Algorithmique avancée : Feuille de TP n^o 1

Structures de données

1 TP encadré + 1 TP non encadré + 1 QEL [1]

Ces TPs doivent être réalisés **en langage C** (lien vers un mémo C). Les algos sur les tas binaires sont disponibles dans le fichier feuille d'algo, ils doivent être implémentés selon ce modèle.

Les éléments du langage **C** ou le processus de compilation/exécution sont considérées comme des pré-requis. L'assistance de l'enseignant de TP privilégiera donc les questions liées à l'algorithmique et aux structures de données.

⚠ À l'issue du TP encadré, vous aurez une séance de TP non encadrée, puis lors de la séance encadrée suivante, un QEL sera organisé pour évaluer l'atteinte des objectifs fixés.

Contexte du sujet

Nous souhaitons développer un algorithme efficace pour l'extraction des k plus grands éléments dans un tableau de n entiers avec $1 \leq k \leq n$ et les afficher dans l'ordre décroissant. Par programme efficace, nous entendons que les traitement doivent se faire en place, avec une complexité en espace en $\Theta(1)$ et une complexité en temps la plus petite possible.

Cette série de travaux pratiques a pour objectif la mise en œuvre en langage **C** et l'évaluation de différentes approches possibles pour résoudre ce problème.

Les données du problèmes à résoudre seront lues, à partir de l'entrée standard ou d'un fichier texte, selon le format suivant :

— un entier donnant la valeur de n,
— un entier donnant la valeur de k,
— n entiers correspondant aux données.

Le programme devra prendre un argument optionnel permettant de définir le nom du fichier d'entrée (sans argument : la lecture des données se fera à partir du flux d'entrée standard).

Après exécution, le programme devra afficher les k valeurs sélectionnées sons la forme d'une liste d'entiers séparés par un espace. Par exemple, le fichier de test suivant :

```
7 3
30 23 12 50 1 2 9
```

contient 7 entiers desquels il faut extraire les 3 plus grands.

Le programme devra donc retourner 50, 30 et 23.

Sur moodle vous avez à votre disposition les fichiers :

— `test7.txt` contenant l'exemple du sujet à 7 éléments,
— `donnees5000.txt` contenant un exemple avec 5 000 entiers,
— `donnees10000.txt` contenant un exemple avec 10 000 entiers
— `tpSD.c` contenant un programme à complèter qui lit et affiche le tableau issu d'un fichier passé en paramètres ou tapé sur l'entrée standard. Pour le compiler et créer l'exécutable tapez : `gcc -g tpSD.c -o tpSD`
Pour l'exécuter tapez : `./tpSD donnees10000.txt`

1. QEL=Questionnaire en Espace Limité.

Exercice 1 : Approche simple avec un tri à bulle

L'algorithme de tri-bulle (bubble sort) d'un tableau d'entier de taille n repose sur une boucle interne repoussant en fin de tableau la valeur la plus grande par comparaisons successives d'un élément avec son suivant et permutation éventuelle. Pour trier un tableau de taille n, on crée une boucle externe qui répète la boucle interne n fois. Comme nous ne voulons que les k meilleurs, il suffit de répeter cette remontée uniquement un nombre k de fois, les k éléments les plus grands seront amenés en fin de tableau et pourront donc être affichés par notre programme.

— Programmez cette solution simple et validez-la sur les jeux de tests fournis.

Exercice 2 : Utilisation d'un tas binaire maximal

Si l'on structure les données dans un tas binaire maximal, l'accès aux k plus grandes valeurs dans l'ordre décroissant revient à effectuer k fois l'opération REMOVE sur le tas.

1. Implémentez la fonction BUILDHEAP pour créer un tas binaire maximal représenté par un tableau. Vous l'expérimenterez sur les jeux de tests fournis.
2. Implémentez la fonction ADD pour un tas maximal. La fonction ADD n'insère un élément que si le nombre d'éléments du tas binaire (appelé SIZE) est inférieur à la capacité du tas (appelée LENGTH).
3. Créez un nouveau tableau statique vide de taille LENGTH (par exemple, le nombre n d'éléments contenus dans le fichier + 5). Remplissez ce tableau en construisant un tas binaire maximal par n appels successifs à ADD avec les éléments du tableau des entiers lus sur les jeux de tests fournis.
4. Comparez le temps de calcul des deux façons de construire le tas pour différentes valeurs de n. Vous pouvez utiliser les fonctions `startMeasuringTime`, `stopMeasuringTime` et `showElapsedTime` fournies dans le fichier `tpSD.c`. Vos deux tas sont-ils toujours égaux ?
5. Programmez la fonction REMOVE puis écrivez un programme réalisant l'extraction des k plus grandes valeurs dans un tableau de n éléments en utilisant l'une des deux façons de construire un tas maximal.

