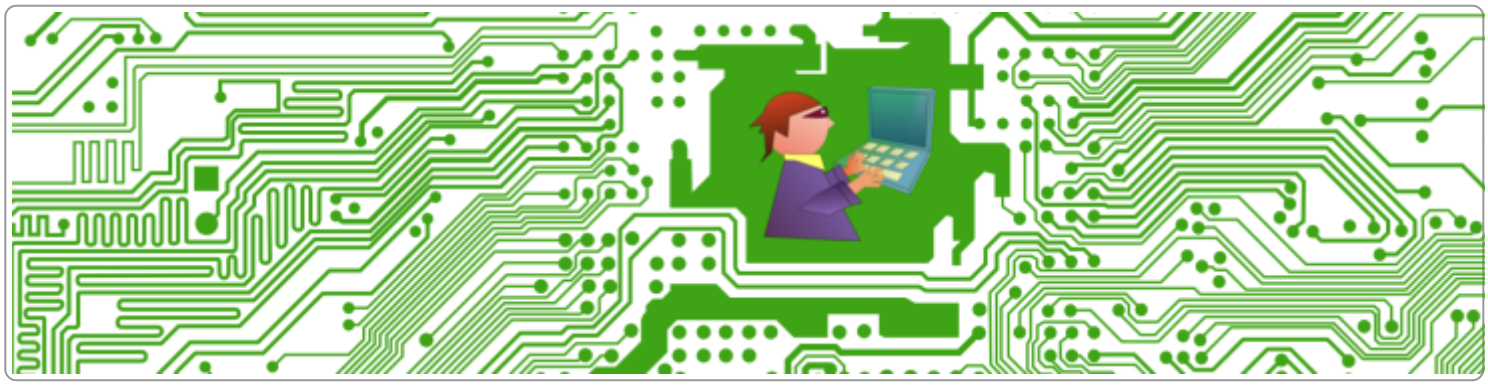


Introduction to Loop Invariants

Posted by admin on January 22, 2012 Go to comments Leave a comment (4)

Introduction

Today, I am going to talk about a confusing concept in algorithm design. I think you have already guessed from the title of this article. Yes, let us talk about the puzzling topic of Loop Invariants. I personally believe it is a mathematical concept more than it is a computer science concept simply because it is directly related to mathematical proofs of correctness. I am not going to approach the topic from a mathematical side though. I will do my best to describe it in words and provide few examples so that the average software engineer or computer science student can get a basic understanding. With that said, let us get started.

When developing computer programs, I bet you, we all use loop invariants in the back of our minds without realizing that, so what on earth is a loop invariant and why should we even care about learning the topic in the first place?. We usually write code not to (Yes not to) solve problems but to implement solutions for problems that are already solved in advance. Solving problems by directly writing code may work for trivial problems but as the complexity increases then tweaking the code through trial and error is not going to be the best approach. There should be a better way to write correct programs.

Programs are developed to process some input and generate an output of particular sort. In Software Engineering, we refer to the characterization of input and output as software requirement specification. On the other hand, there is software design specification to describe the underlying logic and algorithm design. Writing the actual code and testing comes later in the process. As you can see, starting with input then

ending with the desired output means there is eventually a goal to achieve. The question is, how all this relate to the main topic i.e. loop invariants?

Well, the concept of "Loop Invariants" helps us keep focus on the final goal of a given algorithm. It is a smart way to make us confident about the correctness of our code without loosing track of what we are intending to achieve. Loop invariants can be used to prove the correctness of an algorithm, debug an existing algorithm without even tracing the code or develop an algorithm directly from specification. We can not just rely on testing to make sure our programs are correct because testing only shows the presence of bugs not the absence. The only way to verify the correctness of software through testing is to cover all possible input combinations. This is indeed a suicidal approach, it is not practical and impossible to achieve. In summary, loop invariants are used to formally reason about the correctness of code. Let us now talk a little bit about loops before we dive into loop invariants.

Few words about loops

Simple programming constructs such as assignment and "if statements" are not enough to develop non trivial algorithms. In order for an algorithm to achieve something useful, probably it involves one or more loops. Loops are directly related to iteration. Each time a loop executes we call that an iteration. Iteration gradually advances the algorithm towards achieving the desired goal. A loop in the simplest form has a body and Boolean condition called the guard. The body contains the main logic to achieve the desired goal of the algorithm. The guard determines when the loop terminates. In the following section we will show how loops and invariants are related.

What is a loop invariant?

