算法选择题复习

> ***Strassen算法***
>
> **二、Strassen算法**
>
> 那么有没有比 $\Theta(N^3)$ 更快的算法呢?
>
> 1969年，Volker Strassen提出了第一个算法时间复杂度低于 $\Theta(N^3)$ 矩阵乘法算法，算法复杂度为 $\Theta(n^{log_2^7}) = \Theta(n^{2.807})$ 。从下图可知，Strassen算法只有在***对于维数比较大***的矩阵 $(N > 300)$ ，性能上才有很大的优势，可以减少很多乘法计算。

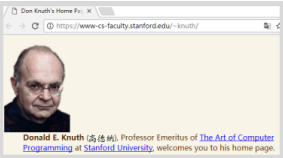
Chapter 0 Preface

- 助教：

  崔绍锞
  闫佳豪
  杨振炜
  闵家旭
  云惟彬
  王泓翔

- 唐纳德.克努特 **1974**年图灵奖 《计算机程序设计艺术》作者 人工智能之父

  （Edsger Wybe **Dijkstra** 与 **Knuth**并称我们这个时代最伟大的计算机科学家）

  Don Knuth's Home Page (stanford.edu)

  唐纳德·克努特（1938-），1974年获得图灵奖，常译为**克鲁斯**，中文名高德纳，算法和程序设计理论与技术的先驱者，经典巨著《计算机程序设计艺术》(The Art of Computer Programming) 的作者，计算机排版系统TEX和METAFONT的发明者，他因这些成就和大量创造性的影响深远的著作而誉满全球。被誉为“人工智能之父”。

- **Nicklaus Wirth 1984**图灵奖 **Pascal**之父 算法 + 数据结构 = 程序：

  - 计算机硬件和软件可以看成两个相互依存但又对立的两个不同体。
  - 算法是软件的核心，也是硬件的灵魂（如：硬件是超大规模集成电路，电信号（0，1）能用来表示逻辑运算，但能进一步进行关系运算、推理、算术运算等，这本身就是算法，或者说，硬件灵魂也是算法【运算规则是算法】）
  - 软件：狭义地讲，就是程序，
    算法 + 数据结构 = 程序
  - 本课程主要从软件视角来学习算法。
  - 算法这么重要，自然地，也是很难学的！

  有一个人因为一句话而得到了图灵奖：Nicklaus Wirth（1984年，图灵奖）——Pascal之父，这个公式对计算机领域的影响程度足以类似物理学中爱因斯坦的“E=MC²”（卡脖子现状的本质就是缺芯少魂，魂，算法！）

- 算法导论：

  《**算法导论**》（Introduction to Algorithms）是麻省理工学院出版社出版的关于计算机中数据结构与算法的图书，作者是托马斯·科尔曼（Thomas H. Cormen）、查尔斯·雷瑟尔森（Charles E. Leiserson）、罗纳德·李维斯特（Ronald L. Rivest）、克利福德·斯坦（Clifford Stein）。第一版刊行于1990年，2009年最新版为第三版。 [1-3]

Chapter 1 The role of algorithms in computing

- **What are algorithms：**

  - **Computing: information processing**
  - **Scientific computing (modeling, computing, verifying)**
    - 数学：公理、规则 => 定理
      - “上帝创造”。描述客观世界，给人们带来无穷乐趣和痛苦
    - (例如：Pi的小数点后面是否存在1000个连续的7？若存在，请找出或证明，若不存在，请证明。)
    - 1900, Hilbert (1862~1943), 巴黎世界数学家大会，“是否存在一个通用的过程（算法），可以自动判定任意命题是否正确？”

    - Alan Turing (1912~1954), 1931, undergraduate in Cambridge Univ.
    - Before 1936, no scientific computing. Turing, 论可计算数及其在判定问题中的应用, 1936.
    - John Von Neumann (1903~1957), 1946, first electronic computer.

    - 算法：过程、工具
      - “人创造”。方便人们研究数学；解决实际问题。

图灵设计的状态自动转移，就是机器指令的例行程序。图灵的指令系统单一不够完善，总结起来主要有两条。第一，没能将指令存储起来重复使用。第二，没能形成实现程序的结构设计。由于这两点缺陷，使图灵机还不能成为处理各种任务的计算机。图灵机欠缺的这两点恰被冯诺依曼提出的程序数据存储的思想解决了。

- **characteristics of algorithm：**

  The characteristics of the algorithm

  - **Output：at least one.**
  - **Correct：An algorithm is said to be correct**
    if, for every input instance, it halts with the correct output.（能停机，且结果正确）
  - **Feasible（可行性，可编程实现）**
  - **Practical (feasible actually，实际可行)**

  - **Incorrect algorithm**
    - **might not halt at all on some input instances, or （不能停机）**
    - **might halt with an answer other than the desired one （停机但结果不正确）**
    - **can sometimes be useful (if error rate can be controlled)**

Chapter 2 Framework for analyzing algorithms

- 几种等价的计算模型

  几种等价的计算模型：原始递归函数，λ 演算，图灵机，RAM

  Computing model (计算模型)

  - RAM, Random-access machine 随机存取器 (Von-neumann, 冯.诺依曼模型)：带着可随机访问的内存，主要是能用电子元器件做出来。实际普遍使用的计算模型，通过程序数据存储的思想解决了图灵奖的不足。
  - 图灵机（七元组的符合体系，抽象的计算机器）：状态自动转移，就是机器指令的例行程序。但指令不能存储起来重复使用，没能形成实现程序的结构设计，由于这两点缺欠，使图灵机还不能成为能够处理各种任务的计算机。
  - λ演算（丘奇，数理逻辑意义上）：可以被称为最小的通用程序设计语言。它包括一条变换规则 (变量替换) 和一条函数定义方式，λ演算之通用在于，任何一个可计算函数都能用这种形式来表达和求值。
  - 原始递归函数（哥德尔，数学意义上）：一种能停机的递归函数，可以用图灵机计算。

Chapter 3 Growth of functions

- 比较时间复杂度的表示方式：

$$o \approx < \ ; \ O \approx \leq \ ; \ \Theta \approx = \ ; \ \Omega \approx \geq \ ; \ \omega \approx >$$

渐近紧界：

For a given function $g(n)$, we denote by $\Theta(g(n))$ the set of functions

$\Theta(g(n)) = \{ f(n)$: there exist positive constants c_1, c_2 , and n_0 such that

$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0 \}.$$

渐近上界：

***O* – notation:** For a given function $g(n)$, we denote by $O(g(n))$ the set of functions

$O(g(n)) = \{f(n)$: there exist positive constants c and n_0 such that $0 \leq f(n) \leq c\, g(n)$ for all $n \geq n_0\}$.

渐近下届：

Ω – notation: For a given function $g(n)$, we denote by $\Omega(g(n))$ the set of functions
$\Omega(g(n)) = \{ f(n)$: there exist positive constants c and n_0 such that $0 \le c\, g(n) \le f(n)$ for all $n \ge n_0 \}$.

非渐近紧的上界：

$o(g(n)) = \{f(n)$: for any positive constants $c > 0$, there exits a constant $n_0 > 0$ such that $0 \le f(n) < c\, g(n)$ for all $n \ge n_0\}$.

非渐近紧的下界：

- The ω-notation denotes an lower bound that is not asymptotically tight. Formally, define $\omega(g(n))$ as the set

  $\omega(g(n)) = \{ f(n)$: for any positive constants $c > 0$, there exits a constant $n_0 > 0$ such that $0 \le c\, g(n) < f(n)$ for all $n \ge n_0\}$.

