Lecture Outline (CSCI3160-17F, 1st week)

CAI Leizhen
CSE-CUHK-HK-CHN

September 7, 2017

Keywords: Algorithm, correctness.

1. *Algorithmics*: Design and analysis of algorithms.
2. *Algorithmic problem* Π: A specification of all legal inputs I_Π, and a function $f_\Pi : I_\Pi \to O_\Pi$ that maps each input $i \in I_\Pi$ to its desired output $f_\Pi(i) \in O_\Pi$.
3. *Algorithm*: A finite sequence of basic instructions for solving an algorithmic problem (i.e., computing $f_\Pi(i)$ for each legal input i) that terminates in a finite number of steps. The underlying mathematical model for an algorithm is a Turing machine program.
4. *Three main concerns*: Correctness (need to be proved mathematically), efficiency (time and space), and simplicity (easy to understand).
5. *Correctness*: Algorithm $\mathcal{A}$ solves problem Π if for every legal input i to $\mathcal{A}$, the algorithm terminates in a finite number of steps and produces $f_\Pi(i)$ as the output.
6. Incorrect behavior of an algorithm: (a) Terminates normally but with wrong output, (b) doesn't terminate at all (infinite loop), and (c) aborts execution abnormally.
7. *Partial correctness*: For every legal input i, $\mathcal{A}$ produces $f_\Pi(i)$ as the output once $\mathcal{A}$ terminates for i. *Total correctness*: $\mathcal{A}$ is partially correct and $\mathcal{A}$ terminates for all legal inputs.
8. Consider the following algorithm to destroy directed cycles in a digraph G by reversing edges.

   while G contains a directed cycle C **do**
   choose an edge e in C and reverse the direction of e.
   end while

   Partial correctness of the algorithm is obvious as the digraph contains no directed cycle when the algorithm terminates. However, the algorithm may not terminate at all, and therefore it is not a correct algorithm for solving the problem.

9. It could be extremely difficult to determine whether an algorithm terminates for all legal inputs. For the following famous $3x + 1$ problem, it is unknown whether it halts for all positive integers x.

   while $x > 1$ **do**
   if x is even **then** $x \leftarrow x/2$
   else $x \leftarrow 3x + 1$;
   end while

10. *Fibonacci number*: $f_0 = 0$, $f_1 = 1$ and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

    We can easily write a recursive algorithm to compute the n-th Fibonacci number using the recurrence relation, and the correctness of the algorithm is obvious in this case.

    function f(n): return the value of f_n.

    case $n = 0$: **return** 0;
    $n = 1$: **return** 1;
    otherwise return $f(n-1) + f(n-2)$.

    We can also easily derive a correct algorithm using an array to store values of f_i's. However, the correctness of the following algorithm is less obvious and need to be proved.

    Fibonacci(n): compute Fibonacci number f_n.
    $i \leftarrow 1$; $j \leftarrow 0$;
    for $k = 1$ **to** n **do**
    $j \leftarrow i + j$;
    $i \leftarrow j - i$;
    end for;
    print j.

    Correctness: The algorithm clearly terminates after n iterations, and we need to prove that j contains the value of f_n when the algorithm terminates.

    Let i_k and j_k be the values of i and j, respectively, at the end of the k-th iteration. We wish to show that $j_n = f_n$. Note that the algorithm works correctly for $k = 0$ as it returns 0 as output.

    Idea: Find an invariant related to the output and use induction to prove that the invariant holds for every iteration.

    Claim (invariant): For every $k \geq 1$, $j_k = f_k$ and $i_k = f_{k-1}$.

    The claim implies that $j_n = f_n$, and we can use induction on k to prove the claim.

    For $k = 1$, we have $j_1 = 1 = f_1$ and $i_1 = 0 = f_0$, and the claim is true. Assume that the claim is true after the $(k-1)$-th iteration and consider the situation after the k-th iteration.

By the algorithm, we have

$$j_k = i_{k-1} + j_{k-1} = f_{k-2} + f_{k-1} = f_k,$$

and

$$i_k = j_k - i_{k-1} = f_k - f_{k-2} = f_{k-1}.$$

Therefore the claim also holds after the k-th iteration.

11. *Multiplication*: Compute the product of two non-negative integers x and y.

```
p ← 0;
while x > 0 do
        if x is odd then p ← p + y;
        x ← ⌊x/2⌋;
        y ← 2y;
end while;
print p.
```

Let x_i, y_i and p_i, respectively, be values of x, y and p after the i-th iteration.

Claim (invariant): For all $i \geq 0$, $xy = p_i + x_i y_i$.

When the algorithm terminates after the i^*-th iteration, $x_{i^*} = 0$ and therefore $p_{i^*} = xy$. We can use induction on i to establish the claim.

The claim is clearly true for $i = 0$ as $p_0 = 0$, $x_0 = x$ and $y_0 = y$. Assume that the claim is true for $i - 1$ and consider the situation after the i-th iteration.

Case 1. x_{i-1} is even. By the algorithm, we have

$$\begin{cases} p_i = p_{i-1} \\ x_i = x_{i-1}/2 \\ y_i = 2y_{i-1} \end{cases}$$

Since $xy = p_{i-1} + x_{i-1}y_{i-1}$ (induction hypothesis), we have $xy = p_i + x_i y_i$ and the claim holds for i.

Case 2. x_{i-1} is odd. In this case, the algorithm yields

$$\begin{cases} p_i = p_{i-1} + y_{i-1} \\ x_i = (x_{i-1} - 1)/2 \\ y_i = 2y_{i-1}. \end{cases}$$

Using the induction hypothesis $xy = p_{i-1} + x_{i-1}y_{i-1}$, we obtain

$$xy = p_i + (x_{i-1} - 1)y_{i-1} = p_i + (2x_i)(y_i/2) = p_i + x_i y_i,$$

and the claim also holds for this case.

12. There are unlimited possibilities of algorithms for solving a problem, and it often calls for creativity in designing new, fast, and clever algorithms. We have discussed 6 different algorithms for computing the product of two positive integers.