DESIGN AND ANANYSIS ALGORITHMS

LECTURE 1

Algorithm Fundamentals

Reference links:

Prof. Xukai Zou, Computer Science Dept.,

Indiana University Purdue University Indianapolis

http://cs.iupui.edu/~xzou/

Prof. Pasi Fränti, School of Computing,

University of Eastern Finland

http://cs.uef.fi/pages/franti/asa/

Open Classroom from Stanford Univ.

http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms

Lecture outline

- Some key concepts
  - Problem
  - Algorithm + Data Structure ⇨ Program
- Brief about the Complexity and Analysis of Algorithm
- Turing Machine
- Primitive Recursive Function
- Exercises

Some key concepts

- ✓ Problem
- ✓ Algorithm
- ✓ Data Structure
- ✓ Program

Problem

- Problem in informatics/computer science
  - A class of problems with the same specification.
  - Example:
    - Solving: $ax^2 + bx + c = 0$ not $2x^2 + 5x + 3 = 0$.
    - Find path from city ***A*** to city ***B*** on a map.
  - **A problem = <Inputs, Outputs>**
- Can the problem be solve?
- How difficult is the problem?
- How to model real-life problems to algorithmic problems?

Algorithm

- Algorithm:

  Is a finite sequence of instructions (well-defined, computer-implementable) for solving a probblem.

  Input $\xrightarrow[\text{Algorithm}]{}$ Output

- Features of algorithms
  - Generality: must apply to a set of defined inputs;
  - Finiteness: must stop after certain instructions;
  - Definiteness: must give a unique result of the problem;
  - Effectiveness: should use suitable time and resource.

Algorithm

- Expressing algorithms
  - Human language;
  - Flowcharts;
  - Pseudo-code.

Algorithm

- Expressing algorithms
  - Human language;
  - Flowcharts;
  - Pseudo-code.
  - Example:

Euclid's algorithm

Step 1 If $n = 0$, return the value of m as the answer and stop; otherwise, proceed to Step 2.

Step 2 Divide m by n and assign the value of the remainder to r.

Step 3 Assign the value of n to m and the value of r to n. Go to Step 1.

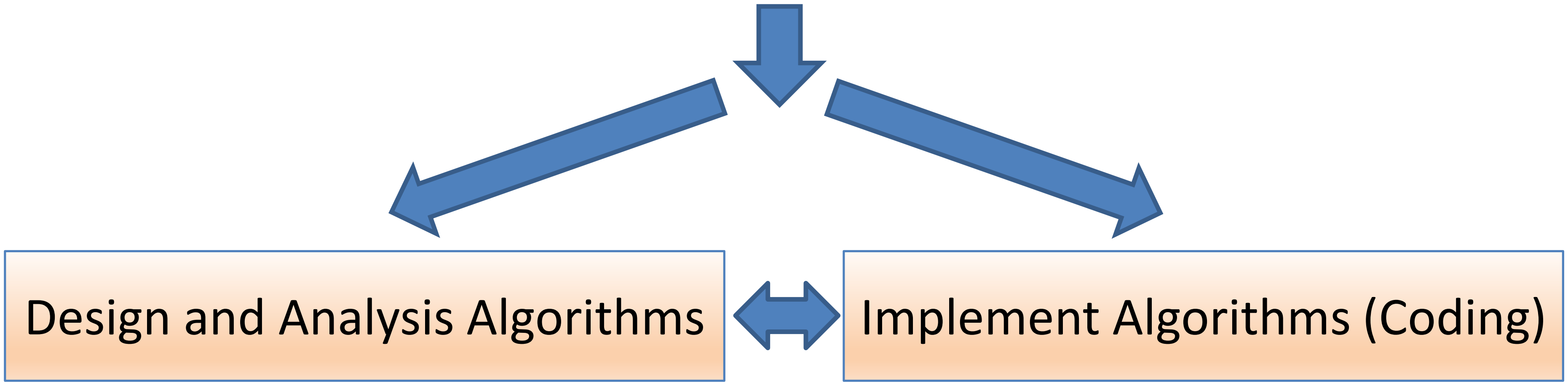
[Anany's book P32]

ALGORITHM *Euclid*(m, n)

//Computes gcd(m, n) by Euclid's algorithm
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while $n \neq 0$ **do**
    $r \leftarrow m \bmod n$
    $m \leftarrow n$
    $n \leftarrow r$
return m

Algorithm

- Research on algorithms
  - Solving problem by algorithmic: which problem can be solved by the algorithm, which cannot.
  - Algorithm Optimization: Find better algorithms.
  - Algorithm Implementation: Implement algorithms on computers.



