Implementation of algorithms and data structures

1. 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 2023/24

Last change 2. října 2023

The course

Goals

- Learn how to implement advanced algorithms and data structures without tedious debugging
- Learn how to write and use various tests
- Learn to structure code and write API

Entrance expectations

- Knowledge of some programming language (e.g. C/C++, Python, Java, C#)
- Theoretical knowledge algorithms and data structures from bachelor study
- Experience in implementing basic algorithm (e.g. graph search)

Communication

Web

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

E-mail

fink@ktiml.mff.cuni.cz

Recodex

- Enroll into my group on Recodex
- Submit your fully working programs here

Gitlab

- Create a new private git repository on `https://gitlab.mff.cuni.cz/`
- Give me (finkj1am) access (developer)

Passing conditions

- Implement 3 algorithms or data structures
- Programming language: negotiable but it must be available on recodex

Assignments

1. **Red-black trees**
   Martin Mareš, Tomáš Valla: Průvodce labyrintem algoritmů, CZ.NIC, 2017
   Robert Sedgewick: Left-leaning Red-Black Trees, doi:10.1.1.139.282
2. **Network flows (Goldberg algorithm)**
   Martin Mareš, Tomáš Valla: Průvodce labyrintem algoritmů, CZ.NIC, 2017
   Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein: Introduction to algorithms, The MIT Press, 2001
   `http://mitp-content-server.mit.edu:18180/books/content/sectbyfn?collid=books_pres_0&id=8030&fn=Chapter%2026.pdf`
3. **Maximum matching in general unweighted graphs (Blossom algorithm)**
   Cunningham, Cook, Pulleyblank, Schrijver: Combinatorial optimization, John Wiley & Sons, 1997

The first assignment: Left-leaning Red-black trees

Implement the following operations

- Insert element
- Delete element
- Find the k-th smallest element
- Martin Mareš, Tomáš Valla: Průvodce labyrintem algoritmů, CZ.NIC, 2017

Definition

- Binary search tree where every node is red or black
- The parent of a red node must be black
- Every path from the root to all leaves has the same number of black nodes
- The root and all leaves are black
- If a node has exactly one child then the child is left

Literature

- Martin Mareš, Tomáš Valla: Průvodce labyrintem algoritmů, CZ.NIC, 2017
- Robert Sedgewick: Left-leaning Red-Black Trees, doi:10.1.1.139.282

The first assignment: Left-leaning Red-black trees

Question

How to find the k-th smallest element in a binary tree?

Approach

- In every node remember the size of its subtree
- Faster version: Remember the size of left subtree

General approach

- Read the problem statement
- Study given algorithm and make sure to understand it
- Understand proofs of correctness and complexities
- Split the task into small parts
- Design application programming interface (API)
- Prepare unit tests
- Determine how to data representation (storing data in memory)
- Solve boundary cases (e.g. storing the root and leaves)
- Prepare test of data representation
- Split your code into small functions
- Implement and test functions one by one

Debugging tools

- Localize a bug using various tests
- Visualize data stored in memory
  E.g. draw a graph with vertices and edges
- Print what program does
  E.g. insert a key, rotate a node
- Tools checking dangerous operations
  E.g. memory allocations in C/C++
- Find a minimal example producing an error
- Write more tests
- Debug your program step by step (a very slow process)
- Read your code and check that we understand every single line
- Read the problem statement and literature again
- Find a better design of our program and rewrite it completely
- Take a break; sleep whole night

Type of tests

- Unit testing
- Fuzzy testing
- Integration testing
- System testing
- Acceptance testing
- Installation testing
- Regression testing
- Continuous testing
- Destructive testing
- Software performance testing
- Security testing
- VCR testing
- Internationalization and localization
- ...

Testing methods

Black-box testing

- No knowledge of codes is used
- Test fulfilling the specification
- First write tests, code later

White-box testing

- Requires knowledge of codes
- Covers every part of code
- Reversed engineering: Searching for dangerous inputs for our code
- Verifies the internal structures

Testing reported bugs

For a reported bug, a test is created before fixing.

