Design Automation Renegades

Globetrotting Division

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

October 6, 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
  BODY OF THE PROCEDURE
  // A while loop.
  while [condition]
      [Something]
  // A for loop.
  for Var = [initial value] to [final value]
      [Something]
  // An if-elseif-else block.
  if [Condition1]
      Blah...
  elseif [Condition2]
      Blah...
  elseif [Condition3]
      Blah...
  else
      Blah...
  // A variable assignment.
  blah = A[j]
      // This is indented with a tab.
  // What is the output of this procedure?
  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. $\sum_{n}^{i=1}$
3. $f(x) = \lim_{n \to \infty} \frac{f(x)}{g(x)}$
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.

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

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

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

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

$$
\begin{gathered}
\text{minimize} \sum_{\mathrm{i}=1}^{\mathrm{c}} \mathrm{c_i} \cdot \mathrm{x_i} \\
\underline{x} \in S \\
\text{subject to}: \\
x_1 + x_4 = 0 \\
x_3 + 7 \cdot x_4 + 2 \cdot x_9 = 0
\end{gathered}
$$

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

Proof. This is a proof for BLAH . . . □

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.

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

Labeling an arrow: $\xrightarrow{ewq}$

Chapter 4

Statistics

Chapter 5

C++ Resources

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

1. [26]: http://www.tutorialspoint.com/cplusplus/cpp_stl_tutorial.htm
2. [8] 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) [47], Chp 23
   (b) [46], Chp 23
   (c) [14], Chp 18
   (d) [3], Chp 19
   (e) [10], Chp 1
3. IO Streams:
   (a) [10], Chp 2
   (b) [11], Chp 12. See all of [11–13].
   (c) [47], Chp 10-11
   (d) [46], Chp 10-11
   (e) [29], Chp 16
   (f) [49], Chp 10
   (g) [44], Chp 21
   (h) [3], Chp 28
   (i) [14], Chp 12
   (j) [34], Chp 17
   (k) [24], Chp 8
4. Templates:
   (a) [10], Chp 3
   (b) [9], Chp 16

   (c) [47], Chp 19
   (d) [46], Chp 19
   (e) [29], Chp 24
   (f) [49], Chp 6
   (g) [2], book; typelist - Chp 3
   (h) [44], Chp 18
   (i) [48], book
   (j) [1], book
   (k) [3], Chp 29
   (l) [14], Chp 11,21
   (m) [24], Chp 16
5. Debugging:
   (a) [10], Chp 11 (especially memory management problems, pp. 533)
6. STL containers:
   (a) [10], Chp 4
   (b) [45], Chp 8
   (c) [29], Chp 25
   (d) [49], Chp 7
   (e) [35], book
   (f) [3], Chp 18
   (g) [14], Chp 15-16
   (h) [34], Chp 16
   (i) [24], Chp 9,11
7. STL algorithms:
   (a) [10], Chp 5
   (b) [29], Chp 25
   (c) [49], Chp 7
   (d) [35], book
   (e) [3], Chp 18
   (f) [14], Chp 15,17
   (g) [34], Chp 16
   (h) [24], Chp 10
8. Function addresses:
   (a) [9], Chp 3, pp. 213
   (b) [47], Chp 8
   (c) [46], Chp 8
9. Dynamic memory management problems:
   (a) [9], Chp 6,13
   (b) [11], Chp 13. See all of [11–13].
   (c) [25], Chp 2-4
   (d) [44], Chp 29
   (e) [3], Chp 14
   (f) [14], Chp 10,22
   (g) [34], Chp 9,12
   (h) [24], Chp 12,13

