问题求解2——导引

马骏

majun@nju.edu.cn

回顾

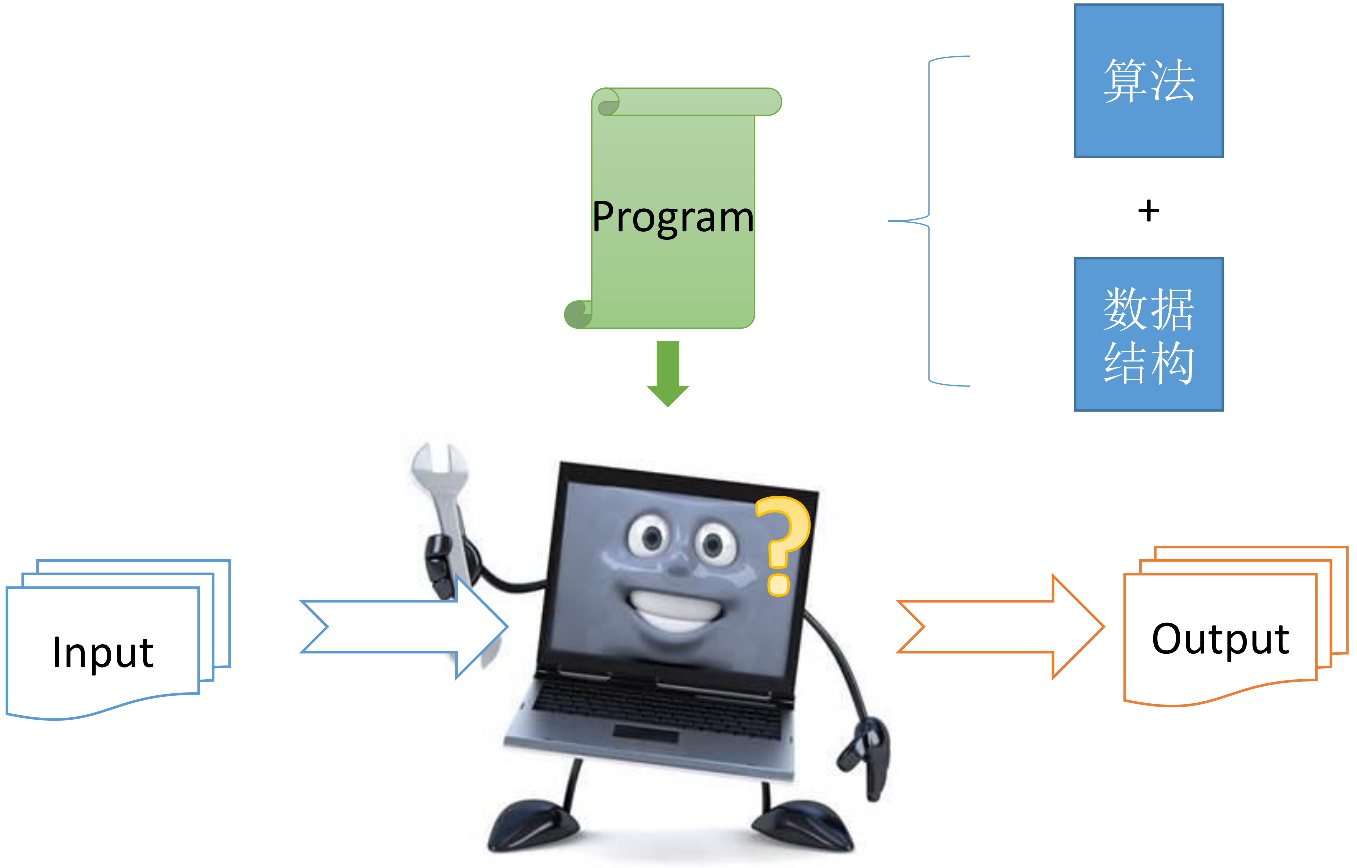
问题求解思想	数学证明方法	离散数学	程序设计基础
Polya 4步骤	Logic/数学归纳法 程序正确性	集合、关系、函数、序、格、布尔代数	基本算法结构 程序正确性

2015.9.22	计算思维导引	
2015.9.24	1-1：为什么计算机能解题	• 理解问题求解的基本过程 • 理解计算机中简单操作为什么能解决复杂问题
2015.9.29	1-2：什么样的推理是正确的	• 掌握命题逻辑与谓词逻辑的基本推导方法
2015.10.08	1-3：常用的证明方法	• 掌握逻辑正确的常用证明方法
2015.10.13	1-4：基本的算法结构 扩展阅读：Goto Statement is Harmful	• 理解基本的算法结构：顺序、分支、循环、子程序、递归 • 理解程序最基本单元的正确性概念
2015.10.27	1-5：数据与数据结构	• 理解数据在计算机问题求解中的核心作用 • 通过例子理解几种常用的数据结构
2015.10.30	1-6：如何将算法告诉计算机	• 理解程序设计语言的基本概念 • 了解程序在计算机中的执行方式

2015.11.12	1-7：不同的程序设计方法	• 了解不同的程序设计方法：函数式、命令式、对象式、逻辑式
2015.11.5	1-8：集合及其运算	• 掌握集合的基本概念以及基本数学性质 • 进一步巩固数学证明能力
2015.11.27	1-9：关系及其基本性质	• 掌握关系的概念与基本数学性质 • 理解等价关系与次序关系的数学性质 • 进一步巩固数学证明能力
2015.12.3	1-10：函数	• 从问题求解的角度理解函数的概念及其重要的数学性质 • 熟悉函数表述方式
2015.12.10	1-11：有限与无限	• 理解无限集合的重要数学性质，理解可数与不可数的差别 • 理解有限过程与无限过程的概念与差别 • 理解测度空间的基本概念
2015.12.17	1-12：偏序关系和格	• 理解集合上的序关系 • 理解格的基本概念 • 理解偏序集与格的联系与区别

