Implementation of algorithms and data structures

5. seminar

Jirka Fink

https://ktiml.mff.cuni.cz/~fink/

Department of Theoretical Computer Science and Mathematical Logic
Faculty of Mathematics and Physics
Charles University in Prague

Summer semestr 2021/22

Last change 16. března 2022

Manually created data in memory

Motivation

- We need to test some parts of codes which are rarely executed
  - E.g. some cases in red-black trees

Manually created data in memory

Motivation

- We need to test some parts of codes which are rarely executed
  - E.g. some cases in red-black trees
- It is hard to create tested configuration using API
  - E.g. How to find a sequence of operations Insert/Delete for every case in red-black trees?

Manually created data in memory

Motivation

- We need to test some parts of codes which are rarely executed
  - E.g. some cases in red-black trees
- It is hard to create tested configuration using API
  - E.g. How to find a sequence of operations Insert/Delete for every case in red-black trees?
- Fuzz tests may create tested configuration but data are too large

Manually created data in memory: Basic idea

```
1  class Node:
2      def __init__(self, key, left, right, parent, is_black, size):
3          self.key = key
4          self.left = left
5          self.right = right
6          self.parent = parent
7          self.is_black = is_black
8          self.size = size
9
10 def test_delete_case_uncle_is_black():
11     root = Node(10, None, None, None, True, 6)
12     p = root.left = Node(5, None, None, root, False, 4)
13     l = p.left = Node(2, None, None, p, True, 2)
14     u = root.right = Node(15, None, None, root, True, 1)
15     ...
16     integrity_test(root)
17     root.delete(2)
18     integrity_test(root)
19     TEST(root.left == u)
20     TEST(u.is_black == False)
21     ...
```

Manually created data in memory: Basic idea

Discussion

- Simple approach to create one small test

Manually created data in memory: Basic idea

Discussion

- Simple approach to create one small test
- Impractical to create a larger test or multiple tests
  - Setting every variable is time consuming,
  - hard to read, and
  - leads to error.

Manually created data in memory: Basic idea

Discussion

- Simple approach to create one small test
- Impractical to create a larger test or multiple tests
  - Setting every variable is time consuming,
  - hard to read, and
  - leads to error.

Approach

- Encode data into compact and clear format, e.g.
  - XML, JSON
  - Custom format, e.g. (3,(1,(0,(),()),(2,(),())),(4,(),()))
  - Use syntax of used programming language

Manually created data in memory: Basic idea

Discussion

- Simple approach to create one small test
- Impractical to create a larger test or multiple tests
  - Setting every variable is time consuming,
  - hard to read, and
  - leads to error.

Approach

- Encode data into compact and clear format, e.g.
  - XML, JSON
  - Custom format, e.g. (3,(1,(0,(),()),(2,(),())),(4,(),()))
  - Use syntax of used programming language
- Write a function reading the chosen format

Encode data: example

```
def subtree(key, is_black, left=None, right=None):
    node = Node()
    node.key = key
    node.is_black = is_black
    node.left = left
    node.right = right
    node.size = 1 + (left.size if left else 0) +
                    (right.size if right else 0)
    if left:
        left.parent = self
    if right:
        right.parent = self

root =
    subtree(5, True,
        subtree(3, False,
            None,
            subtree(4, True)),
        subtree(7, True))
```

Manually created data in memory: Other applications

Methods

- Create data in memory corresponding to tested configuration

Manually created data in memory: Other applications

Methods

- Create data in memory corresponding to tested configuration
- Create data in memory corresponding to expected result

Manually created data in memory: Other applications

Methods

- Create data in memory corresponding to tested configuration
- Create data in memory corresponding to expected result
- Run data consistency tests on both cases

Manually created data in memory: Other applications

Methods

- Create data in memory corresponding to tested configuration
- Create data in memory corresponding to expected result
- Run data consistency tests on both cases
- Run tested function

Manually created data in memory: Other applications

Methods

- Create data in memory corresponding to tested configuration
- Create data in memory corresponding to expected result
- Run data consistency tests on both cases
- Run tested function
- Run data consistency tests on results

Manually created data in memory: Other applications

Methods

- Create data in memory corresponding to tested configuration
- Create data in memory corresponding to expected result
- Run data consistency tests on both cases
- Run tested function
- Run data consistency tests on results
- Compare obtained and expected results

Manually created data in memory: Other applications

Methods

- Create data in memory corresponding to tested configuration
- Create data in memory corresponding to expected result
- Run data consistency tests on both cases
- Run tested function
- Run data consistency tests on results
- Compare obtained and expected results
- These tests often needs access to private members

Compare obtained and expected results

```
1  def compare_trees(tested_tree, expected_tree):
2
3      def recursive(tested_node, expected_node):
4          TEST(tested_node.key == expected_node.key)
5          TEST(tested_node.is_black == expected_node.is_black)
6          if expected_node.left:
7              TEST(tested_node.left)
8              recursive(tested_node.left, expected_node.left)
9          else:
10             TEST(not tested_node.right)
11                 recursive(tested_node.right, expected_node.right)
12         ...
13
14     recursive(tested_tree.root, expected_tree.root)
```

Compare obtained and expected results

```
def compare_trees(tested_tree, expected_tree):

    def recursive(tested_node, expected_node):
        TEST(tested_node.key == expected_node.key)
        TEST(tested_node.is_black == expected_node.is_black)
        if expected_node.left:
            TEST(tested_node.left)
            recursive(tested_node.left, expected_node.left)
        else:
            TEST(not tested_node.right)
                recursive(tested_node.right, expected_node.right)
        ...

    recursive(tested_tree.root, expected_tree.root)
```

```
tested_tree = Tree(...)
tested_tree.integrity_tests()
tested_tree.insert(5)
tested_tree.integrity_tests()
expected_tree = Tree(...)
expected_tree.integrity_tests()
compare_content(tested_tree, expected_tree)
```

Tasks for next week

The first assignment: Left-leaning Red-black trees

- Design and implement API
- Design and implement data representation
- Write unit tests with small and also large number of elements
- Write test checking correctness of data representation
- Implement operations insert and find k-th element
- Implement operation delete
- Write fuzz tests
- Debug your program
- Submit your program on recodex