

Implementation of algorithms and data structures

4. 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

Winter semestr 2026/27

Last change 20. října 2026

Licence: Creative Commons BY-NC-SA 4.0

- When a program returns an incorrect result, how do we find the bug?
- A unit test gives us an input that reproduces an error, but only limited information about where the bug is.
- For data structures, we can test their internal structure after every operation.
- Can we verify the results of intermediate steps of an algorithm?
- Can we also verify the intermediate states of a data structure during a complex operation?

```
1 function quicksort(array a of length n) {  
2     stack = [(1,n)]  
3     while(stack is non-empty) {  
4         i,j = stack.pop()  
5         m = choose_pivot_and_split(a,i,j)  
6         test_split(a,i,j)  
7         if(i < m) stack.push((i,m))  
8         if(m+1 < j) stack.push((m+1,j))  
9         test_stack(a, stack)  
10    }  
11 }
```

What should be tested after splitting

- $i \leq m \leq j$
- All elements between i and m are smaller than all elements between $m+1$ and j
- The array contains exactly the given elements

How can the stack be tested?

- $stack[z][1] < stack[z][2] < stack[z+1][1]$ for every z in the stack except the last one
- When we replace every subarray $stack[z][1] \dots stack[z][2]$ by its minimum and maximum, the resulting array must be sorted

Exercise: How can we test the intermediate results of the following algorithms?

- Mergesort, Heapsort
- Knuth-Morris-Pratt, Aho-Corasick
- Shortest path
- Minimum spanning tree
- Maximum flow (Ford-Fulkerson, Dinic)
- Red-black trees

- We have a search or optimization problem
 - e.g. coloring, matching, scheduling
- Instead of writing a specialized program, we can use SAT/CSP/ILP solvers
 - Problem is described using variables and constraints
- Boolean satisfiability
 - Variables are Boolean
 - Constraints are disjunctive clauses
- Constraint satisfaction programming
 - Variables have finite domains
 - Constraints restrict assignments for subsets of variables
- Linear programming and integer linear programming
 - Variables are real numbers or integers
 - Constraints are linear inequalities

- Possible problems
 - A solver finds a solution that does not satisfy our conditions
 - A solver claims that our model has no solution, although a solution exists
 - An ILP solver finds a suboptimal solution
- Exercise: How do we find the bug and write tests?

- It is hard to see what is stored in memory
- Without the knowledge of the stored data, debugging is difficult
- Visualize the stored data!
- The content of an array can be easily printed to the terminal
- Advanced structures need a graphical presentation

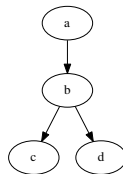
Write the structure of a red-black tree

```
1 void rb_print(rb_tree *tree) {
2     rb_print(tree, tree->root);
3     printf("\n");
4 }
5
6 void rb_print(rb_tree *tree, rb_node *node) {
7     if(!node)
8         printf("L");
9     else {
10        printf("(");
11        rb_print(tree, node->left);
12        printf("%d", node->key);
13        if(node->is_red())
14            printf("R");
15        rb_print(tree, node->right);
16        printf(")");
17    }
18 }
```

Output

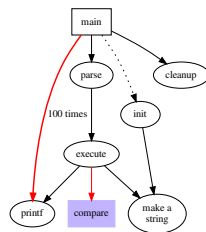
- (((L1L)2(L3L))4(((L6RL)7L)10((L14RL)20L)))
- The output can be improved using ASCII art
- A graphical visualization may be clearer

```
1 digraph graphname {  
2   a -> b -> c;  
3   b -> d;  
4 }
```



Graphs created in DOT: node and edge attributes

```
1 digraph G {
2   size = "4,4";
3   main [shape=box]; /* this is a comment */
4   main -> parse [weight = 8];
5   parse -> execute;
6   main -> init [style=dotted];
7   main -> cleanup;
8   execute -> {make_string printf}
9   init -> make_string;
10  edge [color=red]; // so is this
11  main -> printf [style=bold,label="100 times"];
12  make_string [label="make a\nstring"];
13  node [shape=box,style=filled,color=".7 .3 1.0"];
14  edge [color=red];
15  execute -> compare;
16 }
```



More examples: <https://graphviz.org/gallery/>

Write the structure of a red-black tree using DOT

```
1 void rb_tree_print(rb_tree *tree) {
2     printf("digraph G {\n");
3     rb_node_print(tree, tree->root);
4     printf("}\n");
5 }
6
7 void rb_node_print(rb_tree *tree, rb_node *node) {
8     if(node) {
9         printf("%d [color=%s];\n", node->key, node->is_red ? "red" :
10             "black");
11         if(node->parent)
12             printf("%d -> %d;\n", node->parent->key, node->key);
13         rb_node_print(tree, node->left);
14         rb_node_print(tree, node->right);
15     }
```

Create a Graphviz picture in Python

```
1 import pydot
2
3 graph = pydot.Dot("my_graph", graph_type="graph", bgcolor="yellow")
4
5 # Add nodes
6 graph.add_node(pydot.Node("a", label="Foo"))
7 graph.add_node(pydot.Node("b", shape="circle"))
8
9 # Add edges
10 graph.add_edge(pydot.Edge("a", "b", color="blue"))
11
12 # Output image in png format
13 graph.write_png("output.png")
14
15 # Convert to string
16 output_raw_dot = graph.to_string()
17 # Save it as a DOT-file:
18 graph.write_raw("output_raw.dot")
```

Language bindings (API)

- Python
- Java
- MATLAB
- WordPress
- LaTeX: TikZ, dot2tex: A Graphviz to LaTeX converter

Visualization and IDE integrations

- Graphviz (dot) language support for Visual Studio Code
- Graphviz Visual Editor: A web application
- Qt Visual Graph Editor (C++)
- More resources about Graphviz

Other graph visualization tools

- NetworkX
- Gephi
- igraph

Testing intermediate results

- Check a predicate that must hold after every step, not only the final output
- Quicksort: the split invariant and the contents of the stack
- Data structures: the keys and the colours after every operation

Verifying solver output

- A returned solution can be checked against the model
- A claimed infeasibility or optimality is hard to verify independently

Making the state visible

- Printouts and ASCII art for arrays and small trees
- DOT/Graphviz for complex structures