Data Structure

- Algorithms operate on data (input, output, temporary).
  - ⇨ The ways of organizing data play a critical role in the design and analysis of algorithms.
- Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently.
- General abstract data types (ADT): arrays, files, lists, stacks, queues, trees, graphs,…
  - Specification
  - Implementation ⇔ Algorithm
  - Application

Program

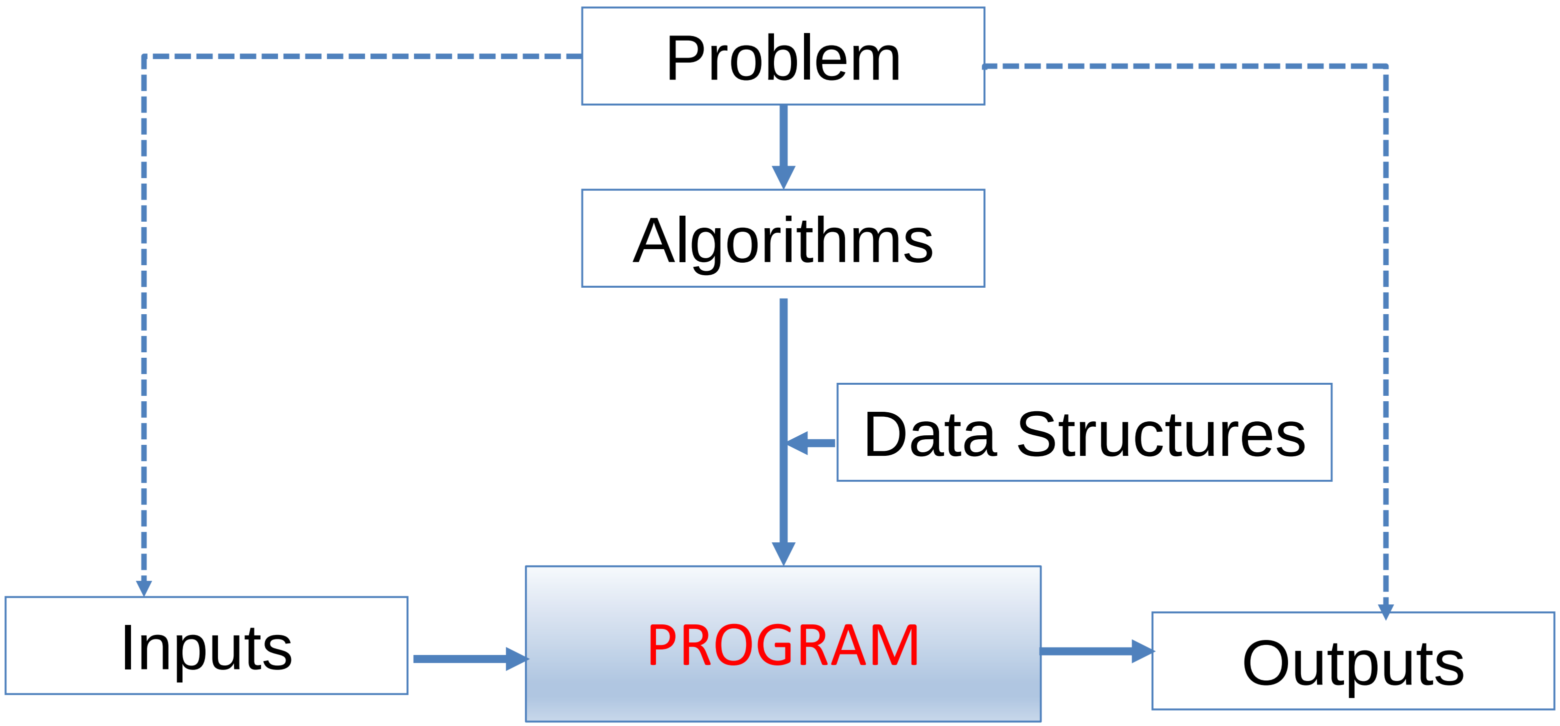
- A computer program is a collection of instructions (written by a programmer) in a programming language to perform specific tasks for solving problem.

