CS101 Introduction to computing

Problem Solving (Computing)

A. Sahu and S. V .Rao
Dept of Comp. Sc. & Engg.
Indian Institute of Technology Guwahati

Outline

- Problem Solving : Process involves
  - Definition, Analysis, Solution Approaches, Correctness, Programming, Testing
- Loop invariant and loop termination
- Many Problem Solving Examples
  - 7 Problems **(Solution Method not given)**
  - 3 problems **(Solution Method given)**

Reference : R G Dromey, "***How to solve it by Computer***", Pearson Education India, 2009

Analysis of Solution Approaches

- Correctness and Efficiency (C & E)
  - Algorithm/Approaches are analyzed for C & E
  - C & E are precise and detailed enough
- **Correctness analysis**
  - To ensure the **algorithm solves** the given problem
  - Involves a mathematical proof that algorithm **satisfies the specification**; termination proofs
- **Efficiency analysis** : To determine
  - **amount of time** or number of operations
  - **amount of memory** required for executing the algorithm

Algorithm

- The algorithm is part of the blueprint or plan for the computer program, an algorithm is:

  "An effective procedure for solving a class of problems in a finite number of steps."

- Every algorithm should have the following 5 characteristic features:
  - **Definiteness:** Each step must be define precisely
  - **Effectiveness :** its **operations** must be **basic enough** to be able to be **done exactly** and in **finite length of time**
  - **Termination:** must terminate after a finite number of steps
  - **Input** and **Output**

Problem Solving Strategies

- **New problems** may require **newer strategies**
- Problem solving skills can be developed **only with experience**
- Main emphasis of the course
  - To expose you to **various problem solving strategies by way of examples**
- The programming languages is for concreteness and execution of your ideas

Problem Solving Strategies

- Given a Problem P
- You may come up many Approaches/ strategies : App1, App2, App3, App4, Appm
- If we are not able prove the correctness by loop termination and loop invariant of some approaches
  - We cannot call that Approaches as Algorithm
- Suppose App2 and App3: We are not able prove the correctness for them , then App2 and App3 are not algorithms by definition
  - Algorithms for P: App1, ~~App2~~, ~~App3~~, App4, Appm

Mathematical Argument

- **Prove the correctness** using **mathematical arguments**
- Proof of Correctness involves two-Step argument
  - **Loop Invariants**
  - **Loop Termination**

Loop invariants

- A **condition** (**logical expression**) involving program variables
  - It holds **initially**
  - If it holds **before start** of iteration, it holds at **the end**;
  - The condition remains invariant under iteration

Please don’t get confuse : Loop invariants

- Please do not get confuse with loop invariant in coding

```
for(i=0;i<10;i++){
        K=20;  //K is loop invariant
        printf(“%d\n”,i*K);
}
```

  - Variable don’t get change over iterations
  - *Used for code optimization*
- **Loop invariant used for proving correctness**
  - Properties don’t get change over iterations
- **Both are different things**

Loop Termination

- **Non termination** is an important **source of incorrectness**.
- **Correctness proof** includes **termination proof**
- Bound on iteration
  - An integer valued expression called **bound function** that **reduces in each iteration**
  - When the bound function reaches 0, loop terminates
- For our example, the bound function is:

  length of the input list yet to be processed

Efficiency Analysis

- How many number of **operations**?
  - In each iteration of the loop, constant number of comparisons
- Can we improve this?
  - If the number is less than 50, there is no need for comparing it with 80.
- Rewrite the algorithm

Problem Solving Example

- Set A **(Solution Method not given)**
  1. Nth Power of X
  2. Square root of a number
  3. Factorial of N
  4. Reverse a number
  5. Finding value of unknown by question answers
  6. Value of Nth Fibonacci Number
  7. GCD to two numbers

Problem Solving Example

- Set B **(Solution Method given)**
  1. Finding values Sin(x) using series sum
  2. Value of PI
  3. Finding root of a function Bisection Methods