So far, we talked about loop invariants in general then described what a loop means. Now the question is how the two are linked together and how is that related to algorithm correctness. Loop invariant is nothing but a condition or a logical expression that must evaluate to true as long as the loop is progressing towards achieving the desired goal. It is directly linked to the body and guard of the loop. Loop invariant for an algorithm is like the GPS device in a car. As long as you are on track then you must reach your destination. In order to identify a loop invariant for a given algorithm you need to find a condition that better describes the goal of the algorithm. As the loop advances that condition remains true and gives an idea about the current progress towards the final goal. By the time the loop guard evaluates to false, the loop terminates and the overall goal must have been achieved.

Loop invariants in words

To better understand the concept behind loop invariants let us indicate a real life example in plain English. Consider a soccer team which typically consists from 11 players. During the game time there must be 11 players from each team in the soccer field. I am assuming no player expulsion. In this case, a soccer game invariant could be the statement "There must be 11 players in the field from each team during the game". As you can see, the total number of players from each team is fixed (11) and does not change even though if a given player needs to be substituted. If one player gets in, another player goes out which means the invariant stays true during the game.

Ask the following questions

In order to identify a correct loop invariant, ask the following questions:

1. Does the initialization (before the loop) make the invariant true?
2. Does the invariant (together with a false loop guard) imply the goal (or post condition)? In other words, does the invariant capture the correctness and the meaning of the loop?
3. Does the loop make progress toward termination (achieving the goal)
4. As the loop progresses towards the goal, does it preserve the invariant?

Where does a loop invariant evaluate to true?

Loop invariant must evaluate to true just before entering the loop and after each iteration as shown in the following pseudo code block:

```
view plain  copy to clipboard  print  ?
01. //Invariant must be true before entering the loop
02. while (Loop guard is true)
03. {
04.     //Loop body
05.     //Invariant must be true after each iteration
06. }
```

Identifying loop invariants

Generally speaking, loop invariants can be identified by applying the following steps:

1. Identify the goal of the loop and write it as a post condition
2. Write the loop specifying the guard (loop condition)
3. Fill in the loop invariant
4. Fix the initialization so that the loop invariant evaluate to true
5. Figure out how to achieve the goal by filling in the body of the loop

In order to demonstrate the steps above let us take an example. Suppose we want to print the elements of an array (A) of (10) integers.

1. Identify the goal of the loop and write it as a post condition

```
view plain  copy to clipboard  print  ?
01. //Elements A[1] to A[10] have been printed
```

2. Write the loop specifying the guard (loop condition)

```
view plain  copy to clipboard  print  ?
01. while (k <= 10)
02. {
03.     k++;
04. }
```

```
05.
06.  //Elements A[1] to A[10] have been printed
```

3. Fill in the loop invariant

```
view plain  copy to clipboard  print  ?

01.  //Invariant: Elements A[1] to A[k-1] have been printed”
02.  while (k <= 10)
03.  {
04.      k++;
05.      //Invariant: “Elements A[1] to A[k-1] have been printed”
06.  }
07.
08.  //Elements A[1] to A[10] have been printed
```

4. Fix the initialization so that the loop invariant evaluate to true

```
view plain  copy to clipboard  print  ?

01.  int k = 1;
02.  //Invariant: “Elements A[1] to A[k-1] have been printed”
03.  while (k <= 10)
04.  {
05.      k++;
06.      //Invariant: “Elements A[1] to A[k-1] have been printed”
07.  }
08.
09.  //Elements A[1] to A[10] have been printed
```

5. Figure out how to achieve the goal by filling in the body of the loop

```
view plain  copy to clipboard  print  ?

01.  int k = 1;
02.  //Invariant: “Elements A[1] to A[k-1] have been printed”
03.  while (k <= 10)
04.  {
05.      print(k);
06.      k++;
07.      //Invariant: “Elements A[1] to A[k-1] have been printed”
08.  }
09.
10.  //Elements A[1] to A[10] have been printed
```

Let us go through the example above step by step:

1. The goal of the loop is to print all elements of the array (recall that array size (10) is given). We simply indicate that the post condition should be "Elements A[1] to A[10] have been printed".
2. Processing an array of (10) elements requires a loop that terminates after (10) iterations. This is trivial for sure but it is only used for the purpose of demonstration. The guard of the loop checks if the index is within the array limits. The loop progresses towards termination as we increment the loop index every iteration.
3. Loop invariant says array elements A[1] to A[k-1] have already been processed. Note that it is directly related to the loop guard and the goal of the loop as well. It is defined in terms of (k) which is the loop index. Note that the loop index (k) is used in the loop guard to terminate the loop when (k > 10). Loop invariant tells us about the current progress towards achieving the goal. In this case the eventual goal is to print the elements of the array. At each iteration we will

be having some value for (k) so the statement "elements A[1] to A[k-1] have already been processed" indicates (k-1) elements have already been processed (processed means printed in our example) so far. In order for the invariant to hold true just before entering the loop, some variables need to be initialized correctly as we will show in the next step.