2015.12.31	1-13：布尔代数	• 理解布尔代数基本概念 • 理解布尔代数与格的联系与区别 • 布尔代数表达式的化简
	1-13：问题的难度	• 理解问题固有难度的概念 • 了解NP-完全理论的基本内容
	1-14：基本计算模型与不可计算性	
	1-15：并行与并发	
	1-16：模算术与费马小定理	

程序=数据结构+算法

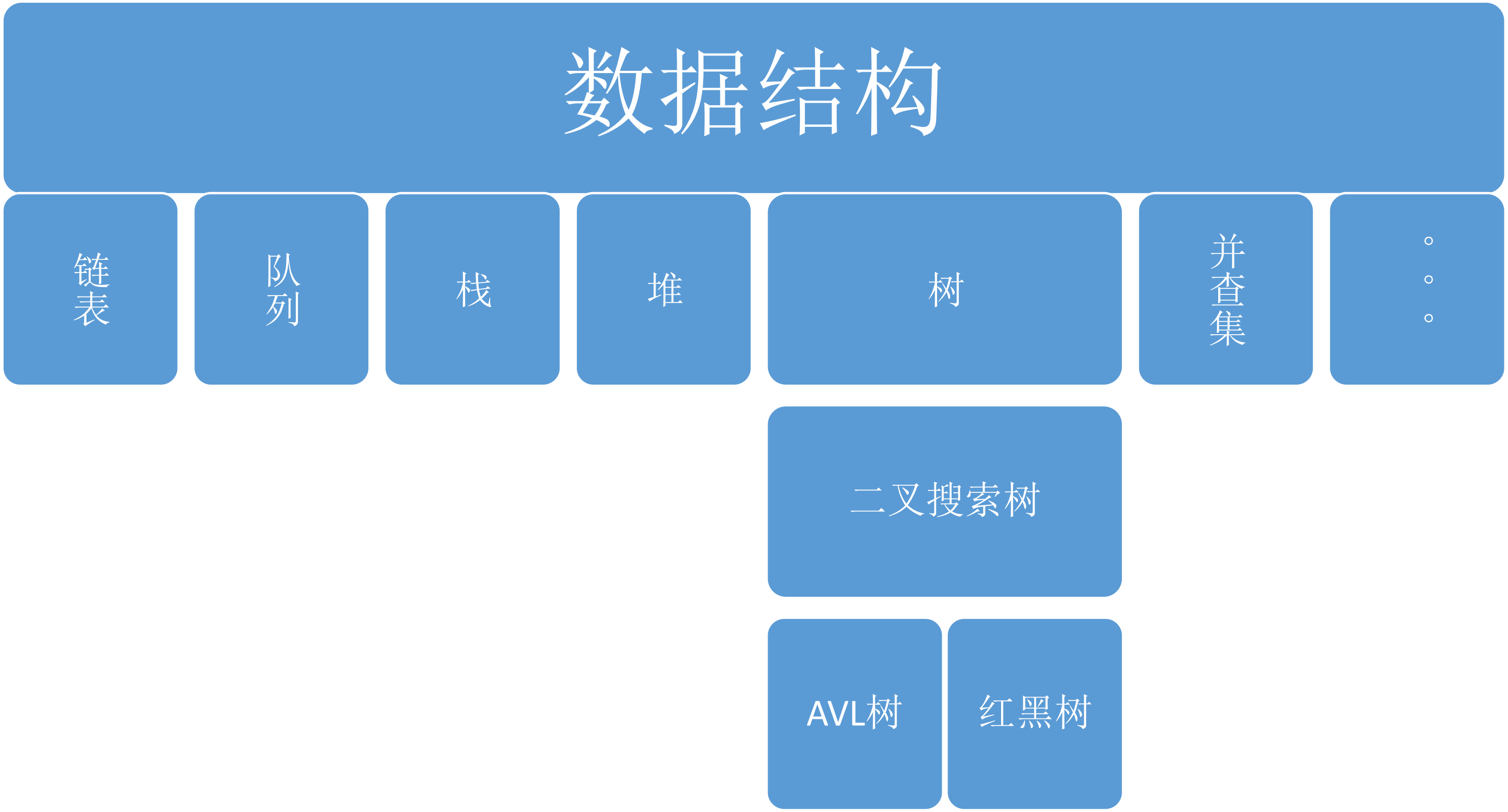


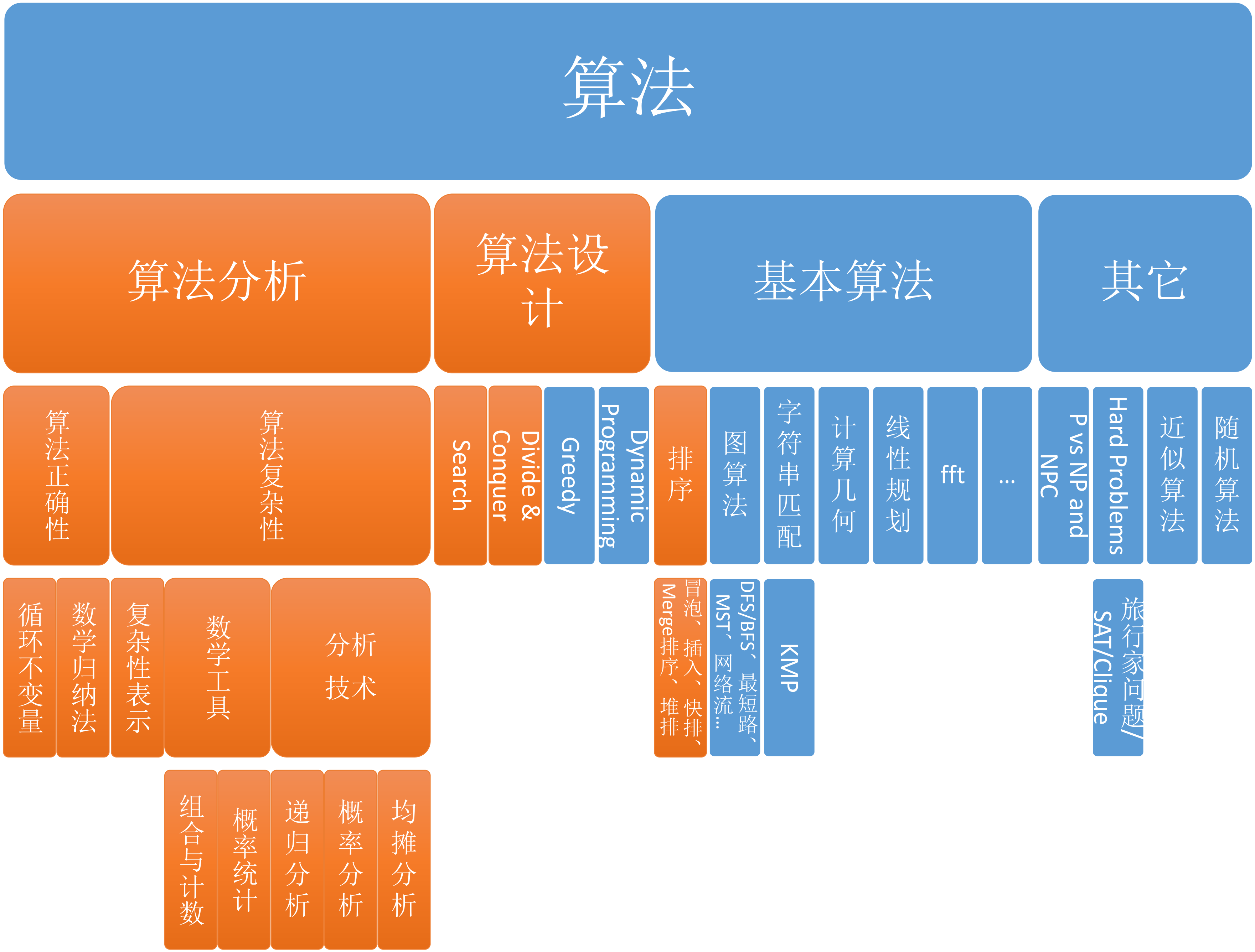
<u>抽象定义</u>：一组数据元素及其之间相互关系

记为：

$$\textbf{Data\_Structure} = \{D, R\}$$

- **D是某一组数据元素集合**
- **R是该集合中所有数据成员之间关系的有限集合**

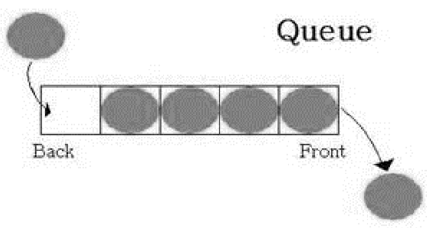




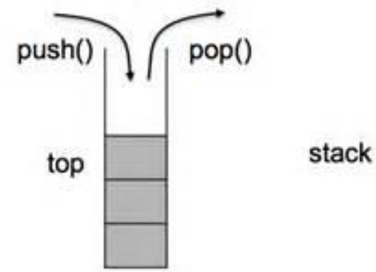
算法
算法分析
算法设计
基本算法
其它
算法正确性
算法复杂性
Search
Divide & Conquer
