UdeM –DIRO- IFT2015H16

TP2 : Monceaux *p*-aires

Dans ce *travail plaisant* (TP), vous allez coder des modifications au monceau, comparer leurs performances relatives expérimentales et théoriques et faire un rapport décrivant vos résultats et analyses.

Introduction

Le monceau est une adaptation du tri par insertion (plus exactement du tri à bulle ou *bubble sort*) à une structure arborescente. Dans le tri par insertion, on ajoute un élément à un tableau ordonné en trois étapes :1- localiser la case d'insertion, 2- déplacer tous les éléments à partir de cette place d'une case vers la droite, et finalement 3- insérer l'élément à sa destination. Lorsqu'on retire le plus petit élément du tableau, il est nécessaire de déplacer tous les éléments restants d'une case vers la gauche. Ainsi, l'insertion est O(n) et la suppression est également O(n). Pas très efficace pour une file de priorité ! Quand on y pense un peu, on voit bien que le monceau fonctionne exactement de la même manière mais sur un arbre binaire balancé. Lors de l'insertion, on remonte l'arbre à partir de la première feuille libre en échangeant les valeurs des feuilles tant que la nouvelle valeur est prioritaire sur la valeur contenue dans la feuille candidate décrivant un chemin de fouille. Ce chemin de fouille est borné par $\log_2 n$ puisqu'il est inscrit dans un arbre balancé et l'insertion est donc $O(\log_2 n)$. La suppression de l'élément de priorité maximale est également $O(\log_2 n)$: on remplace la valeur prioritaire par la valeur dans la feuille en bas et à droite de l'arbre puis on fait descendre cette valeur le long d'un chemin tant qu'il existe un nœud dont la valeur est de priorité supérieure. Ce chemin est de longueur maximale $\log_2 n$ et la suppression est donc dans $O(\log_2 n)$. Ces processus sont tout à fait similaires à l'insertion dans un tableau trié en utilisant le tri par insertion.

L'implantation du monceau ne nécessite pas de structure de données particulière : pas de nœud, pas de pointeur vers les enfants… Ceci est possible car l'arbre binaire du monceau est *complet*.

Définition (arbre complet) : Un arbre binaire est complet si tous ses niveaux sont remplis sauf possiblement le dernier dont les nœuds sont tous à gauche.

Dans un tel arbre, l'enfant gauche d'un nœud en position i se trouve en position $2i + 1$ et son enfant droit est en position $2i + 2$. Son parent est donc en position $\lfloor (i - 1)/2 \rfloor$. Si un nœud n'a pas un certain enfant c'est que la valeur de l'indice résultant est $\geq n$.

Queue de priorité circulaire

Nous venons de voir que le maintient d'une liste triée est $O(n)$ pour l'insertion et pour la suppression. Qu'en serait-il si nous pouvions réduire ce cout de moitié. Pour y arriver, codez votre queue sur une liste circulaire dans un tableau python. Si la liste est circulaire, la suppression de l'élément de priorité maximale est $O(c)$ puisqu'il n'est plus nécessaire de déplacer les éléments du tableau. Codez votre liste circulaire dans un tableau de taille K. Si les indices commencent à 0, pour accéder au *i-ème* élément de la queue, on utilise $(t + i)\%K$, avec $0 \leq t < K$ donnant l'indice du premier élément de la queue; évidemment $i < K$ puisque autrement la queue serait plus longue que la capacité du tableau.

Monceaux *p-aires*

Pourquoi utilise-t-on des monceaux binaires (*2-aires*)?

Définition (arité) : Le nombre d'enfants des nœuds d'un arbre (et généralement de tout graphe) se nomme arité.

L'insertion dans un monceau binaire est $O(\log_2 n)$ car le plus long chemin de la racine à une feuille comprend $\log_2 n$ éléments. Mais si chaque nœud de l'arbre comprenait 5 enfants, le chemin le plus long serait plus court de $\log_5 n / \log_2 n$. Et si l'arbre avait 100 enfants à chaque nœud ? Aux premiers abords on peut avoir l'impression que l'encodage dans un tableau n'est pas compatible avec de telles arités de l'arbre. Mais il n'en est rien. La raison pour laquelle on utilise p=2 réside-t-elle dans la performance du code? dans la simplicité de l'encodage? À vous de me le dire dans votre rapport.

