Algorithmique TD4
L3 Informatique – ENS Cachan

Guillaume Bury

31 janvier 2017

Exercice 1

On souhaite implémenter une pile à l'aide d'un tableau. Si on fixe la taille du tableau a priori, on limite la taille de la pile ; de plus, beaucoup de place est perdue lorsqu'il y a trop peu de données. Pour résoudre ces problémes, on utilise des tableaux dynamiques dont la taille peut changer de telle sorte qu'au moins la moitié du tableau est toujours remplie. L'opération qu'on veut implémenter est l'empilement : elle consiste à insérer l'élément à la premiére place libre lorsque le tableau n'est pas plein ; sinon, on crée un nouveau tableau de taille double du précédent, on copie dedans tous les éléments du premier tableau puis on empile finalement l'élément dedans.

On néglige le coût de l'allocation d'un tableau. Le coùt à mesurer est donc le nombre de copies effectuées.

1. Implémenter la fonction `empiler`.
2. Quelle est la complexité de la fonction `empiler` dans le pire des cas ?
3. Calculer la complexité amortie de la fonction `empiler` lorsqu'on commence par une pile vide.
4. On considère une procédure différente `empiler'` qui consiste à augmenter la taille du tableau d'une constante k plutôt que de la doubler, lorsque le tableau est plein. Calculer la complexité (avec potentiel) de cette procédure.
5. Supposons qu'on veuille également une opération de dépilement `depiler`, qui consiste à effacer le dernier élément du tableau. Si la suppression du dernier élément résulte en un tableau à moitié plein, on crée un tableau de taille divisée par deux et on copie les éléments du tableau dedans. Quelle est la complexité (avec potentiel) des opérations `empiler` et `depiler`.
6. On propose de diviser par deux la taille du tableau quand le nombre d'éléments descend en dessous d'un quart. Calculer la complexité de cette nouvelle version de `depiler`.

Exercice 2

1. Donner l'algorithme de tri rapide
2. Calculer sa complexité en moyenne, en supposant une répartition uniforme des permutations du tableau.

Exercice 3

On considère un ensemble d'objets positionnés dans le plan euclidien. Ces objets sont assimilés à des disques ayant tous le même diamètre d et sont repérés par les coordonnées (abscisse et ordonnée) de leur centre. Formellement, on dispose en entrée d'un tableau T de dimension n dont chaque élément $T[i]$ indique les coordonnées du centre d'un objet. On note $T[i].x$ son abscisse et $T[i].y$ son ordonnée.

Le problème consiste à décider si deux objets du tableau sont en collision, c'est-à-dire si leurs centres sont distants de d ou moins.

On considère que les opérations suivantes se font en temps constant : accéder aux coordonnées du centre d'un objet, comparer les abscisses ou les ordonnées de deux objets, calculer la distance entre deux points, comparer des indices de tableau.

1. Proposer un premier algorithme très simple qui prend en entrée le tableau T et le diamètre d des objets, et qui retourne *vrai* s'il y a (au moins) une collision, *faux* sinon. Évaluer sa complexité en temps et en espace dans le pire des cas.

On tente maintenant une approche de type « diviser pour régner ». On génère deux sous-problèmes en prenant d'une part les $\lfloor \frac{n}{2} \rfloor$ objets de plus petite abscisse et d'autre part les $\lceil \frac{n}{2} \rceil$ objets de plus grande abscisse. On suppose que les deux sous-problèmes ont été traités sans qu'aucune collision n'ait été détectée. Il reste à vérifier l'absence de collision entre un objet de petite abscisse et un objet de grande abscisse.

2. Montrer que les objets à considérer ont leur centre dans une bande verticale de largeur $2d$.
3. On note p le nombre d'objets qui ont leur centre dans cette bande. Par quelle fonction de n peut-on majorer p ?
4. Montrer comment tester l'absence de collisions entre ces objets en temps $O(p)$.
5. Résumer les points précédents en écrivant l'algorithme complet, puis donner sa complexité en temps.
6. Montrer comment adapter l'agorithme pour qu'il retourne les deux objets les plus proches, même lorsqu'il n'y a pas de collision.

Exercice 4

On voudrait rechercher un élément x donné dans un tableau A non trié. On considère les deux algorithmes suivants :

— On choisit une permuation du tableau selon une loi de probabilité uniforme, puis on teste tous les éléments de gauche à droite.

— On choisit un indice i selon une loi de probabilité uniforme. Si $A[i] = x$, alors on retourne oui, sinon on recommence.

1. Proposer une modification de l'algorithme aléatoire afin qu'il termine si tous les indices ont été testés.
2. Calculer l'espérance du nombre d'itérations de l'algorithme aléatoire dans les cas suivants :
   (a) On suppose que x n'est pas dans A.
   (b) On suppose qu'il y a un seul indice i tel que $A[i] = x$.
   (c) On suppose qu'il y a $m > 0$ occurences de x dans A.
3. On suppose qu'il y a $m > 0$ occurences de x dans A, soit $k \geq 1$, pour chaque algorithme, caluler la probabilité que le nombre d'itérations de l'algorithme soit supérieur à k.
4. En déduire une comparaison de l'espérance du nombre d'itérations des deux algorithmes, puis comparer les deux algorithmes.