Analyse d'Algorithmes et Génération Aléatoire

Devoir de Programmation : Sujet 2

Ce projet est construit dans le contexte d'un outil de test de propriétés afin de réaliser de la vérification d'algorithmes manipulant des structures de données.
Le but du devoir consiste à implémenter divers générateurs aléatoires ou exhaustifs puis de mettre en œuvre une batterie de tests l'outil afin de mettre en évidences certaines défaillances du code ou de faire un test avec couverture complète pour les plus petites structures afin de garantir un code exempt de bugs jusqu'à ce seuil pour la taille des données.

Le travail est à effectuer en binôme (ou trinôme après discussion avec A. Genitrini). Le langage de programmation pour la génération de structures est au choix du binôme.
Chaque groupe devra rendre, au plus tard le 30/01/22 à 23h59 une archive, par mail (`Antoine.Genitrini@lip6.fr`) contenant un rapport au format `pdf` ainsi que le code source contenant les différents programmes (et une notice pour la compilation du code, si nécessaire).

Lorsque les questions suivantes demande l'implémentation fonctions ou la mise en place de l'outil de vérification, indiquer dans le rapport le pseudo-code commenté ou les étapes clé et des remarque quant à l'outil de vérification. Il est possible d'insérer des captures d'écran dans l'annexe du rapport.

1 Mise en route

Question 1.1 Lire assez rapidement l'article originel concernant l'outil `QuickCheck` :
`https://www.cs.tufts.edu/~nr/cs257/archive/john-hughes/quick.pdf`

Question 1.2 Faire un peu de bibliographie (sur le web) des divers outils permettant de faire du test de propriété pour divers langages de programmation.
Mettre en avant les outils qui permettraient de réaliser les vérifications suggérées dans la Section 2 ci-dessous, soit grâce à leurs fonctionnalités de base soit en permettant par exemple l'appel à un générateur externe (cf. Section 3 dans l'article relatif à QuickCheck).

2 Vérification d'algorithmes

2.1 Ternary Search Trie

Dans les diverses familles de structures de recherche arborescentes l'une d'entre elles plutôt efficace en pratique est appelée famille des arbres ternaires de recherche. Une idée globale de la structure se trouve sur la page wikipedia dédiée `https://en.wikipedia.org/wiki/Ternary_search_tree`.

Question 2.3 Faire un peu de bibliographie (sur le web) afin de comprendre le fonctionnement global de la structure. Indiquer les liens intéressants et décrire rapidement leurs contenus.

À l'adresse suivante vous trouverez un code (très mal documenté, mais c'est fait exprès...) dont la fonction essentielle consiste à prendre deux arbres ternaires de recherche et de les fusionner en un seul arbre.
`https://www-apr.lip6.fr/~genitrini/doc_ens/AAGA_projet/ternary_trie.py`
Cet algorithme n'est pas correct.

Question 2.4 La base de l'ensemble des mots utilisés dans les œuvres de Shakespeare est disponible à l'adresse :
`https://www-apr.lip6.fr/~genitrini/doc_ens/AAGA_projet/Shakespeare.tar.gz`
Écrire une fonction prenant en entrée un titre d'une œuvre de Shakespeare et un nombre nb de mots qui extrait uniformément au hasard nb mots de l'œuvre en question et qui construit l'arbre ternaire via la suite de mots extraite.

Question 2.5 Insérer au sein de votre code l'algorithme erroné permettant la fusion de deux arbres.

Question 2.6 Trouver une stratégie (simple !) permettant de mettre en avant un bug observé par la fusion de deux arbres. Implémenter cette stratégie.
Indication : Il faut que l'arbre soit un arbre de recherche.

2.2 Arbre binaire de Rémy

Les auteurs de l'article `https://projecteuclid.org/download/pdf_1/euclid.mjms/1567216867` ont exhibé un bug de manière statistique concernant la version de l'algorithme de Rémy présentée dans le livre de Alonso et Schott. Une copie du code est disponible en Annexe (Section 4) de ce projet.

Question 2.7 Implémenter le code fourni présenté en Annexe.

Question 2.8 Rendre l'algorithme précédant déterministe, en lui fournissant en plus de la taille de l'arbre à générer, une liste des valeurs des entiers utilisés pour le choix des nœuds à étendre. Cette liste est concevable en amont de l'algorithme, car les choix aléatoires sont dépendant des numéros d'étapes mais pas (réellement) de la structure arborescente en construction.

