BLM1011 – Bilgisayar Bilimlerine Giriş I

by

Z. Cihan TAYŞİ

Outline

- Fundamental concepts
  - A little history, definition, design & analyzing
- Complexity
- Flow charts
  - Elements, connections
- Algorithms
  - Variable concept, Loops, Array concept, Example problems
- Pseudo codes

A little history

- Muhammad ibn Musa Al-Khwarizmi
  - Few details of al-Khwārizmī's life are known with certainty. He was born in a Persian[3] family and Ibn al-Nadim gives his birthplace as Khwarezm[9] in Greater Khorasan (modern Xorazm Region, Uzbekistan).
  - http://www-groups.dcs.st-andrews.ac.uk/~history/Mathematicians/Al-Khwarizmi.html
- Book on arithmetic:
  - Hindu numeration, decimal numbers, use of zero, method for finding square root
  - Latin translation (c.1120 CE): "Algoritmi de numero Indorum"
- Book on algebra
  - Hisab al-jabr w'al-muqabala

Definition – I

- the central concept underlying all computation is that of the algorithm
  - an algorithm is a ***step-by-step sequence of instructions*** for carrying out some task
- programming can be viewed as the process of designing and implementing algorithms that a computer can carry out.
- a programmer's job is to:
  - create an algorithm for accomplishing a given objective, then
  - translate the individual steps of the algorithm into a programming language that the computer can understand

Definition – II

- Algorithms are well-defined sequence of unambiguous instructions
- must terminate (to produce a result)
- Algorithm description relies on a well-defined “instruction language”

- **Example: Manual Addition**

Describe the method!

```
    123456
+   789001
    ______
    912457
```

Definition – III

- the use of algorithms is not limited to the domain of computing
  - e.g., recipes for baking cookies
  - e.g., directions to your house
- there are many unfamiliar tasks in life that we could not complete without the aid of instructions
  - in order for an algorithm to be effective, it must be stated in a manner that its intended executor can understand
    - a recipe written for a master chef will look different than a recipe written for a college student
  - as you have already experienced, computers are more demanding with regard to algorithm specifics than any human could be

Designing & Analyzing Algorithms

- 4 steps to solving problems (George Polya)
  1. understand the problem
  2. devise a plan
  3. carry out your plan
  4. examine the solution
- **EXAMPLE:** finding the oldest person in a room full of people

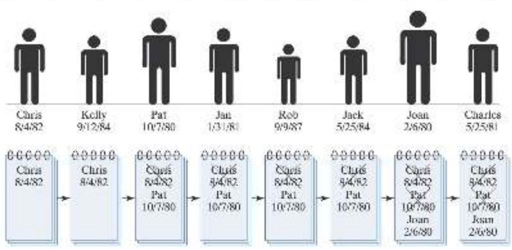
1.understanding the problem

- initial condition – room full of people
- **goal** – identify the oldest person
- assumptions
  - ✓ a person will give their real birthday
  - ✓ if two people are born on the same day, they are the same age
  - ✓ if there is more than one oldest person, finding any one of them is okay

2.we will consider 2 different designs for solving this problem

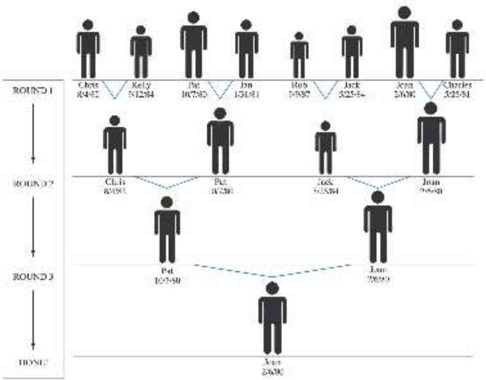
Algorithm #1

- **Finding the oldest person (algorithm 1)**
  1. line up all the people along one wall
  2. ask the first person to state their name and birthday, then write this information down on a piece of paper
  3. for each successive person in line:
     1. ask the person for their name and birthday
     2. if the stated birthday is earlier than the birthday on the paper, cross out old information and write down the name and birthday of this person
- when you reach the end of the line, the name and birthday of the oldest person will be written on the paper