- 传递性 & 自反性 & 对称性 & 反对称性：

$$
\begin{aligned}
&f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \text{ imply } f(n) = \Theta(h(n)),\\
&f(n) = O(g(n)) \text{ and } g(n) = O(h(n)) \text{ imply } f(n) = O(h(n)),\\
&f(n) = \Omega(g(n)) \text{ and } g(n) = \Omega(h(n)) \text{ imply } f(n) = \Omega(h(n)),\\
&f(n) = o(g(n)) \text{ and } g(n) = o(h(n)) \text{ imply } f(n) = o(h(n)),\\
&f(n) = \omega(g(n)) \text{ and } g(n) = \omega(h(n)) \text{ imply } f(n) = \omega(h(n)).
\end{aligned}
$$

- **Reflexivity（自反性）**

$$
\begin{aligned}
&f(n) = \Theta(f(n)),\\
&f(n) = O(f(n)),\\
&f(n) = \Omega(f(n)).
\end{aligned}
$$

- **Symmetry（对称性）**

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)).$$

- **Transpose symmetry（反对称性）**

$$
\begin{aligned}
&f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)),\\
&f(n) = o(g(n)) \text{ if and only if } g(n) = \omega(f(n)).
\end{aligned}
$$

Chapter 4 Recurrences

- 计算时间复杂度的方法：

  How to obtain asymptotic "Θ" or "O" bounds on the recurrencesolution?

  - Substitution method（置换法）: guesses a bound and then use mathematical induction to prove our guess correct.
  - Iteration method（迭代法）: converts the recurrence into a summation and then relies on techniques for bounding summations to solve the recurrence.
  - Recursion-tree method (a kind of iteration method)
  - Master method （主方法，主定理，母函数法）: provides bounds for recurrences of the form $T(n) = aT(n/b)+f(n)$, where $a \ge 1$, $b > 1$, and $f(n)$ is a given function.

- 主方法 & 母函数法：

$$T(n) = aT(n/b) + f(n),$$

$$T(n) = \begin{cases} \Theta\left(n^{\log_b a}\right), & f(n) = O\left(n^{(\log_b a)-\varepsilon}\right) \\ \Theta\left(n^{\log_b a}\lg n\right), & f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta(f(n)), & f(n) = \Omega\left(n^{(\log_b a)+\varepsilon}\right) \text{ and } af(n/b) \le cf(n) \text{ for large } n \end{cases} \quad \begin{matrix} \exists \varepsilon > 0 \\ , c < 1 \end{matrix}$$

- $T(n) = 9T(n/3) + n$

$$a = 9, b = 3, f(n) = n \quad \Rightarrow \quad n^{\log_b a} = n^{\log_3 9} = n^2 = \Theta(n^2)$$

$$\Rightarrow \quad f(n) = O(n^{\log_3 9 - \varepsilon}), \text{where } \varepsilon = 1 \quad \Rightarrow \quad T(n) = \Theta(n^2)$$

- $T(n) = T(2n/3) + 1$

$$a = 1, b = 3/2, f(n) = 1 \quad \Rightarrow \quad n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$$

$$\Rightarrow \quad f(n) = \Theta(n^{\log_b a}) \quad \Rightarrow \quad T(n) = \Theta(\lg n)$$

- $T(n) = 3T(n/4) + n\lg n$

$$a = 3, b = 4, f(n) = n\lg n \quad \Rightarrow \quad n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$$

$$\Rightarrow \quad f(n) = \Omega(n^{(\log_4 3)+\varepsilon}), \text{where } \varepsilon \approx 0.2, \text{ and for sufficiently large } n,$$

$$af(n/b) = 3(n/4)\lg(n/4) \le (3/4)n\lg n = cf(n) \text{ for } c = 3/4$$

$$\Rightarrow \quad T(n) = \Theta(n\lg n)$$

Chapter 5 rand

Chapter 6 Heapsort

Chapter 7 Quicksort

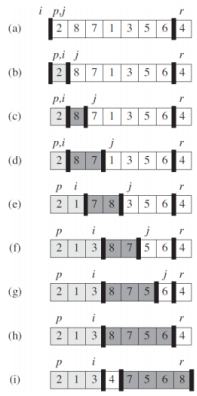
- 算法流程：

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```


```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

$A[p \sim i]$, $\le A[r]$

$A[i+1 \sim r-1]$, $> A[r]$

swap($A[i+1]$, $A[r]$), when termination $0 \le i \le r-1$

分区算法 PARTITION 执行过程中，i 之前（含）的元素小于等于哨兵（标记） $A[r]$， $i+1$ 及其之后的元素大于 $A[r]$。即 $p \sim i$ 的元素小于等于 $A[r]$，其后元素 $A[i+1] \sim A[r-1]$ 比 $A[r]$大。

最后，交换$A[i+1]$与$A[r]$，以保证$A[r]$ 之前的元素都比它小，之后的元素比它大。

- 为什么快排快：

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2      q = RANDOMIZED-PARTITION(A, p, r)
3      RANDOMIZED-QUICKSORT(A, p, q − 1)
4      RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

Intuitively…

第 1 次分区，定位好 1 个元素

第 2 次分区，又定位好 2 个元素（已定位 2^2-1 个）

第 3 次分区，又定位好 4 个元素（已定位 2^3-1 个）

……

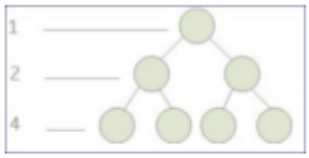
第 k 次分区，又定位好 2^{k-1} 个元素（共定位 2^k-1 个）

2^k-1 = n => k = lg(n+1)
每次分区有最多 n-1 次比较 ⇨ $O(n\lg n)$

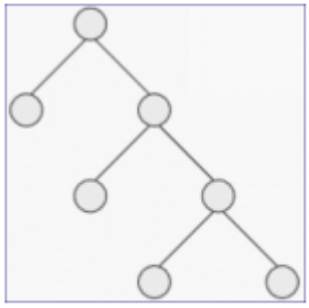
Chapter 8 Sorting in Linear Time

- 满二叉树：

“中国版”
的满二叉树



“外国版”
的满二叉树
（结点要么
叶子，要么
有两个孩子）



- 顺序统计量：

The ith **order statistic** of a set of n elements is the ith smallest element.

✓ the **minimum** of a set of elements is the first order statistic (i = 1).

✓ the **maximum** is the nth order statistic (i = n).

✓ A **median**, informally, is the “halfway point” of the set.

6	12	5	9	2	10	8	7

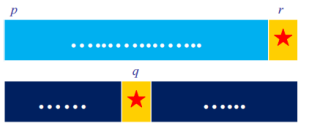
Minimum: 2 （1阶顺序统计量）
Maximum: 12 （n阶顺序统计量）

- 期望线性时间求第k小：

- The general selection problem appears more difficult than the simple problem of finding a minimum.
- Yet, surprisingly, the asymptotic running time for both problems is the same: $\Theta(n)$.

求第 i 小的元素　　Firstly, p is 1, r is n