Greedy
Dynamic Programming
排序
图算法
字符串匹配
计算几何
线性规划
fft
…
P vs NP and NPC
Hard Problems
近似算法
随机算法
循环不变量
数学归纳法
复杂性表示
数学工具
分析技术
冒泡、插入、快排、Merge排序、堆排
DFS/BFS、最短路、MST、网络流…
KMP
旅行家问题/SAT/Clique
组合与计数
概率统计
递归分析
概率分析
均摊分析

数据结构(例)

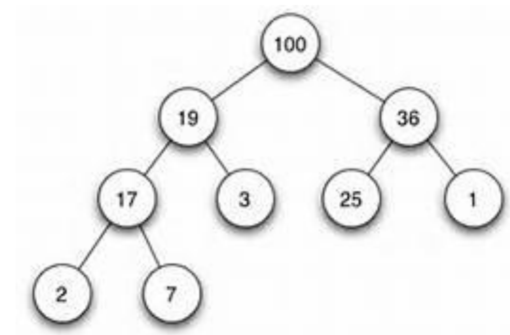
- 队列(Queue)-FIFO



- 栈(Stack)-LIFO

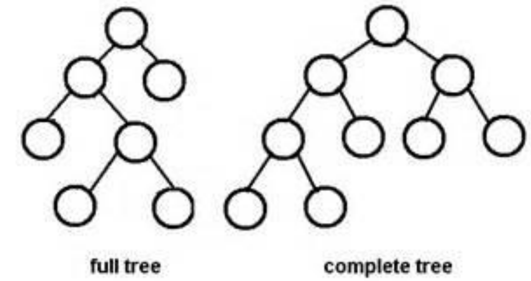


- 堆(Heap)

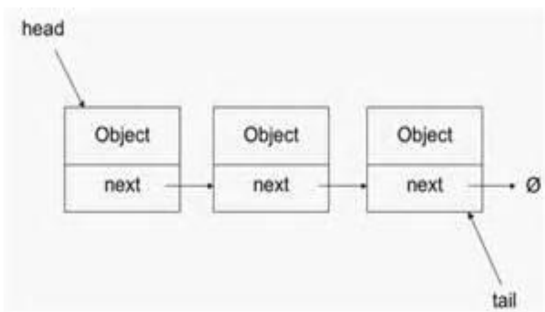


数据结构（例）

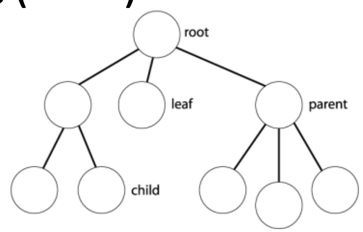
- 二叉树

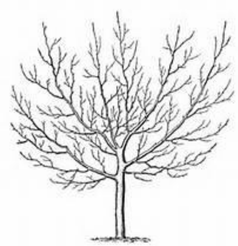


- 链表(Linked List)