Algorithm #2

- **<u>Finding the oldest person (algorithm 2)</u>**
  1. line up all the people along one wall
  2. as long as there is more than one person in the line, repeatedly
     1. have the people pair up (1st with 2nd, 3rd with 4th, etc) – if there are an odd number of people, the last person will be without a partner
     2. ask each pair of people to compare their birthdays
     3. request that the younger of the two leave the line
- **when there is only one person left in line, that person is the oldest**



Algorithm Analysis – I

- determining which algorithm is "better" is not always clear cut
  - it depends upon what features are most important to you
    - if you want to be sure it works, choose the /clearer algorithm
    - if you care about the time or effort required, need to analyze performance
- **algorithm 1** involves asking each person's birthday and then comparing it to the birthday written on the page
  - the amount of time to find the oldest person is ***proportional to the number of people***
  - if you double the amount of people, the time needed to find the oldest person will also double
- **algorithm 2** allows you to perform multiple comparisons simultaneously
  - the time needed to find the oldest person is ***proportional to the number of rounds it takes to shrink the line down to one person***
    - which turns out to be the logarithm (base 2) of the number of people
  - if you double the amount of people, the time needed to find the oldest person increases by a factor of one more comparison

Algorithm Analysis – II

- for algorithm 1:
  - 100 people 5*100 = 500 seconds
  - 200 people 5*200 = 1000 seconds
  - 400 people 5*400 = 2000 seconds
  - . . .
  - 1,000,000 people 5*1,000,000 = 5,000,000 seconds

when the problem size is large, performance differences can be dramatic for example,

assume it takes 5 seconds to compare birthdays

- for algorithm 2:
  - 100 people 5* log 100 = 35 seconds
  - 200 people 5* log 200 = 40 seconds
  - 400 people 5* log 400 = 45 seconds
  - . . .
- 1,000,000 people 5* log 1,000,000 = 100 seconds

Big-oh Notation

- to represent an algorithm's performance in relation to the size of the problem, computer scientists use what is known as Big-Oh notation
  - executing an O(N) algorithm requires time proportional to the size of problem
    - given an O(N) algorithm, doubling the problem size doubles the work
  - executing an O(log N) algorithm requires time proportional to the logarithm of the problem size
    - given an O(log N) algorithm, doubling the problem size adds a constant amount of work
- based on our previous analysis:
  - algorithm 1 is classified as O(N)
  - algorithm 2 is O(log N)

Another Algorithm Example

- SEARCHING: a common problem in computer science involves storing and maintaining large amounts of data, and then searching the data for particular values
  - data storage and retrieval are key to many industry applications
  - search algorithms are necessary to storing and retrieving data efficiently
  - e.g., consider searching a large payroll database for a particular record
    - if the computer selected entries at random, there is no assurance that the particular record will be found
    - even if the record is found, it is likely to take a large amount of time
    - a systematic approach assures that a given record will be found, and that it will be found more efficiently

- There are two commonly used algorithms for searching a list of items
  - sequential search – general purpose, but relatively slow
  - binary search – restricted use, but fast

Sequential Search

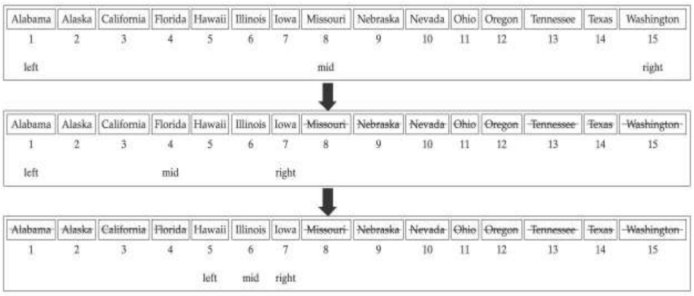
- is an algorithm that involves examining each list item in sequential order until the desired item is found
- for finding an item in a list
  - start at the beginning of the list
  - for each item in the list
    - examine the item - if that item is the one you are seeking, then you are done
    - if it is not the item you are seeking, then go on to the next item in the list
  - if you reach the end of the list and have not found the item, then it was not in the list
- **guarantees** that you will find the item if it is in the list
  - but it is not very practical for ***very large databases***
  - **worst case**: you may have to look at every entry in the list