10. Function overloading:
    (a) [9], Chp 7
    (b) [11], Chp 6. See all of [11–13].
    (c) [47], Chp 8
    (d) [46], Chp 8
    (e) [44], Chp 14
11. Operator overloading:
    (a) [9], Chp 12
    (b) [29], Chp 18
    (c) [44], Chp 15
    (d) [24], Chp 14
12. Constants:
    (a) [9], Chp 8
13. Functions and pointers:
    (a) [9], Chp 11:
        i. use const at the end of accessor functions
        ii. Do not use pointers as instance variables
    (b) [47], 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) [46], Chp 8
    (d) [29], Chp 15,20
    (e) [3], Chp 12-13
    (f) [34], Chp 7-8
    (g) [24], Chp 6
    (h) Elsewhere:
        i. You cannot call a non-const method from a const method. That would 'discard' the const qualifier.:
            A. http://stackoverflow.com/questions/2382834/discards-qualifiers-error
        ii. Pointer to constant data: _const type* variable;_ and _type const * variable;_
            A. http://www.cprogramming.com/reference/pointers/const_pointers.html
        iii. Pointer with constant memory address: _type * const variable = some-memory-address;_
            A. http://www.cprogramming.com/reference/pointers/const_pointers.html
        iv. Constant data with a constant pointer: _const type * const variable = some-memory-address;_ and _type const * const variable = some-memory-address;_
            A. http://www.cprogramming.com/reference/pointers/const_pointers.html
        v. http://stackoverflow.com/questions/1143262/what-is-the-difference-between-const- [27]:
            A. Read it backwards; the first _const_ can be on either side of the type.
            B. "Read pointer declarations right-to-left."
            C. From the answer of Ted Dennison, July 17, 2009. **Rule: The "const" goes after the thing it applies to. Putting const at the very front (e.g., const int *) is an exception to the rule.**
            D. int* – pointer to int
            E. int const * == const int * – pointer to const int
            F. int * const – const pointer to int

G. int const * const == const int * const – const pointer to const int
H. int ** – pointer to pointer to int
I. int ** const – A const pointer to a pointer to an int
J. int * const * – A pointer to a const pointer to an int
K. int const ** – A pointer to a pointer to a const int
L. int * const * const – A const pointer to a const pointer to an int

vi. For the following [27], let: *int var0 = 0;*
A. *const int &ptr1 = var0;* // Constant reference
B. *int * const ptr2 = &var0;* // Constant pointer
C. *int const * ptr3 = &var0;* // Pointer to const
D. *const int * const ptr4 = &var0;* // Const pointer to a const

14. OOD and inheritance:
    (a) [9], Chp 14,15
    (b) [11], Chp 13,14,15. See all of [11–13].
    (c) [47], Chp 9
    (d) [46], Chp 9
    (e) [29], Chp 13-14,21
    (f) [49], Chp 3-4,8
    (g) [3], Chp 24-26
    (h) [14], Chp 4-9
    (i) [34], Chp 10-11,13,14,15
    (j) [24], Chp 7,15,18,19
15. SW engineering issues:
    (a) [3], Chp 21
    (b) [14], Chp 24-26
16. multi-threading:
    (a) [45], Chp 3
17. graphs:
    (a) [45], Chp 7
18. typedef:
    (a) In the sandbox, use the *Make* target *make typedef* to study an example of how *typedef* can be used. When the *header file* defines/specifies the *typedef*, and is included in the *C++ implementation file* and other *C++ implementation file*s that instantiates those objects, it can be used subsequently without additional definition/specification. October 6, 2015.

Books to classify:

1. C++ programming: [17, 22, 23, 32, 33, 38, 39, 41–43]
2. C++ STL: [5–7, 15, 16, 19, 20, 36, 37]
3. C++ -based MPI programming: [21]
4. scientific computing: [30]
5. Boost C++: [28, 31, 40]

Table 5.1: Computational Complexity of Basic Operations of Containers from the *C++ STL*.