Programs = Algorithms + Data Structures

(Niklaus Wirth)

https://en.wikipedia.org/wiki/Niklaus_Wirth

Solving problem on Computer



Complexity and Analysis of Algorithm

- ✓ Design Algorithms
- ✓ Analyze Algorithms

Design Algorithms

- Design Algorithm ⇦ How to find suitable algorithm?
  - Algorithms for specific problem.
  - Algorithm design techniques (Stratergy/Paradigm/Method)
- Some basic algorithm design techniques :
  - Divide-and-Conquer – *Chia để trị*
  - Back-Tracking – *Quay lui*
  - Branch-and-Bound - Nhánh cận
  - Greedy – *Tham lam*
  - Dynamic Programming – *Quy hoạch động*
  - Approximation – *Xấp xỉ*
  - Heuristics

Analyze Algorithms

- How to belive the algorithm?
  - Proving the Correctness – Chứng minh tính đúng.
- How to compare the algorithms?
  - Assessing the Effectiveness – Đánh giá tính hiệu quả.

  Effectiveness: Two kinds of algorithm efficiency

  - Time efficiency - how fast the algorithm runs?

    ⇨ Time complexity – Độ phức tạp thời gian.
  - Space efficiency - how much extra memory it uses?

    ⇨ Space complexity – Độ phức tạp không gian.

Analyze Algorithms

- Example: Compare two sorting algorithms with n=10^6 items.
  - *Insertion sort* has the time complexity $c_1 n^2$.
  - *Merge sort* has the time complexity $c_2 n\log n$.

A little injustice:

- *Insertion sort* with $c_1=2$, and speed 10^9 flops

$$T_{Ins} = 2 \,.\, (10^6)^2 \textit{ flop} \,/\, 10^9 \textit{ flops} = 2000 \; (s)$$

- *Merge sort* with $c_2=50$, and speed 10^7 flops

$$T_{Mer} = 50 \,.\, (10^6 \log 10^6) \textit{ flop} / 10^7 \textit{ flops} \approx 100 \; (s)$$

Merge faster than *Insertion* 20 times

If n increase to 10^7 items the results is 2.3 days vs 13 minutes (!)

Analyze Algorithms

- The complexity of some sorting algorithms

Algorithm	Worst Case	Best Case	In-place	Stable
Selection Sort	O(n^2)	O(n^2)	Yes	No
Insertion Sort	O(n^2)	O(n)	Yes	Yes
Bubble Sort	O(n^2)	O(n^2)	Yes	Yes
Bubble Sort 2	O(n^2)	O(n)	Yes	Yes
Quick Sort	O(n^2)	O($nlogn$)	Yes	No
Merge Sort	O($nlogn$)	O($nlogn$)	No	Yes
Shuffle Sort (Trộn ngẫu nhiên)	?	?		

Analyze Algorithms

- Two ways for analyzing the algorithm
  - By Experimental

    Code – Run – Capture the complexity indicators.

    Pros: Can analyze all programable algorithm.

    Cons: Depend on platform (hardware & software).

  - By Theoretical

    Use mathematic tool to express the complexity as functions of input data size.

    Pros: Not depend on platform; can do with large input data size.

    Cons: Complicates.

Turing Machine

- ✓ Turing Machine Description
- ✓ Turing Machine Structure
- ✓ Turing Machine Example
- ✓ Church-Turing thesis

Turing Machine Description

- A Turing machine is designed as the simplest computing machine, but capable of simulating any algorithm. It is a mathematical concept invented by Alan Turing in 1936.

