Structure de données et algorithmes

Projet 1: Algorithmes de sélection

25 février 2025

L'objectif du projet est d'implémenter, d'analyser théoriquement et de comparer empiriquement différents algorithmes de sélection, c'est-à-dire permettant de trouver la `k`-ième valeur la plus petite dans un ensemble.

Algorithme de sélection

Le problème de sélection consiste à identifier dans un tableau de taille `N` la `k`-ième valeur la plus petite, étant donnée une fonction de comparaison entre valeurs. On supposera dans ce projet que le tableau initial est non nécessairement trié et qu'il est permis de permuter ses éléments. La fonction à implémenter `Select(A, k)` prendra comme argument le tableau `A` et la valeur de `k` et renverra la position dans `A` de la `k`-ième valeur la plus petite de `A`.

Une manière naïve pour résoudre le problème de sélection est de trier entièrement le tableau et de renvoyer simplement `k`. Cette solution est cependant généralement inefficace car elle effectue trop de travail par rapport à ce qui est strictement requis. On se propose d'étudier 4 solutions algorithmiques à ce problème[1]:

1. SelectionSelect
2. HeapSelect
3. QuickSelect
4. FRSelect

Les trois premiers algorithmes sont des adaptations des algorithmes de tri du même nom pour faire la sélection. Le quatrième est une variante censée être plus rapide de l'algorithme `QuickSelect`. Ces quatre algorithmes sont décrits ci-dessous.

SelectionSelect. L'algorithme `SelectionSelect` est une adaption du tri par sélection pour résoudre ce problème. L'idée du tri par sélection est de déterminer l'élément minimal du tableau, d'échanger cet élément avec le premier élément du tableau et d'ensuite procéder de la même manière pour trier le reste du tableau. Pour le problème de sélection, on peut se contenter d'appliquer cette idée jusqu'à avoir déterminé la position du `k`-ième plus petit élément afin d'éviter des calculs inutiles.

[1] Il est à noter qu'une solution théorique optimale au problème est donnée par un algorithme appelé "Median of Medians", qui est également une variante de l'algorithme `QuickSelect`. Cet algorithme, bien que $\Theta(n)$ dans le pire cas, est cependant relativement lent en pratique.

HeapSelect. L'algorithme `HeapSelect` se base sur l'idée du tri par tas pour faire la sélection. Le tri par tas procède en deux étapes: la première consiste à créer un tas à partir du tableau, alors que la deuxième range en fin de tableau les éléments du plus grand au plus petit en extrayant itérativement l'élément au sommet du tas. Si on modifie l'algorithme pour trier le tableau dans le sens des valeurs décroissantes, en utilisant un tas-min plutôt qu'un tas-max, on peut donc interrompre la deuxième étape quand le `k`-ième plus petit élément est celui au sommet du tas.

QuickSelect. L'algorithme `Quickselect` est une adaption du `Quicksort` pour le problème de sélection. L'idée de cet algorithme est comme pour le quicksort de répartir les éléments du tableau en fonction d'une valeur pivot (choisie arbitrairement dans le tableau). Une fois la partition effectuée, plutôt que de faire les deux appels récursifs du `Quicksort`, il est possible de se passer de l'un ou des deux appels dans le cas de la sélection. En effet, si la position du pivot `q` est égale à `k`, l'algorithme peut renvoyer directement `q` comme l'indice recherché. Si `q` $<$ `k` (resp. `q` $>$ `k`), la `k`-ième valeur se trouve à droite (resp. à gauche) du pivot et un seul appel récursif peut être effectué sur le sous-tableau correspondant pour trouver la `k`-ième valeur.

FRSelect. L'algorithme de Floyd-Rivest[2] est une version améliorée de l'algorithme `QuickSelect`. Cette variante permet de réduire potentiellement plus rapidement la taille du sous-tableau par rapport à l'algorithme `QuickSelect`. L'idée pour y parvenir est de déterminer plus intelligement l'élément pivot pour le placer au plus proche de l'élément `k` recherché. Pour ce faire, l'algorithme délimite un sous-ensemble de taille S (avec $S \ll N$) du tableau de départ et il recherche récursivement le k'-ième élément dans ce sous-ensemble avec k' ajusté par rapport à la taille de S pour représenter le même quantile que `k` par rapport N. L'élément ainsi sélectionné sera alors la valeur pivot utilisée pour le partitionnement, qui peut être réalisé comme dans le cas de l'algorithme `QuickSelect`. La taille S du sous-tableau considéré est adaptée à la taille du tableau par une formule assez compliquée, qui permet d'obtenir des bonnes performances théoriques et pratiques. Pour comprendre les détails de cet algorithme, on vous invite à consulter les sources données en note de bas de page.