Binary Search

- involves continually cutting the desired search list in half until the item is found
  - the algorithm is only applicable if the list is ordered
    - e.g., a list of numbers in increasing order
    - e.g., a list of words in alphabetical order
- finding an item in an ordered list
  - initially, the potential range in which the item could occur is the entire list
  - as long as items remain in the potential range and the desired item has not been found, repeatedly
    - examine at the middle entry in the potential range
    - if the middle entry is the item you are looking for, then you are done
    - if the middle entry is greater than the desired item, the reduce the potential range to those entries left of the middle
    - if the middle entry is less than the desired item, the reduce the potential range to those entries right of the middle
- ***by repeatedly cutting the potential range in half, binary search can hone in on the value very quickly***

Binary Search Example

- suppose you have a sorted list of state names, and want to find Illinois
  - start by examining the middle entry (Missouri)
    - since Missouri comes after Illinois alphabetically, can eliminate it and all entries that appear to the right
  - next, examine the middle of the remaining entries (Florida)
    - since Florida comes before Illinois alphabetically, can eliminate it and all entries that appear to the left
  - next, examine the middle of the remaining entries (Illinois)
    - the desired entry is found



Search Analysis

- **sequential search**
  - in the worst case, the item you are looking for is in the last spot in the list (or not in the list at all)
    - as a result, you will have to inspect and compare every entry in the list
  - the amount of work required is proportional to the list size
    - → sequential search is an O(N) algorithm
- **binary search**
  - in the worst case, you will have to keep halving the list until it gets down to a single entry
    - each time you inspect/compare an entry, you rule out roughly half the remaining entries
  - the amount of work required is proportional to the logarithm of the list size
    - → binary search is an O(log N) algorithm
- imagine searching a phone book of the United States (280 million people)
  - sequential search requires at most 280 million inspections/comparisons
  - binary search requires at most log(280,000,000) = 29 inspections/comparisons

Another Algorithm Example – I

- Newton's Algorithm for finding the square root of N
  - start with an initial approximation of 1
  - as long as the approximation isn't close enough, repeatedly
  - refine the approximation using the formula:

newApproximation = (oldApproximation + N/oldApproximation)/2

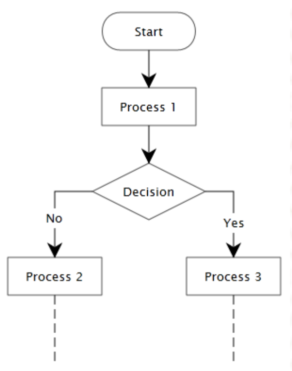
```
Initial approximation = 1
Next approximation = 512.5
Next approximation = 257.2490243902439
Next approximation = 130.16480157022683
Next approximation = 69.22732405448894
Next approximation = 42.00958563100827
Next approximation = 33.19248741685438
Next approximation = 32.02142090500024
Next approximation = 32.0000071648159
Next approximation = 32.0000000000008
Next approximation = 32
```

- **example**: finding the square root of 1024

Another Algorithm Example – II

- Algorithm analysis:
  - Newton's Algorithm does converge on the square root because each successive approximation is closer than the previous one
    - however, since the square root might be a nonterminating fraction it becomes difficult to define the exact number of steps for convergence
  - in general, the difference between the given approximation and the actual square root is roughly cut in half by each successive refinement
    - **demonstrates O(log N) behavior**

Flow charts



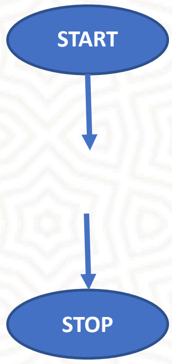
- Elements
- Connections

Flowchart Components – I

- Start & Stop (Begin & End)
- Read / Inputs
- Connections
  - arrows, circles
- Statements/Process
- Decisions
  - if, case/switch, ...
- Loops
  - for, while, do while ...