```
RANDOMIZED-SELECT(A, p, r, i)
1    if p == r
2        return A[p]
3    q = RANDOMIZED-PARTITION(A, p, r)
4    k = q - p + 1
5    if i == k   // the pivot value is the answer
6        return A[q]
7    elseif i < k
8        return RANDOMIZED-SELECT(A, p, q-1, i)
9    else return RANDOMIZED-SELECT(A, q+1, r, i-k)
```



// 随机分区，找到 A 中的第 k 小的元素 $A[q]$

// 左边的 k-1 个元素比 $A[q]$ 小，第 k 大的元素是 $A[q]$

$$T(n) = T(\max(k\text{-}1, n\text{-}k)) + O(n)$$

- **Worst-case running time**

  $T(n) = T(n\text{-}1) + O(n)$,

  $\Theta(n^2)$

- **A special case**

  $q = (r\text{-}p)/2$, then

  $T(n) = T(n/2) + O(n)$,

  $\Theta(n)$

- **Expected running time ?**

  Indicator random variables, $\Theta(n)$? *

1.选择一个基准

从 数组 中选择一个元素作为基准。这个选择可以是随机的，也可以是第一个元素、最后一个元素或中间的某个元素。

2.分区

根据基准元素将数组重新排列，使得所有小于基准的元素都位于基准的左侧，所有大于或等于基准的元素都位于基准的右侧。这个过程与 快速排序 中的分区步骤相同。

3.递归

根据分区结果，确定基准元素的位置。

如果基准元素的位置正好是k-1（因为数组索引通常从0开始），则基准元素就是第k小的数。

如果基准元素的位置小于k-1，则第k小的数位于基准元素的右侧子数组中，继续在右侧子数组中递归查找。

如果基准元素的位置大于k-1，则第k小的数位于基准元素的左侧子数组中，继续在左侧子数组中递归查找，并且k值要减去左侧子数组的大小加1（因为左侧子数组中的元素都不可能是第k小的数）。

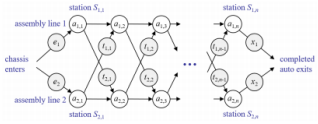
Chapter 9 sort order

略

Chapter 15 dp

- 最优子结构：动态规划重要特点之一

Optimal substructure（最优子结构）



- An optimal solution to a problem (assembly-line scheduling, finding the fastest way through station $S_{i,j}$) contains within it an optimal solution to subproblems (finding the fastest way through either $S_{1,j-1}$ or $S_{2,j-1}$).
  问题的最优解包含其子问题的最优解，p_j contains p_{j-1} within it.
- Optimal substructure is one of the hallmarks of the applicability of dynamic programming.
  最佳子结构是动态规划法的重要特点之一。

- **n个点的BST节点数：**

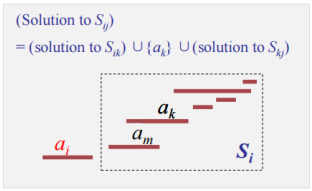
  - The # of BST with n nodes is $\Omega(4^n/n^{3/2})$ （Problem 12-4）.

Chapter 16 Greedy Algorithm

- 最优子结构：

  No general way to tell if a greedy algorithm is optimal, but two key ingredients.
  没有一般化的规则来说明贪心算法是否最优，但有两个基本要点

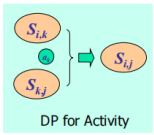
  1. greedy-choice property 贪心算法属性
  2. optimal substructure 最优子结构

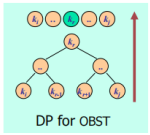


子问题的最优解 + 贪心选择 ⇐ 原问题的最优解

- 贪心和**dp**的区别：

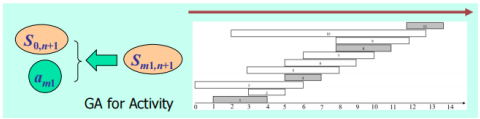
  Dynamic programming
  - 依赖于已知子问题的最优解再作出选择（先解子问题）
  - 求解步骤：bottom-up





  Greedy
  - 直接选择，留更小的子问题进行求解
  - 求解步骤：top-down

  证明子问题的最优解和贪婪选择 ⇒ 原问题的最优解



- 分数背包和**01**背包的区别：
  - 分数背包问题有贪心选择属性
  - 0-1背包问题 无贪心选择属性
  - 求解分数背包问题，先按物品的单位价值 v_i/w_i 降序排序 ($v_i/w_i \geq v_{i+1}/w_{i+1}$ for all i)
  - Time: 排序 $O(n\lg n)$, 贪心选择 $O(n)$

- **Huffman**编码：

An important application: the design of data compression.
Huffman codes, **哈夫曼编码**

1951年，哈夫曼在MIT读博士，导师Robert M. Fano给出一个学期报告的题目“寻找最有效的二进制编码”。哈夫曼发现了基于有序频率二叉树编码的想法，并证明了这个方法是最有效的。哈夫曼使用自底向上的方法构建二叉树，避免了次优算法Shannon-Fano编码的弊端——自顶向下构建树。1952年，哈夫曼发表了《一种构建极小多余编码的方法》，后人称之为Huffman编码。

A method for the construction of minimum-redundancy codes
DA Huffman - Proceedings of the IRE, 1952 - ieeexplore.ieee.org
… In order to avoidthe use of the lengthy term "minimumredundancy," this term will be replaced here by "optimum." Itwill be understood then that, in this paper, "optimum code"means …
☆ 保存 99 引用 被引用次数：10212 相关文章 所有 9 个版本

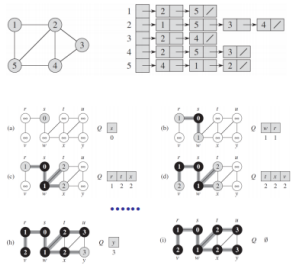
戴维·霍夫曼（David Albert Huffman, 1925-1999），1999年10月17日因癌症去世，享年74岁。但他作为信息论的先驱，对计算机科学、通信等学科所作出的巨大贡献将永远为世人所铭记。
霍夫曼在MIT一直工作到1967年。之后他转入加州大学Santa Cruz分校，是该校计算机科学系的创始人，1970—1973年任系主任。1994年霍夫曼退休。
除了霍夫曼编码外，霍夫曼在其他方面也还有不少创造，比如他设计的二叉最优搜索树算法就被认为是同类算法中效率最高的，因而被命名为霍夫曼算法。

Chapter 22 图算法基础

- 邻接表 & 邻接矩阵空间复杂度：
  - Memory
    - *adjacency-list (AL)* : $\Theta(V+E)$
    - *adjacency-matrix (AM)* : $\Theta(V^2)$
  - *AL*：对不太稠密的图，省空间；搜索高效；方便存储辅助信息（顶点的颜色、访问标记、父节点、子节点、..）
  - *AM*：操作简单
  - 算法具体实现，依赖于采用的数据结构、依赖于采用的程序设计语言。
  - 如果不用STL（Standard Template Library），自己用C语言写，对链表和矩阵运算等操作需要掌握得比较熟练（程序设计和数据结构的基础要求高）。
- **BFS/DFS时间复杂度 & 聚集分析：**

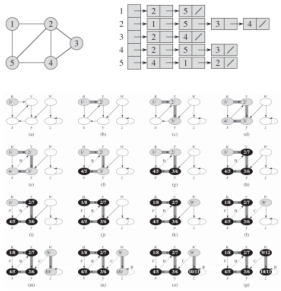
Aggregate analysis: $O(V+E)$

聚集分析：每个顶点入队和出队各一次、每条边检测一次



```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

Aggregate analysis, 聚集分析：顶点为白色时，才会作为递归子树的“源点”进行递归调用（调用后变为非白色，不可逆）；对每个顶点，其邻接表的每条边遍历一次。顶点数 V，所有边数 E。