Container \ Complexity	add	remove	search	size	empty	begin	end
vector	O(1)	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
list	O(1)	O(n)	O(n)	O(1)	O(1)	O(1)	O(1)
queue	O(1) amortized	O(1)	O(n)	O(1)	O(1)	O(1)	O(1)
priority queue	O(log n)	O(log n)	O(n)	O(1)	O(1)	O(1)	???
set	O(log n)	O(log n)	O(log n)	O(1)	O(1)	O(1)	O(1)
multi-set	O(log n)	???	O(log n)	O(1)	O(1)	O(1)	O(1)
map	O(log n)	O(log n)	O(log n)	O(1)	O(1)	O(1)	O(1)
multi-map	O(log n)	???	O(log n)	O(1)	O(1)	O(1)	O(1)
stack	O(1)	O(1)	O(n)	O(1)	O(1)	O(1)	O(1)

5.1 Computational Complexity of C++ Containers

Table 5.1 shows a tabulated summary of containers in the *C++* Standard Template Library (STL) and the computational complexity for each of their common operations: add(element e), remove(element e), search(element e), size(), empty(), begin(), and end().

To conclude, we can get some facts about each data structure:

1. `std::list` is very very slow to iterate through the collection due to its very poor spatial locality.
2. `std::vector` and `std::deque` perform always faster than `std::list` with very small data
3. `std::list` handles very well large elements
4. `std::deque` performs better than a `std::vector` for inserting at random positions (especially at the front, which is constant time)
5. `std::deque` and `std::vector` do not support very well data types with high cost of copy/assignment

This draw simple conclusions on usage of each data structure [4, 18]:

1. Number crunching: use std::vector or std::deque
2. Linear search: use std::vector or std::deque
3. Random Insert/Remove:
4. Small data size: use std::vector
5. Large element size: use std::list (unless if intended principally for searching)
6. Non-trivial data type: use std::list unless you need the container especially for searching. But for multiple modifications of the container, it will be very slow.
7. Push to front: use std::deque or std::list

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] Dov Bulka and David Mayhew. Efficient C++: Performance Programming Techniques. Addison Wesley Longman, Inc., Indianapolis, IN, 2000.

[5] Marshall Cline. C++ FAQ Lite: Frequently asked questions. Available online from the course web page of *CS210 Data Structures and Abstractions Lab*(Spring 2015), Department of Computer Science, Faculty of Science, University of Regina at: http://www.cs.uregina.ca/Links/class-info/210/C++FAQ/; July 10, 2015 was the last accessed date, July 10 2000.

[6] Marshall Cline. C++ FAQ Lite: Frequently asked questions. Available online from the Computer Science Department, B. Thomas Golisano College of Computing and Information Sciences, Rochester Institute of Technology at: http://www.cs.rit.edu/~mjh/docs/c++-faq/; July 10, 2015 was the last accessed date, May 2 2003.

[7] Marshall Cline. C++ FAQ Lite: Frequently asked questions. Available online from the web page of Laura Mensi and Paolo Copello, *Tiscali Italia S.p.A.: Tiscali Webspace: Fanelia Italy – Computer Programming, Psychiatry, Escaflowne, and much more* at: http://web.tiscali.it/fanelia/cpp-faq-en/; July 10, 2015 was the last accessed date, July 28 2011.

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

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

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

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

[12] Tony Gaddis. Starting Out With C++: From Control Structures Through Objects. Addison-Wesley, Boston, MA, seventh edition, 2012.

[13] Tony Gaddis, Judy Walters, and Godfrey Muganda. Starting Out With C++: Early Objects. Addison-Wesley, Boston, MA, seventh edition, 2011.

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

[15] Hewlett-Packard Company staff. Standard template library programmer's guide. Available online in *SGI – The Trusted Leader in High Performance Computing: Tech Archive: Standard xTemplate Library Programmer's Guide* at: http://www.sgi.com/tech/stl/; September 30, 2015 was the last accessed date, 1994.

[16] Hewlett-Packard Company staff. STL complexity specifications. Available online in *SGI – The Trusted Leader in High Performance Computing: Tech Archive: Standard Template Library Programmer's Guide: Design documents: STL Complexity Specifications* at: http://www.sgi.com/tech/stl/complexity.html; September 30, 2015 was the last accessed date, http://www.sgi.com/tech/stl/complexity.html 2014.