On rappelle que le même arbre peut être construit via plusieurs listes (puisqu'on a effacé les étiquettes des nœuds internes).

Question 2.9 Écrire une fonction permettant de tester si deux arbres sont identiques.

Afin de reconnaître si deux arbres (ou sous-arbres) non étiquetés sont identiques, on va associer à chaque arbre une chaîne de caractères construite sur l'alphabet $\{(,)\}$, via la construction suivante. Soit A l'arbre binaire à compresser, et ϕ la fonction suivante :
— Si A est réduit à une feuille, on lui associe le mot vide ε (ainsi $\phi(\bullet) = \varepsilon$) ;
— Si A a un nœud interne et deux enfants G et D qui sont des arbres binaires alors on associe : $\phi(A) = (\ \phi(G)\ )\ \phi(D)$.
On pourrait montrer que ϕ est injective.
Ainsi, pour savoir si deux arbres binaires A et B sont égaux, il suffit de savoir tester si les mots $\phi(A)$ et $\phi(B)$ sont identiques.

Question 2.10 En construisant les plus petits arbres et en générant toutes les listes des valeurs aléatoires possibles, on peut faire un test de couverture. Vérifier que toutes les structures d'arbres n'apparaissent pas de manière uniforme.

Question 2.11 Vérifier que ce n'est pas le cas pour votre implémentation correcte de l'algorithme de Rémy. Quelle est la taille maximale atteinte lors de la vérification (par couverture totale) pour votre algorithme ?

3 Utilisation d'un outil

Question 3.12 Prendre en main un outil *à la QuickCheck* au choix. Il n'a pas forcément besoin d'être adapté au langage de programmation que vous avez choisi.

Question 3.13 Les tas binaires min (ou max, mais on s'intéresse ici aux min) sont décrits à l'adresse :
`https://en.wikipedia.org/wiki/Binary_heap`

Implémenter les fonctions d'insertion permettant la création d'un tas et de suppression de la valeurs minimale (qui retourne un tas).

Question 3.14 Tester votre code via votre outil de vérification. Un travail similaire a été réalisé ici : `https://hypothesis.works/articles/rule-based-stateful-testing/`

Question 3.15 Réaliser les étapes de vérification introduites dans la section précédente (Section 2) au sein de l'outil de test.

4 Annexe

2.1 GENERATION OF BINARY TREES : RÉMY'S ALGORITHM

```
#define N 1000

typedef struct {
     int right_child, left_child, parent;
     int num;
} Node;

Node tree [2*N+1];

// construction of a growing binary tree with n internal nodes (n ≤ N)

// modifies the position of two leaves, does not modify the tree built

void change_leaves (int a, int b)
{
     int parenta, parentb;
     parenta = tree[a]->parent;
```



```
     parentb = tree[b]->parent;
     if (tree [parenta]->right_child == a)
       tree [parenta]->right_child = b;
     else
       tree [parenta]->left_child = b;
     tree[a]->parent = parentb;
     if (tree [parentb]->right_child == b)
       tree [parentb]->right_child = a;
     else
       tree [parentb]->left_child = a;
     tree[b]->parent = parenta;
}

void growing_tree (long n)
{
     long i, number, tmp;

     // built a tree of size 1
     tree [0]->left_child = 1;
     tree [0]->right_child = 2;
     tree [0]->num = 1;
     tree [1]->parent = tree [2]->parent = 0;
     tree [1]->right_child = tree [1]->left_child = -1;
     tree [2]->right_child = tree [2]->left_child = -1;

// the internal nodes will be in the boxes [ 0, i − 1[  of the tree's array
// the leaves in boxes [ i, 2 ∗ i[  of the tree's array

     for (i = 2; i <= n; i++){
         number = random (i); // random number of [ 0, i[
         // the leaf is in the box number+i-1
         change_leaves (i-1, number+i-1);
         tree [i-1]->right_child = 2*i-1;
         tree [i-1]->left_child = 2*i;
         tree [i-1]->num = i;
         tree [2*i-1]->parent = tree [2*i]->parent = i-1;
         tree [2*i-1]->right_child = tree [2*i-1]->left_child = -1;
         tree [2*i]->right_child = tree [2*i]->left_child = -1;
     }
}
```