Exercice 3 : Approche par tri complet de la collection

Lorsque les données du problème sont triées, il est alors trivial d'extraire les k plus grands éléments dans l'ordre décroissant.

L'algorithme de tri par tas, HEAPSORT(T), prend un tableau T en entrée et le trie en place. Ce tri se fait par un BUILDHEAP(T) pour transformer le tableau en Tas binaire **maximum**, puis on opère n REMOVEBIS successifs sur T, où REMOVEBIS est une opération composée d'un REMOVE qui remplace la racine par le dernier élément puis le PERCOLATEDOWN, mais au lieu de détruire l'élément supprimé, on le met à la place du dernier élément dans le tableau, puis on décrémente T.SIZE. Ainsi le maximum est en dernière position. Au cours du HEAPSORT, T.SIZE décroit jusqu'à 0 alors que la capacité du tas reste la même (T.LENGTH= n). À la fin du HEAPSORT, le tableau T est trié par ordre croissant.

1. Implémentez HEAPSORT.
2. Implémentez maintenant la solution consistant à trier les données par un HEAPSORT puis à afficher les k plus grandes.

3. Implémentez une deuxième solution consistant à trier les données par l'algorithme de tri QuickSort (vous pouvez utilisez la fonction `qsort` de la librairie standard C, où la réimplémenter) puis à afficher les k plus grandes.
4. Implémentez une troisième solution consistant à trier les données par l'algorithme de tri MergeSort (dont l'algorithme a été donné en TD).

Comparaison expérimentale de l'efficacité des approches

1. Comparez l'efficacité en temps de calcul des différents tris, pour une valeur fixe de n assez grande (par exemple 10^7).
2. Comparez expérimentalement la méthode du tri à bulles à la meilleure de vos méthodes de tri complet : pour une valeur fixe de n assez grande (par exemple 10^7), déterminez la valeur de k à partir de laquelle le tri à bulles partiel devient plus lent qu'un tri total efficace.
3. Installez valgrind.
4. Lancez les commandes suivantes sur votre exécutable (en supposant qu'il est nommé tpSD) :
   (a) `valgrind --tool=callgrind ./tpSD donnees10000.txt`
   (b) Cette commande crée un fichier texte `callgrind.out.<pid>`.
   (c) La commande suivante vous permet de voir en face du code de votre programme le nombre d'instructions de chaque appel et le pourcentage du temps passé dans chaque ligne.
   (d) `callgrind_annotate callgrind.out.<pid>`
   (e) Notez les "Ir" obtenus pour les différentes fonctions effectuées. Les Irs correspondent-ils bien à la complexité temporelle théorique de l'algorithme que vous avez implémenté.
5. (Optionnel) Retrouvez à présent la complexité spatiale vue en TD pour MergeSort grâce aux commandes suivantes :
   (a) `valgrind --tool=massif ./tpSD donnees10000.txt`
   (b) Cette commande crée un fichier texte : `massif.out.<pid>`.
   (c) `ms_print massif.out.<pid>`
   (d) Vous renvoie des informations sur l'occupation mémoire de la pile d'exécution (stack) et du tas (heap). (# représente le pic d'utilisation, les moments des snapshots)

Travail à faire avant la prochaine séance encadrée

Le questionnaire en espace limité (QEL) demandera de faire tourner les 3 exercices sur des fichiers petits et gros afin de renvoyer les k plus grands entiers d'un tableau.

Il faudra que vous soyez capable de faire des petites modifications des programmes pour répondre à des variantes des questions posées dans cet énoncé. Vous devrez également être capable d'afficher le tas binaire que vous aurez construit.

Il y aura une question concernant l'exécution de `valgrind` sur vos programmes.