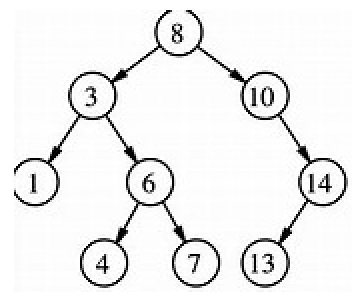


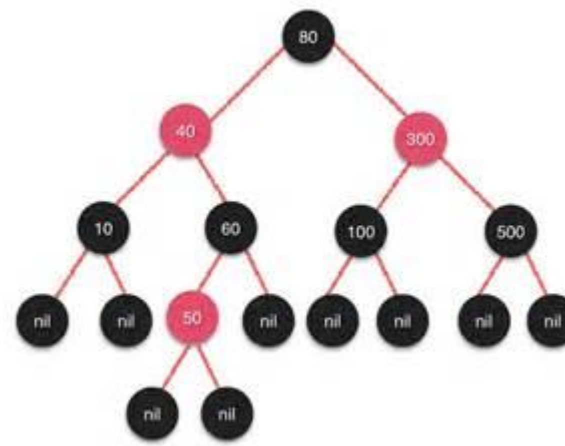
- 树(Tree)





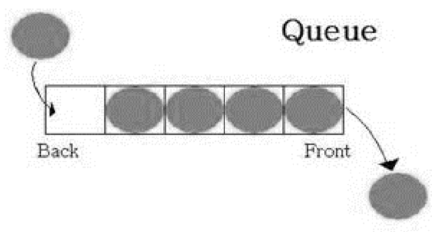
- 二叉搜索树



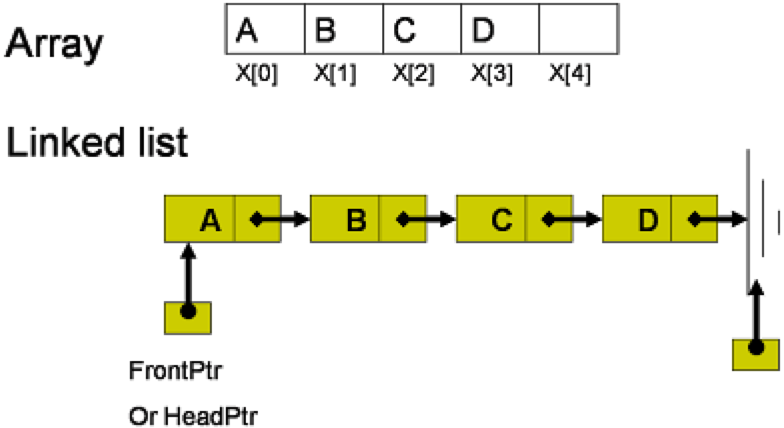


数据结构——
逻辑结构VS存储结构

- 队列(Queue)-FIFO



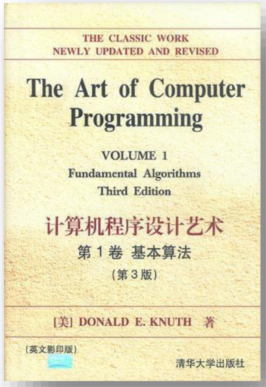
逻辑结构



存储结构

算法(Algorithm)

- Algorithm is the spirit of computing
  - To solve a specific problem (so called an *algorithmic problem*)
  - Combination of basic operations
    - in a precise and elegant way
- Essential issues
  - Model of computation
  - Algorithm design
  - Algorithm analysis