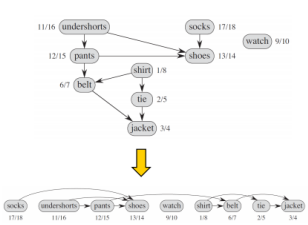
$O(V+E)$

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK
 9  time = time + 1
10  u.f = time
```

- **Topo应用：**

- 许多应用程序使用dag来指示事件之间的优先级。
- 一些应用：最短路径（先拓扑排序，然后需要对每一条边进行松弛操作，后文介绍），并行计算，...



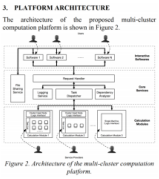


Figure 2. Architecture of the multi-cluster computation platform.

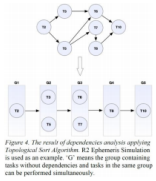


Figure 4. The result of dependencies analysis applying Topological Sort Algorithm. R2 Ephemeris Simulation is used as an example. 'G' means the group containing tasks without dependencies and tasks in the same group can be performed simultaneously.

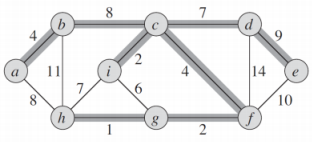
You Song, etc., A Microservice-based Multi-cluster Computation Platform For Space Mission Design, 8th International Systems & Concurrent Engineering for Space Applications Conference, Glasgow, UK, 2018

Chapter 23 最小生成树

- **Kruskal & Prim：**

  Kruskal：只要针对**边**，对于**稀疏图**很有优势

  Prim：针对**稠密图**很有优势



Application: Electronic circuit designs （电子电路设计）

Algorithms: Kruskal , Prim （贪心算法的经典应用）

- Kruskal（**鲁斯卡尔**）算法主要是针对边展开，边数少时效率会非常高，对稀疏图有很大的优势
- Prim （**普里姆**）算法对于稠密图，即边数非常多的情况会好一些

Chapter 24 单源最短路

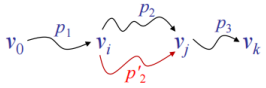
- 无权图最短路：BFS

BFS可用于求解一个无权图的最短路径

- 最优子结构属性：
  - Optimal substructure: Shortest-paths algorithms typically rely on the property that a shortest path between two vertices contains other shortest paths within it.
    最短路径问题具有最优子结构性质
  - **Lemma 24.1** (Subpaths of shortest paths are shortest paths)
    Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \to \mathrm{R}$, let $p = < v_0 , v_1 , \ldots , v_k >$ be a shortest path from vertex v_0 to vertex v_k and, for any i and j such that $0 \le i \le j \le k$, let $p_{ij} = < v_i , v_{i+1} , \ldots , v_j >$ be the subpath of p from vertex v_i to vertex v_j . Then, p_{ij} is a shortest path from v_i to v_j .

    Proof ...

    $v_0 \overset{p_{0i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$

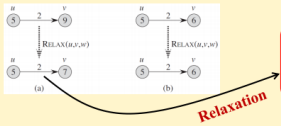


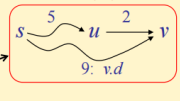
$p = p_1 p_2 p_3$ is shortest, that is, $p = \delta(v_0 , v_k)$ ⇨ p_2 is shortest, that is, $p_2 = \delta(v_i , v_j)$

- 松弛操作：

Relaxation (actually Tight): 对每条边 (u, v) 执行松弛操作，能让源点 s 到 v 的距离 $v.d$ 减少（不增）。

最短路径问题最重要的操作！






```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

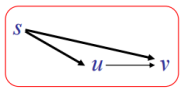
```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

这个步骤实际上为“收紧”过程，称为“松弛”过程，是延续历史的叫法。正如哈夫曼编码里的“前缀码”，实际上应为“前缀无关码”。本章的**每个算法都会调用INITIALIZE-SINGLE-SOURCE，然后反复对每一条边执行relax操作。**

- 最短路径和松弛的性质：

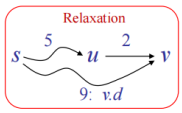
  最短路径和松弛的性质：

  - Triangle inequality，三角不等式 (Lemma 24.10)
    For any edge $(u, v) \in E$, we have $\delta(s, v) \le \delta(s, u) + w(u, v)$.

  - Upper-bound property，上界属性 (Lemma 24.11)
    We always have $v.d \ge \delta(s, v)$ for all vertices $v \in V$, and once $v.d$ achieves the value $\delta(s, v)$, it never changes.

  - Convergence property，收敛属性 (Lemma 24.14)
    If $s \rightsquigarrow u \rightarrow v$ is a shortest path in G for some $(u, v) \in V$, and if $u.d = \delta(s, u)$ at any time prior to relaxing edge (u, v), then $v.d = \delta(s, v)$ at all times afterward.
    假设 $s \rightsquigarrow u \rightarrow v$ 为一条最短路径，如果对边(u, v) 进行松弛前，$u.d$ 达到最短路径，即 $u.d = \delta(s, u)$，则对边(u, v) 进行松弛后，$v.d$ 达到最短路径（即最短路径已经求出），即 $v.d = \delta(s, v)$

- **Bellman-Ford算法**：BF算法

  允许图有负权和环路

  时间复杂度：$O(VE)$

BF algorithm solves the single-source shortest-paths problem in the **general case** in which edge weights may be negative, cycle may exists.

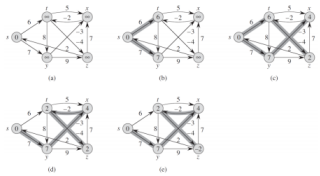
BF算法，通用的算法（允许图有负边和环路）：对顶点进行遍历，针对每个顶点，对所有的边进行松弛操作。

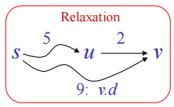
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1   //遍历顶点
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)   //松弛边
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

Running Time?

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```





BF算法，通用的算法（允许图有负边和环路）：对顶点进行遍历，针对每个顶点，对所有的边进行松弛操作。

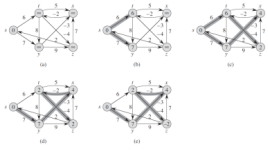
Running Time: 对每个顶点（共 $|V|$-1 个顶点，第 2 行的遍历），所有边都执行一次松弛操作（共 E 条边，第 3 行的遍历），因此，执行时间为 $O(VE)$.

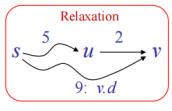
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1   //遍历顶点
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)   //松弛边
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

检测到有负环路（可以把图中(t, x)的权值改为1，此时有负环路，验证该算法的错误），此时不存在最短路径。

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```





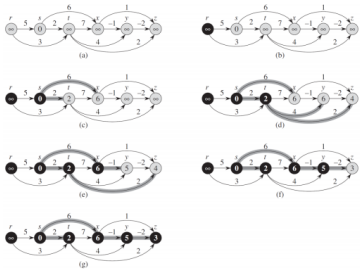
- **基于Topo排序的算法**：TS算法

  只能针对有向无环图（可以有负权）

  时间复杂度：$O(V + E)$

By relaxing the edges of a weighted dag (directed acyclic graph) $G = (V, E)$, (supports negative edges, but acyclic), according to a topological sort of its vertices.

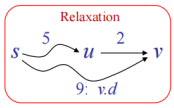
考虑有向无环路图（可以有负边）这类特别情况，先对顶点进行拓扑排序，然后依序对边进行松弛。



```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u, taken in topologically sorted order
4      for each vertex v ∈ G.Adj[u]
5          RELAX(u, v, w)
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
1  for each vertex v ∈ G.V
2      v.d = ∞
3      v.π = NIL
4  s.d = 0
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```



- **Dijkstra算法 & goto有害论：**

  只能针对无负环有向图

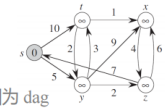
  采用的贪心策略

  时间复杂度：$O(V + E\lg V)$

The running time of DJ depends on how we implement the min-priority queue. It may be

$$O(V^2)\,,\ O(V + E\lg V)\,,\ \text{or}\ O(V\lg V + E)$$

- **Dijkstra** 算法：能求解包括环路的有向图（但要求边是非负的）的最短路径问题。
- 24.2 Topo可解决含“负边”的最短路问题，但要求图为 dag
- 24.1 BF方法可解决包括“负边”和“环路”的最短路问题



Edsger Wybe Dijkstra
1930/5/11-2002/8/6
1972年图灵奖获得者

① 提出“goto有害论”
② 提出信号量和PV原语
③ 解决了“哲学家聚餐”问题
④ 最短路径算法和银行家算法
⑤ Algol 60的设计者和实现者
⑥ THE操作系统的设计者和开发者

与Knuth并称为我们这个时代最伟大的计算机科学家的人

A note on two problems in connexion with graphs
EW Dijkstra
Edsger Wybe Dijkstra: his life, work, and legacy, 2022 · dl.acm.org

We start the construction by choosing an arbitrary node as the only member of set A, and by placing all branches that end in this node in set II. To start with, set I is empty. From then onwards we perform the following two steps repeatedly. Step 1. The shortest branch of set II is removed from this set and added to set I. As a result one node is transferred from set B to set A.