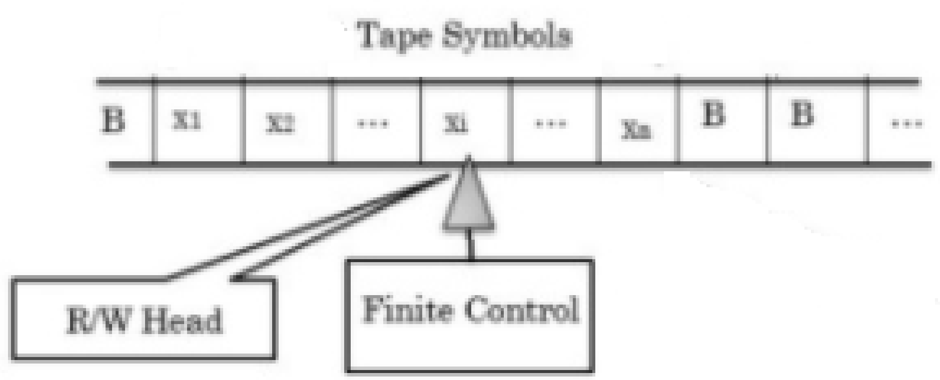
  https://en.wikipedia.org/wiki/Turing_machine

- Alan Mathison Turing (1912–1954) was an English logician, mathematician, cryptanalyst. He is considered to be the father of computer science and artificial intelligence.

  https://en.wikipedia.org/wiki/Alan_Turing

Turing Machine Structure

- Physical description: Turing machine consisted of
  - *An infinite long tape*, on which symbols could be printed.
  - *A read/write head* that can read and write symbols on the tape and move the tape left and right one cell at a time.
  - *A set of state* that stores the states and a *finite table of instruction* control the *head* machine.



Turing Machine Structure

- Turing machine operation:
  - Set input on the tape.
  - The read/write head moves along the tape in steps by instructions until stop or halt state reached.
  - The output given by symbols on the tape.

Demonstration of Turing Machine operation

(file .flv, open with VLC media player)

Turing Machine Structure

- Formal definition:

Turing machine can be formally defined as a 4-tuple:

$$M = (K, \Sigma, \delta, s)$$

Where:

K - set of states of M. $s, h \in K$ are predefined states (start, halt).

Σ - set of symbols/characters of M.

$K \cap \Sigma = \varnothing$ and $\blacktriangleright, \blacksquare \in \Sigma$ are predefined synbols (begin, end)

δ is transition function formed as partial function:

$$\delta : K \times \Sigma \rightarrow \Sigma \times K \times \{\rightarrow, \leftarrow, -\}.$$

$$\delta(t,k) = (k_1, t_1, \{\rightarrow / \leftarrow / -\})$$

- Replace symbol $k \rightarrow k_1$, change state $t \rightarrow t_1$;
- Move read/write head by direction $\{\rightarrow / \leftarrow / -\}$

Turing Machine Example

- Turing Machine example: $M_1 = (K, \Sigma, \delta, s)$

  Where:

  K = {s,q,h} s – start, h – halt.

  $\Sigma = \{0, 1, \blacktriangleright, \blacksquare\}$ $\blacktriangleright$ - begin, $\blacksquare$ - end

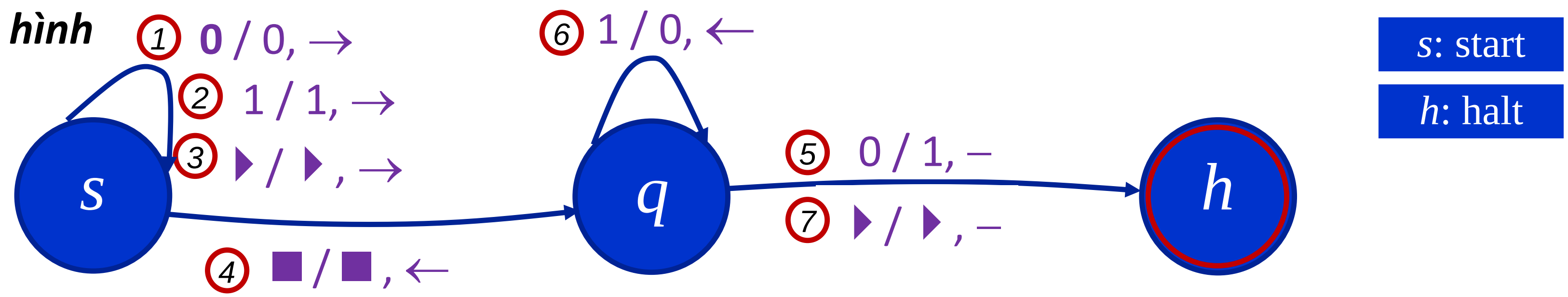
  Table of transition functions δ :

