**Algorithmique / Travaux pratiques**

S09 – Traitement d’erreurs. Tests systématiques





27.04.2015

Adriano De Almeida Silva – T-1f

Alex Travasso – T-1f

1. **Voici les erreurs qui ont été révélées :**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1er scénario | 2ème scénario | 3ème scénario |
| a) | **X** |  |  |
| b) | **X** | **X** | **X** |
| c) |  |  |  |
| d) |  |  |  |

Voici un scénario qui détecte l’erreur c :

static boolean findBugs04() {

IntStack s = new IntStack(5);

try {

s.push(7);

} catch (Exception e) {

return true;

}

return false;

}

Voici un scénario qui détecte l’erreur d :

static boolean findBugs05() {

IntStack s = new IntStack(5);

IntStack s2 = new IntStack(12);

s.push(3);

if (s2.isEmpty()) {

return false;

} else {

return true;

}

}

Voici un bug qu’aucun des 3 scénarios ne détecterais :

Ligne 13 : return buf[top-1] **au lieu de** return buf[top]

1. **Voici ci-dessous la méthode qui vérifie si un résultat d’un tri est correct.**

static boolean isSortingResultCorrect(int[] givenInput, int[] observedOutput) {

int temp = observedOutput[0];

boolean test;

// creation HashMap

HashMap<Integer, Integer> tab = new HashMap();

// Teste si les deux tableaux ont la même taille.

if (givenInput.length != observedOutput.length) {

return false;

}

// Verifie que les elements du tableau de sortie sont dans l'ordre.

for (int i = 0; i < observedOutput.length; i++) {

if (temp < observedOutput[i]) {

return false;

}

temp = observedOutput[i];

}

// Verifie combien de fois apparait un même chiffre, pour chaque element

// du tableau.

for (int i = 0; i < givenInput.length; i++) {

if (tab.containsKey(givenInput[i])) {

tab.replace(givenInput[i], tab.get(givenInput[i]++));

} else {

tab.put(givenInput[i], 1);

}

}

// diminue dans la hashmap chaque fois qu'un chiffre contenu comme clé

// apparait dans le tableau de sortie.

for (int i = 0; i < observedOutput.length; i++) {

if (tab.containsKey(observedOutput[i])) {

tab.replace(observedOutput[i], tab.get(observedOutput[i]--));

} else {

tab.put(observedOutput[i], 1);

}

}

// Verifie que la hashmap contient que des valeurs à 0 pour chaque clés.

for (int i = 0; i < observedOutput.length; i++) {

if (tab.get(observedOutput[i]) != 0) {

return false;

}

}

return true;

}

1. **Voici ci-dessous le code de la pile de l’exercice (1) avec des assertions.**

package s09;

public class IntStack {

private int buf[];

private int top;

public IntStack() {

this(10);

}

// PRE: cap must be > 0

public IntStack(int cap) {

assert(cap > 0);

buf = new int[cap];

top = -1;

}

public boolean isEmpty() {

return top == -1;

}

// PRE: stack must not be empty

public int top() {

assert(!isEmpty());

return buf[top];

}

// PRE: stack must not be empty

public int pop() {

assert(!isEmpty());

int a = buf[top];

top--;

return a;

}

public void push(int x) {

checkSize();

top++;

buf[top] = x;

}

private void checkSize() {

if (top == buf.length - 1) {

int[] t = new int[2 \* buf.length];

for (int i = 0; i < buf.length; i++)

t[i] = buf[i];

buf = t;

}

}

}