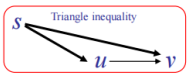
ACM Digital Library

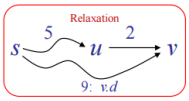
☆ 保存 99 引用 被引用次数: 36124 相关文章 所有 39 个版本

- 总结：聚集分析

Summary

- 单源最短路径问题的关键技术
  - 三角不等式
  - 松弛原理
  - 最小优先队列
  - DFS and BFS
  - 拓扑排序





- 三个算法
  - Bellman-Ford(BF) 算法（通用，可包括负边和环路）
  - 基于拓扑排序的BF算法（可包括负边，但须是有向无环路图dag）
  - Dijkstra(DJ) 算法（贪心策略，最小优先队列）（可环路，但不能有负边）
- 复杂度分析：聚集分析

方法	负边	环路
BF	√	√
topo sort	√	
Dijkstra		√

	环路	非环路
负边	BF	topo sort BF
非负边	Dijkstra BF	DJ topo sort BF

Chapter 25 Floyd

- **Floyd-warshall** 布尔矩阵的传递闭包：

  时间复杂度：$O(n^3)$

The structure of a shortest path

- **The Floyd-Warshall** algorithm considers the intermediate vertices of a shortest path, where an *intermediate* vertex of a simple path $p = \langle v_1, v_2, \ldots, v_l \rangle$ is any vertex of p other than v_1 or v_l, that is, any vertex in the set $\{v_2, v_3, \ldots, v_{l-1}\}$.

  简单路径 p 的端点是 v_1 和 v_l，其他点是 p 的“之间”顶点

Floyd-warshall 来源于floyd，其原理基于warshall提出的基于布尔矩阵的传递闭包。

[PDF] Algorithm 97: shortest path
RW Floyd - Communications of the ACM, 1962 - dl.acm.org
¢ onlment This procedure will perform different order arithmetic operations with b and c, putting the result in a. The order of the operation is given by op. For op= 1 addition is performed. …
☆ 保存 99 引用 被引用次数: 6015 相关文章 所有 3 个版本

A theorem on boolean matrices
S Warshall - Journal of the ACM (JACM), 1962 - dl.acm.org
… Given two boolean matrices A arid B, we define the boolean product AAB as that matrix whose (i, j)th entry is vk(a~/, A bki). We define tile boolean sum AVB as that matrix whose (i, j)th …
☆ 保存 99 引用 被引用次数: 2563 相关文章 所有 7 个版本

- 算法思路：基于dp

  即规定 i, j 之间可以经过哪些顶点

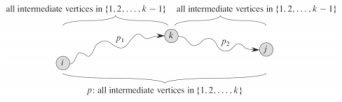
p 是从 i 到 j 的最短路径 $st\text{-}path\text{-}\delta(i,j)$，“之间”顶点 from $\{1, 2, \ldots, k\}$：

- 如果 k 不是路径 p 上的顶点，则 $st\text{-}path\text{-}\delta(i,j)$ 的之间顶点来自于 $\{1, 2, \ldots, k\text{-}1\}$，即，之间顶点 from $\{1, 2, \ldots, k\}$ 的 $st\text{-}path\text{-}\delta(i,j)$ 就是之间顶点 from $\{1, 2, \ldots, k\text{-}1\}$ 的 $st\text{-}path\text{-}\delta(i,j)$；
- 如果 k 是路径 p 上的顶点，则 $st\text{-}path\text{-}\delta(i,j)$ 由两部分构成 $i \xrightarrow{p_1} k \xrightarrow{p_2} j$，其中 p_1 是之间顶点来自于 $\{1, 2, \ldots, k\text{-}1\}$ 的 $st\text{-}path\text{-}\delta(i,k)$，$p_2$ 同理。

A recursive solution to the all-pairs shortest-paths problem

Let $d_{ij}^{(k)}$ be the weight of a $st\text{-}path\text{-}\delta(i,j)$ for which all intermediate vertices are in the set $\{1, 2, \ldots, k\}$. When $k = 0$, a $path\text{-}p(i,j)$ has no intermediate vertices at all. Such a path has at most one edge. We define recursively

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1. \end{cases}$$



Because for any path, all intermediate vertices are in the set $\{1, 2, \ldots, n\}$, the matrix $D^{(n)} = (d_{ij}^{(n)})$ gives the final answer: $d_{ij}^{(n)} = \delta(i,j)$ for all $i, j \in V$.

p 是从 i 到 j 的最短路径 $st\text{-}path\text{-}\delta(i,j)$，$p$ 的“之间”顶点 from $\{1, 2, \ldots, k\}$，其长度（权值）为 $d_{ij}^{(k)}$：

- 如果 k 不是路径 p 上的顶点，则 $st\text{-}path\text{-}\delta(i,j)$ 的之间顶点来自于 $\{1, 2, \ldots, k\text{-}1\}$，即，之间顶点 from $\{1, 2, \ldots, k\}$ 的 $st\text{-}path\text{-}\delta(i,j)$ 就是之间顶点 from $\{1, 2, \ldots, k\text{-}1\}$ 的 $st\text{-}path\text{-}\delta(i,j)$；
- 如果 k 是路径 p 上的顶点，则 $st\text{-}path\text{-}\delta(i,j)$ 由两部分构成 $i \xrightarrow{p_1} k \xrightarrow{p_2} j$，其中 p_1 是之间顶点来自于 $\{1, 2, \ldots, k\text{-}1\}$ 的 $st\text{-}path\text{-}\delta(i,k)$，$p_2$ 同理。

- **Floyd**全局最短路的路径构造：

  不断缩小，可以 $O(n)$ 求得最短路径

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1. \end{cases}$$

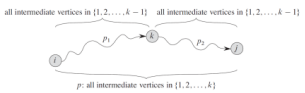
Constructing a shortest path （构造最短路径）

We compute a sequence of matrices $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$, where $\Pi = \Pi^{(n)}$ and we define $\pi_{ij}^{(k)}$ as the predecessor of vertex j on a shortest path from vertex i with all intermediate vertices in the set $\{1, 2, \ldots, k\}$.

$\pi_{ij}^{(k)}$ 表示从 i 到 j 的最短路径（之间顶点 from $\{1, 2, \ldots, k\}$）中 j 的前驱节点

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$



对最短路径 $i \to j$，
(1) k 不在最短路径上，最短路径的之间顶点是 $\{1, 2, \ldots, k\text{-}1\}$，因此 $\pi_{ij}^{(k)} = \pi_{ij}^{(k-1)}$
(2) k 在最短路径上，$i \to j$ 中 j 的前驱 $\pi_{ij}^{(k)}$ 显然就是的 $k \to j$ 中 j 的前驱 $\pi_{kj}^{(k-1)}$

Chapter 26 最大流

- 最大流：又称为网络的最大流量问题，或最小分割问题。

  - **Capacity**: a maximum rate at which the material can flow through the conduit.
    容量：管道能通过的最大 rate（单位时间的容量）
  - **Flow conservation**: the rate at which material enters a vertex must equal the rate at which it leaves the vertex.
    流守恒：流入一个顶点 material = 流出该顶点的 material
  - **Maximum-flow problem**: we wish to compute the greatest rate at which we can ship material from the source to the sink without violating any capacity constraints.
    最大流问题：在容量允许的情况，从源点到汇点能 ship 的最大容量（单位时间情况下，即 rate）

- **Ford-Fulkerson**方法：福特-福克森方法

时间复杂度：$O(E \cdot f^*)$，f^* 为最大流

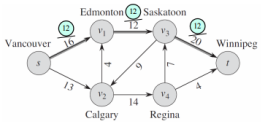
流过大时，F-F算法失效

残留容量：双向的，反向流可以抵消正向流

A "method" rather than an "algorithm". （福特-福克森方法，由Ford 和Fulkerson于1956年提出的方法）

The Ford-Fulkerson method depends on three important ideas:

