Representing Algorithms

210CT 2015/16

Week B, block 3

1 Algorithms again

In which an odd sequence of thoughts brings enlightenment later

1.1 What is an algorithm?

- A way of doing things
- A **finite list of instructions**
- An **effective** method
- Made of **well-defined** steps
- A process of turning an **input state** into an **output state**
- All of the above.

1.2 What do we need to express them?

- Almost anything
- We use the ideas of **Turing Completeness** and **Turing equivalency** to describe computers, processors and programming languages
  - (all the same thing from a certain abstract view)
- To be equivalent to all other machines, we need:
  1. Conditional branching
  2. Modifiable state

1.3 Conditional... er...

- Conditional branching
  - In assembly, `jne`, for example
  - In many others `if`

1.4 Modifiable what now?

- Modifiable state
  - Variables or just plain memory access
  - Or a disk
  - Or any other way to hold data
- A program can be described as a series of changes to state
  - Yes, even the functional ones
- The data of a program can be referred to as its "state space"
- If we can't remember any numbers or change some output, we can't do much

1.5 That's all?

- Yes.
- The single instruction computer (SIC, or OISC (one instruction set computer)) can do anything you can do with any other computer
- From a single instruction that modifies state and execution path, it is possible to build any other instruction and therefore any other program

1.6 subleq

- This SIC has the instruction subleq
  - subtract-and-branch-if-less-than-zero
- Example: subleq a b c
  - Subtract value in memory address a from value in memory address b

  - store result in memory address b
  - if the value is less than or equal to 0, jump to c

- Fun fact: since we don't have any other instructions, we can just show the parameters

```
9 10 3 11 9 0 12 12 0 14 100 -1 0
```

1.7 subleq

Try looking at it in triples…

```
9 10 3, 11 9 0, 12 12 0, 14 100 -1 0
```

And with addresses…

```
 9 10  3,11  9  0,12 12  0,14 100 -1, 0
 0  1  2, 3  4  5, 6  7  8, 9 10  11, 12
```

1.8 subleq

- Let's run it!

```
9 10 3, 11 9 0, 12 12 0, 14 100 -1, 0
execute at 0
9 10 3, 11 9 0, 12 12 0, 14 86 -1, 0
execute at 3 (not jumping, just moving on)
9 10 3, 11 9 0, 12 12 0, 15 86 -1, 0
execute at 6 (not jumping, just moving on)
9 10 3, 11 9 0, 12 12 0, 15 86 -1, 0
execute at 0 (jumped!)
9 10 3, 11 9 0, 12 12 0, 15 71 -1, 0
execute at 3 (not jumping, just moving on)
9 10 3, 11 9 0, 12 12 0, 16 71 -1, 0
execute at 6 (not jumping, just moving on)
9 10 3, 11 9 0, 12 12 0, 16 71 -1, 0
execute at 0 (jumped!)
9 10 3, 11 9 0, 12 12 0, 16 55 -1, 0
...and so on
```

1.9 subleq

- Rather than build programmes at this level, higher-level instructions can be built from this one
- A subtract instruction (`sub`)is made when the third parameter is set to the next address.
  - Because if it jumps, it jumps to where it would have gone next anyway
- A branch (`jmp`) can be made like this: `subleq Z Z c` where `Z` is some unused location in memory
  - Because any number subtracted from itself is zero
- An add instruction: `add a,b -> sub Z Z, sub a Z, sub Z b`
  - Because: first make Z=0, then take a from it (so 5 becomes -5, for eg.) then take this from b

2 Pseudo-code

2.1 Not quite a programming language

- But is something close
- It is not close enough to any particular language to give anyone an unfair advantage
- Provides a language independent way to describe an algorithm
- Formal enough to convert into any programming language
- According to the theory, we need very little int he way of instructions, but it's nice to have some higher-level stuff too

2.2 Assigment

- Pseudo-code uses ← for assignment instead of '=' used by most programming languages
  - Sometimes written as <-
- '=' is used for comparison instead of '=='

- Multiple assignment is achieved with: $a \leftarrow b \leftarrow 0$ (a and b are variables)
- Arrays in Pseudo-code sometimes start at 1, unlike most programming languages which start at 0
  - Depends on the author

2.3 Indentation

- Code blocks are defined with indentation, this will be familiar to Python programmers e.g.

```
if(proposition p is true)
   this gets done if p is true
   this also gets done if p is true
this gets done regardless of whether or not p is true
```

- This also applies to loops and functions

2.4 Loops and Conditionals

- Loops, such as while and for retain their commonly understood meanings, as do if and else statements
- Loop counters retain their value after the loop has finished

2.5 Functions

- Capitalise function name, include parameters in brackets
- e.g: DO-STUFF(THING1, DATE)

2.6 Example

- Insertion Sort in Plain English
  - Start with the 2nd item of the list
  - Compare the current value with the values before it
  - Keep shifting it to the left while it's greater than the previous value
  - Go to the next item in the list and repeat
- (for ref: `http://www.sorting-algorithms.com/insertion-sort`)

2.7 Example

- In Pseudo-code

```
INSERTION-SORT(A)
for i ← 1 to length[A]
    key ← A [i]
    j ← i
    while j > 0 and A [j - 1] > key
          A [j] ← A [j - 1]
          j ← j - 1
    A [j] ← key
```

2.8 Example

- C++ Version

```
void insertionSort(int A[], int length){
  for(int i = 1; i < length; i++) {
    int key = A[i];
    int j = i;
    while(j > 0 && A[j - 1] > key) {
      A[j] = A[j - 1];
      j = j - 1;
    }
    A[j] = key;
  }
}
```

2.9 Example

- Python Version

```
def insertionSort(A):
    for i in range(1, len(A)):
        key = A[i]
        j = i
        while j > 0 and A[j - 1] > key:
            A[j] = A[j - 1]
            j = j - 1
        A[j] = key
```

3 Homework

3.1 Pre-homework

1. Write a program that displays your name 10 times
2. Write a function that draws a square of stars of a size given as a parameter
3. Write a program to open a file and display its contents in captials

Stars example:

```
********
********
********
********
********
********
********
********
```

- If you take more than a minute or two for any of these, you need to practice programming
  - Write more code. Write at least one small program every day
  - Use code-academy: `http://www.codecademy.com/` (If you did C++ last year, consider the python classes here and just write your C++ code in your own editor)
  - Watch the videos and read the notes from Google on Python programming (`https://developers.google.com/edu/python/`)
  - Make use of the programming support centre (`https://gitlab.com/coventry-university/programming-support-lab/wikis/home`)

3.2 Homework

1. A linear search of a list involves checking every element in order until the target is found. Write the pseudocode for a linear search.
2. Now imagine an algorithm that examines a list for duplicates. To determine if at least one duplicate exists, the algorithm examines item 1 and then compares it to all other items. If it finds a match, the search is over. If not, it tries item 2 and so on. Write the pseudocode for this algorithm.

3.3 Alternative Homework

- Write a virtual subleq SIC machine.
- You only need to work on "machine code" represented as space-separated sequences of numbers.
- Jumping to address -1 should terminate the program
- Jumping to address -2 should just jump forward one (similar to the `sub` instruction above)
- Writing to address -1 should output the ASCII character of the number written (e.g `-65 -1 -2` outputs "A")
- Demonstrate it works with "hello world!"

Emacs 23.3.1 (Org mode 8.0.3)