Problem 1

The n^{th} power of X

The n^{th} power of X

- **Problem:** Given some integer x. write a program that computes the nth power x, where n is positive integer considerably greater than 1.
- Evaluating expression $p=x^n$

```
Prod=1;
for  (i=1; i<=n; i++){
    Prod= Prod * x;
}
```

- Naïve or straight-forward approach

  How many multiplication: n

  Require n steps

Assumption : all basic operations on integers take constant time

The n^{th} power of X

- **Is there any better approach?**
- From basic algebra
  - if n is even == > $X^n = X^{n/2} . X^{n/2}$
  - If n is odd and n=2m+1 ==> $X^n = X^{2m+1} = x^m . x^m . x$
- From this above fact, can we calculate X^n in fewer steps
- Approach
  - Binary representation of n,
  - X^{23} Example $23 = (10111)2 = 1x2^4 + 0x2^3 + 1x2^2 + 1x2^1 + 1x2^0 = 16+0+4+2+1$
  - Start from right to left
    - $1x2^4 + 0x2^3 + 1x2^2 + 1x2^1 + 1x2^0$

$\longleftarrow$

Approach/Algorithm

1. Initialize the power sequence and product variable *(let initial value of n is n0=n)*

Product=1; ProdSequence=x;

2. Do while n > 0 repeat

   2.1 if the next most binary digit of n is one then **Product = Product * ProdSecuence;**

   2.2 n = n /2;

   2.3 ProdSecuence *= ProdSecuence;

//**Invariant Product*ProdSecuence^{n}=x^n0 , n>=0**

Assumption : all basic operations on integers take constant time

Approach

- Binary representation of n,
- X^{23} Example
  $23=(10111)2=1x2^4+0x2^3+1x2^2+1x2^1+1x2^0 = 16+0+4+2+1$
- Start from right to left

$$1x2^4+0x2^3+1x2^2+1x2^1+1x2^0$$

- Approach
  - Successive generation of x, x^2, x^4, x^8, x^{16}, ...
  - Inclusion of the current power member into accumulated product when the corresponding binary digit is 1

Approach

Odd number or Right Most Bit

Before Loop

1	0	1	1	1	
$P=X^7.X^{16}=X^{23}$	$P=X^7$	$P=X^3.X^4$ $=X^7$	$P=X.X^2$ $=X^3$	$P=P.PS$ $=X$	**$P=1$**
X^{32}	X^{16}	X^8	X^4	X^2	**$PS=X$**
$N=0$	$N=1$	$N=2$	$N=5$	$N=11$	**$N=23$**
$X^{23}.(X^{32})^0$ $=X^{23}$	$X^7.(X^{16})^1$ $=X^{23}$	$X^7.(X^8)^2$ $=X^{23}$	$X^3.(X^4)^5$ $=X^{23}$	**$X.(X^2)^{11}$** $=X^{23}$	**$P*PS^n$ $=1.X^{23}$**

Loop Invariant

C –Code for X^n

```
int n, x, Prod, ProdSeq;
// Put code for Input n, x
Prod=1; ProdSeq=x;
while(n > 0)     {
  if ((n%2)==1){
     Prod=Prod*ProdSeq;
     }
  n=n/2;
  ProdSeq = ProdSeq* ProdSeq;
}
//Put code to Display Prod as Xⁿ
```

Assumption : all basic operations on integers take constant time

Problem 2

The square root problem : sqrt(X)

One Strategy

- Given a guess ***a*** for square root of m
  - ***m/a*** falls on the opposite side
  - *(a + m/a)/2*, can be the next guess
  - **Why this guess? Make next guess closer to sqrt(m) based on current guess.**
- This gives rise to the following solution
  - start with an arbitrary guess, r_0
  - generate new guesses r_1, r_2, etc by using the averaging formula.
- When to terminate?
  - when the successive guesses **differ by a given small number**

