Introduction to Algorithms

Module Code: CELEN086

Lecture 2

(10/10/24)

Lecturer & Convenor: Manish Dhyani
Email: Manish.dhyani@nottingham.edu.cn

Office hours

When and where?

Day of Week	Time Slot	Venue
Wednesday	11:00 to 12:00	YANG Fujia Building-412(TB)
Thrusday,	10:00 to 11:00	PB217
Friday,	14:00 to 15:00	YANG Fujia Building-328+(TB)

No need to make an appointment.

Drop in at the venues at scheduled time with your questions.

Topics to Cover Today

1. Review of structure of Algorithm
2. Trace an Algorithm
3. Boolean variables and operators
4. Nested If else statements
5. Calling sub algorithm inside main Algorithm
6. Check digit concept using Luhn Algorithm

- Write an algorithm **Diff** that returns the difference between two numbers. For instance:
    - **Diff**(5,2) = 3
    - **Diff**(7,12) = 5
    - **Diff**(3,3) = 0

Algorithm : Diff(x, y)
requires: two numbers x and y
returns: the difference between x and y

Header of algorithm

```
1.  If x>y then
2.      return x-y
3.  else
4.      return y-x
5.  endif
```

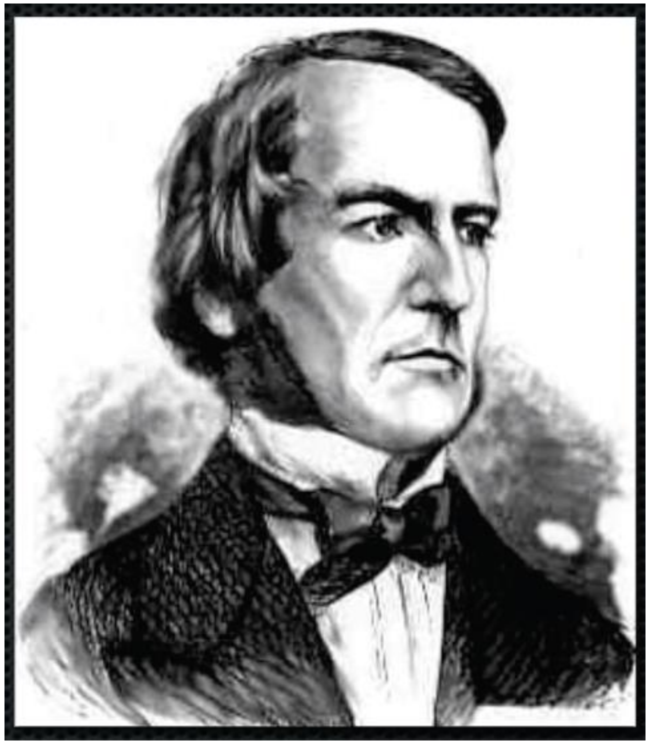
Body of algorithm

Trace algorithm Diff(x, y) When x = 7 , y =12

line 1 if (7>12) then   7>12 == false
Skip line 2
line 3 else
line 4 return 12-7 i.e. 5
line 5 endif

Data structures: Booleans

- **George Boole** (1815 - 1864)
  - **the inventor of Boolean logic**
- A Boolean value is either
  - **True** or
  - **False**
- You can inspect a Boolean using **if-then-else**.



Note : To represent True we are using symbol T or 1 and for False we are using F or 0.

Example: Simple if else

```
Algorithm NOT($b$)
Require: A boolean $b$
Return: True if $b$ is False;
        False if $b$ is True.
1: if $b$ then
2:     return False
3: else
4:     return True
5: endif
```

b	NOT(b)
True	False
False	True

Example :AND

Algorithm 3 And(b, c)

Require: Two booleans b and c

Return: True if both b and c are True; False otherwise.

```
1: if $b$ then
2:     if $c$ then
3:         return True
4:     else
5:         return False
6:     endif
7: else
8:     return False
9: endif
```

(Used Nested if statement)

Why are Booleans important?

- Booleans **control** the **flow** of your program.
- This allows you to write code that can **branch**. Code that is more than just a list of instructions.

Practice

XOR(P,Q) **Exclusive or**

Write an algorithm for XOR(P,Q).

P	Q	XOR(P,Q)
F	F	F
F	T	T
T	F	T
T	T	F

Algorithm: XOR(P,Q)

Requires: Two Boolean values P, Q.

Returns: A Boolean variable

```
1.  if  P  then
2.      if Q  then
3.          return False
4.      else
5.          return True
6.      endif
```

Practice

XOR(P,Q) **Exclusive or**

Write an algorithm for XOR(P,Q).

P	Q	XOR(P,Q)
F	F	F
F	T	T
T	F	T
T	T	F

Algorithm: XOR(P,Q)

Requires: Two Boolean values P, Q.