Vous allez encoder cinq variantes du monceau-p-aire. Avec p=2, p=3, p=4, p=5 et avec p variable donné en argument. Cela vous donnera la chance de vérifier si vous pouvez optimiser les performances avec des valeurs fixes de p relativement à la version où p est variable.

Le problème des *n* plus grands nombres

Nous sommes intéressés par le problème de sélection des n plus grands nombres d'un grand ensemble de k éléments. La solution naïve à ce problème est $O(n + k \log k)$ qui se réduit à $O(k \log k)$ lorsque $n \ll k$. La solution consiste à (1) trier l'ensemble et (b) énumérer les n premiers. Mais lorsque k est grand, d'une part il peut être impossible de mettre l'ensemble en mémoire et d'autre part, la performance peut être améliorée. Nous aimerions une solution qui ne nécessite pas de stocker les données (nous voulons une solution *en-ligne*) et la composante en k de la complexité grand-O doit donc être linéaire. Nous solutionnons ce problème en utilisant une file de priorité de taille maximale n avec la politique *min-heap* (c'est à dire qu'il sera facile d'en retirer la plus petite valeur en tout temps). Lisons un à un les k éléments et si un nouvel élément est plus grand que le plus petit élément contenu dans la file de priorité (ou si la file comprend moins de n éléments) on ajoute cet élément à la file et, si ce faisant nous dépassons la capacité de la file, il faudra en retirer la valeur minimale. À la fin de l'exécution, énumérons les valeurs comprises dans le tableau. Si la file de priorité est $O(\log n)$ pour l'insertion et la suppression, le travail sera donc $O(k \log n)$ résultant en un gain marqué. (Notez que nous n'avons pas nécessairement demandé que les n éléments retenus soient triés à l'affichage.)

Description du travail

Encodez les classes suivantes en python 3.4 : (a) QP_insertion (une queue de priorité sur un tableau en utilisant le tri par insertion), (b) QP_circulaire (une queue de priorité sur un tableau circulaire tel que décrit plus haut), (c) Monceau (le monceau binaire classique ou p-aires avec p=2), (d) MP3 (monceau 3-aire), (e) MP4 (monceau 4-aire), (f) MP5 (monceau 5-aire), (g) MPP (Monceau p-aire avec p variable).

La capacité de la queue de priorité est passée en paramètre au constructeur et la valeur de p est donnée en paramètre pour la classe MPP. Les classes doivent toutes supporter au moins les opérations suivantes : (1) ajouter(valeur) qui ajoute la valeur donnée à la queue de priorité si la capacité de la queue n'est pas atteinte ou si la valeur est plus grande que la plus petite valeur dans la file; suivant() qui retourne l'élément de priorité maximale et réarrange la queue de priorité; et liste() qui retourne le tableau

comprenant toutes ses valeurs. Les valeurs contenues dans ces monceaux sont simplement des nombres et leurs valeurs correspondent à leurs priorités. Les files de priorité encodées sont toutes des min-heap. C'est à dire que les chiffres les plus petits ont des priorités plus importantes.

Écrivez une boucle python qui encode un générateur de points aléatoires. Utilisez-là pour générer $k = 10^8$ valeurs et invoquez vos classes (pour MPP, utilisez $p \in [2, 3, 4, 5, 7, 15, 31]$) avec $n \in [10, 10^2, 10^3, 10^4, 10^5, 10^6]$ et mesurez le temps d'exécution dans chaque cas. Faites un graphique de ces 13 séries de résultats de 6 durées chacune. Ce graphique fait partie de votre rapport (voir ci-bas).

Rapport

Votre rapport doit être remis en format pdf. AUCUN AUTRE FORMAT NE SERA ACCEPTÉ. Il doit comprendre EXACTEMENT les pages suivantes :

(1) Page titre comprenant vos noms, matricule, date de remise, numéro du cours ainsi que la phrase 'TP2 : Monceaux p-aires'.
(2) Votre graphique des résultats. Une ligne joint chaque série de points, une légende exacte et un court texte décrivant le graphique.
(3) Analyse théorique : Pour chaque classe, donnez la complexité des opérations pertinentes (insertion, suppression du plus petit élément, etc).
(4) Analyse numérique : Discutez les résultats obtenus sur le graphique en fonction de vos analyse théoriques. Expliquez toutes les déviations. Que s'est-il passé?
(5) Discussion et conclusion. En particulier, croyez-vous qu'il soit possible de faire un algorithme en-ligne plus rapide pour le problème des n plus grands nombres?

Ainsi, votre rapport fera exactement 5 pages.