STT	t	k	$\delta(t,k) = (k_1, t_1, \{\rightarrow / \leftarrow / -\})$
1	s	0	$(0, s, \rightarrow)$
2	s	1	$(1, s, \rightarrow)$
3	s	$\blacktriangleright$	$(\blacktriangleright, s, \rightarrow)$
4	s	$\blacksquare$	$(\blacksquare, q, \leftarrow)$
5	q	0	$(1, h, -)$
6	q	1	$(0, q, \leftarrow)$
7	q	$\blacktriangleright$	$(\blacktriangleright, h, -)$

Turing Machine Example

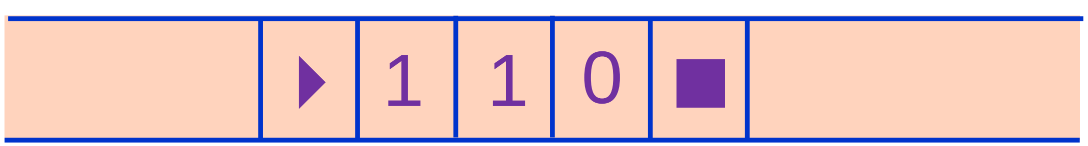
- Operation of M_1 **Demo**

Mô hình



Hoạt động

$$n=101 \xrightarrow{M1} 110 = n+1$$



Step	Func
1	3
2	2
3	1
4	2
5	4
6	6
7	5
8	-

Turing Machine Example

- Operation of M_1:

Operate with other inputs:

- $n = 1001$ Result: ouput value, number of steps?
- $n = 1011$ Result: output value, number of steps?

Unexpected instance of M_1 and solution: M_1 with input:

- n = 11 Result: ouput value, number of steps?

What trouble?

Improve M_1?

Chusch -Turing thesis

- The Turing machine as a computational model
  - Turing machines can be used to simulate any algorithm that can be performed with a computer.
  - Turing machine is an efficient model for describing the functionality of computer processors.

“Turing Machine ≡ Algorithm”

- Church-Turing thesis and the variations:
  - “A function on the natural numbers can be calculated by an effective method if and only if it is computable by a Turing machine”.
  - A variation of the thesis: If a problem can be solved by a finite and definite instructions, then it can be solved with a computer program.

https://en.wikipedia.org/wiki/Church%E2%80%93Turing_thesis

Turing Machine and Algorithm

- Turing Machine and Conway's Game of Life

  P. Rendell, *Turing Machine Universality of the Game of Life,* Complexity and Computation 18, Springer 2016, DOI 10.1007/978-3-319-19842-2_2

  Play Game of life online: https://playgameoflife.com/

Primitive Recursive Function

- ✓ Introduction PRF
- ✓ PRF definitions
- ✓ Prove a function is PRF

Primitive Recursive Function

- Definition: Primitive Recursive Function (PRF) are among the number-theoretic functions. These functions take *n* arguments (*n*-ary), which are the natural numbers (nonnegative integers) {0, 1, 2, ...} to the natural numbers.
- Primitive recursive function is computable. Mean exist a Turing machine *M* to compute the value of function *f* if *f* is a PRF.

Reference link:

https://en.wikipedia.org/wiki/Primitive_recursive_function

Primitive Recursive Function

- The basic primitive recursive functions are given:
  - **Constant function**: The 0-ary constant function 0 is PRF.
  - **Successor function**: The 1-ary successor function *S*, which returns the successor of its argument is PRF. S(x) = x + 1.
  - **Projection function**: For every $n \geq 1$ and each i with $1 \leq i \leq n$, the n-ary projection function P_i^n, which returns its i-th argument, is PRF.

