Feuille de TD N^o 4 : Tableaux et quelques Tris

1 Hachage

Un hétérogramme est un mot ne contenant pas deux fois la même lettre. Quels sont les plus longs que vous trouvez ?

Rangez les lettres de a à k dans un tableau T[1..13].
Le faire avec le hachage avec listes chaînées,
le hachage linéaire, le double hachage.
On suppose que h et $h2$ de ces lettres sont :

lettre	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*
h(*lettre*)	13	8	3	8	4	13	2	2	5	9	1
$h2$(*lettre*)	5	6	11	4	3	12	1	9	4	7	3

2 Sélections

1. (facultatif) Écrire un algorithme qui rend les éléments minimal ET maximal d'un tableau de n entiers. Faire moins de environ 2n comparaisons. Donner un code. Est-il possible de se passer d'un tableau annexe ? De ne pas modifier le tableau en entrée ?
2. Que fait l'algorithme ci-dessous ? Quelle est, en nombre de comparaisons entre éléments du tableau, à O(1) près, la complexité au pire ? au mieux ? en moyenne (en supposant les éléments distincts, et les permutations équiprobables) ?

```
si T[0] < T[1] alors { max <- T[1]; sousmax <- T[0]; } sinon { max <- T[0]; sousmax <- T[1]; }
pour k de 2 a taille-1 faire si T[k] > sousmax
                             alors si T[k] > max alors { sousmax <- max ; max <- T[k] ; }
                                                 sinon { sousmax <- T[k] ; }
```

3. Donner un algorithme qui rend le sous-maximal d'un tableau de n entiers en $n + \theta(ln(n)) + O(1)$ comparaisons au pire.

3 Dichotomie et variantes

1. (facultatif) La recherche trichotomique procède de façon similaire a la recherche dichotomique, mais en découpant les segments en 3. A chaque itération, on compare avec l'élément situé au tiers, puis si besoin avec l'élément situé au deux tiers. Comparez le nombre d'itérations, et le nombre de comparaisons en moyenne et au pire avec des recherches dichotomique et trichotomique.
2. (facultatif) La recherche dans un dictionnaire diffère de la dichotomie en deux points : On fait une estimation de l'emplacement du mot (on n'ouvre pas un dico au milieu si on cherche "zèbre") et on ne sait pas viser précisément un emplacement (l'humain ne sait pas ouvrir du premier coup un livre de 1000 pages à la page 500). Notez que la longueur de l'intervalle de recherche n'est pas une mesure pertinente de la distance au but, si vous cherchez "zèbre"et tombez sur "zébu", vous êtes proche, pourtant l'intervalle de recherche [abaca .. zébu] est grand. C'est pourquoi nous regarderons la distance au but du dernier mot considéré, donc ici la distance [zèbre, zébu]. On modélise la recherche en disant que si l'on vise le but depuis un certain mot à distance d, on se trompe de 10% et l'on va à distance 0.9*d ou 1.1*d. Estimez la complexité de cette recherche, comparez avec la dichotomie
3. (facultatif) T[1..N] contient des réels de [0..1] tirés au hasard puis triés. Pour rechercher un réel r dans ce tableau, on procède comme en dichotomie, mais au lieu de regarder au milieu de l'intervalle courant [i,j], on fait une interpolation linéaire (on regarde en $i + (j - i) * (r - T[i])/(T[j] - T[i]) + O(1)$). De nouveau, la longueur de l'intervalle de recherche est non pertinente et on s'intéressera à la distance du but au bord le plus proche. On modélise cette recherche en disant que si le mot visé est à distance d d'un bord, on se trompe de $\sqrt{d}$ (une histoire d'écart-type). Estimez la complexité de cette recherche. Comparez.
4. (facultatif) Donnez le code de la recherche de la position du dernier élément inférieur strict à x dans un tableau trié T[0..N-1].

4 Insertion dans une liste triée

L réels triés sont rangés dans les premières cases d'un tableau T[0..M-1], avec M>L. On considère le code ci-dessous pour insérer un nouvel élément x. Quelle est la complexité au pire en nombre de comparaisons entre floats de cet algorithme ? Peut-elle être améliorée ? Si oui, comment, et quelle est la nouvelle complexité ? Itou avec la complexité au pire en nombre d'affectations de floats ?

```
L ++             \* il y a un element de plus *\
T[L-1] <- x        \* le nouvel element est range a droite *\
           \* puis il est deplace vers la gauche jusqu'a sa position finale : *\
P <- L-1
tant que (P > 0) et ( T[P] < T[P-1] )
faire     {  echanger T[P] et T[P-1]
              P-- }
```

5 Tri de 5 éléments (facultatif)

Montrez qu'un algorithme de tri à base de comparaisons de 5 éléments nécessite au moins 7 comparaisons au pire. Existe-t-il un algorithme qui effectue 7 comparaisons au pire ?

