Design Automation Renegades

Boilerplate Code: Data Structures and Algorithms for Design Automation

Zhiyang Ong [1]

Report on
Common Data Structures and Algorithms
Found in Bolierplate Code for
Design Automation Software

September 29, 2015

[1]Email correspondence to: ✉ ongz@acm.org

Abstract

This report describes the design and implementation of common data structures and algorithms, as well as "computational engines" that are found in electronic design automation (EDA) software.

Data structures and algorithms for digital VLSI and cyber-physical system design include: binary decision diagrams (BDDs), AND-inverter graphs (AIGs), and their associated algorithms for optimization, traversal, and other operations (such as graph matching). Common computational engines for digital systems would include: optimization and verification engines for deterministic and nondeterministic finite state machines; decision procedures for the boolean satisfiability problem (SAT solvers) and satisfiability modulo theories (SMT solvers); quantified boolean formula (QBF) solvers; and SAT and SMT solvers for maximum satisfiability (i.e., Max-SAT and Max-SMT solvers).

Regarding EDA problems that require numerical computation (in digital, analog, or mixed-signal VLSI design), the data structures and algorithms for circuit simulation based on sparse graph would be required. In addition, techniques for model order reduction shall be implemented.

Computational engines for statistical and probabilistic analyses or stochastic modeling can include data structures and algorithms for partially observable Markov decision processes (POMDPs) and Markov chains. Tools for analyses of queueing systems (based on queueing theory) should be included.

Regarding cyber-physical systems and mixed-signal circuits, hybrid automata can be used to represent these circuits and systems.

Optimization engines for EDA include: solvers for different types of mathematical programming, such as linear programming (LP), integer linear programming (ILP), mixed-integer linear programming (MILP), quadratic programming (QP), convex programming (CP), geometric programming (GP), and second-order conic programming (SOCP); solvers for pseudo-boolean optimization (PBO solvers) and weighted-boolean optimization (WBO); and meta-heuristics (e.g., evolutionary algorithms, simulated annealing, and ant colony optimization).

Algorithms shall be implemented using parallel programming, in a scalable style. In addition, considerations shall be given to the use of constraint programming.

More stuff to be included. . .

Revision History

Revision History:
1. Version 0.1, December 23, 2014. Initial copy of the report.
2. Version 0.1.1, September 16, 2015. Added sections for mathematics and statistics, and the abstract.

Contents

Chapter 1

Algorithms

This section documents algorithms that I have implemented for my C++ -based boilerplate code repository.

A template for typesetting algorithms is shown in PROCEDURE 1.

NAME OF THE ALGORITHM(*ARGUMENTS*)

    ∥ Input ARGUMENT #1: Definition1
    ∥ Input ARGUMENT #2: Definition2
1   BODY OF THE PROCEDURE
    ∥ A while loop.
2   **while** [condition]
3   [Something]
    ∥ A for loop.
4   **for** *Var* = [initial value] **to** [final value]
5   [Something]
    ∥ An if-elseif-else block.
6   **if** [Condition1]
7   Blah. . .
8   **elseif** [Condition2]
9   Blah. . .
10 **elseif** [Condition3]
11   Blah. . .
12 **else**
13   Blah. . .
    ∥ A variable assignment.
14 *blah* = $A[j]$
      ∥ This is indented with a tab.
    ∥ What is the output of this procedure?
15 **return**

Chapter 2

Data Structures

2.1 Graphs

2.1.1 Directed Graphs

2.1.1.1 Functions that need to be implemented

2.1.1.2 Binary Decision Diagrams (BDDs)

2.1.1.3 AND-Inverter Graphs (AIGs)

2.1.2 Undirected Graphs

Chapter 3

Mathematics

Math symbols that I use frequently:
1. $\mathbb{N}$
2. $\sum\limits_{n}^{i=1}$
3. $f(x) = \lim\limits_{n \to \infty} \dfrac{f(x)}{g(x)}$
4. $\varnothing$
5. q

A 3×3 matrix: $\begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix}$

Here is an equation:

$$\iint_{\Sigma} \nabla \times \mathbf{F} \cdot \mathrm{d}\mathbf{\Sigma} = \oint_{\partial\Sigma} \mathbf{F} \cdot \mathrm{d}\mathbf{r}. \tag{3.1}$$

Here is an equation that is not numbered.

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$$

Here is the set of Maxwell's equations that is numbered.

