Programmation avancée

Projet 2: Filtre médian

Pierre GEURTS – Jean-Michel BEGON – Romain MORMONT

25 octobre 2018

L'objectif de ce projet est de vous familiariser avec la conception, l'implémentation et l'analyse de structures de données dans un contexte de traitement du signal, et plus particulièrement du filtre médian pour des signaux uni-dimensionnels.

Le but sera de comparer les performances d'un point de vue pratique et théorique de différentes implémentations.

1 Filtre médian

Le filtre médian consiste à faire glisser une fenêtre de taille impaire fixée sur un signal et à remplacer la valeur au centre de la fenêtre par la médiane des valeurs dans celle-ci (figure 1).

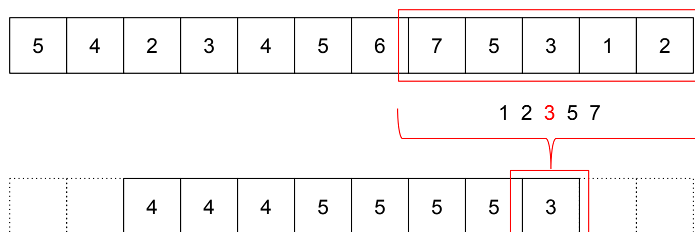


FIGURE 1 – Filtre médian unidimensionnel

Plus formellement, soit un signal discret $S = s_1, s_2, \ldots, s_N$, l'application d'un filtre médian de taille w impaire ($w \leq N$) à S donne le signal $M = m_1, m_2, \ldots m_{N-w+1}$ tel que

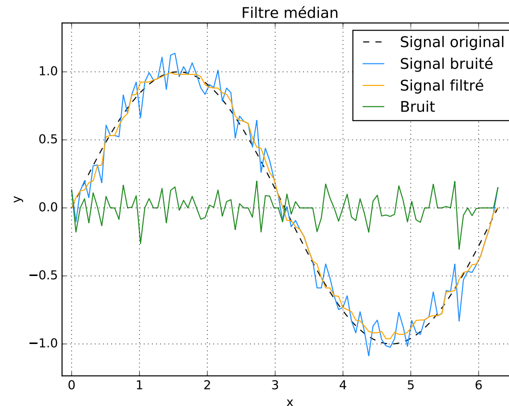
$$m_i \text{ est la médiane de } s_i, s_{i+1}, \ldots, s_{i+w-1}, \quad i = 1, \ldots, N - w + 1$$

Le filtre médian a de nombreuses applications en traitement du signal. Son application la plus répandue est le filtrage des hautes fréquences, comme le bruit impulsionnel (voir figure 2a pour une illustration sur des images). Cette étape peut également servir à la soustraction de fond (figure 2), lorsqu'on veut analyser uniquement le bruit.

2 Implémentation

Un solution naïve à ce problème consisterait à trier les valeurs de la fenêtre, à chaque glissement, la médiane étant alors l'élément central du tableau ainsi trié. On vous propose

(a) Débruitage (source : `https://en.wikipedia.org/wiki/Median_filter`)



(b) Background soustraction

FIGURE 2 – Exemples d'application du filtre médian

d'implémenter et de comparer trois approches visant à améliorer les performances par rapport à cette approche naïve. La première remplace le tri de l'approche naïve par un algorithme plus efficace de recherche de la médiane basée sur une modification de l'algorithme de tri rapide. La deuxième tente de faire le tri plus rapidement à chaque glissement en exploitant le fenêtre triée précédente. Enfin, la troisième est basée sur l'utilisation de deux structures de tas.

Dans chaque cas, l'implémentation est décomposée en deux parties.

La première, que nous vous fournissons, concerne la logique de déplacement de la fenêtre sur le signal au travers de la fonction SIGNALMEDIAN définie dans le fichier `Signal.h`.

La seconde partie concerne le calcul de la médiane. Celui-ci passe par l'implémentation de la structure `MedianQueue` définie dans le header du même nom, et qui comporte notamment les fonctions suivantes :

MQCREATE(A, W) qui crée une `MedianQueue` et l'initialise avec les w premiers éléments du tableau A.

MQMEDIAN(M) qui renvoie la valeur de la médiane des éléments contenus dans la structure M. Cette étape doit être $\Theta(1)$.

MQUPDATE(M, X) qui met à jour la structure M en remplaçant la plus ancienne valeur qui s'y trouve par x.

Nous vous fournissons l'implémentation naïve (`NaiveMedianQueue.c`), ainsi qu'une application (`main.c`) permettant de lire, charger et filtrer un signal.

Nous vous fournissons également deux fichiers signaux, celui de la figure 1 et un signal représentant la température horaire à la station d'Uccle entre le 23 mars 2017 et le 23 mars 2018. Pour information, les fichiers signaux (extension `.sgl`) sont des fichiers texte composés de deux lignes. La première contient un entier non-signé correspondant à la taille N du signal. La seconde ligne contient N réels (avec le point comme séparateur décimal) séparés par un espace, correspondant aux valeurs du signal.