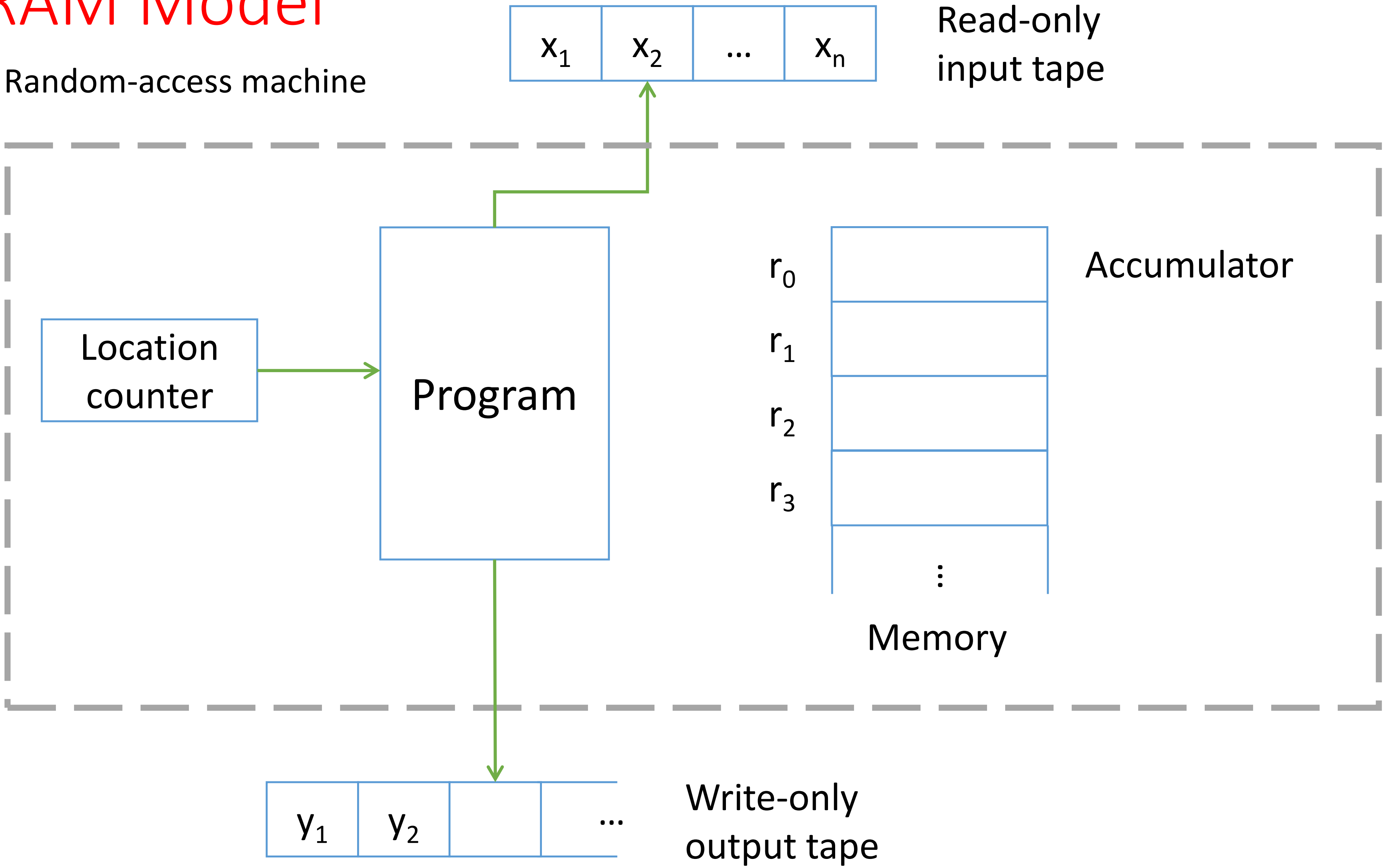


Model of Computation

- Problems
  - Why the algorithms we learn can run almost everywhere?
  - Why the algorithms we learn can be implemented in any language?
- Machine-independent algorithms run on an abstract machine
  - Turing machine: over-qualify
  - RAM model: simple but powerful

RAM Model

Random-access machine



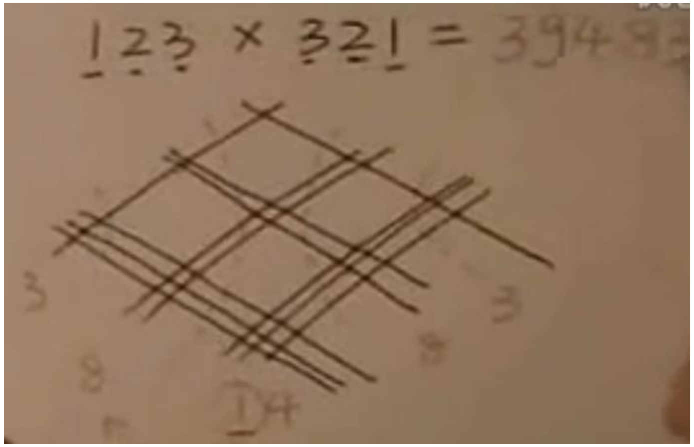
The RAM Model of Computation

- Each *simple operation* takes one time step
  - E.g., key comparison, +/-, memory access, …
- Non-simple operations should be decomposed
  - Loop
  - Subroutine
- Memory
  - Memory access is a simple operation
  - Unlimited memory

Example： Multiplex

- $123*321=\underbrace{123+123+\ldots+123}_{321}=39483$

$$
\begin{array}{cccccc}
 & & & 1 & 2 & 3 \\
\times & & & 3 & 2 & 1 \\
\hline
 & & & 1 & 2 & 3 \\
 & & 2 & 4 & 6 & \\
+ & 3 & 6 & 9 & & \\
\hline
 & 3 & 9 & 4 & 8 & 3
\end{array}
$$



Probably the Oldest Algorithm

- Euclid Algorithm

Problem
- Find the greatest common divisor of two non-negative integers *m* and *n*

Specification

Input: non-negative integer m, n
Output: gcd(m, n)

Euclid algorithm

[E1] n divides m, the remainder -> r
[E2] if r = 0 then return n
[E3] n -> m; r-> n; goto E1

Euclid algorithm – recursive version

Euclid(m,n)
[E1] if n=0 then return m
[E2] else return Euclid(n, m mod n)

程序正确性

- 如何证明一个算法是正确的？
  - 对所有合法输入均正确
  - 输入可能无穷

- 常用技术；
  - 循环不变量
  - 数学归纳法

To Create an Algorithm

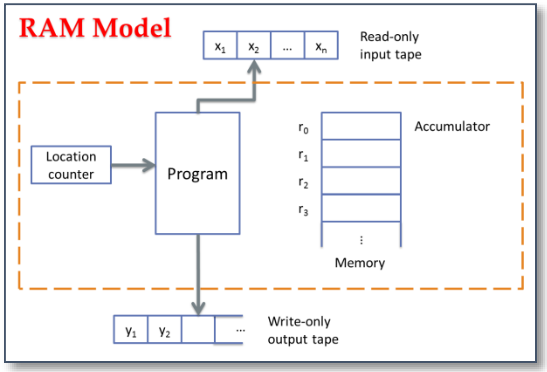
- Algorithm design
  - Composition of simple operations, to solve an algorithmic problem
- Algorithm analysis
  - Amount of work done / memory used
    - In the worst/average case
  - Advanced issues
    - Optimality, approximation ratio, …

Algorithm Analysis

- Criteria
  - **Performance metrics**
- Worst case
  - Best case?
- Average case
  - Average cost?
- Advanced topics
  - Lower bound, optimality, …

