Loop Invariants

https://cs.pomona.edu/classes/cs140
Outline

Topics and Learning Objectives
• Practice writing loop invariants

Exercises
• Loop Invariant
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 Invariant Proofs

• A procedural way to prove the correctness of some code with a loop

• Very similar to inductive proofs for recursive algorithms
**FUNCTION** SumArray(array)

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

How do we prove that this code sums all values in the given array?

Some useful syntax:

- `array[start ..= end]` is the subarray
  - Including `array[start]`, `array[end]`, and everything in between

- `array[start ..< end]` is the subarray
  - Including `array[start]`, excluding `array[end]`, and including everything in between
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
Loop Invariant

1. **At the start of the iteration with** <reference the looping variable>, ...
   - For example: “index j”

2. **The** <reference to partial solution> ...
   - For example: “subarray array[0 ..= j−1] consists of the elements originally in array[0 ..= j−1]...”

3. **<something about why the partial solution is correct>**.
   - For example: “In sorted order.”
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?
Example

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

At the start of the iteration with index $i$, the variable $sum$ is the sum of all values in the subarray $array[0 \ldots < i]$. 

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

At the start of the iteration with index \( i \), the variable \( \text{sum} \) is the sum of all values in the subarray \( \text{array}[0..<i] \).
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, \( i = 0 \). There are no numbers in the subarray \( \text{array}[0 \ldots < i] \). The sum of no terms is the identity for addition (0).
Example

FUNCTION SumArray(array)

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

Maintenance:
Upon entering an iteration with index $i$, assume that $\text{sum}$ is equal to the sum of all values in the subarray $\text{array}[0 ..< i]$: 

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

The current iteration adds $\text{array}[i]$ to $\text{sum}$ and then increments $i$, so that the loop invariant holds upon 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 with i = n. According to the loop invariant, sum is equal to the sum of all values in the subarray array[0 ..< i]:

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

which is the sum of all values in the array.
A more complex example: 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 \( S \).
Dijkstra’s Algorithm

**DIJKSTRA** (G, w, s)

1. S = null
2. Q = G.V
3. while Q is not null
   3.1. u = EXTRACT-MIN(Q)
   3.2. S = S union {u}
   3.3. for each vertex v adjacent to u
       3.3.1. RELAX(u, v, w)

**Initialization:**
Initially, S = null and so the invariant is trivially true.

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

\textsc{Dijkstra} (G, w, s)

\begin{align*}
S &= \text{null} \\
Q &= G.V \\
\text{while } Q \text{ is not null} \\
&\quad u = \text{EXTRACT-MIN}(Q) \\
&\quad S = S \cup \{u\} \\
&\quad \text{for each vertex } v \text{ adjacent to } u \\
&\quad \text{RELAX}(u, v, w)
\end{align*}

\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

**DIJKSTRA** (G, w, s)

S = null

Q = G.V

while Q is not null

u = **EXTRACT-MIN**(Q)

S = S union {u}

for each vertex v adjacent to u

**RELAX**(u, v, w)

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

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