The Approach

Input float m, e, **assume:** m>0, 0< e > 1
Output float r_1, r_2

Loop Invariant :

|(r2*r2-m) | <= |(r1*r1-m)|, |r1- r2|> e

1. $r_1 = m/2$, $r_2 = r_1$
2. **Do**

   2.1 $r_1 = r_2$

   2.2 $r2 = (r_1+m/r_1)/2$

while $(|r_1 - r_2| > e)$

C Code : Square root of m

```
float m, e, r1, r2;
// Put code for Input m, e
r1=m/2;  r2=r1;
do    {
    r1=r2;
    r2=(r1+m/r1)/2;
} while(abs(r1-r2) > e)
//Put code to Display root as r2
```

Analysis of the Approach

- Is it **correct**? Find the **loop invariant** and **bound function**
- Can the algorithm be **improved?**
- More general techniques available
  - **Numerical analysis**
- NA: Newton Raphson's for square root

$F(x) = x^2 - m = 0$

$x_{k+1} = x_k - F(x_k)/F'(x_k) = x_k - (x_k^2 - m)/2x_k$

$\mathbf{x_{k+1} = (x_k + m)/2}$

Factorial Computation

- Given a number **n**, compute the **factorial of n**
- Assume **n >= 0**
- What is factorial?
  - 0! = 1, 1! = 1, 2! = 1*2 = 2
  - 3! = 1*2*3 = 6
  - 4! = 1*2*3*4* = 24
- **n! = 1*2*...*(n-1)*n**, for **n>=1**

Assumption : all basic operations on integers take constant time

The algorithm/Approach

- **Observation:** For n>=1, n! is (n-1)! multiplied by n
- **Strategy:** Given n, compute n! by successively computing 1!, 2!, etc. till n!

Input n, **Output** Fact

1. **initialize** fact to 1 and index to 1
2. **do while** (index < = n) steps 2.1 and 2.2

   2.1 fact = fact * index

   2.2 index = index + 1

Analysis of Factorial Algorithm

- Is the solution **correct**?
- **Loop invariant**: At the beginning of each iteration,
  - **fact** holds the partial product **1 * . . . * (index-1)**
- When the loop terminates, **index = (n+1)**
  - fact then holds **(1 * ... * n)**
- Does the loop terminate?
  - There is a **bound function**: (n + 1 – index)
  - The bound function always >= 0
  - It decreases in each iteration

Reversing Digit of a Number

Problem: Reversing the Digits of an integer

Examples:

Input: 58902

Output: 20985

Input: 4300

Output: 34

Reversing Digit of a Number

Problem: Reversing the Digits of an integer

Examples:

Input: 58902 Output: 20985

$R(58902) = 2x10^4+R(5890)$

// you need to know how many digit before hand

$= 2x10^4+0x10^3+R(589)$

$= 2x10^4+0x10^3+9x10^2+R(58)$

$= 2x10^4+0x10^3+9x10^2+8x10^1+R(5)$

$= 2x10^4+0x10^3+9x10^2+8x10^1+5$

= **2**0000+**0**000+**9**00+**8**0+5

Decreasing power of 10

We can think as a polynomial $(2.x^4+0.x^3+9.x^2+8.x+5)$ evaluated at 10..

Reversing Digit of a Number

Problem: Reversing the Digits of an integer

Examples:

Input: 58902   Output: 20985

Try to use the concept of polynomial evaluation using hornor's rule

R(58902) = 2+R(5890)

= 2x10+0+R(589)

= (2x10+0)x10+9+R(58)

= ((2x10+0)x10+9)x10+8+R(5)

