COSC 2011 Section N

Tuesday, March 20 2001

Overview

- Review of Assertions
  - Pre and post conditions
- Loop Invariants
- Bubble Sort Algorithm
- Mathematical Induction
  - Definition
  - Examples
- Assignment 1 Notes/Questions?

Assertions - Review: (1)

- Assertions:
  - A logical condition that is assumed to hold at some point of program execution.
  - Precondition:
    - Some condition before the start of execution.
    - May not have any!
  - Postconditions:
    - If precondition is true and code is executed, these conditions will now hold.

Assertions - Review: (2)

- Notation:

$\{P\} \rightarrow S \rightarrow \{Q\}$

If ***precondition*** P is true before execution of statements S, then the ***postcondition*** Q will be true after execution.

Examples:

1. $\{b=1\} \rightarrow b = b+1 \rightarrow \{b=2\}$
2. {true} $\rightarrow$ stack.push("y") $\rightarrow$ {stack is non-empty}

Loop Invariants: (1)

- An assertion that remains true each time the statements of a loop are executed
  - Tells us something about the values of the loop variables while executing a loop.
  - Should be true at the beginning of each iteration, including the first!

Loop Invariants: (2)

- Consider the following loop:

while C **do** S

'p' is a loop invariant if:

$(p \wedge C) \rightarrow \{S\} \rightarrow p$

- ◆If p is a loop invariant and p is true before the loop is executed and the loop terminates then:
  - ⋆p is true and C is not true.

Loop Invariants: (3)

- Conclusion:

$\{p\}$ **while** C **do** S $\rightarrow \{p \wedge \neg C\}$

- Book's Variation:
  - ◆To prove p, define S in terms of a series of smaller statements $S_0, S_1, \ldots, S_k$
    - ⋆S_0 is true before loop
    - ⋆If S_{i-1} is true before iteration i then S_i is true after iteration i is over.
    - ⋆S_k implies p is true.

Loop Invariants: (4)

- Examples:
  - ◆Array maximum problem
    - ⋆Find the max. element in array A of n elements:

```
Algorithm arrayMax(A, n):
  Input: An array A storing n integers.
  Output: The maximum element in A.
 currentMax ← A[0]
 for i ← 1 to n −1 do
   if currentMax < A[i] then
     currentMax ← A[i]
 return currentMax
```


Loop Invariants: (4)

- Arguing its Correctness:
  - ◆Use a simple argument
    - ⋆***currentMax*** starts out being first element in A
    - ⋆Beginning of ith iteration, ***currentMax*** = max. of first i elements of A
    - ⋆Since currentMax is compared to A[i] in iteration i, if this claim is true before this iteration it will be true after it for i+1
    - ⋆After n-1 iterations, ***currentMax*** = max element

Loop Invariants: (5)

Algorithm arrayFind(x, A)

Input: element x & n-element array A
Output: index i s.t. x = A[i] or –1 if no element in A equals x

i = 0
while i < n **do**
if x == A[i] then
  return i
else
  i = i + 1
return -1

Loop Invariants: (7)

- Correctness (Book Method):
- Define series of statements S_i:
- ***Claim***: At the beginning of each iteration i:

S_i: x is not equal to any of the first i elements of A

- ◆True at beginning of first iteration since no elements among the first 0 in A
- ◆In iteration i, compare x to A[i] & return i if they are equal

Loop Invariants: (8)

- ★When x = A[i], returning is correct.
- ◆If x is not equal to A[i] then we found one more element not equal to x
  - ★increment index i.
    - S_i is true at start of next iteration.
- ◆If loop terminates without returning index, then it must be true i == n.
  - ★S_k is true – No elements in A equal to x.

Bubble Sort Algorithm (1)

- Bubble-Sort Algorithm
  - ◆Sorts a sequence of elements in a sequence in non-decreasing order.
  - ◆Performs a series of passes over the sequence
  - ◆In each pass, elements are scanned in increasing rank, from rank 0 to the end of the sequence.