- residual networks（剩余网络（残留网络、残差网络），核心思想：存在一些边，在其上还能增加额外流，这些边就构成了残留网络的增广路径【增益路径】）
- augmenting paths（增益路径，增广路径）
- cuts（割、分割、切割）




```
FORD-FULKERSON-METHOD(G, s, t)
1  initialize flow f to 0
2  while there exists an augmenting path p in the residual network G_f
3        augment flow f along p
4  return f
```

Residual networks

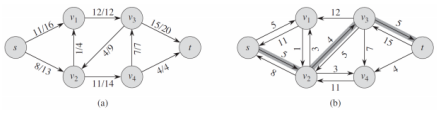
residual capacity

（残留容量：是双向的）

边上还能增加的额外流

反向流最多抵消正向流

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E, \\ f(v,u) & \text{if } (v,u) \in E, \\ 0 & \text{otherwise.} \end{cases}$$



Example: let $u \leftarrow s$, $v \leftarrow v_1$, there are $c(u, v) = 16$ and $f(u, v) = 11$, then we can increase $f(u, v)$ by up to $c_f(u, v) = 5$ units before we exceed the capacity constraint on edge (u, v). We also wish to allow an algorithm to return up to 11 units of flow from v to u, and hence $c_f(v, u) = 11$.

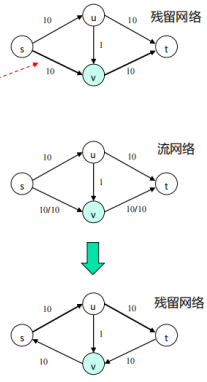
- **E-K算法：**（Edmonds-karp算法）

  在F-F算法中使用**BFS**寻找增广路径

  时间复杂度：$O(V \cdot E^2)$

  （E-K算法实际就是F-F算法的一种改进（或者一种具体实现），因此，F-F称为方法

```
EDMONDS-KARP(G, s, t)                                   O(V·E²)
1  for each edge (u, v) ∈ E
2      f[u, v] ← 0
3  while there exists a path p from s to t in the residual network G_f (using BFS)
4      c_f(p) ← min {c_f(u, v) : (u, v) is in p}
5      for each edge (u, v) in p
6          if (u, v) ∈ E
7              f[u, v] ← f[u, v] + c_f(p)
8          else f[v, u] ← f[v, u] - c_f(p)
```



*证明思想：关键边（增广路径 p 上的最小容量边）。

沿着 p 增加流一次，关键边消失；边 (u, v) 最多$O(V)$次作为关键边（源点 s 到顶点 v 的最短路径随着流增加而单调增加）；共E条边；E-K算法执行中的关键边数量 $O(V E)$（关键边全部消失，不再有增广路径，最大流找到）。每次找增广路径和给边增加流的操作，时间为 $O(E)$。总时间 $O(V E^2)$。

该算法最初由 Yefim Dinitz 于1970年发表，并由 Jack Edmonds 和Richard Karp 于1972年独立发表。E-K算法实际就是F-F算法的一种改进（或一种具体实现），因此，F-F称为方法!

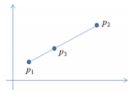
Chapter 33 计算几何

- 直线的凸组合：

- A ***convex combination***(凸组合) of two distinct points $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ is any point $p_3 = (x_3, y_3)$ such that for some α in the range $0 \le \alpha \le 1$, we have

  $x_3 = \alpha x_1 + (1 - \alpha) x_2$ and $y_3 = \alpha y_1 + (1 - \alpha) y_2$.

  We also write that $p_3 = \alpha p_1 + (1 - \alpha) p_2$.



- p_3 is any point that is on the line passing through p_1 and p_2 and is on or between p_1 and p_2 on the line.

两个点 p_1 和 p_2 的凸组合 p_3 是线段 p_1p_2 上的任意一个点（含端点）。

- 凸包算法：

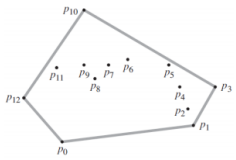
  Graham：$O(n \lg n)$，找到最下边/左边的点，按相对极角排序

  Jarvis：$O(nh)$，h为凸包上的点数，也是找极角最小

  Andrew算法为Graham的一个变种，每次求一个半壳

  Some algorithms that compute the convex hull of a set of n points:

  - Graham's scan (格雷厄姆), runs in $O(n\lg n)$ time
  - Jarvis's march (贾维斯), runs in $O(nh)$ time, where h is the number of vertices of the convex hull.
  - Additional several methods
    - *incremental method*, $O(n\lg n)$
      增量法（每次增加一点，更新当前凸包）
    - *divide-and-conquer method*, $O(n\lg n)$
    - *prune-and-search method*, $O(n\lg h)$
      剪枝-搜索法：先找凸包上部分，然后找下部分



Chapter 30 FFT

> *采用了分而治之的思想*

- 傅里叶变换：

让·巴普蒂斯·约瑟夫·傅里叶（Baron Jean Baptiste Joseph Fourier，1768-1830），男爵，法国数学家、物理学家，1768年3月21日生于欧塞尔，1830年5月16日卒于巴黎。
1817年当选为科学院院士。主要贡献是在研究《热的传播》和《热的分析理论》时创立了一套数学理论，对19世纪的数学和物理学的发展都产生了深远影响。

$s(t) = \sin(60\pi t) + \sin(30\pi t)$?

$s(t) = \sin(60\pi t) + \sin(30\pi t) + \sin(90\pi t)$?

$$S(\omega) = \int e^{-i\omega t} s(t) \mathrm{d}t$$

- FT的基本思想首先由傅里叶提出，所以以其名字来命名以示纪念。傅里叶变换是一种特殊的积分变换。它能将满足一定条件的某个函数表示成正弦基函数的线性组合或者积分。在不同的研究领域，傅里叶变换具有多种不同的变体形式，如连续傅里叶变换和离散傅里叶变换。
- 数学上看，用简单表示来对复杂函数的深入研究。正弦函数在物理上是被充分研究而相对简单的函数类，这一想法跟化学上的原子论想法何其相似！奇妙的是，现代数学发现傅里叶变换具有非常好的性质，使得它如此的好用和有用，让人不得不感叹造物的神奇。
- 哲学上看，"分析主义"和"还原主义"，就是要通过对事物内部适当的分析达到增进对其本质理解的目的。比如近代原子论试图把世界上所有物质的本源分析为原子，而原子不过数百种而已，相对物质世界的无限丰富，这种分析和分类无疑为认识事物的各种性质提供了很好的手段。

- 秦九韶算法：

秦九韶（1208年 - 1268年），汉族，出生于普州（今四川安岳县）。南宋著名数学家，与李冶、杨辉、朱世杰并称宋元数学四大家。

- A *coefficient representation* of a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$

  $a = (a_0, a_1, \cdots, a_{n-1})^T$, column vectors

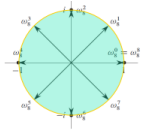
秦九韶（1208年 - 1268年），汉族，出生于普州（今四川安岳县），南宋著名数学家，与李冶、杨辉、朱世杰并称宋元数学四大家。

- Convenient for certain operations. For example,
  - *evaluating* $A(x)$ at a given point x_0, takes time $\Theta(n)$ using *Horner's rule*:

    $A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x_0(a_{n-2} + x_0(a_{n-1}))))$.　　秦九韶算法
  - *adding* two polynomials represented $a = (a_0, a_1, \cdots, a_{n-1})^T$ and $b = (b_0, b_1, \cdots, b_{n-1})^T$, takes $\Theta(n)$ time: we just produce the coefficient vector $c = (c_0, c_1, \cdots, c_{n-1})$, where $c_j = a_j + b_j$ for $j = 0, 1, \cdots, n-1$.