[17] Cay S. Horstmann. C++ for Everyone. John Wiley & Sons, Hoboken, NJ, second edition, 2012.

[18] Nicolai M. Josuttis. The C++ Standard Library: A Tutorial and Reference. Addison-Wesley, Reading, MA, 1999.

[19] Nicolai M. Josuttis. The C++ Standard Library: A Tutorial and Reference. Pearson Education, Upper Saddle River, NJ, second edition, 2012.

[20] Björn Karlsson. Beyond the C++ Standard Library: An Introduction to Boost. Pearson Education, Boston, MA, 2006.

[21] George Em Karniadakis and Robert M. Kirby II. Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation. Cambridge University Press, Cambridge, U.K., 2003.

[22] Jayantha Katupitiya and Kim Bentley. Interfacing with C++: Programming Real-World Applications. Springer-Verlag Berlin Heidelberg, Heidelberg, Germany, 2006.

[23] Andrew Koenig and Barbara Moo. Accelerated C++: Practical Programming by Example. C++ In-Depth Series. Addison-Wesley, Boston, MA, 2000.

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

[25] 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.

[26] 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.

[27] Peter Mortensen. What is the difference between const int*, const int * const, and int const *? Available online from *Stack Exchange Inc.: Stack Overflow: Questions* at: http://stackoverflow.com/questions/1143262/what-is-the-difference-between-const-int-const-int-const-and-int-const; October 1, 2015 was the last accessed date, March 13 2015.

[28] Arindam Mukherjee. Learning Boost C++ Libraries: Solve Practical Programming Problems Using Powerful, Portable, and Expressive Libraries from Boost. Packt Publishing, Birmingham, West Midlands, England, U.K., July 2015.

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

[30] Joe Pitt-Francis and Jonathan Whiteley. Guide to Scientific Computing in C++. Undergraduate Topics in Computer Science. Springer-Verlag London, London, U.K., 2012.

[31] Antony Polukhin. Boost C++ Application Development Cookbook: Over 80 practical, task-based recipes to create applications using Boost libraries. Packt Publishing, Birmingham, West Midlands, England, U.K., 2013.

[32] Constantine Pozrikidis. Introduction to C++ Programming and Graphics. Springer Science+Business Media, LCC, New York, NY, 2007.

[33] Stephen Prata. C++ Primer Plus. Sams Publishing, Indianapolis, IN, fifth edition, 2005.

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

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

[36] Chris Riesbeck. Standard C++ containers. Available online from *Prof. Chris Riesbeck's web page: Programming: Useful C++ / Unix Resources*, Computer Science Division, Department of Electrical Engineering and Computer Science, Robert R. McCormick School of Engineering and Applied Science, Northwestern University at: `http://www.cs.northwestern.edu/~riesbeck/programming/c++/stl-summary.html`; September 30, 2015 was the last accessed date, July 3 2009.

[37] Robert Robson. Using the STL: The C++ Standard Template Library. Springer-Verlag Berlin Heidelberg New York, Heidelberg, Germany, second edition, 2000.

[38] Philip Romanik and Amy Muntz. Applied C++: Practical Techniques for Building Better Software. C++ In-Depth Series. Addison-Wesley, Boston, MA, 2003.

[39] Walter Savitch. Problem Solving with C++. Pearson Education, Boston, MA, seventh edition, 2009.

[40] Boris Schäling. The Boost C++ Libraries. Self-published, 2012.

[41] Edward Scheinerman. C++ for Mathematicians: An Introduction for Students and Professionals. Chapman & Hall/CRC, Boca Raton, FL, 2006.

[42] Herbert Schildt. C++: The Complete Reference. McGraw-Hill, Berkeley, CA, third edition, 1998.

[43] Herbert Schildt. C++ from the Ground Up. McGraw-Hill/Osborne, Berkeley, CA, third edition, 2003.

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

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

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

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

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

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