Les petits caractères

Ce travail se fait seul ou en équipe de deux.

Avertissement au sujet du plagiat

Tout travail montrant des évidences de plagiat se méritera la note de zéro; les individus dont les travaux sont copiés étant responsables au même titre que les individus qui dupliquent le travail d'autrui. Vous pouvez utiliser des informations obtenues sur Internet (vous ne pouvez pas dupliquer du code obtenu sur Internet), dans un livre ou dans une autre source : citez toutes vos sources! Soyez sérieux, faites votre travail, consultez les démonstrateurs. Si vous vous trouvez dans une discussion avec des collègues et que vous croyez que vous avez échangé suffisamment d'idées pour que vos solutions et analyses se ressemblent, dites-le franchement dans le rapport.

Fichiers à remettre sur Studium

Créez un répertoire dont le nom est matricule-ift2015H16-TP2 (remplacez 'matricule' par votre matricule bien sûr) avec la structure donnée ci-bas. Ensuite, utilisez l'utilitaire zip (ou la commande compress sur un mac) pour créer un fichier d'archive avec l'extension .zip (évitez les .rar et autres variations sur le thème svp). La remise consiste en un seul fichier, l'archive .zip résultante. Structure du répertoire à remettre :

matricule-ift2015H16-TP2/

équipiers.txt : ce fichier texte nomme les membres de l'équipe.

explications.txt : (fichier optionnel) Si vous désirez transmettre de l'information au correcteur, vous pouvez vous expliquer dans ce fichier texte.

matricule-ift2015H16-TP2-rapport.pdf : votre rapport.

code/ : ce répertoire comprend vos fichiers python.

Fonctionnalité

Un code est fonctionnel s'il remplit les spécifications et qu'il ne fait pas autre chose. Un programme qui ne répond pas aux tâches demandées n'est pas fonctionnel bien sûr. Mais un code qui enjolive sa sortie ne l'est pas non plus ! Par exemple : si un programmeur décide d'augmenter son interface d'entrées/sorties il devient impossible d'utiliser son travail dans une chaîne d'outils prédéterminée et cela réduit la fonctionnalité de ses programmes. Lorsque vous produisez du code devant remplir des spécifications assurez-vous de remplir exactement les fonctionnalités.

Robustesse

Évidemment plus un programme comprends d'erreurs, moins il est robuste.

De plus, un programme d'ordinateur doit se comporter de manière consistante malgré les variations dans les données qui lui sont fournies en entrée et malgré les conditions variables d'exécution. En particulier lorsqu'une situation incongrue est détectée pendant l'exécution, le programme doit refuser de générer des résultats erronés et aviser l'utilisateur qu'une situation imprévue s'est produite. La politique « crash early, crash hard » est tout à fait bienvenue dans la plupart des cas (sauf évidemment si vous écrivez du code pour piloter un avion.)

Beauté

Un programme d'ordinateur n'est pas qu'une suite d'instructions destinées à être exécutée par un ordinateur ! C'est un objet abstrait destiné à être modifié par des humains. Un programme doit pouvoir être :

corrigé de ses erreurs (même les programmeurs très expérimentés font des erreurs de codage),

modifié (le comportement d'un programme devrait pouvoir être ajusté pour de nombreuses raisons n'ayant rien à voir avec le programmeur),

validé (la relecture critique de code reste une méthode très efficace d'assurance qualité), etc.

Ainsi, un programmeur utile écrit son code avec deux cibles en tête : la machine et l'humain. Parmi les alternatives, on doit favoriser l'usage d'idiomes communs et faciles à comprendre (par un programmeur compétent) et constamment s'efforcer à produire du code limpide. La marque

d'un bon programmeur est la beauté de son code.

Correction négative

Le code non fonctionnel n'est pas recevable et la correction sera très sévère en ce qui concerne les remises partiellement fonctionnelles. La remise de TP ne réalisant pas les spécifications pourrait vous coûter jusqu'à la totalité des points.

Vous perdrez des points si vous utilisez un utilitaire produisant une archive qui n'est pas au format demandé ou si la structure de votre répertoire de remise n'est pas conforme aux demandes.

Correction positive

fonctionnalité: 3.33 pts,
robustesse : 3.33, pts
beauté : 3.33 pts.

Retard

20% par jour (tranches de 24h)

Soupçon de plagiat

zéro, 0, nil, zip, quedalle.

Date de remise

Mardi le 8 mars 23h55.

Bon travail et bonne chance