- 群论：
  - $\omega_n = e^{2\pi i/n}$: principal nth root of unity 〔单位 n 次复根的基元〕
  - $e^{iu} = \cos(u) + i\sin(u)$ 〔欧拉公式〕
    the exponential of a complex number

    $\omega_n^k = e^{2\pi ik/n} = \cos(2\pi k/n) + i\sin(2\pi k/n)$,
    $k = 0, 1, \cdots, n-1$
  - All of the other complex nth roots of unity are powers of ω_n. The n complex nth roots of unity $\{\omega_n^k, k = 0, \cdots, n-1\}$ form a group.



群的快读
集合S以及定义在S上的运算，满足性质：
①封闭性，②单位元，③结合律，④逆元。
如，整数及加法构成一个群。
进一步定义，交换群，有限群等。

埃瓦里斯特·伽罗瓦（1811年10月25日-1832年5月31日），1811年10月25日生，法国数学家。现代数学中的分支学科群论的创立者。用群论彻底解决了根式求解代数方程的问题，而且由此发展了一整套关于群和域的理论，人们称之为伽罗瓦理论，并把其创造的“群”叫作伽罗瓦群（Galois Group）。在世时在数学上研究成果的重要意义没被人们所认识，曾呈送科学院3篇学术论文，均被退回或遗失。后转向政治，支持共和党，曾两次被捕。21岁时死于一次决斗。
个人成就：使用群论的想法去讨论方程式的可解性，整套想法现称为伽罗瓦理论，是当代代数与数论的基本支柱之一。他系统化地阐释了为何五次以上之方程式没有公式解，而四次以下有公式解。他解决了古代三大尺规作图问题中的两个：“三等分任意角不可能”，“倍立方不可能”。

- 最美丽的公式 —— 欧拉公式：

最美丽的公式：**欧拉公式**

$e^{iu} = \cos(u) + i\sin(u)$

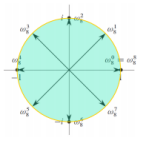
$$e^{i\pi} + 1 = 0$$

理由如下：
1. 最重要的自然常数的 e 含于其中。自然对数的底，大到飞船的速度，小至蜗牛的螺线，谁能够离开它？
2. 最重要的常数 π 含于其中。世界上最完美的平面对称图形是圆。“最伟大的公式”能够离开圆周率吗？（还有 π 和 e 是两个最重要的无理数！）
3. 最重要的运算符号 + 含于其中。加号最重要：因为其余符号都是由加号派生而来，减号是加法的逆运算，乘法是累计的加法……
4. 最重要的两个元在里面。零元 0，单位 1，是构造群，环，域的基本元素。如果你看了有关《近世代数》的书，你就会体会到它的重要性。
5. 最重要的虚单位 i 也在其中。虚单位 i 使数轴上的问题扩展到了平面，而在哈密尔的 4 元数与 凯莱的 8 元数中也离开不了它。
6. 这个公式的美在于其精简。她没有多余的字符，却联系着几乎所有的数学知识。有了加号，可以得到其余运算符号；有了 0，1，就可以得到其他的数字；有了 π 就有了圆函数，也就是三角函数；有了 i 就有了虚数，平面向量与其对应，也就有了哈密尔的 4 元数，现实的空间与其对应；有了 e 就有了微积分，就有了和工业革命时期相适宜的数学。

- **DFT的意义：**

  多项式在特殊点的取值（单位复根）

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \quad \begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ & & \cdots\cdots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix} = V(x_0, x_1, \cdots, x_{n-1}) \cdot a \quad (30.4)$$



- wish to evaluate a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ at $x = \omega_n^0, \omega_n^1, \omega_n^2, \cdots, \omega_n^{n-1}$
- without loss of generality, assume that $n = 2^m$, if not, let $a_{n+k} = 0$
- Discrete Fourier Transform (DFT): let A is given in coefficient form: $a = (a_0, a_1, \cdots, a_{n-1})^T$, let $x_k = \omega_n^k$, define y_k, for $k = 0, 1, \cdots, n-1$, by

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj}, \quad \begin{pmatrix} y_0 \\ y_1 \\ \cdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ & & \cdots\cdots & & \\ 1 & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \cdots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \cdots \\ a_{n-1} \end{pmatrix}$$

$y = \mathrm{DFT}_n(a)$：数学意义上，多项式在特殊点的取值（单位复根）

- 蝴蝶变换其实是几个稀疏矩阵的乘积：

$$\omega = e^{2\pi i/8}$$

$$\begin{pmatrix} y_0 \\ y_4 \\ y_2 \\ y_6 \\ y_1 \\ y_5 \\ y_3 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{4\cdot1} & \omega_n^{4\cdot2} & \cdots & \omega_n^{4(n-1)} \\ 1 & \omega_n^{2\cdot1} & \omega_n^{2\cdot2} & \cdots & \omega_n^{2(n-1)} \\ 1 & \omega_n^{6\cdot1} & \omega_n^{6\cdot2} & \cdots & \omega_n^{6(n-1)} \\ 1 & \omega_n^{1\cdot1} & \omega_n^{1\cdot2} & \cdots & \omega_n^{1(n-1)} \\ 1 & \omega_n^{5\cdot1} & \omega_n^{5\cdot2} & \cdots & \omega_n^{5(n-1)} \\ 1 & \omega_n^{3\cdot1} & \omega_n^{3\cdot2} & \cdots & \omega_n^{3(n-1)} \\ 1 & \omega_n^{7\cdot1} & \omega_n^{7\cdot2} & \cdots & \omega_n^{7(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \omega^0 & & & & & & \\ 1 & \omega^4 & & & & & & \\ & & 1 & \omega^2 & & & & \\ & & 1 & \omega^6 & & & & \\ & & & & 1 & \omega^1 & & \\ & & & & 1 & \omega^5 & & \\ & & & & & & 1 & \omega^3 \\ & & & & & & 1 & \omega^7 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & \omega^0 & 0 & & & & \\ 0 & 1 & 0 & \omega^0 & & & & \\ 1 & 0 & \omega^4 & 0 & & & & \\ 0 & 1 & 0 & \omega^4 & & & & \\ & & & & 1 & 0 & \omega^2 & 0 \\ & & & & 0 & 1 & 0 & \omega^2 \\ & & & & 1 & 0 & \omega^6 & 0 \\ & & & & 0 & 1 & 0 & \omega^6 \end{pmatrix} \cdot \begin{pmatrix} 1 & & & & \omega^0 & & & \\ & 1 & & & & \omega^0 & & \\ & & 1 & & & & \omega^0 & \\ & & & 1 & & & & \omega^0 \\ 1 & & & & \omega^4 & & & \\ & 1 & & & & \omega^4 & & \\ & & 1 & & & & \omega^4 & \\ & & & 1 & & & & \omega^4 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

Ex_GCD 扩展欧几里得算法

- 给定 2 个不全为零的整数 a, b，令 gcd(a, b) 表示整数 a 和 b 的最大公约数
- x|y 表示 x 可以整除 y ，即 y 是 x 的倍数
- 扩展欧几里得算法用于快速求解 gcd(a, b)，同时可以计算出 x_0 和 y_0，使得 $ax_0 + by_0 = gcd(a, b)$

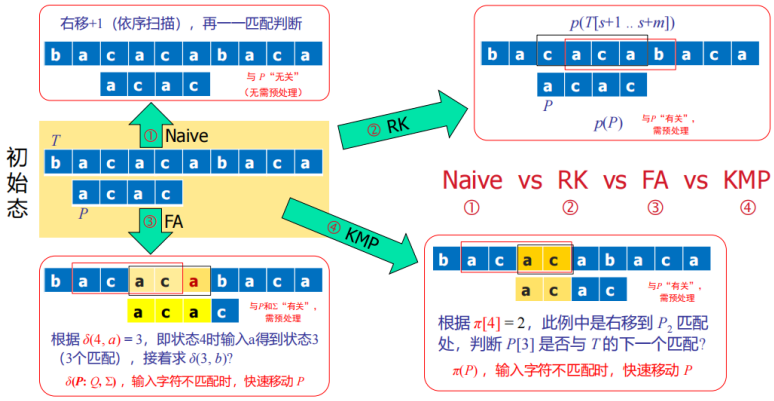
- **裴蜀定理**
  - 结论一：设 a, b 是不全为零的整数，对任意整数 x, y，满足 ax+by 能被 gcd(a,b) 整除，即 gcd(a, b) | ax + by
    (即， 0 < gcd(a, b) ≤ ax + by ；即， k*gcd(a, b) = ax + by，其中1 ≤ k)
  - 结论二：存在整数 x, y，使得 gcd(a, b) = ax + by
    (即，gcd(a, b) ≤ ax + by 的等号可取得，即 gcd 是 a 和 b 的最小的正线性组合)

Chapter 32 String Matching

- 前缀 & 后缀：
  - $\omega \sqsubset x$: string ω is a prefix of x, if $x = \omega y$ for some $y \in \Sigma^*$.