Implémentation

Vous devez remplir 4 fichiers implémentant chacun une des quatres solutions:

- `SelectionSelect.c`;
- `HeapSelect.c`;
- `QuickSelect.c`
- `FRSelect.c`

Ces quatre fichiers devront fournir la fonction `select` telle que définie dans le fichier `Select.h` (fourni). Vous pouvez ajouter autant de fonctions supplémentaires que nécessaires (qui devront être déclarées statiquement).

Le prototype de la fonction `select` est le suivant:

```
size_t select(void *array, size_t length, size_t k,
              int (*compare)(const void *array, size_t i, size_t j),
              void (*swap)(void *array, size_t i, size_t j))
```

[2]Voir ce document https://people.csail.mit.edu/rivest/pubs/FR75b.pdf ou la page wikipédia https://en.wikipedia.org/wiki/Floyd-Rivest_algorithm.

où `array` est un pointeur vers le début du tableau à trier (de type `void *`, pour pouvoir être appliquée sur tout type de tableau), `length` est sa longueur, `k` est la position dans l'ordre de l'élément à trouver, `compare` est une fonction renvoyant un entier respectivement plus petit, égal ou plus grand que 0 si l'élément à la position `i` dans `array` est plus petit, égal ou plus grand que l'élément à la position `j` et `swap` est une fonction permettant d'échanger les éléments du tableau aux positions `i` et `j`. Par exemple, on peut utiliser cette fonction sur un tableau d'entiers de la manière suivante:

```
void swapInt(void *array, int i, int j) {
   int temp = ((int*)array)[i];
   ((int*)array)[i] = ((int*)array)[j];
   ((int*)array)[j] = temp;
}

int compareInt(void *array, int i, int j) {
   return (((int*)array)[i] - ((int*)array)[j]);
}

int array[5] = {4, 2, 1, 3, 0};
size_t pos = select(array, 5, 3, compareInt, swapInt);
```

Comme indiqué ci-dessus, la fonction `select` peut permuter les éléments du tableau mais uniquement en utilisant la fonction `swap` donnée en argument. Les éléments doivent être comparé uniquement avec la fonction `compare`. La fonction `select` devra renvoyer la position du `k`-ième élément le plus petit dans le tableau après appel de la fonction. On supposera que l'argument `k` est compris entre 0 et `length` $-$ 1 inclus. Appelée avec `k` $= 0$ (respectivement `k` $=$ `length` $-$ 1), la fonction devra ainsi renvoyer l'élément minimum (respectivement maximum du tableau).

Vos implémentations devront être *les plus efficaces possibles* tout en suivant les consignes données. L'implémentation des algorithmes `SelectionSelect` et `HeapSelect` n'appelle pas de commentaires particuliers. Pour l'implémentation de `QuickSelect`, vous avez plusieurs degrés de liberté en ce qui concerne le choix du pivot et de la fonction de partitionnement. Pour cette dernière, réfléchissez à une manière d'éviter autant que possible l'occurrence des pires cas. Pour l'implémentation de `FRSelect`, respectez à la lettre les pseudo-codes proposés aux deux sources données plus haut pour ce qui est du choix du pivot. Vous êtes libres par contre dans le choix de l'implémentation de la fonction de partitionnement.

Vos fichiers seront compilés avec `gcc` et les flags de compilation suivants:

```
--std=c99 --pedantic -Wall -Wextra -Wmissing-prototypes
```

Ceci implique que le projet doit être réalisé dans le standard C99. La présence de *warnings* impactera négativement la cote finale. Un fichier qui ne compile pas recevra une cote nulle.

