CPSC 331
Tutorial2

Department of Computer Science
University of Calgary

Search Problem

- Consider the "**Search" problem** that was introduced as an example at the beginning of the lecture notes
- This problem was specified by a pair of preconditions and postconditions

Search Problem

Precondition P_1: Inputs include

- n: a **positive integer**
- A: **an integer array** of length n, with entries A[0], A[1], …, A[n−1]

key: An integer found in the array (ie, such that A[i] = key for at least one integer i between 0 and n−1)

Postcondition Q_1:

Output is the integer i such that $0 \le i < n$, $A[j] \neq key$ for every integer j such that $0 \le j < i$, and such that $A[i] = key$

Inputs (and other variables) have not changed

This pair of precondition and postcondition describe what should happen for a "successful search."

Search Problem

***Precondition* P_2:** Inputs include
n: a positive integer
A: an integer array of length n, with entries A[0], A[1], …, A[n−1]
key: An integer not found in the array (ie, such that A[i] ≠ key for every integer i between 0 and n−1)

***Postcondition* Q_2:**
A notFoundException is thrown
Inputs (and other variables) have not changed

This pair of precondition and postcondition describe what should happen for an "unsuccessful search."

Search Problem

One, very widely used, algorithm for this problem is a "linear search:" compare the given key to A[i] for increasing i — that is, for i = 0, 1, 2,…
— until you *either* discover an integer i such that 0 ≤ i < n and A[i] = key

(so that **i** is the value that should be returned — because you checked values in increasing order and this is the first such value that you discovered), or until you have checked all the array entries (so you know that key is not stored in the array, and an exception should be thrown).

Questions

a)

Write code (using **pseudocode** as used in the lecture notes, but that also includes statements to return values and to throw exceptions) for the linear search algorithm that has been described above.

Solution part a

```
int LinearSearch(T key)
  i = 0
  while (i < n) and (A[i] ≠ key) do
    i = i + 1
  end while
  if i < n then
    return i
  else
    throw KeyNotFoundException
  end if
```

Questions

b) Write a loop invariant for the loop in the algorithm.

Recall from the notes that a loop invariant is an assertion that is true ***immediately after*** **the loop termination test** and ***immediately before*** **any execution of the loop body.**

Write down all the information that you know at the beginning of the first, second, and third executions of the loop body. See whether there is a pattern that you can identify that will help you to write down what is know after the k^{th} execution of the loop body, for larger k.

Solution part b

Loop invariant:
the following properties are satisfied at the beginning of each execution of the loop body:

- i is an integer such that $0 \leq i < n$
- $A[j] \neq key$ for $0 \leq j \leq i$
- A and key have not been changed

Explanation part b

- The first part is to give upper and lower bounds on the loop index *i*. Before the first execution of the loop body we have that $i = 0$, and if we have $i \geq n$ in the termination test the loop terminates. Thus, $0 \leq i < n$.

Explanation part b

- Finally, we need to provide one or more logical statements succinctly describing the effect of the loop on the loop's variables. The purpose of the example, searching for *key* in the array, comes from the post-condition provided.
- The progress made towards this goal after an iteration of the loop comes from the fact that, before an iteration of the loop, we know that *key* is not equal to any of the array elements with index $j \leq i$.
- Putting all this together, we have that the properties defined above represent a loop invariant for **LinearSearch**.

Questions

- c)

Once you believe that you have discovered the assertions that are needed, try to complete **a proof of "partial correctness.**" You may need to cycle back and forth, once or twice, between this step and the previous one, because your assertions are not strong (or complete) enough for a proof to be written down.

Solution part c

- **The first step** is to prove that this loop invariant is correct, using mathematical induction on number of executions of the loop body (*i*) as follows:

  • Prove the base case, i.e. that the loop invariant holds before the first execution of the loop body when $i = 0$.

  • Assume (inductive hypothesis) that the loop body is executed at least $i \geq 0$ times and that the loop invariant is satisfied at the beginning of the *i*th execution.

  • Show that if there is a $i + 1$st execution of the loop body, then the loop invariant holds immediately before that execution.

The second step is to prove that when the loop terminates, the truth of the loop invariant implies the post conditions.

- a complete proof of partial correctness is given in the solution pdf

Questions

- **d)**

  try to identify a “loop variant” for the loop in your program and use this to prove termination as well.

Prove that this loop terminates by giving a loop variant for it

Solution to part d

- We claim that $f(n, i) = n - i$ is a loop variant for this loop. To prove this, we note that $f(n, i)$ is an integer-valued function (because n and i are both integers), and show that $f(n, i)$ satisfies the remaining two properties of a loop variant:

- $f(n, i) = n - i$ decreases after every iteration of the loop because i increases and n remains constant.

- $f(n, i) \leq 0$ when $i \geq n$, and the for-loop terminates when $i = n$.

Explanation to part d

- To find a correct loop variant, we need to construct a function involving the loop's variables that decreases after each iteration of the loop body and implies the termination of the loop when ≤ 0.

- In this case, the loop variant will be a function of the length of the array *n* and the loop index *i*, as these are the only variables involved in deciding the loop's termination.

Explanation to part d

- The second condition ($key = A[i]$) could also cause the loop to terminate but always earlier than the first condition, so it does not need to be included in the loop variant **(think of the loop variant as a "worst-case" termination condition)**.

- The loop terminates when $i = n$, so the function must be ≤ 0 whenever $i \geq n$ and > 0 whenever $i < n$ (note that i increases after each execution of the loop body).
- Putting these together, we see that $f(n, i) = n - i$ satisfies the requirements.