Algorithm Analysis

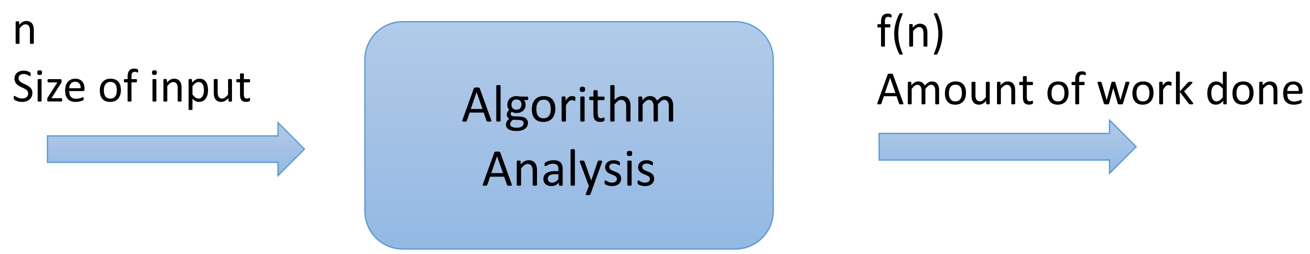
- Criteria
  - **Critical operation**
  - How many critical operation are conducted
- For example



Algorithmic problem	Critical operation
Sorting, selection, searching String matching	Comparison (of keys)
Graph traversal	Processing a node/edge
Matrix multiplication	Multiplication

Algorithm Analysis

- Amount of work done
  - usually depends on size of the input
  - usually does not depend on size of the input only



Worst-case Complexity

- W(n)
  - Upper bound of cost
    - For any possible input of size n
  - $W(n) = \max_{I \in D_n} f(I)$

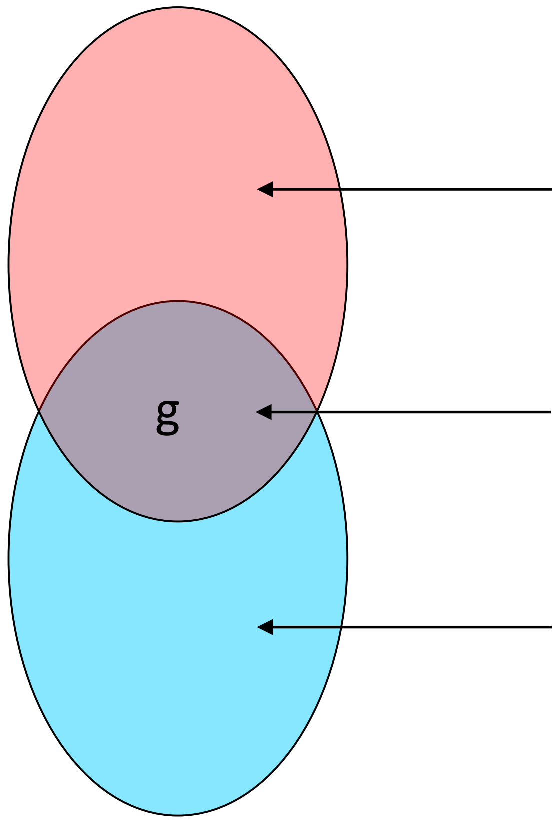
Average-case Complexity

- A(n)
  - Weighted average
  - $A(n) = \sum_{I \in D(n)} \Pr(I)\, f(I)$
- A special case
  - Average cost
    - Total cost of all inputs, averaged over the input size
  - $Average(n) = \frac{1}{|D(n)|} \sum_{I \in D(n)} f(I)$

How to Compare Two Algorithms

- Algorithm analysis, with *simplifications*
  - Measuring the cost by the number of critical operations
  - Large input size only
    - Only the **leading term** in $f(n)$ is considered
    - Constant coefficient is ignored
- Capturing the essential part in the cost in a mathematical way
  - Asymptotic growth rate of $f(n)$