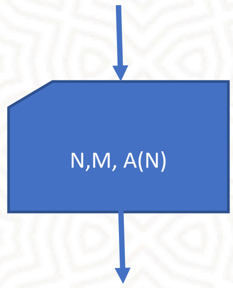
Flowchart Components – II

- START
  - shows the start of the algorithm
  - each flowchart must have a START point
- STOP
  - shows where the algorithm ends
  - each flowchart must have a STOP point



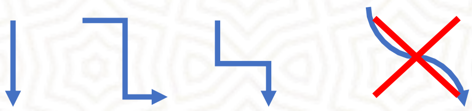
Flowchart Components – III

- Input
  - shows variables, which are supplied by the user
  - each variable is separated by a comma
- Reading arrays
  - option 1 : use the input box in a for loop
  - option 2 : use A(N) notation.
    - **read variable N before A(N) !**



Flowchart Components – IV

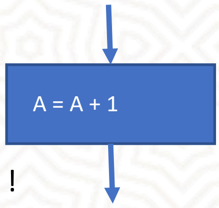
- Arrows
  - connects two elements of the flow
  - show the direction of the flow
  - **moves in rectangular fashion**



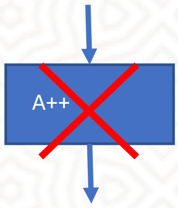
- Circles
  - use them to simplify the connections
    - nested loops, nested controls, and so on…
  - **pay attention to numbering !**

Flowchart Components – V

- Statements/Process
  - includes at least one of the following statement types
    - arithmetic, assignment, and so on...

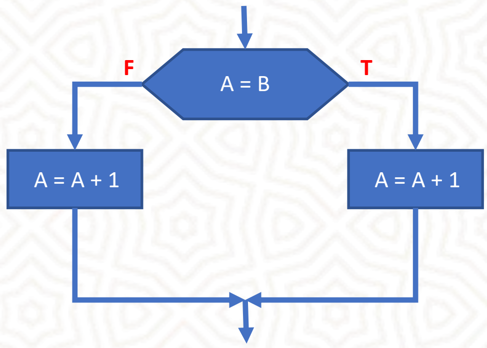


  - algorithm is in dependent of the programming language !
    - do NOT use specific operators
    - do NOT use specific variables



Flowchart Components – VI

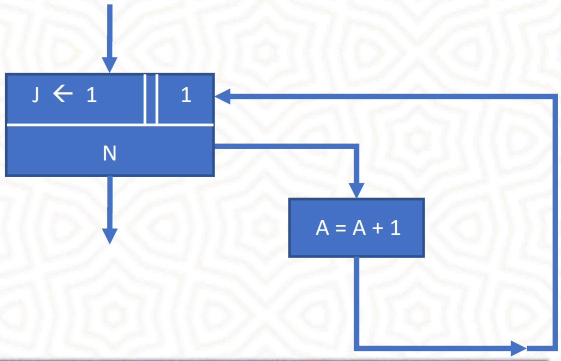
- IF Statements
  - always indicate **(T)**rue and **(F)**alse branches
  - each if statement MUST have the (T)rue branch
  - (F)alse branch is optional



- It is a good idea to use F / T branches on the same side at each of the if statements