Returns: A Boolean variable

```
1. if P == Q then
2.         return  False
3. else
4.           return True
5. endif
```

Practice exercise:

- Write an algorithm **Or** that takes two Booleans *b* and *c*, that returns False if both *b* and *c* are False, and True otherwise.
- Try to write **And** and **Or** using just **one if-then-else statement**.

Practice exercise:

Write an algorithm for NOR(P,Q).

NOR(P,Q) **NOT OR**

P	Q	NOR(P,Q)
F	F	T
F	T	F
T	F	F
T	T	F

Write an algorithm for NAND(P,Q).

NAND(P,Q) **NOT AND**

P	Q	NAND(P,Q)
F	F	T
F	T	T
T	F	T
T	T	F

Compounded conditional statements

Sometimes, we can simplify nested IF or multiple IF structures:

```
if P
    if Q
        statement 1
…
```

$\Rightarrow$

```
if (P && Q)
        statement 1
…
```

```
if P
    statement 2
endif
if Q
    statement 2
endif
…
```

Same statements!

$\Rightarrow$

```
if (P || Q)
        statement 2
…
```

Example

```
let x=0
if a>=50
    if b!=100
        x=1
    endif
endif
if c<20
    x=1
endif
return x
```

```
let x=0
if (a>=50) && (b!=100)
    x=1
endif
if c<20
    x=1
endif
return x
```

```
let x=0
if (a>=50) && (b!=100) || (c<20)
    x=1
endif
return x
```

Main and Sub algorithm

Note : main and sub-algorithms, breaking down the tasks into smaller, reusable components.

Find the Largest of Four Numbers (Using Sub-Algorithm)
Main Algorithm: FindLargestOfFour
Sub Algorithm: FindLargestOfTwo

Sub-Algorithm: FindLargestOfTwo
Header:
Algorithm: FindLargestOfTwo(n1,n2)
Input: n1, n2 (integers)
Return: largest of the two numbers
Body:

```
1. if n1 >= n2 then
2.     return n1
3. else
4.     return n2
5. endif
```

Main Algorithm: FindLargestOfFour
Header:
Algorithm: FindLargestOfFour(n1,n2,n3,n4)
Input: n1, n2, n3, n4 (integers)
Return: largest of the four numbers
Body:

```
1. Let largest1 = FindLargestOfTwo(n1, n2)
2. Let largest2 = FindLargestOfTwo(n3, n4)
3. Let largestFinal = FindLargestOfTwo(largest1, largest2)
4. return largestFinal
```

Example: the Luhn algorithm

- Have you ever bought anything on the Internet?
- Did you make a typo when entering your credit card number?
- How does amazon know that a credit card number is wrong, without having contacted your bank?

Credit/debit cards

- My credit card number consists of 16/19 digits:
    - the first few identify the bank
    - the next numbers identify my account
    - one number is a ‘dummy’ – it’s only there to make the whole credit card number valid.
- We’ll deal with “mini credit cards” with 4 numbers, but the principles are exactly the same!

Luhn explained

- Take the numbers:
  - 8 7 6 3 $\rightarrow$ 5 7 6 3
- Counting from the right, double every second number.
  - 16 7 12 3 $\rightarrow$ 10 7 12 3
- If the result is more than 9, subtract 9.
  - 7 7 3 3 $\rightarrow$ 1 7 3 3
- Add all numbers.
  - 7 + 7 + 3 + 3 = 20 $\rightarrow$ 1 + 7 + 3 + 3 = 14
- Is the sum divisible by ten?
  - Yes $\rightarrow$ No

Which numbers are valid mini credit cards:

A. 1 3 4 2 Not valid

B. 1 4 1 2 Valid

C. 8 3 7 5 Valid

D. 9 5 0 0 Not valid

- **Take** the numbers
- Counting from the right, **double** every second number
- If the result is more than 9, **subtract** 9
- **Add** all numbers
- Is the sum **divisible** by 10?

the Luhn algorithm(sub–algorithm)

Algorithm: LuhnDouble(x)

Requires: A number x

Returns: The Luhn-double of x

```
1: if $2 \times x > 9$ then
2:     return $2 \times x - 9$
3: else
4:     return $2 \times x$
5: endif
```

- **Take** the numbers
- Counting from the right, **double** every second number
- If the result is more than 9, **subtract** 9

the Luhn algorithm(main algorithm)

```
Algorithm  Luhn(a, b, c, d)
Requires: Four numbers a,b,c and d
Returns: True if a b c d is a valid
  mini credit card number; False
  otherwise
    let c′= LuhnDouble(c)
    let a′= LuhnDouble(a)
    return ((a′+b+c′+d) mod 10 == 0)
```

Declare new variables!

- **Take** the numbers
- Counting from the right, **double** every second number
- If the result is more than 9, **subtract** 9
- **Add** all numbers
- Is the sum **divisible** by 10?