Relative Growth Rate

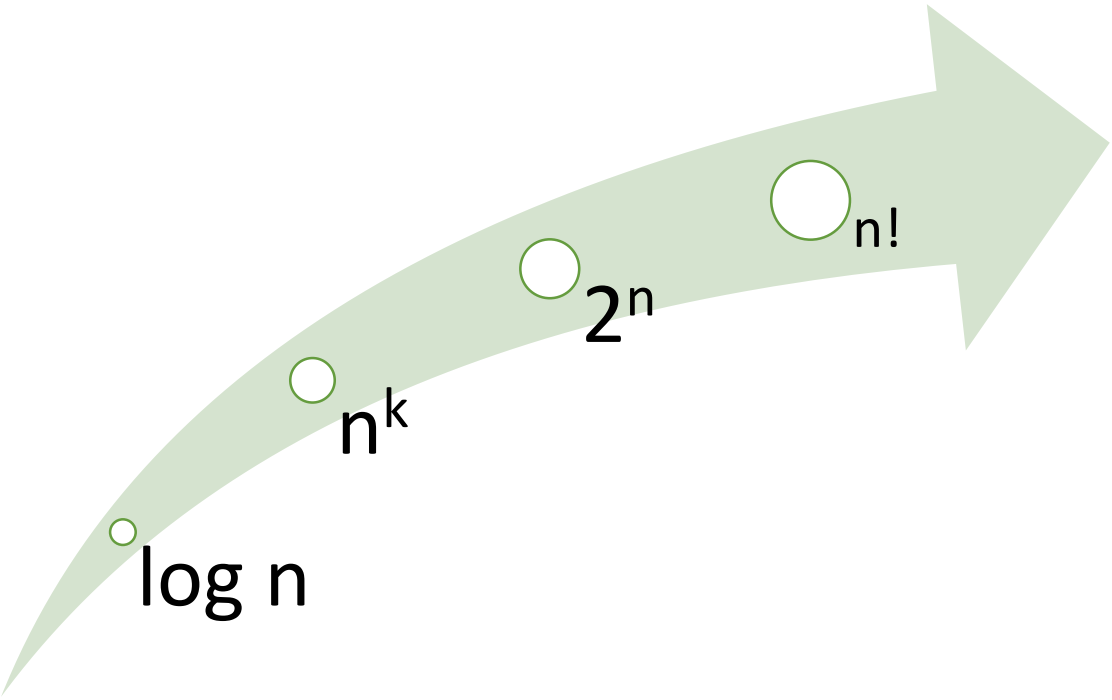


Ω(g):functions that grow at least as fast as g

Θ(g):functions that grow at the same rate as g

O(g):functions that grow no faster as g

Asymptotic Growth Rate



Some Empirical Data

Effect of the Asymptotic Behavior

algorithm		1	2	3	4
Run time in *ns*		$1.3n^3$	$10n^2$	$47n\log n$	$48n$
time for size	10^3	1.3*s*	10*ms*	0.4*ms*	0.05*ms*
	10^4	22*m*	1*s*	6*ms*	0.5*ms*
	10^5	15*d*	1.7*m*	78*ms*	5*ms*
	10^6	41*yrs*	2.8*hrs*	0.94*s*	0.48*s*
	10^7	41*mill*	1.7*wks*	11*s*	48*s*
max Size in time	sec	920	10,000	1.0×10^6	2.1×10^7
	min	3,600	77,000	4.9×10^7	1.3×10^9
	hr	14,000	6.0×10^5	2.4×10^9	7.6×10^{10}
	day	41,000	2.9×10^6	5.0×10^{10}	1.8×10^{12}
time for 10 times size		$\times1000$	$\times100$	$\times10+$	$\times10$

on 400Mhz Pentium II, in C
from: Jon Bentley: *Programming Pearls*

算法分析工具与方法

- 数学工具
  - 组合计数
  - 概率统计
- 分析方法
  - 递归分析
    - Master Theory
  - 概率分析
  - 均摊分析

程序设计方法/思想

- 搜索(Searching)
- 分治法(Divide & Conquer)
- 动态规划(Dynamic Programming)
- 贪心(Greedy)
- 随机算法
- …

搜索(Searching)+剪枝(Pruning)

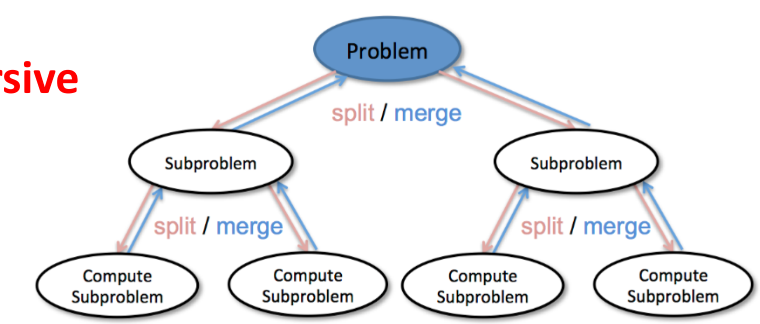
- Examine all possible elements until you find the answer.
- Sometimes it maybe the only way to solve a algorithmic problem
- Information can be used to prune some impossible elements

- Data Structure and Searching
  - Searching progress can be accelerated if the data is well organized
    - E.g., Dictionary
  - Data Structure for Searching:
    - Binary Searching Tree
      - AVL. Red/Black Tree
    - HashTable
  - Searching in Tree
    - BFS/DFS…
  - Searching in well ordered array
    - Binary Search

分治法(Divide & Conquer)

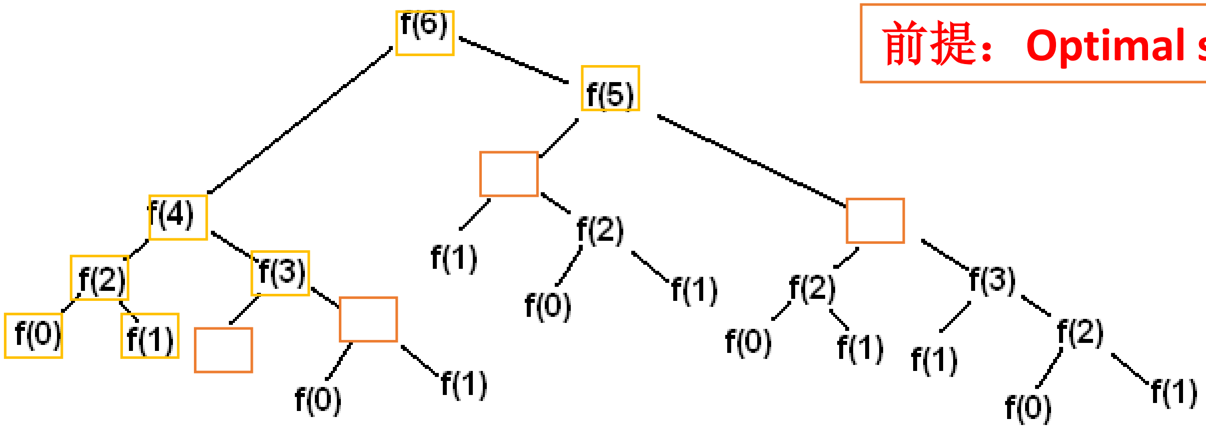
- Divide a BIG problem into some (disjoint) sub-problems
- Solve each sub-problems
- Combine solutions to all sub-problems to obtain the solution to the original BIG problem.

