Lecture 3: Algorithmic Thinking

What are we facing? What to do?

- What are we facing?
  - Problems
- What to do?
  - Understand the problems
  - Understand our capability
  - Devise a plan to solve the problems (with regard to our capability)
  - Check the correctness of our plan
  - Carry out the plan
  - Examine the solution (check the result of our plan)

What are we facing? What to do?

- Problem: How much did I earn last year?
- Understand the earning problem:
  - I got salary $w USD every month.
  - There are 12 months last year.
- Understand my capability (which is useful to know my earning)
  - I am able to do calculations: addition, multiplication, etc.
- Devise a plan to solve the earning problem
  - Lookup my monthly salaries at my payment stubs: w1, w2, …, w12
  - Initialize, my earning, s = 0
  - Set s = s + w1, s = s+w2, …, s = s+w12
  - At the end, s will accumulate the amount I have earned last year.
- Check the correctness of my plan
  - Do I have incomes other than the salaries? If not, the plan is okay; otherwise, I need to revise my plan to include the other incomes.
- Carry out my plan above
- Examine the solution (check the result) for the earning problem
  - Check the correctness of my calculations

Mathematician’s steps of problem solving

- Understand the problem.
- Devise a plan. (The capability of mathematics is assumed.)
- Carry out the plan.
- Examine the solution.

George Polya (1887 – 1985)

Computer scientist's steps of problem solving

- Understand the **problem**
- Devise an **algorithm** to solve the problem
  - An algorithm is a series of steps that can be followed to solve the problem
- Detail the algorithm into a computer **program** so that a computer can execute each step of it
  - Steps are in computer instructions
  - Or steps are in high level **computer language** which can be translated into detailed computer instructions
- Compile the program into **machine language** (binary-form instructions)
- Execute the program
- Debug the program: Test it and see if there are mistakes.
- Maintenance: Modify the program to handle newly discovered bugs, changing problem setting

An algorithm example (The idea of steps!)

- Problem: Who is the oldest person in the room?
- Your capability:
  - Identify each person
  - Ask each person for his/her age
  - Have any person move to any place in the room
- Algorithm:
  1. Have all the people to move to left side of the room
  2. Pick a person from the left side, ask his/her age
     1. If his/her age is older than the person standing before you
        1. Have the person before you move to right side of the room
        2. Have the picked person from the left side move to the place before you
     2. Otherwise, ask the picked person to go to the right side of the room
  3. Repeat step 2, until there are no one left on the left side
  4. The person who stands before you is the oldest person in the room

Algorithm to solve a problem

- An algorithm is a series of steps that can be followed to solve the problem.

Another algorithm example (The idea of efficiency)

- Problem: Who is the oldest person in the room?
- Your capability:
  - Identify each person
  - Have any pair of persons compare their ages, and identify the older person
  - Have any person move to any place in the room
- Algorithm:
  1. Have all the people form a line in the room
  2. As long as there is more than one person in the line, repeatedly
     1. Have the people pair up. If there are an odd number of people, the last person will remain without partner
     2. Ask each pair of people to compare their ages
     3. Request the younger of the two leave the line.
  3. When there is only one person left in the line, that person is the oldest

Comparison of algorithms

- The first algorithm requires n steps for n people in the room
- The second algorithm requires |log(N)| steps for n people in the room

Computer Program

- A program is a series of instructions that specify exactly what the computer is supposed to do. Instructions can be
  - in high level programming languages
  - Or in machine language (binary coded instructions)
- Stored program scheme: The programs in machine language are stored in computer memory, the CPU fetches and executes the instructions in the program one by one.

Computer Programming languages

- High level programming languages: Languages that provide high-level abstractions and constructs (which correspond to machine level operations) for solving problems using computers.
  - Assembly languages: specify instructions using words instead of binary numbers
  - C language, C++, JAVA, JavaScript, etc.:
    - specify instructions using words
    - abstract memory cells into variables (with names) rather than memory addresses in binary number
    - abstract more than one machine instructions into single high-level instructions, e.g. instructions of various arithmetic computations, conditional choices, repetition, and functions
- Machine language:
  - specify instructions in binary number
  - directly correspond to CPU operations
  - Memory is accessed using binary number coding address

From algorithm to program

- The steps in algorithms depend on the capability of “you”; in the computer case, it depends on the capability of the computer
- Each implementable algorithm step must correspond to computer instruction steps
- Design algorithm in terms of computer instruction is tedious
- Instead, we
  1. design algorithm in human-sense steps;
  2. convert the human-sense steps into instructions of high level computer languages: e.g. Java, C, C++, etc.
  3. convert the instructions in high level computer languages into machine language
- Who does the coversion (some mappings again!)
  - Human: convert human-sense steps into high level computer language steps
  - Computer: convert high level language steps into machine language steps

Example of high level language

In C language:

```
double w1 = 4000;
double w2 = 4020;
double w3 = 4000;
double s = 0;
s = s + w1;
s = s + w2;
s = s + w3;
```

In machine language (symbols replace the binary numbers):

Memory layout:

```
0: 4000
1: 4020
2: 4000
3: 0;
4: Load R0, 3;
5: Load R2, 0;
6: ADD R3, R2, R0
7: Store 3, R3
8: Load R0, 3;
9: Load R2, 1;
10: ADD R3, R2, R0
11: Store 3, R3
12: Load R0, 3;
13: Load R2, 2;
14: ADD R3, R2, R0
15: Store 3, R3
```

Compiler and Interpreter

- Compiler: Translate a program in the high level language into a equivalent program in the machine language before the execution.
- Interpreter: Translate a program in the high level programming, **piece by piece,** into the machine language; execute the obtained piece of program in the machine language before the subsequent piece of the program is being translated into the machine language.

Compiling vs. Interpreting

- Compiling
  - Translate the program once for all
  - Run quickly without the translation overhead
  - The effect of the instructions in high level language is not immediately available
- Interpreting
  - Translate the program when it is to be executed
  - Run relatively slow because of the translation overhead
  - The effect of the instructions in high level language is immediately available

Software life-cycle

- Analyze the problem
- Specify the problem strictly
- Devise algorithm (and software architecture: how to organize the data, and organize the program.)
- Coding: implement the algorithm in some computer language(s)
- Testing
- (Documentation, and software training and support)
- Maintenance

Bugs

- A **software bug** is an error, flaw, mistake, failure, or fault in a computer program that
  - prevents it from working as intended,
  - or produces an incorrect result.
- Examples of bugs
  - The Mars Climate Orbiter was lost in space in 1999 because of a bug in calculating its route: different metric systems have been used without conversion from one to another.
  - A patriot missile failed to intercept a scud fired at US troops in 1991 because of bug in calculating time: the unit of internal clock is 1/10 second, but the measure was multiplied by 1/10 to produce time in second in stead of the correct fact 10.

Summary

- Algorithm and algorithm thinking
- Computer program
- Stored program scheme
- Computer programming language, high level language, machine language
- Compiler and interpreter
- Software life-cycle