CSE 331
Software Design & Implementation

James Wilcox
Autumn 2021
Lecture 4½ – Reasoning Wrap-up
Updates

• Lots going on!
  – HW1 due Monday, October 11 at 5pm
  – HW3 due Thursday, October 14 at 11pm
  – HW2 due the following Monday, October 18 at 5pm
Interview Question
Sorted Matrix Search

Problem Description

Given a matrix $M$ (of size $m \times n$), where every row and every column is sorted, find out whether a given number $x$ is in the matrix.
Sorted Matrix Search

Given a sorted matrix $M$ (of size $m \times n$), where every row and every column is sorted, find out whether a given number $x$ is in the matrix.

(darker color means larger)
Sorted Matrix Search

Given a sorted matrix M (of size m x n), where every row and every column is sorted, find out whether a given number x is in the matrix.

(One) Idea: Trace the contour between the numbers ≤ x and > x in each row to see if x appears.
Sorted Matrix Search Code

Partial Invariant: $M[i,0], ..., M[i,j-1] < x \leq M[i,j], ..., M[i,n-1]$

- for each $i$, holds for exactly one $j$
- holds when we are in the right spot in row $i$
Sorted Matrix Search Code

Initialization:

Partial Invariant: \( M[i,0], \ldots, M[i,j-1] < x \leq M[i,j], \ldots, M[i,n-1] \)

How do we get the invariant to hold with \( i = 0 \)?
- no easy way to initialize it so the invariant holds
- we need to search...
Sorted Matrix Search Code

Initialization: $i$

New goal: $M[0,0], ..., M[0,j-1] < x \leq M[0,j], ..., M[0,n-1]$

- will need a loop to find $j$
- Loop invariant: $x \leq M[0,j], ..., M[0,n-1]$
  - weakening of the new goal
  - decrease $j$ until we get $M[0,j-1]$ to also hold
Sorted Matrix Search Code

Initialization:

```
int i = 0;
int j = ?
{{ Inv: x ≤ M[i,j], ..., M[i,n-1] }}
while ( ?? )
  ??
  {{ M[i,0], ..., M[i,j-1] < x ≤ M[i,j], ..., M[i,n-1] }}
```

What is the easiest way to make this hold initially?
Sorted Matrix Search Code

Initialization:

```
int i = 0;
int j = n;
{% Inv: x \leq M[i,j], ..., M[i,n-1] %}
while ( ?? )
    ??
{% M[i,0], ..., M[i,j-1] < x \leq M[i,j], ..., M[i,n-1] %}
```
Sorted Matrix Search Code

Initialization:

```
int i = 0;
int j = n;
{{ Inv: x ≤ M[i,j], ..., M[i,n] }}
while ( ?? )
  ??
  {{ M[i,0], ..., M[i,j-1] < x ≤ M[i,j], ..., M[i,n] }}
```

When does the postcondition hold? (Careful!)
Sorted Matrix Search Code

\[
\begin{aligned}
\text{Initialization:} & \\
\text{int } i = 0; & \\
\text{int } j = n; & \\
\{ \text{ Inv: } x \leq M[i,j], \ldots, M[i,n-1] \} & \\
\text{while } (j > 0 \text{ and } x \leq M[i,j-1]) & \\
\quad \text{??} & \\
\{ \text{ M[i,0], } \ldots, \text{ M[i,j-1] } < x \leq \text{ M[i,j], } \ldots, \text{ M[i,n-1] } \} & \\
\end{aligned}
\]
Sorted Matrix Search Code

Initialization:

```
int i = 0, j = n;
{{ Inv: x ≤ M[i,j], ..., M[i,n-1] }}
while (j > 0 && x <= M[i,j-1]) {
    ??
    j = j - 1;
}
{{ M[i,0], ..., M[i,j-1] < x ≤ M[i,j], ..., M[i,n-1] }}
```
Sorted Matrix Search Code