    $(P_i^n(x_1,..,x_i,\ldots x_n)=x_i)$

Primitive Recursive Function

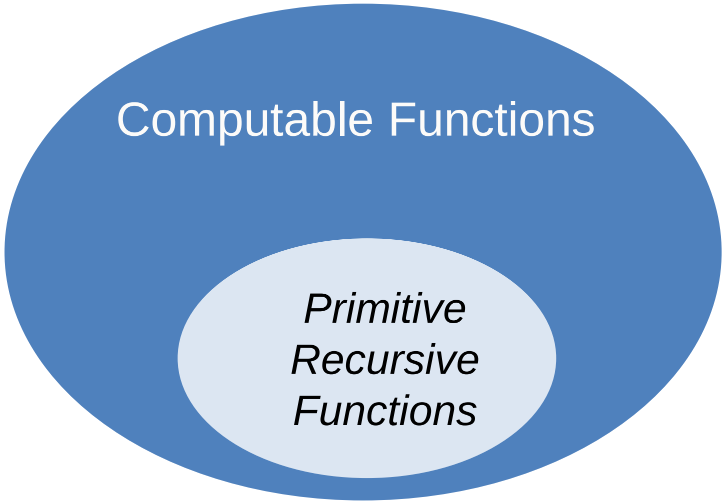
- More complex PRF obtained by applying the operations:
  - **Composition**: Given f is a k-ary PRF, and $g_1,\dots,g_k$ are k m-ary PRF, the composition of f with $g_1,\dots,g_k$, i.e. the m-ary $h(x_1,\dots,x_m) = f(g_1(x_1,\dots,x_m),\dots,g_k(x_1,\dots,x_m))$ is PRF.
  - **Primitive recursion**: Given f, a k-ary PRF, and g, a $(k+2)$-ary PRF, the $(k+1)$-ary function h is defined as the primitive recursion of f and g:

    $h(0,x_1,\dots,x_k) = f(x_1,\dots,x_k)$ and

    $h(S(n),x_1,\dots,x_k) = g(h(n,x_1,\dots,x_k),n,x_1,\dots,x_k)$ is PRF

Primitive Recursive Function

- Identifications:
  - PRFs are computable, mean exist a Turing machine or algorithm to compute the value of the function.
  - But set of PRF are not all the computable fuctions



Primitive Recursive Function

- Some provable PRFs:
  - Addition: add(a,b)= $a+b$
  - Multiplication: mul(a,b)= $a \times b$
  - Exponentiation: exp(a,b)= a^b,
  - Factorial: fac(a) = a!
  - Predecessor: pred(a) = (a>0 ? a-1 : 0)
  - Proper subtraction: proper_sub(a,b) = ($a \geq b$? a-b : 0)
  - Minimum: minimum(a_1, ... a_n)
  - Maximum: maximum (a_1, ... a_n)

Primitive Recursive Function

- Rewrite defined definitions about PRF:

  (1) Constant function: C=const

  (2) Successor function: $S(x) = x+1$

  (3) Projection function: $P_i^n(x_1,..,x_i,\ldots x_n)=x_i$

  (4) Composition: If $g_1,...,g_k$. are m-ary PRF and f is k-ary PRF then:

  $h(x_1,...,x_m) = f(g_1(x_1,...,x_m),...,g_k(x_1,...,x_m))$ is PRF.

  (5) Primitive recursion: If f,g are k-ary and k+2-ary, and h is k+1-ary function such:

  $h(0,x_1,...,x_k) = f(x_1,...,x_k)$,

  $h(S(n),x_1,...,x_k) = g(h(n,x_1,...,x_k),n,x_1,...,x_k)$, is a PRF.

Primitive Recursive Function

- Example of proving a function is PRF:

  Addition: add$(a,b)= a+b$

  Prove add function is a PRF?

  - add$(0,b) = 0 + b = b = P_1^1(b)$ is PRF since P is PRF (1)
  - add$(a+1,b) = a+1+b = a+b+1 =$ add$(a,b)+1$

    $=$ S(add(a,b)) $=$ S(P_1^3(add$(a,b),a,b)$)

    is PRF since S, P are PRFs and Composition (2)

  - From (1) and (2) get add is PRF by Primitive recursion.

Exercises

- Build Turing Machine
- Prove Primitive Recursive Function
- Problem specification, design algorithm and program.
- More detail in Hw1_AlgorithmFundamental.doc