2.1 Variante basée sur le QUICKSORT

L'idée de cette variante est de remplacer le tri complet de la fenêtre tel que fait dans l'approche naïve par un algorithme plus efficace de recherche de la médiane. En s'inspirant de

l'algorithme du QUICKSORT, il est possible de calculer la médiane d'un tableau sans avoir à trier celui-ci entièrement. En effet, connaissant les bornes du sous-tableau à trier, ainsi que la position du pivot, on peut déterminer dans quel sous-tableau va se situer la médiane et donc éviter de faire un des deux (ou les deux) appels récursifs.

Cette solution doit faire l'objet du fichier `QuickMedianQueue.c`.

2.2 Variante basée sur le tri

Dans l'approche naïve, la nouvelle fenêtre est retriée complètement à chaque glissement. Dans cette seconde variante, on vous demande de proposer une solution plus efficace en se basant sur le fait qu'une nouvelle fenêtre ne diffère de la précédente que sur un élément. L'idée est de conserver un tableau avec les éléments de la fenêtre courante trié et, lors d'un appel à MQUPDATE, de remplacer l'élément le plus ancien par le nouveau dans ce tableau et réordonner les éléments le plus efficacement possible en prenant en compte le fait que le tableau était trié avant le remplacement.

Cette solution doit faire l'objet du fichier `SortedMedianQueue.c`.

2.3 Variante à base de tas

Cette variante repose sur l'utilisation d'un tas-max pour stocker les $w/2$ plus petits éléments de la fenêtre de taille w et d'un tas-min pour stocker les autres éléments, permettant ainsi d'accèder rapidement à la médiane (*Où se trouve-t-elle ?*). Le but de la fonction MQUPDATE(M, x) est de mettre à jour efficacement la structure en maintenant la propriétés de tas des deux tas, ainsi que leurs tailles. Pour ce faire, vous devrez implémenter un mécanisme efficace de remplacement d'une valeur dans un tas. Avec cette approche, il est possible d'obtenir une complexité strictement inférieure à celle obtenue avec les deux autres approches (dans le pire cas). Cette solution doit faire l'objet du fichier `HeapMedianQueue.c`.

3 Analyse

Dans votre rapport, nous vous demandons de répondre aux questions suivantes :

Analyse théorique

1.1 Donnez le pseudo-code de la fonction MQUPDATE(M, X) dans le cas du tri. Tout en précisant ceux-ci, analysez la complexité en temps de la fonction par rapport à w, la taille du filtre, au pire et au meilleur cas.

1.2 Donnez le pseudo-code de la fonction MQUPDATE(M, X) dans le cas de la variante du quicksort. Tout en précisant ceux-ci, analysez la complexité en temps de la fonction par rapport à w, la taille du filtre, au pire et au meilleur cas.

1.3 Donnez le pseudo-code d'une fonction HEAPREPLACE qui remplace une valeur par une autre dans un tas. Tout en précisant ceux-ci, analysez la complexité en temps de la fonction par rapport à n, le nombre d'éléments dans le tas, au pire et au meilleur cas.

1.4 Donnez le pseudo-code de la fonction MQUPDATE(M, X) dans le cas des tas. Tout en précisant ceux-ci, analysez la complexité en temps de la fonction par rapport à w, la taille du filtre, au pire et au meilleur cas.

1.5 Analysez la complexité au pire et meilleur cas de la fonction SIGNALMEDIAN en fonction de l'implémentation du filtre par rapport à N, la taille du signal et w, la taille du filtre.

Analyse empirique

2.1 Comparez empiriquement et discutez les différentes implémentations sur de grands signaux et pour différentes tailles de filtre.

2.2 Discutez vos résultats empiriques au regard des résultats théoriques.

4 Deadline et soumission

Le projet est à réaliser **par groupe de deux étudiant(e)s** pour le **1 décembre 2018, 23h59** au plus tard. Il doit être soumis via la plateforme de soumission (`http://submit.montefiore.ulg.ac.be/`).

Le projet doit être rendu sous la forme d'une archive `tar.gz` contenant :

1. Votre rapport (8 pages maximum) au format PDF. Soyez bref mais précis et respectez bien la numération des (sous-)questions.
2. Les fichiers `HeapMedianQueue.c`, `QuickMedianQueue.c` et `SortedMedianQueue.c`.

Comme dans le projet précédent, l'algorithme permettant de calculer la médiane est choisi à la compilation. Par exemple, vous pouvez compiler le fichier `main.c` avec la variante naïve à l'aide de la commande :

```
gcc main.c Signal.c NaiveMedianQueue.c --std=c99 -lm -o filter
```

Le programme peut alors s'utiliser comme suit :

```
./filter -o filtered_sig.sgl temperatures23.03.17_23.03.18.sgl 25
```

En outre, nous utiliserons les flags de compilation habituels (`--pedantic -Wall -Wextra -Wmissing-prototypes -DNDEBUG`).

Ces contraintes impliquent que

- — Les noms des fichiers doivent être respectés.
- — Le projet doit être réalisé dans le standard C99.
- — La présence de *warnings* impactera négativement la cote finale.
- — Un projet qui ne compile pas sur les machines ms8xx recevra une cote nulle (pour la partie code du projet).

Un projet non rendu à temps recevra une cote globale nulle. En cas de plagiat avéré, l'étudiant se verra affecter une cote nulle à l'ensemble des projets.

Les critères de correction sont précisés sur la page web des projets.

Bon travail !