TP d'Algo n° 2

EPITA ING1 2013; E. RENAULT

Objectif : Implémenter une recherche dichotomique pour réduire le nombre de comparaisons du tri par insertion, comme vu en TD ce matin.

1 Les aventuriers de l'élément perdu...

Dans un premier temps nous voulons implémenter une simple recherche linéaire qui est de complexité $O(n)$ puisqu'elle parcourt au maximum tous les éléments du tableau.

Question 1 : Écrivez la fonction `linear_search()` qui cherche l'élément `element` dans le tableau non trié `tab` contenant `count` éléments de taille `size`, avec la fonction de comparaison `compare`. La fonction retourne un pointeur sur l'élément trouvé, `NULL` sinon.

```
void* linear_search(void* element,
                    void* tab,
                    size_t count,
                    size_t size,
                    int (*compare)(void* a, void* b));
```

Question 2 : Écrivez la fonction `binary_search()` qui effectue une recherche dichotomique de `element` dans le tableau `tab`, **supposé trié**. Attention, comme dans le TD, cette fonction doit retourner soit un pointeur sur la position où se trouve l'élément recherché dans le tableau, soit un pointeur sur la position où il faudrait insérer l'élément.

```
void* binary_search(void* element,
                    void* tab,
                    size_t count,
                    size_t size,
                    int (*compare)(void* a, void* b));
```

2 A la recherche du temps perdu...

Nous voulons maintenant optimiser l'algorithme de tri insertion en utilisant la recherche dichotomique pour déterminer la position de la clef.

Question 3 : Écrivez la fonction `generic_binary_insert_sort()` qui effectue cette recherche dichotomique dans le tableau avant de décaler tous les successeurs. (Faites-le en un seul appel à `memmove()` !)

```
void generic_binary_insert_sort(void* tab,
                                size_t count,
                                size_t size,
                                int (*compare)(void* a, void* b));
```

3 Recherché par Interpol !

On a coutume de dire que la recherche dichotomique est celle qu'on pratique quand on cherche dans un dictionnaire. Ce n'est pas tout à fait vrai. Par exemple quand j'ouvre mon dictionnaire pour chercher ce qu'est un *xyste*, je ne commence pas ma recherche au milieu du dictionnaire. Je sais que les mots qui commencent par `x` sont beaucoup plus proche de la fin que du début.

Dans cette partie nous appliquons cette approche à la recherche dans un tableau d'entier triés. On supposes que nos entiers sont répartis uniformément. En fonction des valeurs des entiers de début et fin du sous-tableau considéré, nous pouvons interpoler (avec une règle de trois qu'il vous faut trouver) une position proche de l'entier recherché. À partir de là, on jette une partie du tableau et on recommence.

Question 4 : Écrivez `interpolation_search()` qui retourne un pointeur sur l'élément recherché, ou un pointeur sur l'endroit où il faudrait l'insérer.

```
int* interpolation_search (int element,
                           int* tab,
                           size_t count,
                           int (*compare)(void* a, void* b));
```

Testez cela sur une une recherche dans un tableau d'entiers aléatoire. Comparez le nombre de comparaisons effectuées par `binary_search()` et `interpolation_search()`.

Question 5 : Votre fonction `interpolation_search()` marche-t-elle sur un tableau contenant un unique élément n répété fois ? Corrigez si ce n'est pas le cas.

Question 6 : Écrivez `interpolation_insert_sort()` basée sur `interpolation_search()`.

```
void interpolation_insert_sort(int* tab,
                               size_t count,
                               int (*compare)(void* a, void* b));
```

Comparez le nombre de comparaisons effectués par `binary_insert_sort()` et `interpolation_insert_sort()` sur un tableau d'entiers aléatoires.

4 Pour ceux qui arrivent ici...

Prenez de l'avance sur le TP3, il est long !