Example: the Luhn algorithm

Algorithm Luhn(a, b, c, d)

Requires: Four numbers a,b,c and d

Returns: True if a b c d is a valid mini credit card number; False otherwise

```
let $c'$ = LuhnDouble($c$)
let $a'$ = LuhnDouble($a$)
return (($a'+b+c'+d$) mod 10 ≡ 0)
```

Call other functions!

- **Take** the numbers
- Counting from the right, **double** every second number
- If the result is more than 9, **subtract** 9
- **Add** all numbers
- Is the sum **divisible** by 10?

Example: the Luhn algorithm

Algorithm Luhn(a, b, c, d)

Requires: Four numbers a,b,c and d

Returns: True if a b c d is a valid mini credit card number; False otherwise

```
let $c'$ = LuhnDouble($c$)
let $a'$ = LuhnDouble($a$)
return (($a'+b+c'+d$) mod 10 = = 0)
```

Return complicated expressions!

- **Take** the numbers
- Counting from the right, **double** every second number
- If the result is more than 9, **subtract** 9
- **Add** all numbers
- Is the sum **divisible** by 10?

Sub-algorithm

It is a common practice to break down large algorithms/codes into smaller sub-algorithms (also called sub-routines), and then call them in a main algorithm with an order of execution.

Luhn Algorithm

Sub-algorithm

Algorithm: LuhnDouble(x)
Requires: a number x
Returns: the Luhn-double of x

```
1. if 2*x>9
2.       return 2*x-9
3. else
4.       return 2*x
5. endif
```

calling sub-algorithm

Main algorithm

Algorithm: Luhn(a, b, c, d)
Requires: four single-digit numbers
Returns: True if *abcd* is a valid mini credit card number; False otherwise

```
1. let c′ = LuhnDouble(c)
2. let a′ = LuhnDouble(a)
3. let s = a′+ b+ c′+ d
4. return (s mod 10) == 0
```

Trace main/sub-algorithms with the given examples!!

Line 4: a Boolean statement, takes value T/F

Trace : the Luhn algorithm

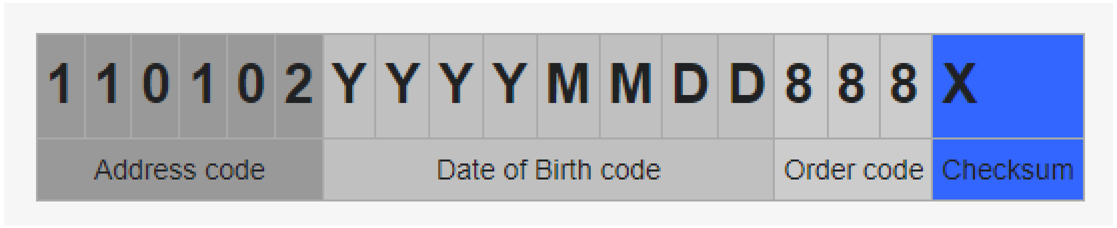
Input = 7680
∴ a = 7 ; b = 6 ; c = 8 ; d =0

```
Algorithm  Luhn(7, 6, 8, 0)
Requires: Four numbers 7,6,8 and 0
Returns:???
    let c′= LuhnDouble( 8)
    let a′= LuhnDouble(7)
    return ((5+6+7+0) mod 10 = = 0)
```

Example

Chinese Resident Identity Card Number

- 18-digit number



1	1	0	1	0	2	Y	Y	Y	Y	M	M	D	D	8	8	8	X
Address code						Date of Birth code								Order code			Checksum

中华人民共和国
居 民 身 份 证
SAMPLE
签发机关 区县级公安局名称
有效期限 20YY.MM.DD – YYYY.MM.DD/长期
姓 名 某某某
性 别 男/女 民 族 汉
出 生 19/20YY 年 M 月 DD 日
住 址 省级行政区名称-区县级行政区名称-具体住宿地址
彩色免冠照片
公民身份号码 110102YYYYMMDD888X

- last digit is made for validating check

“modulus 11” algorithm

Note: content in this slide won’t be assessed in exam.

Source: https://en.wikipedia.org/wiki/Resident_Identity_Card

Point to Note:

The Luhn Algorithm starts at the end of the number, from the last right digit to the first left digit. One must multiply by 2 all digits of even rank, going from left to right as aforementioned. If the double of a digit is equal or superior to 10, replace it by the sum of its digits. The find the sum s of all digits found. The control digit c is equal to c=(10−(smod10)mod10).
(A friendly reminder: mod refers to the modulo operation. Modulo is the name of the calculus of the remainder in the Euclidean division. The modulo calculator returns the rest of the integer division.).

Keywords used in todays lecture

Requires

Returns

Assignment

Boolean

Check digit

Sub algorithm

Review Activity

URL : https://forms.office.com/r/6kyTqgSHdR

Introduction to Boolean algebra
and modulo 10 algorithm