6 Parcours de tableaux

1. Une pile est gérée par un tableau[0..M-1] et un entier P hauteur de pile. Si P dépasse M, on alloue un nouveau tableau de longueur $2M$, on recopie le vieux tableau dans le nouveau et on continue avec le nouveau. Quel sont les coûts au pire et amorti de "empile" ? On souhaite que M reste $O(P)$, même après des dépilements. Quelle nouvelle action introduire ?
2. Écrire le code de la procédure `decale (inout T[], i,j, k)` qui décale de k cases la zone T[i..j] du tableau T[0..N-1]. La zone "glisse", et telle Attila, efface tout et met des 0 sur son passage. Le code supposera qu'il n'y a pas de débordements de tableau ($0 \leq i \leq j \leq N-1 \wedge 0 \leq i+k \wedge j+k \leq N-1$). Exemples : Avec l'appel `decale(T[],4,6,4)`, le tableau [2,10,8,5,9,6,3,1,4,7,12,13] devient [2,10,8,5,0,0,0,0,9,6,3,13]. Avec l'appel `decale(T[],4,6,-2)`, le même tableau devient [2,10,9,6,3,0,0,1,4,7,12,13].
3. Écrire un algorithme qui prend en entrée un mot donné par un tableau $T[0..N-1]$ et rend vrai ssi il contient un facteur carré. Un carré est un mot $uu, u \neq \epsilon$, le mot $piftoctoccccpapaf$ contient six carrés, *abacbcabacabcbacbcabcbacabacbcabcba* est sans carré.
4. Écrire une procédure qui prend en entrée deux tableaux triés croissants T1[0..n1-1] et T2[0..n2-1] et écrit dans un troisième tableau T3 la fusion (union triée avec multiplicité) de ces deux listes.
5. Un tableau T[0..N-1] contient N objets $x_0, ..., x_{N-1}$ numérotés de 0 à N-1 (par exemple, N=13, et les objets sont des cartes à jouer). Le numéro d'un objet est donné par la fonction `Num` (On a donc $Num(x_j) = j$). On souhaite déplacer les objets de façon à ce que chaque objet x soit placé dans la case $T[Num(x)]$. Faire un premier code avec un tableau annexe $V[0..N-1]$, puis un second sans tableau annexe. Quelles sont les complexités de vos algos ?
6. Proposez des algos pour déterminer si tous les éléments d'un tableau $T[1..N]$ sont différents.

7 Quicksort (tri rapide)

1. Donnez une version de la procédure `separe` (inout T[], in d, in f, out pp) qui fait une passe de gauche à droite. Temporairement, il y aura le pivot, puis des éléments plus petits, puis des plus grands, puis des éléments non regardés.
2. Écrire une version itérative de Quicksort. (facultatif) Quel est l'ordre de grandeur de la hauteur maximale de la pile lors de l'exécution de cette version ? Modifier votre algorithme pour réduire la hauteur maximale de pile au pire.
3. Donnez un algorithme de séléction inspiré de quicksort qui prend en entrée `T[0..N-1]`, un entier k et rend le k^{eme} élément (celui qui serait en position k si on triait le tableau). Donnez la complexité au pire et estimez celle en moyenne.
4. (facultatif) SurpriseSelection procède comme QuickSelection, mais on complique la manière de chercher le pivot:

   On regroupe les n éléments par paquets de 5 éléments. On cherche le médian de chacun de ces paquets, puis on appelle récursivement SurpriseSelection pour chercher le médian de ces médians. Ce dernier est le pivot.

   Si l'intervalle de départ est de taille k, quelle taille au max et au min ont les deux intervalles résultats de la séparation ? En déduire une formule de récurrence pour la complexité au pire de cette sélection. Estimer la complexité au pire de cette sélection.

8 Heapsort (tri par tas)

1. Complétez le tableau ci-dessous pour en faire un tas contenant les entiers de 1 a 12. Utiliser les algos du cours pour insérer une nouvelle occurrence de 3 puis retirer l'élément minimal. Faire des dessins pour justifier vos réponses.

			10	3		5					7

2. Écrire la procédure EstTas(T[0..H-1]) qui rend vrai ssi T[0..H-1] est un tas.
3. Lorsque l'on remonte ou descend un élément dans un tas, on effectue k échanges, où k est la distance dans l'arbre entre les positions de départ et d'arrivée. Chaque échange se fait en 3 affectations, on effectue donc $3k$ affectations. Améliorez.
4. Lorsque l'on descend un élément dans un tas depuis la racine, en règle générale, cet élément va arriver dans un niveau à distance $\theta(1)$ du fond. Utiliser cette propriété pour améliorer la complexité en nombre de comparaisons.
5. Donnez la procédure `remplace(inout T,t,i,x)`, qui suppose que `T[1..t]` est un tas et que $1 \leq i \leq t$, et a pour effet de remplacer $T[i]$ par x, puis de déplacer les éléments de sorte que `T[1..t]` soit toujours un tas.

9 Fusion de k listes

On veut fusionner k listes triées en une seule. Fusionner deux listes de longueur l_1 et l_2 coûte $l_1 + l_2 - 1$ comparaisons.

1. Montrez qu'on peut fusionner les k listes en $\sim (n * ln(k))$ comparaisons, où n est la somme des longueurs des listes. Comment s'appelle l'algorithme obtenu si les listes initiales sont des singletons ?
2. $k = 3$, on fusionne deux listes, puis le résultat avec la troisième. Quelles listes choisissez-vous de fusionner initialement ?
3. Soit S une séquence de fusions. Soit A_S l'arbre valué par des listes : les feuilles sont les listes de départ, un nœud interne marque la fusion des listes fils. Donnez la complexité de S en fonction des longueurs et des positions dans A_S des listes de départ.
4. Montrez qu'il existe une séquence de fusion optimale dans laquelle on commence par fusionner les deux listes les plus courtes.
5. Donnez un algorithme simple permettant de déterminer une séquence optimale de fusions. Précisez la structure de données avec laquelle vous mémorisez les longueurs des listes qu'il vous reste à fusionner pour une fusion quelconque, et pour le tri fusion.

10 Bucket sort (facultatif)

Pour trier N réels aléatoires de [0,1[, on prend un tableau[0..N-1] de listes chaînées initialisées à NULL, on ajoute chaque élément à trier x en case T[⌊x*N⌋], puis on trie chaque case du tableau du tableau avec le tri par séléction simple, puis on concatène les cases du tableau. Quelle est la complexité en moyenne de cet algorithme ? Cela contredit-il un théorème du cours ?