Un fichier `Makefile` vous est fourni qui crée 4 exécutables à partir de vos implémentations en utilisant un fichier `main.c` également fourni. Ces exécutables vérifient d'abord l'exactitude de la fonction `select` sur un tableau de taille 1000 et lance ensuite un test sur les quatre types de tableaux précisés ci-dessous pour des tableaux de taille 10000 et le calcul de la médiane (`k` $= 5000$). Vous pouvez tester d'autres tailles de tableaux et valeurs de `k` en les donnant en arguments sur la ligne de commande. Vous êtes libres évidemment de modifier ce fichier `main.c` pour répondre aux questions ci-dessous.

Analyse théorique

Dans le rapport, on vous demande de répondre aux questions suivantes:

1. Donnez dans une table les complexités en temps et en espace dans le pire cas et le meilleur cas de vos 4 implémentations en fonction de la taille `N` du tableau et de la valeur de `k`. Justifiez

brièvement chacun des cas en précisant si nécessaire les spécificités de vos implémentations. Pour l'algorithme `FRSelect`, vous pouvez consulter des sources externes pour déterminer ces complexités (en les citant).

2. En adaptant l'analyse de complexité du cas moyen du `Quicksort` vue au cours et sous les mêmes hypothèses, montrez que la complexité moyenne de `Quickselect` est $\Theta(\mathtt{N})$ dans le cas où `k` est fixé à 0 (recherche du minimum).

3. Un algorithme de sélection est stable s'il respecte l'ordre des éléments identiques, c'est-à-dire s'il tient compte de l'ordre initial de ces éléments identiques pour déterminer le `k`-ième plus petit. Pour chacun des 4 algorithmes, précisez s'ils sont stables ou pas en justifiant brièvement votre réponse.

Expérimentations

1. Mesurez les temps de calcul et le nombre de comparaisons des 4 algorithmes dans le cas de tableaux aléatoires, de tableaux ordonnés de manière croissante et décroissante et de tableaux ne contenant que des valeurs identiques pour la recherche de la médiane ($\mathtt{k} = \lfloor \mathtt{N}/2 \rfloor$ et du 10-ième centile ($\mathtt{k} = \lfloor \mathtt{N}/10 \rfloor$). Reportez ces temps de calcul dans deux tables (une pour chaque valeur de `k`) au format ci-dessous:

Type de tableau	aléatoire			croissante			décroissante			constante		
Taille	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6	10^4	10^5	10^6
SelectionSelect HeapSelect QuickSelect FRSelect												

2. Commentez ces résultats. Pour chaque type de tableau:

   - comparez l'évolution des temps de calcul en fonction de la taille du tableau aux complexités théoriques,
   - commentez l'ordre relatif des différents algorithmes,
   - discutez de l'impact du paramètre `k`.

Remarques:

- Les fonctions pour générer les 4 types de tableaux vous sont fournies dans le fichier `IntArray.c` et le fichier `main.c` réalise une expérience pour chaque type de tableau.
- Les temps reportés doivent être des temps moyens établis sur base de 10 expériences au moins. C'est important en particulier dans le cas d'un tableau aléatoire.
- Précisez bien dans vos commentaires les spécificités de vos implémentations qui ont une influence sur les résultats théoriques (par exemple, le choix du pivot et la fonction de partitionnement utilisée par le `QuickSelect`).

Date de remise et soumission

Le projet est à réaliser ***seul ou en binôme*** pour le ***dimanche 23 Mars 2025 à 23h59*** au plus tard. Le projet est à remettre via Gradescope (http://gradescope.com, voir code cours sur Ecampus)

Les fichiers suivants doivent être remis:

1. votre rapport (6 pages maximum) au format PDF et nommé `rapport.pdf`. Soyez bref mais précis et respectez bien la numération des (sous-)questions;

2. les quatre fichiers `SelectionSelect.c`, `HeapSelect.c`, `QuickSelect.c` et `FRSelect.c`.

Respectez bien les extensions de fichiers ainsi que les noms des fichier `*.c` (en ce compris la casse). N'incluez aucun fichier supplémentaire.

Un projet non rendu à temps recevra une cote globale nulle. La soumission du projet suppose que vous avez pris connaissance des règles en terme de plagiat rappelées sur Ecampus. En cas de plagiat[3] avéré, le groupe se verra affecter une cote nulle à l'ensemble du cours.

Les critères de correction sont précisés sur Ecampus.

Bon travail !

[3]Des tests anti-plagiat seront réalisés.