The Fundamentals: Algorithms, the Integers, Matrices

Algorithms

March 16, 2016
Outline

1. Algorithms
Definition

An algorithm is a finite series of precise instructions for performing a computation or solving a problem.
Definition

An algorithm is a finite series of precise instructions for performing a computation or solving a problem.

Example (Algorithm: Maximum element in finite sequence)

```plaintext
procedure max(a1, a2, ..., an: integer)
    max := a1
    for i := 2 to n
        if (ai > max) then max = ai
        { max largest on a1 - ai inclusive }
        { max largest on a1 - an inclusive }
```
Properties of Algorithms

**Input**  An algorithm has input values from a specified set
Properties of Algorithms

Input  An algorithm has input values from a specified set
Output From each set of input values, an algorithm produces output values, the solution, from a specified set
Properties of Algorithms

**Input**  An algorithm has input values from a specified set

**Output**  From each set of input values, an algorithm produces output values, the solution, from a specified set

**Definiteness**  The steps of the algorithm are defined precisely
Properties of Algorithms

Input  An algorithm has input values from a specified set
Output From each set of input values, an algorithm produces output values, the solution, from a specified set
Definiteness The steps of the algorithm are defined precisely
Correctness The algorithm should produce the correct output for each input value
Properties of Algorithms

**Input**  An algorithm has input values from a specified set

**Output**  From each set of input values, an algorithm produces output values, the solution, from a specified set

**Definiteness**  The steps of the algorithm are defined precisely

**Correctness**  The algorithm should produce the *correct* output for each input value

**Finiteness**  The solution must be produced in a finite number of steps
Properties of Algorithms

Input: An algorithm has input values from a specified set.

Output: From each set of input values, an algorithm produces output values, the solution, from a specified set.

Definiteness: The steps of the algorithm are defined precisely.

Correctness: The algorithm should produce the correct output for each input value.

Finiteness: The solution must be produced in a finite number of steps.

Effectiveness: It must be possible to perform each step in the algorithm precisely and in a finite amount of time.
Properties of Algorithms

- **Input**: An algorithm has input values from a specified set.
- **Output**: From each set of input values, an algorithm produces output values, the solution, from a specified set.
- **Definiteness**: The steps of the algorithm are defined precisely.
- **Correctness**: The algorithm should produce the *correct* output for each input value.
- **Finiteness**: The solution must be produced in a finite number of steps.
- **Effectiveness**: It must be possible to perform each step in the algorithm precisely and in a finite amount of time.
- **Generality**: The procedure should apply to all problems of the given form, not just a single input value.
Example (Algorithm: Linear Search of Finite Sequence)

```plaintext
procedure linear(x: integer, a1, ..., an: distinct integer)
    i := 1
    while (i ≤ n ∧ x ≠ a_i) do
        i := i + 1
        { no x yet and more to search }
    if i ≤ n then location := i
    else location := 0
    { location is 0 if not found; index of matching value otherwise }
```
Algorithm

Binary Search

Example (Algorithm: Binary Search of Sorted Finite Sequence)

```
procedure binary(x: integer,
                 a_1, a_2, ..., a_n: increasing integer)

low := 1
high := n
{ search interval [low, high] }

while (low < high) do
    mid := ⌊(low + high)/2⌋
    if (x > a_mid) then low := mid + 1
    else high := mid

{ low = high (could not be <; why not?) and,
  if x is found, x = a_low }

if (x = a_low) then location := low
else location := 0
{ location is 0 if not found; index of matching value otherwise }
```
Algorithms

Binary Search

Theorem

binary always terminates

Proof.

The only way it can not terminate is to be stuck in the while loop. In the loop, \( low \leq mid \leq high \)

\( high - low \) is the size of the range to be searched.
Algorithms

Binary Search

Theorem

*binary always terminates*

Proof.

The only way it can **not** terminate is to be stuck in the **while** loop. In the loop, \( low \leq mid \leq high \)

\( high - low \) is the size of the range to be searched.

\( high' - low' \), the value after the loop, is smaller:
Theorem

*binary always terminates*

Proof.

The only way it can **not** terminate is to be stuck in the **while** loop.