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0} \tag{3.2}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{3.3}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{3.4}$$

$$\nabla \times \mathbf{B} = \mu_0 \left(\mathbf{J} + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t} \right) \tag{3.5}$$

$$\text{minimize} \sum_{i=1}^{c} c_i \cdot x_i$$

$$\underline{x} \in S$$

$$\text{subject to}:$$

$$x_1 + x_4 = 0$$

$$x_3 + 7 \cdot x_4 + 2 \cdot x_9 = 0$$

$$f(n) = \begin{cases} case-1 & : \text{n is odd} \\ case-2 & : \text{n is even} \end{cases} \tag{3.6}$$

Proof. This is a proof for BLAH ... $\qquad\square$

Theorem 3.1. *TITLE of theorem. My theorem is...*

Axiom 3.1. *TITLE of axiom. Blah...*

Cases of putting a bracket/parenthesis on the right side of the equation.

$$\left.\begin{array}{l} B' = -\partial \times E, \\ E' = \partial \times B - 4\pi j, \end{array}\right\} \quad \text{Maxwell's equations}$$

Labeling an arrow: $\xrightarrow{ewq}$

Chapter 4

Statistics

Chapter 5

C++ Resources

Some C++ and C++ STL resources are:

1. [11]: http://www.tutorialspoint.com/cplusplus/cpp_stl_tutorial.htm
2. [4] and CplusplusCom2015: http://www.cplusplus.com/reference/stl/
3. http://en.cppreference.com/w/cpp/container
4. http://www.cs.wustl.edu/~schmidt/PDF/stl4.pdf
5. Pointers to functions: http://www.cplusplus.com/doc/tutorial/pointers/

 C++ topics:

1. Function objects:
    (a) https://en.wikipedia.org/wiki/Functional_(C%2B%2B)
    (b) http://stackoverflow.com/questions/356950/c-functors-and-their-uses
    (c) http://www.cprogramming.com/tutorial/functors-function-objects-in-c++.html

2. Strings:
    (a) [18], Chp 23
    (b) [17], Chp 23
    (c) [8], Chp 18
    (d) [3], Chp 19
    (e) [6], Chp 1

3. IO Streams:
    (a) [6], Chp 2
    (b) [7], Chp 12
    (c) [18], Chp 10-11
    (d) [17], Chp 10-11
    (e) [12], Chp 16
    (f) [20], Chp 10
    (g) [15], Chp 21
    (h) [3], Chp 28
    (i) [8], Chp 12
    (j) [13], Chp 17
    (k) [9], Chp 8

4. Templates:
    (a) [6], Chp 3
    (b) [5], Chp 16

(c) [18], Chp 19

(d) [17], Chp 19

(e) [12], Chp 24

(f) [20], Chp 6

(g) [2], book; typelist - Chp 3

(h) [15], Chp 18

(i) [19], book

(j) [1], book

(k) [3], Chp 29

(l) [8], Chp 11,21

(m) [9], Chp 16

5. Debugging:

(a) [6], Chp 11 (especially memory management problems, pp. 533)

6. STL containers:

(a) [6], Chp 4

(b) [16], Chp 8

(c) [12], Chp 25

(d) [20], Chp 7

(e) [14], book

(f) [3], Chp 18

(g) [8], Chp 15-16

(h) [13], Chp 16

(i) [9], Chp 9,11

7. STL algorithms:

(a) [6], Chp 5

(b) [12], Chp 25

(c) [20], Chp 7

(d) [14], book

(e) [3], Chp 18

(f) [8], Chp 15,17

(g) [13], Chp 16

(h) [9], Chp 10

8. Function addresses:

(a) [5], Chp 3, pp. 213

(b) [18], Chp 8

(c) [17], Chp 8

9. Dynamic memory management problems:

(a) [5], Chp 6,13

(b) [7], Chp 13

(c) [10], Chp 2-4

(d) [15], Chp 29

(e) [3], Chp 14

(f) [8], Chp 10,22

(g) [13], Chp 9,12

(h) [9], Chp 12,13

10. Function overloading:
    (a) [5], Chp 7
    (b) [7], Chp 6
    (c) [18], Chp 8
    (d) [17], Chp 8
    (e) [15], Chp 14

11. Operator overloading:
    (a) [5], Chp 12
    (b) [12], Chp 18
    (c) [15], Chp 15
    (d) [9], Chp 14

12. Constants:
    (a) [5], Chp 8

13. Functions and pointers:
    (a) [5], Chp 11:
          i. use const at the end of accessor functions
          ii. Do not use pointers as instance variables
    (b) [18], Chp 8:
          i.   Pass-by-reference: e.g., void init(vector<double> &v)
          ii. Pass-by-const-reference: e.g., void print(const vector<double> &v)
          iii. Pass-by-value: e.g., void fn(int x)
    (c) [17], Chp 8
    (d) [12], Chp 15,20
    (e) [3], Chp 12-13
    (f) [13], Chp 7-8
    (g) [9], Chp 6
    (h) Elsewhere:
          i. http://stackoverflow.com/questions/2382834/discards-qualifiers-error
          ii. You cannot call a non-const method from a const method. That would 'discard' the
              const qualifier.

