

TP2 ASD - Recherche auto-adaptative

Le but du TP consiste à simuler trois types de recherche auto-adaptative (algorithmes qui s'adaptent au fil de leur utilisation de façon à optimiser la recherche des clés les plus souvent demandées) et à en estimer le coût.

1- Programmation et test des trois méthodes de recherche auto-adaptative.

Chacune de ces méthodes cherche une clé dans un tableau et, quand la clé est trouvée :

- **Méthode a)** : avance d'une place la clé cherchée dans le tableau (en l'échangeant avec celle qui la précède) ;
- **Méthode b)** : met en tête de tableau la clé cherchée (sans changer l'ordre des autres clés) ;
- **Méthode c)** : incrémente le nombre d'accès à la clé, puis trie le tableau selon l'ordre décroissant des accès.

Chacune des 3 méthodes travaille sur son tableau ELEM de taille N (avec N=15 par exemple) initialisé par $ELEM[i]=i$, $i \in [0, N-1]$.

Pour la dernière méthode, un tableau CPT de taille N, correspondant au nombre de recherches déjà effectuées pour chaque élément de ELEM, est initialisé à 0.

⇒ En plus des 3 méthodes de recherche, penser à écrire une procédure unique contenant toutes les initialisations de tableaux, et également à écrire une procédure d'affichage d'un tableau.

2- Evaluation de la complexité des trois méthodes en comptant le nombre de comparaisons d'éléments de tableau effectuées par chacune d'elles.

⇒ Modifier les procédures précédentes afin d'intégrer les calculs de complexité.

3- Analyse du comportement et de la complexité des trois méthodes dans un contexte d'utilisation adapté (i.e. certains éléments sont plus souvent recherchés que d'autres)

Pour cela, on effectue un certain nombre de recherches de clés dans les trois tableaux distincts, mais identiquement initialisés, selon les 3 méthodes a), b) ou c) (associées aux différents tableaux).

Les 3 algorithmes à analyser sont des algorithmes de recherche auto-adaptative ; il faut donc être dans le cas où l'on a effectivement besoin de ce type de recherche, c'est-à-dire qu'il existe des éléments que l'on cherche plus souvent que d'autres.

Des fonctions permettant de simuler l'utilisation d'un tel environnement sont disponibles, via le fichier *TP2.zip* sur Moodle, dans le fichier *rech_auto.h* avec leur implémentation dans le fichier *rech_auto.c*.

- la fonction *void init_env_psal(int aff)*
pour initialiser l'environnement (avec *aff*=1 pour le mode verbeux),
- la fonction *int tirage_cle_env_psal(int aff)*
pour tirer la prochaine clé à rechercher selon les règles de l'environnement (avec *aff*=1 pour le mode verbeux).

⇒ Modifier votre programme principal en appelant les deux fonctions précédentes afin de pouvoir comparer les trois méthodes dans un contexte d'utilisation adapté.

Pour aller plus loin : principe de la simulation de ce contexte d'utilisation

La simulation est basée sur une méthode de tirage des clés à chercher pseudo-aléatoire qui, bien qu'utilisant *random* (et peu importe les éléments qu'elle tire), génère plus souvent certaines clés.

Pour cela, on définit un tableau PSAL de taille N, initialisé (pseudo-)aléatoirement par des valeurs strictement croissantes et strictement supérieures à 0, contenant les fréquences cumulées d'accès aux clés (cf. initialisation de PSAL).

À chaque tirage, pour obtenir la clé à rechercher, on tire un nombre (pseudo-)aléatoire k compris entre 0 et PSAL[N-1]. L'indice i du tableau PSAL tel que $PSAL[i-1] < k \leq PSAL[i]$ correspond à cette clé (avec *i*=0 si $k \leq PSAL[0]$).

Initialisation de PSAL :

On initialise PSAL par ordre croissant, que l'on dit être des fréquences cumulées.

Par exemple :

8	15	19
---	----	----

Lors du tirage suivant du nombre aléatoire k compris entre 0 et $PSAL[N-1] = PSAL[2] = 19$, si $k = 0, 1, \dots, 8$, on considère que c'est la case 0 qui est concernée, donc la clé 0 ;

si $k = 9, 10, \dots, 15$, on considère que c'est la case 1 qui est concernée, donc la clé 1 ;

si $k = 16, 17, \dots, 19$, on considère que c'est la case 2 qui est concernée, donc la clé 2.

Donc la première case, qui a déjà été la plus tirée, a 9 chances de se voir proposer de nouveau, alors que la troisième, qui a déjà été tirée 4 fois (19-15), n'a que 4 chances d'être proposée.

Par conséquent, on distingue encore plus un élément qui a beaucoup été tiré par rapport aux autres éléments.