Loop Invariants

https://courses.missouristate.edu/anthonyclark/325
Outline

Topics and Learning Objectives
• Practice writing loop invariants

Assessments
• Loop Invariant activity
Extra Materials

• **Chapter 2** of *Introduction to Algorithms*, Third Edition

• [https://www.win.tue.nl/~kbuchin/teaching/JBP030/notebooks/loop-invariants.html](https://www.win.tue.nl/~kbuchin/teaching/JBP030/notebooks/loop-invariants.html)
Loop Invariants

A loop invariant is a **predicate** (a statement that is either true or false) with the following properties:

1. It is true upon entering the loop the first time. **Initialization**

2. If it is true upon starting an iteration of the loop, it remains true upon starting the next iteration. **Maintenance**

3. The loop terminates, and the loop invariant plus the reason that the loop terminates gives you the property that you want. **Termination**
How to perform a proof by loop invariant

1. State the loop invariant
   1. A statement that can be easily proven true or false
   2. The statement must reference the purpose of the loop
   3. The statement must reference variables that change each iteration

2. Show that the loop invariant is true before the loop starts

3. Show that the loop invariant holds when executing any iteration

4. Show that the loop invariant holds once the loop ends
Example

FUNCTION SumArray(array)
    sum = 0
    i = 0
    WHILE i < array.length
        sum = sum + array[i]
        i = i + 1

1. State the loop invariant
   1. A statement that can be easily proven true or false
   2. The statement must reference the purpose of the loop
   3. The statement must reference variables that change each iteration

What would be a good loop invariant for proving this procedure?
Upon entering an iteration of the loop, sum = the sum of all values with index < i

FUNCTION SumArray(array)
    sum = 0
    i = 0
    WHILE i < array.length
        sum = sum + array[i]
        i = i + 1

1. State the loop invariant
   1. A statement that can be easily proven true or false
   2. The statement must reference the purpose of the loop
   3. The statement must reference variables that change each iteration
Example

**FUNCTION** SumArray(array)
    sum = 0
    i = 0
    WHILE i < array.length
        sum = sum + array[i]
        i = i + 1

Upon entering an iteration of the loop, sum = the sum of all values with index < i

1. Initialization
2. Maintenance
3. Termination
Example

FUNCTION SumArray(array)
    sum = 0
    i = 0
    WHILE i < array.length
        sum = sum + array[i]
        i = i + 1

Initialization:
Upon entering the first iteration, i = 0. No numbers have an index < i. The sum of no terms is the identity for addition (0).
Example

**FUNCTION** SumArray(array)

```plaintext
    sum = 0
    i = 0
    WHILE i < array.length
        sum = sum + array[i]
        i = i + 1
```

Maintenance:
Upon entering an iteration, with a given value of \( i \), suppose that the loop invariant is true, so that:

\[
    sum = \sum_{i=0}^{i-1} array[i]
\]

The current iteration adds \( A[i] \) to sum and then increments \( i \), so that the loop invariant holds entering the next iteration.
Example

FUNCTION SumArray(array)
    sum = 0
    i = 0
    WHILE i < array.length
        sum = sum + array[i]
        i = i + 1

Termination:
The loop terminates once i = n. According to the loop invariant

\[ sum = \sum_{i=0}^{i-1} array[i] = \sum_{i=0}^{n-1} array[i] \]

which is all values in the array.
A more complex example: Dijkstra’s Algorithm

\[ \text{DIJKSTRA} (G, w, s) \]

\begin{itemize}
  \item \text{S} = \text{null}
  \item \text{Q} = G.V
  \item \text{while} \ Q \text{ is not null}
    \begin{itemize}
      \item \text{u} = \text{EXTRACT-MIN}(Q)
      \item \text{S} = \text{S union } \{\text{u}\}
      \item \text{for each vertex } v \text{ adjacent to } u
        \begin{itemize}
          \item \text{RELAX}(u, v, w)
        \end{itemize}
    \end{itemize}
\end{itemize}

Loop Invariant:
At the start of each iteration of the while loop, \( v.d = \text{delta}(s, v) \) for each vertex \( v \) in \( S \).
Dijkstra’s Algorithm

\[ \text{DIJKSTRA} \ (G, w, s) \]

\[ S = \text{null} \]
\[ Q = G.V \]

\[ \text{while} \ \text{Q is not null} \]
\[ \quad u = \text{EXTRACT-MIN}(Q) \]
\[ \quad S = S \cup \{u\} \]
\[ \quad \text{for each vertex v adjacent to u} \]
\[ \quad \quad \textbf{RELAX}(u, v, w) \]

\[ \text{Initialization:} \]
Initially, \( S = \text{null} \) and so the invariant is trivially true

\[ \text{Loop Invariant:} \]
At the start of each iteration of the while loop, \( v.d = \text{delta}(s, v) \) for each vertex \( v \) in \( S \).
Dijkstra’s Algorithm

\textsc{Dijkstra} (G, w, s)
\begin{itemize}
\item \textit{S} = null
\item \textit{Q} = G.V
\item \textbf{while} \textit{Q} is not null
\begin{itemize}
\item \textit{u} = \textsc{Extract-Min}(\textit{Q})
\item \textit{S} = \textit{S} union \{\textit{u}\}
\item \textbf{for} each vertex \textit{v} adjacent to \textit{u}
\begin{itemize}
\item \textsc{Relax}(\textit{u}, \textit{v}, w)
\end{itemize}
\end{itemize}
\end{itemize}

\textbf{Loop Invariant:}
At the start of each iteration of the while loop, v.d = \text{delta}(s, v) for each vertex v in S.

\textbf{Maintenance:}
<long proof by contradiction on page 661 of the textbook>
Dijkstra’s Algorithm

\[ \text{DIJKSTRA} \ (G, w, s) \]

\[ S = \text{null} \]
\[ Q = G.V \]
\[ \text{while } Q \text{ is not null} \]
\[ u = \text{EXTRACT-MIN}(Q) \]
\[ S = S \cup \{u\} \]
\[ \text{for each vertex } v \text{ adjacent to } u \]
\[ \text{RELAX}(u, v, w) \]

**Loop Invariant:**
At the start of each iteration of the while loop, \( v.d = \text{delta}(s, v) \) for each vertex \( v \) in \( S \).

**Termination:**
At termination, \( Q = \text{null} \) which, along with our earlier invariant that \( Q = V - S \), implies that \( S = V \).
Thus, \( u.d = \text{delta}(s, u) \) for all vertices in \( G.V \).