COMP 1002

Logic for Computer Scientists

Lecture 31
Analysis of algorithms

• Putting it all together:
  – Using **logic** to describe what an algorithm is doing
  – and **induction** to show that it does that correctly

  – Using **recurrence** relations to see how long it takes in the worst case.
    • With **O-notation** to talk about the time.

  – and **probabilities/expectation** to try to see how long it might take on average.
Example: search in an array

• Given:
  – an array $A$ containing $n$ elements,
  – and a specific item $x$

• Goal: find the index of $x$ in $A$, if $x$ is in $A$.
  – Which box contains $\circ$? Box 4.
Example: search in an array

- **Given:**
  - an array $A$ containing $n$ elements,
  - and a specific item $x$

- **Goal:** find the index of $x$ in $A$, if $x$ is in $A$.
  - Which box contains $\bullet$? Box 4.
Example: search in an array

- **Precondition**: what should be true before a piece of code (or the whole algorithm) starts
  - E.g.: A is an array of numbers and A is not empty and x is a number.

- **Postcondition**: what should be true after a program (piece of code) finished.
  - E.g. If the program returned value k, then A[k]=x
    • or k=-1, if x is not in A.
Example: search in an array

- **Precondition**: A is an array containing x

- **Postcondition**: Returned k such that $A[k] = x$
Example: search in an array

- **Precondition:** A is an array containing x

```plaintext
Algorithm `arraySearch(A, x)`
Input array A of n integers, number x
Output k such that A[k]=x

i = 0
out = -1

while out < 0 do
    if A[i] = x then
        out = i
        i = i + 1

return out
```

- **Postcondition:** Returned k such that A[k]=x
arraySearch algorithm

Algorithm arraySearch(A, x)
Input array A of n integers, number x
Output k such that A[k]=x

∃i ∈ {0 ... n−1} A[i] = x

i = 0
out = -1

∃i ∈ {0 ... n−1} A[i] = x ∧ i = 0 ∧ out = -1

while out < 0 do
    if A[i] = x then
        out = i
        i = i+1

A[out] = x

return out

Program returned k such that A[k]=x

• A = [5, 10, 8, 7]
• x = 8
• out = 2
Loop invariant

- **Loop invariant**: a condition that is true on each iteration of the loop
  - Implied by loop precondition
  - Implies the loop postcondition
  - Implies next loop iteration is correct

- \( I(k): i = k \land ((\text{out} = i \land A[\text{out}] = x) \lor (\exists j > i \ A[j] = x)) \)

- Guard condition: condition in the while loop
  - \( G = \text{“out < 0”} \)

- Loop is correct when:
  - precondition \( \rightarrow I(0) \)
  - for all \( k \), \( G \land I(k) \rightarrow I(k + 1) \)
  - If \( k_0 \) is the smallest number such that \( \neg G \),
    then \( \neg G \land I(k_0) \rightarrow \text{postcondition} \)

- **Termination**: proof that \( \exists k_0 \) such that after \( k_0 \) iterations \( G \) becomes false

\[
\begin{align*}
\exists i \in \{0 \ldots n - 1\} \ A[i] = x & \land \\
& \land i = 0 \land \text{out} = -1 \\
\text{while } \text{out} < 0 \text{ do } \quad & \\
\quad & \text{if } A[i] = x \text{ then } \\
\quad & \quad \text{out} = i \\
\quad & \quad i = i + 1 \\
A[\text{out}] = x &
\end{align*}
\]
Proving the loop invariant

- By induction on i:
  - Base case: I(0)
    - \( \exists i \in \{0 \ldots n - 1\} \ A[i] = x \land i = 0 \land \land out = -1 \)
      Implies I(0)
    - \( i = 0 \land ((out = 0 \land A[out] = x) \lor (\exists j > i \ A[j] = x)) \)
  - Assume I(k): \( i = k \land ((out = i \land A[out] = x) \lor (\exists j > i \ A[j] = x)) \)
  - Show: if \( G \), then I(k+1): \( i = k + 1 \land ((out = i \land A[out] = x) \lor (\exists j > i \ A[j] = x)) \)
    - i=k+1 because of “i=i+1” statement
    - If A[i]=x, then \((out = i \land A[out] = x)\) holds
    - Otherwise, \((\exists j > i \ A[j] = x)\) holds.
  - Otherwise, if \( \neg G \), postcondition holds:
    - in this case, \((out = i \land A[out] = x)\) should have been true in I(k), for i=k.
    - So A[out]=x

\( \exists i \in \{0 \ldots n - 1\} \ A[i] = x \land \land i = 0 \land out = -1 \)

while \( out < 0 \) do
  if \( A[i] = x \) then
    \( out = i \)
    \( i = i+1 \)
\( A[out] = x \)
Correctness of recursive programs

Algorithm \textit{arraySearch}(A, x)
\textbf{Input} array A of \textit{n} integers, number x
\textbf{Output} \textit{k} such that \text{A}[\textit{k}]=x, -1 if no such \textit{k}