= (((2x10+0)x10+9)x10+8)x10+5

Increasing power of 10

Approach : Digit Reversal

Input: ***N*** is k digit number to be reversed

Output: ***RevNum*** the reversed number

1. q = N
2. ***RevNum*** = 0
3. **Do while** (q > 0) steps 3.1,3.2,3.3

    3.1 rem = q **mod** 10

    3.2 ***RevNum*** = ***RevNum*** *10 + rem

    3.3 q = q / 10

Invariant: After jth iteration q=$\{d_1\}\{d_2\}..\{d_{k-j}\}$ and RevNum=$\{d_k\}\{d_{k-1}\}...\{d_{k-j-1}\}$

C Code to reverse a number

```
int n, RevNum, Rem, q;
// Put code for Input n
q=n;
RevNum=0;
while(n != 0)    {
    Rem = n%10;
    RevNum=RevNum*10+ Rem;
    n=n/10;
}
//Put code to Display RevNum
```

Problem 5

Finding value of Unknown integer X

Unknown Number Problem: Version 1

- **Problem:** Given an unknown integer X in the range R_{min} and R_{max} ($R_{min} \leq X < R_{max}$) , We need to find the value of X by asking Boolean queries of type **a==x**, **a>x**, **a<x, a>=x and a<=x**
- **Goal:** is to minimize the number of question to find the value of X
- Is the problem definition **clear?**

Approach

- **Problem:** Given an unknown integer X in the range R_{min} and R_{max} ($R_{min} \leq X < R_{max}$), We need to find the value of X by asking Boolean queries of type **a==x**, **a>x**, **a<x, a>=x and a<=x**

- Start from R_{min} and go upto R_{max}, one by one

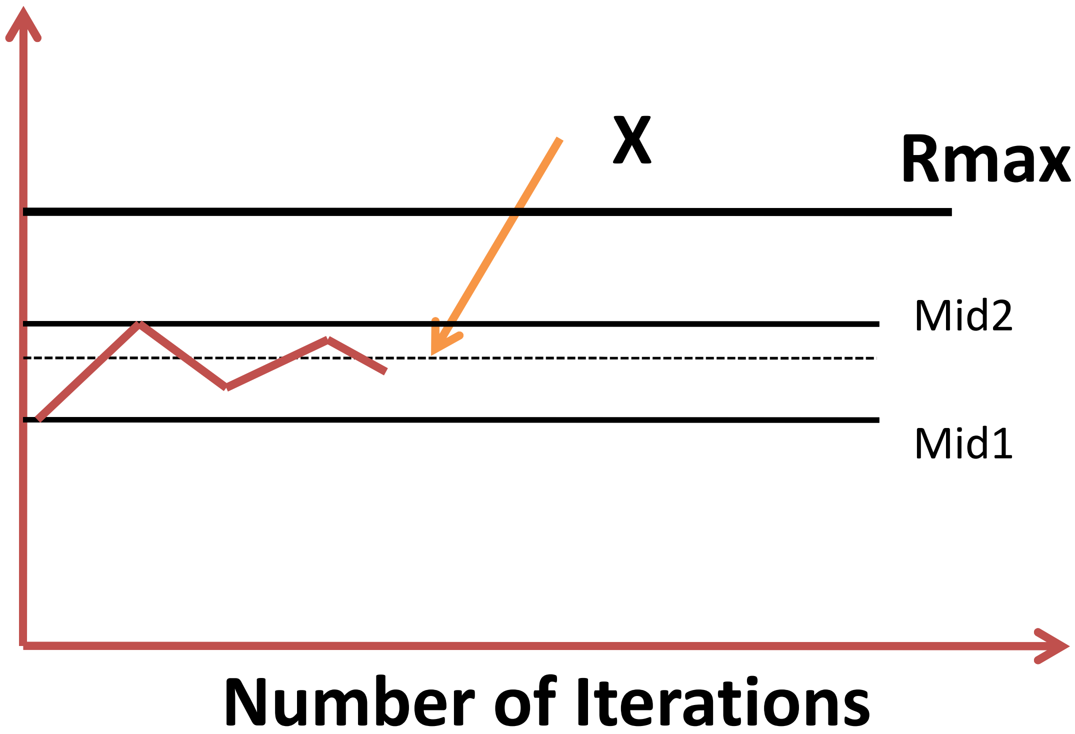
```
for(a=Rmin, a<Rmax; a++){
    if(x==a) break;
}
//Print  value of X is a
//Number of step required is X-Rmin
```

