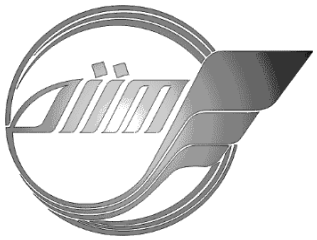
МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ



УДУНТ ННІ ДІІТ

Кафедра «Комп'ютерні інформаційні технології»

Лабораторна робота №1

з дисципліни «Якість програмного забезпечення та тестування»

на тему: «Тестування методами білої скриньки»

Виконав:
студент гр.ПЗ1911
Сафонов Д.Є.
Прийняв:
Шинкаренко В.І.

Дніпро, 2022

Тема. Тестування методами білої скриньки.
Мета. Отримати практичні навички тестування методами білої скриньки.
Завдання. Підготувати дві чи більше функцій для тестування. Функції повинні містити не менше шести керуючих конструкцій (розгалужень і циклів). Зазначені структури мають бути вкладеними. Виконати тестування функцій методами білої скриньки.
Текст. github

Специфікація

Функція 1. Додавання елемента до бінарного дерева.
Заголовок fun add(root: Node?, key: Key, value: Data): Node
Вхід:

- root — дерево, у яке треба додати елемент. Явний вид.
- key — ключ нового елемента. Явний вид.
- value — значення нового елемента. Явний вид.
- Key — тип ключа. Неявний вид — параметр узагальнення батьківського класу.
- Data — тип даних. Неявний вид — параметр узагальнення батьківського класу.

Вихід:

- Дерево з доданим елементом — в явному та неявному виді (якщо початкове дерево не є пустим).

Функція 2. Задача про ферзів.
Заголовок fun solve(board: Board): Boolean
Вхід:

- board — Початкова дошка. Явний вид.
- _n — розмірність задачі. Неявний вид — поле класу.

Вихід:

- У явному виді булеве значення яке показує чи вирішена задача.
- У неявному виді — змінена початкова дошка.

Тести

Функція 1.

1. Додавання вершини у пусте дерево
   Вхід:
   - root = null
   - key = 2
   - value = 2
   - Key = Int
   - Data = Int

   Вихід:
   - Дерево з одного вузла (2, 2)

2. Додавання вершини у ліве піддерево
   Вхід:
   1. root = Дерево:
      - (3, 3, left, right)
      - left = (2, 2, null, null)
      - right = (4, 4, null, null)
   2. key = 1
   3. value = 1
   4. Key = Int
   5. Data = Int

   Вихід: Дерево:
      - (3, 3, left, right)
      - left = (2, 2, child, null)
      - right = (4, 4, null, null)
      - child = (1, 1, null, null)
3. Додавання вершини у праве піддерево
   Вхід:
   1. root = Дерево:
      1. (3, 3, left, right)
      2. left = (2, 2, null, null)
      3. right = (4, 4, null, null)
   2. key = 5
   3. value = 5
   4. Key = Int
   5. Data = Int

   Вихід: Дерево:
      1. (3, 3, left, right)
      2. left = (2, 2, null, null)
      3. right = (4, 4, null, child)
         child = (5, 5, null, null)
4. Перезапис елемента
   Вхід:
   1. root = Дерево:
      1. (3, 3, left, right)
      2. left = (2, 2, null, null)
      3. right = (4, 4, null, null)
   2. key = 3
   3. value = 1
   4. Key = Int
   5. Data = Int

   Вихід: Дерево:
      1. (3, 1, left, right)
      2. left = (2, 2, null, null)
      3. right = (4, 4, null, null)

Функція 2.

З розмірністю 4 існує тільки 2 рішення — жоден з ферзів не має стояти на діагоналях.

1. Рішення 1

   Вхід:

			Ферзь

   Вихід:

   true

	Ферзь		
			Ферзь
Ферзь			
		Ферзь	

2. Рішення 1

   Вхід:

			Ферзь

   Вихід:

   true

		Ферзь	
Ферзь			
			Ферзь
	Ферзь		

3. Дошка, яку не можна вирішити

   Вхід:

Ферзь			

   Вихід:

   false, дошка не зміниться

4. Дошка неправильної розмірності

   Вхід:

   Вихід: Виключення

Покриття

Функція 1.

Таблиця покриття умов

Функція	add			_assign		_search			
Умова / Тест	*root == null*	*prev == null*	*key < prev.key*	*node?.*	*?:*	*curr != null*	*!found*	*key == curr.key*	*key < curr!!.key*
1	✓					×			
2	×	×	✓	×	✓	✓	✓	×	✓
3	×	×	×	×	✓	✓	✓	×	×
4	×	✓				✓	✓	✓	

✓ - виконано хоча б раз
× - не виконано
пуста клітинка — не досягнуто

_search: *curr?.left* та *curr?.right* не враховані тому що вони завжди мають однаковий результат та можуть бути замінені на *curr!!.left* та *curr!!.right*

a?.b - в мові котлін те саме що *if (a == null) null else a.b* — в таблиці якщо повертається *null* - ×
a ?: b - в мові котлін те саме що *if (a == null) b else a* - в таблиці якщо повертається *b* - ✓

Таблиця покриття рішень

Функція	add			_assign		_search		
Напрям / Тест	*root == null*	*prev == null*	*key < prev.key*	*node?.*	*?:*	*curr != null && ! found*	*key == curr.key*	*key < curr!!.key*
1	✓					×		
2	×	×	✓	×	✓	✓	×	✓
3	×	×	×	×	✓	✓	×	×
4	×	✓				✓	✓	

Функція 2.

Таблиця покриття умов

Функція	solve		_solve				_anyQueensInCells	isSafe
Умова Тест	*board.size != _n*	*i.size != _n*	*col >= _n*	*_anyQueensInCells(board, colIndices)*	*isSafe(board, i, col)*	*_solve(board, col + 1)*	*board[c.first][c.second]*	*!_anyQueensInCells(board, cells)*
1	×	×	✓	✓	✓	✓	✓	✓
2	×	×	✓	✓	✓	✓	✓	✓
3	×	×	×	✓	✓	×	✓	✓
4	✓							

Таблиця покриття рішень

Функція	solve	_solve				_anyQueensInCells	isSafe
Умова Тест	*board.size != _n || board.any { i -> i.size != _n }*	*col >= _n*	*_anyQueensInCells(board, colIndices)*	*isSafe(board, i, col)*	*_solve(board, col + 1)*	*board[c.first][c.second]*	*!_anyQueensInCells(board, cells)*
1	×	✓	✓	✓	✓	✓	✓
2	×	✓	✓	✓	✓	✓	✓
3	×	×	✓	✓	×	✓	✓
4	✓						

Висновки

Розроблені функції комбінують різні види циклів, умов та рекурсію, що дозволяє сформувати складні розгалуження. Усі тести окрім однієї пари мають неоднакові таблиці покриття умов та рішень.