4. We should initialize (k = 1) so that the invariant holds true just before entering the loop. The invariant says "Array elements A[1] to A[k-1] have already been processed" but we know that no elements have been processed yet before entering the loop. When (k = 1) the invariant evaluates to "Array elements A[1] to A[0] have been already processed". Our array starts at (1) so the range A[1] to A[0] contains no elements. After the first element is printed out inside the loop we increment the index (k) so we get (2) and the invariant evaluates to "Array elements A[1] to A[1] have been already processed" which simply means the first element have already been printed out which is correct. As the loop progresses toward termination the invariant keeps holding true.
5. The last step is to fill the loop body so that the final goal is achieved. In our case it is trivial to print one array element. A simple print statement will do the job.

Summary

Loop invariants are used to reason about the correctness of computer programs. Intuition or trial and error can be used to write easy algorithms however when the complexity of the problem increases, it is better to use formal methods such as loop invariants.

We are done with part 1. In the next article we will discuss a new loop invariant example "Array Sum".

Please use the comments section below for questions, corrections or feedback. Thanks for reading.

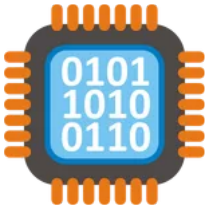
Search Terms...

- loop invariant tutorial

More from my site

Python for loop

Python Power Function



Computer science vs computer engineering vs software engineering

Difference Between User Level Threads and Kernel Level Threads

Understanding Digital Certificates and SSL

Sorting Algorithms Explained by Examples

Like this:

★ Like

Be the first to like this.

Algorithms and Data Structures, Articles and Tutorials Loop Invariant

← Dynamic Programming Integer Knapsack

Loop Invariant Array Sum →

Leave a comment ? 4 Comments.

Charlie Guan June 15, 2013 at 2:37 PM Reply

Can I just say thank you so much! This is so helpful!!

I really appreciate your clean, understandable presentation, and the fact that you took so much effort writing and editing for something that may not give you any financial benefit. I really admire your kind heart for teaching!

I know I have no right to say "keep working because we like it!", since as I said, this probably don't give you any financial benefit; however, I just want you to know that your work is an inspiration and a big help for a lot of people! (although most of them didn't take the time to leave a comment to tell you)

Thanks again!

Charlie Guan

Mohammed Abualrob June 15, 2013 at 2:42 PM

Reply

Thanks for the kind words

Eduardo February 19, 2016 at 8:11 AM

Reply

Very good job.
Loop invariant is generally used do prove the correctness of an algorithm. Should be interesting articles about correctness and completeness.

Anonymous January 1, 2017 at 9:18 AM

Reply

It helped me a lot.
You explained it in simple way

Leave a Reply

Enter your comment here...

Subscribe

Get notified when new articles are posted

Email

Subscribe

Featured

Python REST API Example
Python mutex example
Python For Loop
Python unit test example
Python power function
Palindrome in Python
Python anagram solver
Repeated characters in Python
Second largest number in Python
Python character image generator

Categories

Algorithms and Data Structures
Articles and Tutorials
Code Snippets
Computer Networks
Computer Security
Database Systems
General Topics
Interview Questions
Math and Probability
Operating Systems
Software Quality Assurance

Popular

Difference between Multiprogramming, Multitasking,…
Difference Between User Level Threads and Kernel Level…
Monolithic Kernel vs Microkernel OS Architectures
Difference between computer architecture and computer…
Difference Between System Call, Procedure Call and Function
Iterative Binary Search Function
Sikuli Java Tutorial
Sikuli Selenium Robot Framework Tutorial
Dynamic Programming Assembly Line Scheduling
File Allocation Methods in Operating System

Tags

Cpp
Recursion
Big O
String
Optimization
Perl
Java
Factorial

Combinatorial Lecture Notes Python

Interview Divide & Conquer Sort

Hash Dynamic Programming SQL

Loop Invariant C# Windows

UNIX Binary Search Pixel Bender

Greedy Flex Pixel Shader SSL

Linux Android Tree Sikuli

Linked List Ubuntu Bitwise

Having Clause Exponential PHP

Unity3d Stack Robot Framework

Selenium Wavelet RegExp

Jython LZW RPC-Xml

Edge Detection SCCS Google

DBI