Test-driven development

Development cycle

1. Write tests
2. Run tests and check that all fails
3. Implement the program
4. Use tests for debugging
5. Refactorization and clean up
6. Write documentation

Advantages

- Tests may contain bugs, we verify that tests fails as expected
- Writing unit tests verifies usability of interface

Unit testing

Motivation

Verify correctness of new features and preserving functionality after changes in codes.

Description

- Software testing method by which individual units of source code are tested to determine whether they are fit for use
- Tests should be independent

Advantages

- Unit tests can be run repeatedly
- Fast discovery of a bug when code is changed
- Example of usage of a library

Limitations and disadvantages

- Unit tests only proves that a program contains a bug
- Unit tests cannot prove that a program is correct (halting problem)
- Unit tests are not supposed to verify integration of modules
- Unit tests uses API without verifying internal data correctness

Unit testing: Sum of two numbers

```
1 public class TestAdder {
2   @Test
3   public void testSumPositiveNumbersOneAndOne() {
4     Adder adder = new AdderImpl();
5     TEST(adder.add(1, 1) == 2);
6   }
7   @Test
8   public void testSumPositiveNumbersOneAndTwo() {
9     Adder adder = new AdderImpl();
10    TEST(adder.add(1, 2) == 3);
11  }
12  @Test
13  public void testSumPositiveNumbersTwoAndTwo() {
14    Adder adder = new AdderImpl();
15    TEST(adder.add(2, 2) == 4);
16  }
17  @Test
18  public void testSumZeroNeutral() {
19    Adder adder = new AdderImpl();
20    TEST(adder.add(0, 0) == 0);
21  }
22  ...
23 }
```

Unit testing: Graph

```
1  public abstract class AbstractGraphTest {
2    MutableGraph<Integer> graph;
3    @Before
4    public void init() {
5      graph = createGraph();
6    }
7    @After
8    public void validateGraphState() {
9      validateGraph(graph);
10   }
11   @Test
12   public void nodes_oneNode() {
13     addNode(N1);
14     TESTThat(graph.nodes().containsExactly(N1);
15   }
16   @Test
17   public void nodes_noNodes() {
18     TESTThat(graph.nodes().isEmpty();
19   }
20   @Test
21   public void adjacentNodes_oneEdge() {
22     putEdge(N1, N2);
23     TESTThat(graph.adjacentNodes(N1)).containsExactly(N2);
24     TESTThat(graph.adjacentNodes(N2)).containsExactly(N1);
25   }
26 }
```

Source: Wikipedia: Unit testing
https://github.com/google/guava/blob/master/guava-tests/test/com/google/common/graph/AbstractGraphTest.java

How create unit tests?

Initial questions about unit tests

- How should a unit be tested?
  E.g. which library should be used to write tests?
- What should be tested?
  E.g. what should be the content of unit tests?

Libraries for unit tests

- Python: unittest — Unit testing framework
- Julia: Unit testing (standard library)
- Java: JUnit 5
- C#: Unit test basics
- C/C++: Tens of libraries

Wikipedia: List of unit testing frameworks

What should be tested?

General hints

- Write test verifying that task is fulfilled
- Test boundary cases

Test sorting function

```
// First, small simple tests
TEST(sort([5,7,9,3]) == [3,5,7,9])
TEST(sort(['d','a','z']) == ['a','d','z'])
TEST(sort(["one","two","three"]) == ["one","three","two"])

// Tricky cases, some cases may depend on documentation
TEST(sort([]) == [])
TEST(sort([1,2,1,2,1]) == [1,1,1,2,2])
TEST(sort([1,False,5,True]) == [False, 1, True, 5])
```

Creating larger and random tests will be discussed later.

What should be tested?

Examples of tests of binary search trees

1. Create a new empty tree and destroy it
2. Create a new empty tree, insert one element and destroy the tree
3. Insert more elements
4. Delete some elements
5. Delete all elements
6. Combine insertion and deletions
7. Check the counter of the number of elements
8. Find existing and non-existing elements
9. Insert existing and delete non-existing elements
10. Find and delete an element in an empty tree

The first assignment: Left-leaning Red-black trees

Tasks for next week

- Create a new private git repository on `https://gitlab.mff.cuni.cz/`
- Give me (finkj1am) access (developer)
- Read and understand literature
- Design API
- Design data representation
- Write unit tests
- Use git every day!