublon et on recommence).
     (b) S'il est plus petit, on le regarde à gauche, sinon on regarde à droite.
     (c) Si il y a un noeud dans la direction regardée, alors on compare la valeur à ajouter à celle de ce noeud et on recommence (en allant à gauche ou a droite).
     (d) Sinon, on ajoute un nouveau noeud à l'arbre dans cette direction.
   — Quelle taille en mémoire fait cette structure ? Est-ce significatif (au moins 10 fois plus ou 10 fois moins) par rapport à la taille en mémoire du tableau ?
2. L'arbre ainsi créé est déjà trié. Le processus de création est-il significativement différent en nombre d'opérations et de mémoire utilisée qu'un tableau ? (on utilisera à nouveau des variables pour compter le nombre de lecture et d'écritures nécessaires)
3. A présent, on s'intéresse à la lecture. Prendre un nombre aléatoire entre 0 et 1 000 000 et chercher sa présence dans l'arbre. Le nombre de lectures est-il significativement différent à utiliser un tableau ?

Conclusion

1. Pour chaque partie, présenter les résultat que vous avez obtenus (de manière graphique si possible).
2. Entre les trois méthodes, quelle est celle qui est la plus économe en mémoire ? Quelle est celle qui est la plus rapide pour construire le jeu de données, pour le trier, et pour le lire ? La plus simple à coder ? (la dernière question, subjective, ne sera pas notée mais mérite d'être posée néanmoins !)

Grille de notation

Chaque ligne corresponds à 1 point sauf si c'est précisé autrement.

- Rendu contenant une source et un fichier compilé qui ne crash pas
- Instructions de compilations claires et qui fonctionnent (ligne de commande ou makefile)
- Programme simple d'utilisation et qui marque clairement ses résultats
- Code clair et commenté lorsque c'est nécessaire
- Partie 1
  - Création des données
  - Tri des données
  - Lecture des données
  - Surveillance des ressources système
- Partie 2
  - Création des données
  - Tri des données
  - Lecture des données
  - Surveillance des ressources système
- Partie 3
  - Création des données (2 points)
  - Lecture des données
  - Surveillance des ressources système
- Conclusion
  - Présentation des résultats (2 points)
  - Evaluation des résultats (2 points)