Bubble Sort Algorithm (2)

- At each position in each pass, an element is compared with its neighbour.
- If in wrong order, elements are swapped.
- Total of n passes are performed.
- In first pass, when largest element is swapped, it will be swapped until it reaches the end of the sequence.
- In the second pass, the seconds largest element is found etc.
- Running Time: $O(n^2)$

Bubble Sort Algorithm (3)

Algorithm Bubblesort(*sequence*):

Input: sequence of integers *sequence*

Postcondition: *sequence* is sorted & contains the same integers as the original sequence

length = length of *sequence*

for *i* = 0 **to** *length* **-** 1 **do**
  for *j* = 0 **to** *length* **-** *i* **-** 2 **do**
    if *j*th element of *sequence* >
      (*j*+1)th element of *sequence*
    then
      swap *j*th and (*j*+1)th element of *sequence*

Bubble Sort Algorithm (4)

- Loop Invariant – Outer Loop:
  - Last i elements of *sequence* are sorted and are all greater or equal to the other elements of the sequence.
- Loop Invariant – Inner Loop:
  - Same as outer loop and the jth element of *sequence* is greater or equal to the first j elements of s*equence*.

Bubble Sort Algorithm (5)

- Running Time Analysis:
  - Assume access to and swap of elements takes O(1) time.
  - Running time of ith pass:

    O(sum[n-i+1])
  - Can re-write it as:

O(n + (n+1) + … + 2 + 1)

O(sum(i)) i = 1…n

By proposition 3.4: sum(i) = [n(n+1)] / 2

Mathematical Induction: (1)

- What is a formula for the sum of the first *n* positive integers?:
  - Sums for n = 1,2,3,4,5 are:

    1= 1

    1+3 = 4

    1+3+5 = 9

    1+3+5+7 = 16

    1+3+5+7+9 = 25
  - Appears to be n^2.
    - How do we prove it?

Mathematical Induction: (2)

- Mathematical Induction is used to prove "statements" such as this.
  - Used extensively to prove results about about a large variety of discrete objects:
    - Algorithm complexity.
    - Program correctness.
    - Theorems about graphs, trees
    - Wide range of equalities and inequalities.

Mathematical Induction: (3)

- Can be used only to prove results obtained some some other way:
  - Not a tool for discovering formulas or theorems!
- Many theorems state P(n) is true for all positive integers n
  - Mathematical induction is used to prove assertions (propositions) of this kind.

Mathematical Induction: (4)

- Used top prove statements of the form $\forall n P(n)$, for all positive integers..
- A proof by mathematical induction that P(n) is true for all positive integers consists of two steps
  1. Basis Step: show P(1) (or n = some other finite value) is true.
  2. Inductive Step: Show $P(n) \rightarrow P(n+1)$ is true for every positive integer n.

Mathematical Induction: (5)

- P(n) is called the ***inductive hypothesis.*** When both steps are done, then we have shown $\forall n P(n)$.

$$[P(1) \wedge \forall n(P(n) \rightarrow P(n+1)] \rightarrow \forall n P(n)$$

  - To prove the inductive step for every n, we need to show P(n) cannot be false when P(n) is true.
    - Assume P(n) is true & show that under this assumption, P(n+1) must be true.

Mathematical Induction: (6)

- **Remark**: It is not assumed P(n) is true for all positive integers! Only shown that if it is assumed P(n) is true then P(n+1) is also true.
- When using induction, we show that P(1) is true. Then since P(1) implies P(2), P(2) must be true. Then P(3) is true because P(2) implies P(3). Continuing along these lines, P(k) is true for any positive integer k.

Mathematical Induction: (7)

- **Useful Illustration**:
  - Consider a line of people, person 1, person 2 etc. A secret is told to the first person and each person tells the secret to the next person in line.
  - Let P(n) be the statement that person n knows the secret.
    - P(1) is true since it was told to first person.
    - P(2) is true since person 1 tells person 2 and so on…

Mathematical Induction: (9)

- **Another Illustration:**
  - Infinite row of dominos, labeled 1,2,3,….,n & each domino is standing up.
  - Let P(n) be the statement that domino n is knocked over.
  - If the first domino is knocked over, P(1) is true.
  - If whenever first domino is knocked over – P(1) is true, it knocks the (n+1)th domino over – $P(n) \rightarrow P(n+1)$ is true, then all dominos are knocked over!