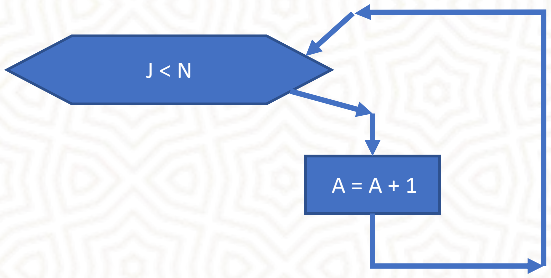
Flowchart Components – VIII

- **FOR loops**
  - are used for creating a loop for distinct number of iterations
  - Contains three parts
    - Initial conditioning
    - Terminating condition
    - increment/decrement



Flowchart Components – IX

- **While loops**
  - are used to create loops with unknown number of iterations
    - mostly depends on satisfying a condition



Pseudo Codes

- A different way to express an algorithm
  - A combination of human readable directives and programming codes
  - can be written in several ways
  - always define your inputs and outputs
  - show intermediate variable types, if possible
  - numbering the lines is a good idea !

Input: A nonempty string of characters $S_1 S_2 \ldots S_n$, and a positive integer n giving the number of characters in the string.
Output: See the related problem below.
Procedure:

```
Get n
Get S1 S2 ... Sn
Set count = 1
Set ch = S1
Set i = 2
While i ≤ n
   If Si equals ch
      Set count = count + 1
   Set i = i + 1
Print ch, ' appeared ', count, ' times.'
Stop
```

Problem 1.1 *What is printed if the input string is* pepper*?*

Problem 1.2 *What is printed if the input string is* CACCTGGTCCAAC*?*

What is a Variable ?

- **Variables** are used to store information to be referenced and manipulated in a computer program.
- They also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves.
- It is helpful to think of variables as containers that hold information. Their sole purpose is to label and store data in memory. This data can then be used throughout your program.

Example #1

- Please find the minimum of the given three numbers...

- **We will receive 3 numbers from the user**
- **We will find the minimum among those numbers**

Example #2

- Please find the properties of a given triangle

- **We will receive dimensions (3 edges) of a triangle**
- **We will state the properties of the triangle**
  - **if all edges are same => EŞ KENAR**
  - **if just two edges are equal => İKİZ KENAR**
  - **if none of the edges are equal => ÇEŞİT KENAR**

Example #3

- Develop an algorithm that uses minimum number of bills to pay a given amount of money.

- **user will supply the amount of money**
- **we already know the available bills (200, 100, 50, 20, 10, 5)**

Loops

- Deterministic loops
  - The number of iterations of such a loop are known in advance, even before the loop has started.
  - Most counting loops are deterministic.
  - Before they start, we can say how many times they will execute
- non-Deterministic loops
  - A loop that is driven by the response of a user is not deterministic, because we cannot predict the response of the user.
  - Non-deterministic loops usually are controlled by a Boolean,
  - and the number of iterations is not known in advance

A Range of numbers

- find the average of given numbers

- **First, user will tell us number (N) of elements it will give**
- **Then he/she will supply the numbers one by one**
- **We should add up numbers and after the last element divide the summation by N**

Fibonacci numbers

- Please find the N^{th} Fibonacci number

- **Fibonacci numbers**
  - $F_{i+1} = F_i + F_{i-1}$
  - **starts from 1 and goes like 1, 1, 2, 3, 5, 8, 13, 21**

Newton's Square Root

- We will calculate the square root of a given number

- **user will supply a number**
- **we also need a threshold value**
  - **preferably supplied by the user**
- **you know the formula !**
  - **calculate a new estimation using the formula**
  - **iterate till the difference between iterations is smaller then the threshold**

Arrays

- An array is a group of related data values (called elements) that are grouped together.
- **In most cases,** all of the array elements must be the **same data type**.

Organizing An Array

- Please reverse the position of elements in a given array

- **user will supply number of elements**
- **user will supply elements of the array**
- **we will switch first element with the last and continue to do it till all elements are switched**

Organizing an array

- Please develop an algorithm that finds the position of the element with the minimum value

- **user will supply number of elements**
- **user will supply elements of the array**
- **we will examine each element and decide if it is minimum**
- **if yes, we will store its position**