Recursive



动态规划(Dynamic Programming)

- Dynamic programming, like the divide-and-conquer method, solves problems by combining the solutions to sub-problems
- divide-and-conquer algorithms partition the problem into **disjoint** sub-problems
- In contrast, dynamic programming applies when the **sub-problems overlap—subproblems share subsubproblems.**



前提：Optimal substructure

贪心(Greedy)

- Basic idea:
  - Selecting one “optimal” sub-problem instead of checking all sub-problem
  - Local optimal leads to global optimal
- Greedy is Good some of the time, but not always good, depending on the property of problem
    - **Greedy-choice property**
    - **Optimal substructure**

Example1

- 有一个背包，背包容量是M=150。有7个物品，物品不可以分割成任意大小。要求尽可能让装入背包中的物品总价值最大，但不能超过总容量。
- 物品 A B C D E F G
  - 重量w: 30kg 30kg 30kg 30kg 30kg 30kg 30kg
  - 价值p: 10$ 40$ 30$ 50$ 35$ 40$ 30$

目标函数：$\sum p_i$最大
约束条件是装入的物品总重量不超过背包容量：$\sum w_i <= M(M = 150)$

贪心策略：每次选择价格最高的物品，直至无法继续添加

Example2

本题不能通过贪心求解，但是DP可以！

- 有一个背包，背包容量是M=150。有7个物品，物品不可以分割成任意大小。要求尽可能让装入背包中的物品总价值最大，但不能超过总容量。
- 物品 A B C D E F G
    - 重量w: 35kg 30kg 6kg 50kg 40kg 10kg 25kg
    - 价值p: 10$ 40$ 30$ 50$ 35$ 40$ 30$

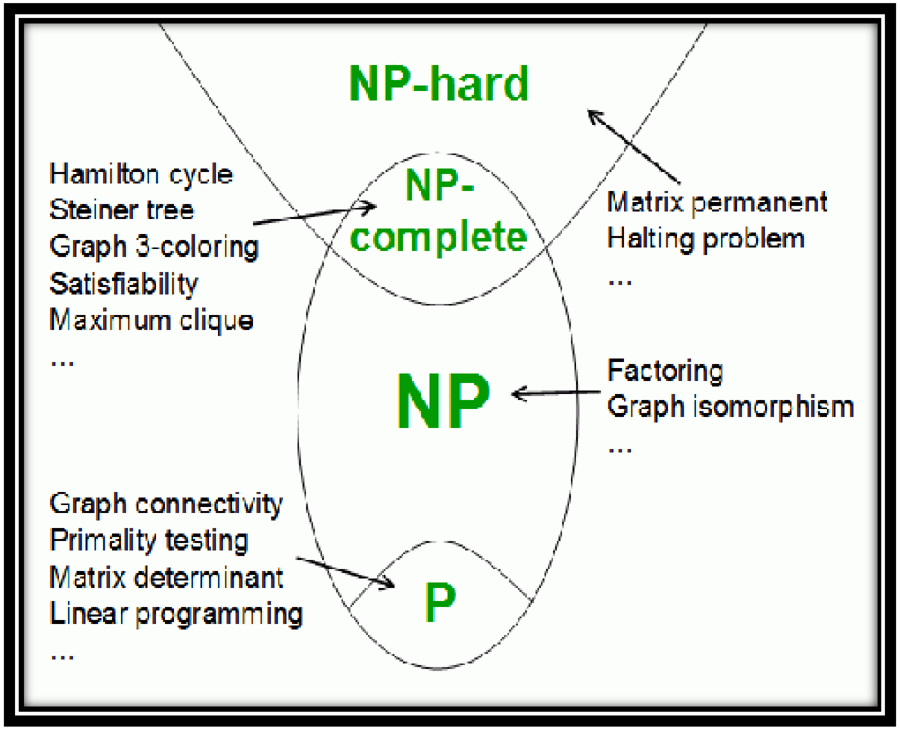
目标函数：$\sum p_i$最大
约束条件是装入的物品总重量不超过背包容量：$\sum w_i <= M(M = 150)$

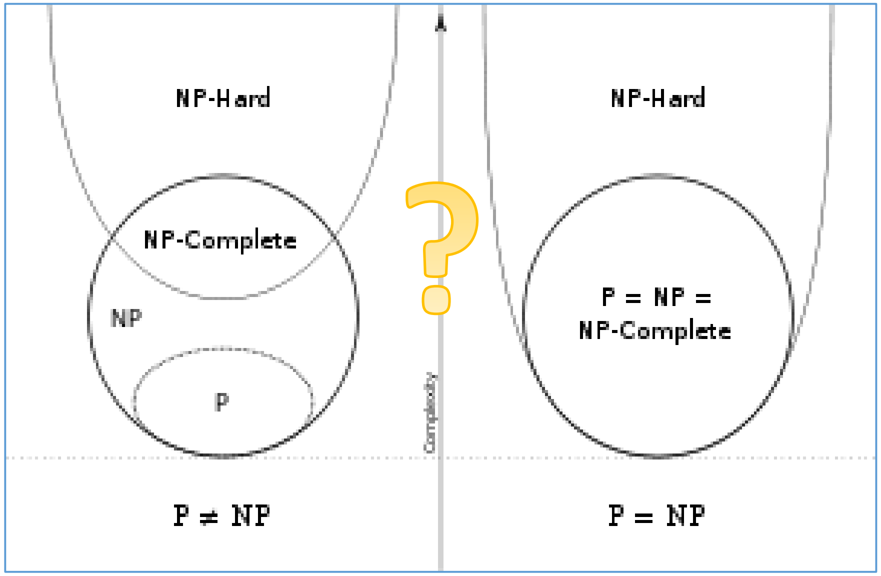
贪心策略：每次选择价格最高的物品，直至无法继续添加

贪心策略：每次选取重量最小物品

贪心策略：每次选取单位重量价值最大的物品

How Hard a Problem can be? P, NP and NP Complete





近似算法

Suppose that we are working on an optimization problem in which each potential solution has a positive cost, and we wish to find a near-optimal solution. Depending on the problem, we may define an optimal solution as one with maximum possible cost or one with minimum possible cost; that is, the problem may be either a maximization or a minimization problem.

We say that an algorithm for a problem has an ***approximation ratio*** of $\rho(n)$ if, for any input of size n, the cost C of the solution produced by the algorithm is within a factor of $\rho(n)$ of the cost C^* of an optimal solution:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \le \rho(n) . \tag{35.1}$$

- How to measure "good enough"?
  - Approximation ratio

随机算法(Randomized Algorithm)

- Random algorithm can be used to obtain acceptable solution.
- Example:
  - To calculate the area of an irregular plane area;
  - To calculate the volume of an irregular object;