Initialization:

\[
\begin{align*}
\text{int } i &= 0, \ j = n; \\
\{ \text{ Inv: } x &\leq M[i,j], \ldots, M[i,n-1] \} \\
\text{while } \ (j > 0 \ \&\& \ x \leq M[i, j-1]) \ {\{ x \leq M[i,j], \ldots, M[i,n-1] \text{ and } x \leq M[i,j-1] \}} \\
&\quad \quad \quad \quad \quad \{ x \leq M[i,j-1], \ldots, M[i,n-1] \} \\
&\quad \quad \quad \quad \quad \{ x \leq M[i,j], \ldots, M[i,n-1] \} \\
\{ M[i,0], \ldots, M[i,j-1] < x \leq M[i,j], \ldots, M[i,n-1] \}\end{align*}
\]
Sorted Matrix Search Code

Initialization:

\[
\begin{array}{|c|c|c|c|c|}
\hline
& & & & j \\
\hline
& & & & \\
\hline
& & & & \\
\hline
& & & & \\
\hline
& & & & \\
\hline
& & & & \\
\hline
i & & & & \\
\hline
\end{array}
\]

\[
\text{int } i = 0, j = n;
\]

\{
\text{Inv: } x \leq M[i,j], ..., M[i,n-1] \}

while (j > 0 && x \leq M[i,j-1]) {

    j = j - 1;

}

\{
\text{M[i,0], ..., M[i,j-1] < x \leq M[i,j], ..., M[i,n-1]} \}
int i = 0;
int j = n;

{{ Inv: $x \leq M[i,j]$, ..., $M[i,n-1]$ }}

while ($j > 0$ && $x \leq M[i,j-1]$)
    j = j - 1;

{{ $M[i,0]$, ..., $M[i,j-1] < x \leq M[i,j]$, ..., $M[i,n-1]$ }}
Sorted Matrix Search Code

That finds the right column in row 0
• can now check $M[0,j] = x$ (if $j < n$)
• if not, we can move onto the next row
  – $x$ cannot be anywhere in the row if it’s not at $M[i,j]$  
  – set $i = i + 1$

Process continues in each row thereafter...
Sorted Matrix Search Code

- Make progress by setting \( i = i + 1 \)
- When \( i \) increases, the invariant may be broken
  - we have \( x \leq M[i,j] \leq M[i+1,j] \) since columns are sorted
  - and \( M[i+1,j] \leq M[i +1,j+1], \ldots, M[i +1,n-1] \) since rows are sorted
  - so we get \( x \leq M[i +1,j], \ldots, M[i +1,n-1] \)
Sorted Matrix Search Code

- Make progress by setting $i = i + 1$
- When $i$ increases, the invariant may be broken
  - we have $x \leq M[i+1,j], ..., M[i+1,n-1]$
  - may need to restore invariant for $M[i,0], ..., M[i,j-1] < x$
  - decrease $j$ until it holds again...
    - when have we seen this before?
    - initialization
Sorted Matrix Search Code

- Make progress by setting $i = i + 1$
- When $i$ increases, the invariant may be broken
  - we have $x \leq M[i+1,j], \ldots, M[i+1,n-1]$
  - may need to restore invariant for $M[i,0], \ldots, M[i,j-1] < x$
  - could copy and paste the same loop
    - or you can do it with one copy
Sorted Matrix Search Code

**instead of**

```c
int i = 0, j = n;
[move j left]
{{ Inv: M[i,0], ..., M[i,j-1] < x ≤ M[i,j], ..., M[i,n-1] }}
while (i != m) {
    i = i + 1;
    [move j left]
}
```

**we can write**

```c
int i = 0, j = n;
while (i != m) {
    [move j left]
    {{ M[i,0], ..., M[i,j-1] < x ≤ M[i,j], ..., M[i,n-1] }}
i = i + 1;
}
```
Sorted Matrix Search Code

```cpp
int i = 0;
int j = n;

while (i != m) {
    {{ Inv: x ≤ M[i,j], ..., M[i,n-1] }}
    while (j > 0 && x ≤ M[i,j-1])
        j = j - 1;
    {{ M[i,0], ..., M[i,j-1] < x ≤ M[i,j], ..., M[i,n-1] }}
    if (j < n && x == M[i,j])
        return true;
    i = i + 1;
}
return false;
```
int i = 0;
int j = n;
{{ Inv: x not in M[k,l] for k < i and x ≤ M[i,j], ..., M[i,n-1] }}

while (i != m) {
    {{ Inv: x not in M[k,l] for k < i and x ≤ M[i,j], ..., M[i,n-1] }}
    while (j > 0 && x <= M[i,j-1])
        j = j - 1;