In the loop, \( \text{low} \leq \text{mid} \leq \text{high} \)

\( \text{high} - \text{low} \) is the size of the range to be searched.

\( \text{high}' - \text{low}' \), the value after the loop, is smaller:

If \( x > a_{\text{mid}} \), \( \text{low}' > \text{low} \)

Otherwise, \( \text{high}' < \text{high} \) because

\( \text{low} \neq \text{high} \); \( [ \ ] \) of average less than max

Range to be searched is smaller on each iteration of loop; range initially finite so value must cross 0 terminating the while loop.
Example (Problem)

Given: Amount of change to make, \( n \in \mathbb{Z}^+ \)
Sequence of coins: \( c_1 > c_2 > \ldots > c_r \)

Describe an algorithm to solve this problem.
Example (Algorithm: Greedy Algorithm for Making Change)

```plaintext
procedure change(integer n,
                  decreasing integer c_1, c_2, ..., c_r)

change = {}
for i = 1 to r
    while n ≥ c_i
        add c_i to change
        n = n - c_i
    // c_i > n - no more of this coin fit
// Assuming all values can be built of coins, change is set
// of coins making change n (since n should be 0)
```
Lemma

If \( n \in \mathbb{Z}^+ \), then \( n \) cents in change using American coins under a half dollar, using the fewest coins possible, has at most two dimes, at most one nickel, at most four pennies, and cannot have two dimes and a nickel. The amount of change excluding quarters cannot exceed 24 cents.
Theorem

The greedy algorithm produces correct change in the fewest number of coins using American coins under a half dollar.
**Halting Problem**

**Definition**

The **Halting Problem** is the problem of constructing a program, $H$, that takes two parameters: $P$, another computer program and $I$, input for $P$. $H$ should report “halts” or “loops forever” depending on whether or not $P$ halts on input $I$.

$$H(P, I) = \begin{cases} 
  "\text{halts}" & \text{if } P(I) \text{ halts} \\
  "\text{loops forever}" & \text{if } P(I) \text{ does not halt}
\end{cases}$$
Programs as Data

A program is encoded in some way (Fortran, pseudo-code, Java, bytecode). An encoding can be expressed as a sequence of symbols across some alphabet.
Any alphabet can be re-encoded using strings of bits.
Any program can be expressed as a sequence of bits.
A program, encoded as a sequence of bits, can be given as input to a program. The receiving program may or may not respect the “right” interpretation.
Programs as Data

A program is encoded in some way (Fortran, pseudo-code, Java, bytecode). An encoding can be expressed as a sequence of symbols across some alphabet.

Any alphabet can be re-encoded using strings of bits.

Any program can be expressed as a sequence of bits.

A program, encoded as a sequence of bits, can be given as input to a program. The receiving program may or may not respect the “right” interpretation.

Any program that takes a single input parameter can be passed itself (or rather, its own encoding) as its input.
Halting Problem

Definition (A)

\[ H(P, I) = \begin{cases} 
"\text{halt}" & \text{if } P(I) \text{ halts} \\
"\text{loop}" & \text{if } P(I) \text{ does not halt} 
\end{cases} \]
Halting Problem

Definition (A)

\[ H(P, I) = \begin{cases} 
\text{"halt"} & \text{if } P(I) \text{ halts} \\
\text{"loop"} & \text{if } P(I) \text{ does not halt}
\end{cases} \]

Definition

FSOC Assume \( H \) exists. Construct \( D \)

\[ D(P) = \begin{cases} 
\text{loop forever} & \text{if } H(D, P) \text{ halts} \\
\text{return} & \text{if } H(D, P) \text{ does not halt}
\end{cases} \]
Halting Problem

**Definition (A)**

\[ H(P, I) = \begin{cases} 
\text{"halt"} & \text{if } P(I) \text{ halts} \\
\text{"loop"} & \text{if } P(I) \text{ does not halt} 
\end{cases} \]

**Definition**

FSOC Assume \( H \) exists. Construct \( D \)

\[ D(P) = \begin{cases} 
\text{loop forever} & \text{if } H(D, P) \text{ halts} \\
\text{return} & \text{if } H(D, P) \text{ does not halt} 
\end{cases} \]

What does \( H(D, D) \) return?