Approach-2

- **Is there any better approaches**
- Why to test one by one?
- Test at middle and set new range
  - Mid $=(R_{min}+R_{max})/2$
  - If (x==Mid) found
  - If (X>Mid) R_{min}=Mid+1 else R_{max}=Mid
- Binary Search….

Approach-2

- Test at middle and set new range
  - Mid =$(R_{min}+R_{max})/2$; If (x==Mid) found
  - If (X>Mid) R_{min}=Mid+1 else R_{max}=Mid



Rmax=255, Rmin=0, X=155

Mid1=127, Rmin=128
Mid2=191, Rmax=191
Mid3=159, Rmax=159
Mid4=143, Rmin=144
Mid5=151, Rmin=152
Mid6=155 ....Done

Approach-2

- **Is there any better approaches**
- Why to test one by one?
- Test at middle and set new range
  - Mid =(R_{min}+R_{max})/2
  - If (x==Mid) found
  - If (X>Mid) R_{min}=Mid+1 else R_{max}=Mid

```
while (Rmin<Rmax){
   mid=(Rmin+Rmax)/2;
   if(x==mid)return found;//print mid
   if (x>mid) Rmin=mid+1;
    else Rmax=mid;
}
```

Analysis: Approach-2

- Test at middle and set new range
  - Mid =(Rmin+Rmax)/2
  - If (x==Mid) found
  - If (X>Mid) Rmin=Mid+1 else Rmax=Mid
- Number of test:
  - 2 per iterations
  - Number of iteration : $\mathbf{Log_2 (R_{max}\text{-}R_{min})}$

Unknown Number Problem: Version 2

- **Problem:** Given an unknown integer X, We need to find the value of X by asking Boolean queries of type **a==x**, **a>x**, **a<x, a>=x and a<=x**
- **Goal:** is to minimize the number of question to find the value of X
- Is the problem definition **clear?**

Approach-1

- **Problem:** Given an unknown integer X, We need to find the value of X by asking Boolean queries of type **a==x**, **a>x**, **a<x, a>=x and a<=x**
- Start from 1 and go upto X, one by one

```
a=1;
while(a<X){{
     if(x==a) break;
     a=a+1;
}
//Print  value of x is a
//Number of step required is a-Rmin
```

Approach-2

- **Problem:** Given an unknown integer X, We need to find the value of X by asking Boolean queries of type a==x, a>x, a<x
- Is there any better approaches?
- Start from 1 but go at faster pace and find a range
  - Instead of **a = a+1 ,** use a = a+100

```
a=1;
while(a<X){a=a+100;}
// x will be between [a-100]<= X < a
Find Using previous method: Binary
search for X between R_min and R_max
```

Approach-2

- Start from 0 but go at faster pace and find a range : Instead of a = a+1 use a = a+M

```
a=1;
while(a<X){
    a = a + M;
}
// x will be between [a-M]<= X < a
Find Using previous method: Binary
search for X between Rmin and Rmax
```

- How good it is ?
- Number of steps: $X/M + \text{Log}_2 M$
- Can it be done better?

Approach-3

- Start from 0 but go at faster pace and find a range : Instead of a = a+1, use a = a*2

```
a=1;
while(a<X){
    a = a * 2;
}
// x will be between [a/2]<= X < a
Find Using previous method: Binary
search for X between Rmin and Rmax
```

- How good it is ?
- Number of steps **$\text{ceil}(\text{Log}_2 X) + \text{ceil}(\text{Log}_2 X)$**

  $\approx \log_2 X$