    {{ x not in M[k,l] for k < i and M[i,0], ..., M[i,j-1] < x ≤ M[i,j], ..., M[i,n-1] }}
    if (j < n && x == M[i,j])
        return true;
    i = i + 1;
}
return false;
Worksheet
Reasoning Summary
Reasoning Summary

• Checking correctness can be a mechanical process
  – using forward or backward reasoning

• This requires that loop invariants are provided
  – those cannot be produced automatically

• As long as you document your loop invariants, it should not be too hard for someone else to review your code
Documenting Loop Invariants

- Write down loop invariants for all non-trivial code
- They are often best avoided for “for each” loops:

```java
{{ Inv: printed all the strings seen so far }}
for (String s : L)
    System.out.println(s);
```
Documenting Loop Invariants

• Write down loop invariants for all non-trivial code

• They are often best avoided for “for each” loops:

```java
// Print the strings in L, one per line.
for (String s : L)
    System.out.println(s);
```
Documenting Loop Invariants

• Write down loop invariants for all non-trivial code

• They are often best avoided for “for each” loops:

\[
\text{{{ Inv: B has }2\times x + 1\text{ for each element }x\text{ removed so far }}}
\]

\[
\text{for (int x : A)}
\]
\[
\text{B.add(2* x + 1);}
\]
Documenting Loop Invariants

• Write down loop invariants for all non-trivial code

• They are often best avoided for “for each” loops:

```java
for (int x : A)
    B.add(2*x + 1);
```
Documenting Loop Invariants

- Write down loop invariants for all non-trivial code
- They are often best avoided for “for each” loops.
- Invariants are more helpful when a variable incorporates information from multiple iterations
  - e.g., \{ s = A[0] + \ldots + A[i-1] \}
- *Use your best judgement!*
Reasoning Summary

• You can check correctness by reasoning alone

• Correctness: tools, inspection, testing
  – reasoning through your own code
  – do code reviews

• Practice!
  – essential skill for professional programmers
Reasoning Summary

• You will eventually do this in your head for most code

• Formalism remains useful
  – especially tricky problems
  – interview questions (often tricky)
    • see last example…
Next Topic…
A Problem

“Complete this method such that it returns the location of the largest value in the first $n$ elements of the array $arr$.”

```java
int maxLoc(int[] arr, int n) {
  ...
}
```
One Solution

```java
int maxLoc(int[] arr, int n) {
    int maxIndex = 0;
    int maxValue = arr[0];
    // Inv: maxValue = max of arr[0] .. arr[i-1] and
    //      maxValue = arr[maxIndex]
    for (int i = 1; i < n; i++) {
        if (arr[i] > maxValue) {
            maxIndex = i;
            maxValue = arr[i];
        }
    }
    return maxIndex;
}
```

Is this code correct?

What if \( n = 0 \)?
What if \( n > \text{arr.length} \)?
What if there are two maximums?
A Problem

“Complete this method such that it returns the location of the largest value in the first n elements of the array arr.”

```java
int maxLoc(int[] arr, int n) {
    ...
}
```

Could we write a specification so that this is a correct solution?

- throw IllegalArgumentException if n <= 0
- throw ArrayOutOfBoundsException if n > arr.length
- return smallest index achieving maximum
Morals

• You can all write the code correctly

• Writing the specification was harder than the code
  – multiple choices for the “right” specification
    • must carefully think through corner cases
  – once the specification is chosen, code is straightforward
  – (both of those will be recurrent themes)

• Some math (e.g. “if n <= 0”) often shows up in specifications
  – English (“if n is less or equal to than 0”) is often worse
How to Check Correctness

• Step 1: need a specification for the function
  – can’t argue correctness if we don’t know what it should do
  – surprisingly difficult to write!

• Step 2: determine whether the code meets the specification
  – apply reasoning
  – usually easy with the tools we learned