14. OOD and inheritance:
    (a) [5], Chp 14,15
    (b) [7], Chp 13,14,15
    (c) [18], Chp 9
    (d) [17], Chp 9
    (e) [12], Chp 13-14,21
    (f) [20], Chp 3-4,8
    (g) [3], Chp 24-26
    (h) [8], Chp 4-9
    (i) [13], Chp 10-11,13,14,15
    (j) [9], Chp 7,15,18,19

15. SW engineering issues:
    (a) [3], Chp 21
    (b) [8], Chp 24-26

16. multi-threading:
    (a) [16], Chp 3

17. graphs:
    (a) [16], Chp 7

5.1 Computational Complexity of C++ Containers

Table 5.1: Computational Complexity of C++ Containers

Container \ Complexity	add	remove	search	size	empty	begin	end
vector	O(1)	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)

Acknowledgments

I would like to thank Ms. Deepika Panchalingam for motivating me to revise basic data structures and algorithms for internship and job interviews.

Bibliography

[1] David Abrahams and Aleksey Gurtovoy. C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond. C++ In-Depth Series. Pearson Education, Boston, MA, 2005.

[2] Andrei Alexandrescu. Modern C++ Design: Generic Programming and Design Patterns Applied. C++ In-Depth Series. Addison-Wesley, Indianapolis, IN, 2001.

[3] Alex Allain. Jumping into C++. Cprogramming.com, San Francisco, CA, 2012.

[4] cplusplus.com. The C++ resources network. Available online at: http://www.cplusplus.com/; April 2, 2014 was the last accessed date, 2014.

[5] Bruce Eckel. Thinking in C++: Introduction to Standard C++, volume 1. Prentice Hall, Upper Saddle River, NJ, second edition, 2000.

[6] Bruce Eckel and Chuck Allison. Thinking in C++: Practical Programming, volume 2. Prentice Hall, Upper Saddle River, NJ, 2003.

[7] Tony Gaddis. Starting Out With C++: From Control Structures Through Objects. Pearson Education, Boston, MA, sixth (brief) edition, 2010.

[8] Marc Gregoire. Professional C++. John Wiley & Sons, Indianapolis, IN, third edition, 2014.

[9] Stanley B. Lippman, Josée Lajoie, and Barbara E. Moo. C++ Primer. Addison-Wesley, Upper Saddle River, NJ, fifth edition, 2013.

[10] Scott Meyers. Effective C++: 55 Specific Ways to Improve Your Programs and Designs. Addison-Wesley Professional Computing Series. Pearson Education, Upper Saddle River, NJ, third edition, 2005.

[11] Mohtashim. C++ STL tutorial. Available online at *Tutorials Point: C++ Tutorial: C++ STL Tutorial*: http://www.tutorialspoint.com/cplusplus/cpp_stl_tutorial.htm; September 17, 2015 was the last accessed date, 2015.

[12] Steve Oualline. Practical C++ Programming. Programming Style Guidelines. O'Reilly Media, Sebastopol, CA, second edition, 2003.

[13] Stephen Prata. C++ Primer Plus: Developer's Library. Pearson Education, Upper Saddle River, NJ, sixth edition, 2012.

[14] Greg Reese. C++ Standard Library Practical Tips. Charles River Media Programming Series. Charles River Media, Hingham, MA, 2006.

[15] Herbert Schildt. C++: The Complete Reference. Osborne Complete Reference Series. McGraw-Hill/Osborne, Berkeley, CA, fourth edition, 2003.

[16] Herbert Schildt. The Art of C++. McGraw-Hill/Osborne, Emeryville, CA, 2004.

[17] Bjarne Stroustrup. Programming: Principles and Practice Using C++. Pearson Education, Boston, MA, 2009.

[18] Bjarne Stroustrup. Programming: Principles and Practice Using C++. Pearson Education, Upper Saddle River, NJ, second edition, 2014.

[19] David Vandevoorde and Nicolai M. Josuttis. C++ Templates: The Complete Guide. Pearson Education, Boston, MA, 2003.

[20] Dirk Vermeir. Multi-Paradigm Programming using C++. Springer-Verlag London Berlin Heidelberg, London, U.K., 2001.