\textbf{if} \text{A}[0] = \textit{x} \textbf{then}
\hspace{1cm} \text{return} 0
\textbf{else if} \textit{n} > 1 \textbf{then}
\hspace{1cm} \text{first} = \textit{arraySearch}(A[0..\frac{n}{2} - 1], x)
\hspace{1cm} \text{second} = \textit{arraySearch}(A[\frac{n}{2}, n - 1], x)
\hspace{1cm} \textbf{if} \text{second} > 0 \textbf{then}
\hspace{2cm} \text{return} \text{second} + \frac{n}{2}
\hspace{1cm} \textbf{else}
\hspace{2cm} \text{return} \text{first}
\textbf{else}
\hspace{1cm} \text{return} -1

Use strong induction!
Assume both calls return correct value
Show that the program returns correct value
Running time: worst case

• **Precondition:** A is an array containing x
  
  – **Therefore, in the worst scenario need to check all n boxes A[i]**

  – **Running time: O(n)**

---

**Algorithm arraySearch(A, x)**

**Input** array A of n integers, number x

**Output** k such that A[k]=x

```
i = 0
out = -1
while out < 0 do
    if A[i] = x then
        out = i
        i = i + 1
    i = i + 1
return out
```
Running time: average case

- **What is the expected number of steps before x is found?**
  - Depends on the probability of x being in each cell.
  - Or whether there is only one x, or can be many

```plaintext
Algorithm arraySearch(A, x)
Input array A of n integers, number x
Output k such that A[k] = x

i = 0
out = -1
while out < 0 do
    if A[i] = x then
        out = i
        i = i + 1
return out
```
Bernoulli trials and repeated experiments

• Suppose an experiment has two outcomes, 1 and 0 (success/failure), with \( \Pr(1) = p \).
  
  – Such experiment is called a **Bernoulli trial**.

• What happens if the experiment is repeated multiple times (independently from each other?)

  – A sample space after carrying out \( n \) Bernoulli trials is a set of all possible \( n \)-tuples of elements in \{0,1\} (or \{success, fail\}).
  
  – Number of \( n \)-tuples with \( k \) 1s is \( \binom{n}{k} \)
  
  – Probability of getting 1 in any given trial is \( p \), of getting 0 is \( 1-p \).
  
  – Probability of getting exactly \( k \) 1s (successes) out of \( n \) trials is \( \binom{n}{k} p^k (1 - p)^{n-k} \)
    
    • Called binomial distribution
  
  – Probability of getting the first success on exactly the \( k^{th} \) trial is \( p(1 - p)^{k-1} \)

• How many trials do we need, on average, to get a success?
Running time: average case

- Suppose probability of $x$ being in any cell is $p$
  - Can have many $x$ in $A$
- Then probability of finding $x$ in $k$ steps is $p(1 - p)^{k-1}$
- Let random variable $X$ denote the number of loop iterations till $x$ is found
- $E(X) = \Sigma_{i \in \mathbb{N}} i \cdot \Pr(X = i) = \frac{1}{p}$
- Expect to find $x$ in $O(1/p)$ steps

Algorithm $\text{arraySearch}(A, x)$
Input array $A$ of $n$ integers, number $x$
Output $k$ such that $A[k] = x$

$i = 0$
$out = -1$
while $out < 0$ do
  if $A[i] = x$ then
    $out = i$
    $i = i + 1$
return $out$
Running time: average case

- Suppose there is just one \( x \) in \( A \)
- Probability of finding \( x \) in each step is \( \frac{1}{n} \)
- Let random variable \( X \) denote the number of loop iterations till \( x \) is found
- \( E(X) = \sum_{i=n} \frac{1}{n} i \cdot \Pr(X = i) = \frac{1}{n} \sum_{i=1}^{n} i = \frac{(n + 1)}{2} \)
- Expect to find \( x \) in the middle of \( A \)
- Running time \( O(n) \)

Algorithm arraySearch(\( A, x \))
Input array \( A \) of \( n \) integers, number \( x \)
Output \( k \) such that \( A[k] = x \)

\[
\begin{align*}
&i = 0 \\
&\text{out} = -1 \\
&\text{while } \text{out} < 0 \text{ do} \\
&\quad \text{if } A[i] = x \text{ then} \\
&\quad \quad \text{out} = i \\
&\quad \quad i = i+1 \\
&\text{return out}
\end{align*}
\]
More to come...

- You will see a lot of algorithm analysis and use of the concepts we developed in COMP 2002 and beyond.
  - Logic, sets, relations and graphs for specification, modeling problems and describing what you are doing.
  - Logic, induction and models of computation for proving program correctness and analysis of problem complexity.
  - Recursive definitions of algorithms, counting and probability for algorithm performance and problem solving.
- With the million dollar problem rearing its head every now and then

Have fun!