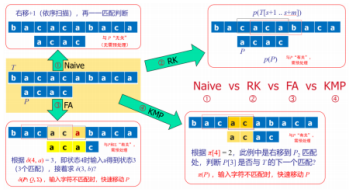
prefix (ω 是 x 的前缀)　　suffix

  - $\omega \sqsupset x$: ω is a suffix of x, if $x = y\omega$ for some $y \in \Sigma^*$.
    - If $\omega \sqsubset x$ or $\omega \sqsupset x$, then $|\omega| \le |x|$.
    - The empty string ε is both a suffix and a prefix of every string.
    - For example, we have ab $\sqsubset$ abcca and cca $\sqsupset$ abcca.
    - For any strings x and y and any character a, we have $x \sqsupset y$ if and only if $xa \sqsupset ya$.
    - $\sqsubset$ and $\sqsupset$ are transitive relations.
- 字符串匹配算法比较：



Algorithm	Preprocessing time	Matching time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\|\Sigma\|)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

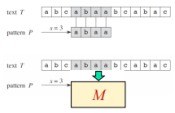
	关键	特征
FA	求 $\delta(P: Q, \Sigma)$	输入字符 $T[i]$ 不匹配时，快速移动 P，每个 $T[i]$ 匹配一次
KMP	求 $\pi(P)$	输入字符 $T[i]$ 不匹配时，快速移动 P，每个 $T[i]$ 可能匹配多次

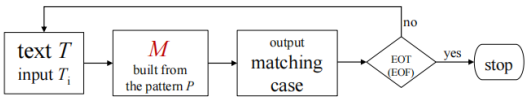


- 有限自动机（有穷自动机） & 作者：

[引用] The design and analysis of computer algorithms
AV Aho, JE Hopcroft - 1974 - Pearson Education India
☆ 保存 99 引用 被引用次数：15831 相关文章 所有 9 个版本 »

- Many string-matching algorithms build a finite automaton (Machine) that scans the text T for all occurrences of the pattern P.
- These string-matching automata are very efficient:
  - they examine each text character *exactly once* ;
  - taking constant time per text character.





- **KMP**：作者：**Knuth - Morris - Pratt**

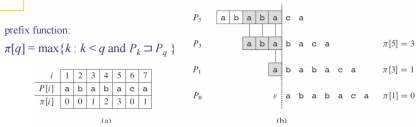
KMP is a linear-time string-matching algorithm due to Knuth, Morris, and Pratt.

The accounting method

Running time? chapter17 Amortized analysis (accounting)

```
KMP-MATCHER(T, P)                                   Θ(n)
1  n = T.length
2  m = P.length
3  π = COMPUTE-PREFIX-FUNCTION(P)
4  q = 0
5  for i = 1 to n
6      while q > 0 and P[q + 1] ≠ T[i]
7          q = π[q]
8      if P[q + 1] == T[i]
9          q = q + 1
10     if q == m
11         print "Pattern occurs with shift" i − m
12         q = π[q]
```

```
COMPUTE-PREFIX-FUNCTION(P)                          Θ(m)
1  m = P.length
2  let π[1 .. m] be a new array
3  π[1] = 0
4  k = 0
5  for q = 2 to m
6      while k > 0 and P[k + 1] ≠ P[q]
7          k = π[k]
8      if P[k + 1] == P[q]
9          k = k + 1
10     π[q] = k
11 return π
```



KMP algorithm avoids computing the transition function δ, and its matching time is $\Theta(n)$ using just an auxiliary function π, which we precompute from the pattern in time $\Theta(m)$ and store in an array $\pi[1 \,..\, m]$.

Chapter 17 平摊分析

- 平摊分析：
  - In an amortized analysis, we average the time required to perform a sequence of data-structure operations over all the operations performed.
    在多个操作中，求一个操作的平均时间
  - With amortized analysis, we can show that the average cost of an operation is small, if we average over a sequence of operations, even though a single operation within the sequence might be expensive.
  - Amortized analysis differs from average-case analysis in that probability is not involved; an amortized analysis guarantees the average performance of each operation in the worst case.
    平摊分析不需要考虑输入的概率密度
  - Amortized cost, 分摊消费：在一个操作上的平均消费 (cost)
- 包含方法：
  - 聚集分析（aggregate analysis）——凸包
  - 记账法（the accounting method）——凸包，KMP
  - 势能法
- 聚集分析：

- In aggregate analysis, we show that for all n, a sequence of n operations takes worst-case time $T(n)$ in total.
- In the worst case, the average cost, or amortized cost, per operation is therefore $T(n)/n$.

- 记账法：
  - **Amortized cost** : the amount we charge an operation.
    - **Credit** : when an operation's amortized cost exceeds its actual cost, we assign the difference to specific objects in the data structure as credit.
      分配的消费券大于实际价格，多余券留下，作为信用，将来用（某些分摊金额小于实际花费时使用）
    - Credit can help pay for later operations whose amortized cost is less than their actual cost.
  - We denote the actual cost of the ith operation by c_i and the amortized cost of the ith operation by $\hat{c}_i$, we require
    $$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$
    For all sequences of n operations.
  - The total credit
    $$\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$$
- 势能法：
  - The potential method of amortized analysis represents the prepaid work as "potential energy" (potential), which can be released to pay for future operations.
  - We associate the potential with the data structure (DS) as a whole rather than with specific objects within the data structure.
  - The potential method works as follows:
    - We will perform n operations, starting with an initial DS D_0.
    - c_i : the actual cost of the ith operation ($i = 1, 2, \ldots, n$).
    - D_i : the DS that results after applying the ith operation to DS D_{i-1}.
    - Φ : A potential function Φ maps D_i to a real number $\Phi(D_i)$, which is the potential associated with D_i.
  - **The amortized cost** $\hat{c}_i$ of the ith operation with respect to potential function Φ is defined by
    $$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$



- **The amortized cost** $\hat{c}_i$ of the ith operation with respect to potential function Φ is defined by
$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$
- The total amortized cost of the n operations is
$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)\end{aligned}$$

- We usually just define $\Phi(D_0)$ to be 0 and then show that $\Phi(D_i) \geq 0$ for all i .
- Different potential functions may yield different amortized costs.

Chapter 34 NP Complete Problems

- **NP Complete Problems**：

千僖年数学难题（2000-5-24，美国的克雷(Clay)数学研究所，在巴黎法兰西学院宣布每一个悬赏一百万美元）

一、贝赫(Birch)和斯维讷通 - 戴尔(Swinnerton-Dyer)猜想

二、霍奇(Hodge)猜想

三、纳维叶 - 斯托克斯(Navier-Stokes)方程

四、P (多项式算法可解)问题对NP ("非确定性问题")

五、庞加莱(Poincare)猜想

This question turned out to be extraordinarily difficult. Nearly a century passed between its formulation in 1904 by Henri Poincaré and its solution by **Grigoriy Perelman**, announced in preprints posted on ArXiv.org in 2002 and 2003.
俄罗斯数学家格里戈里·佩雷尔曼在预印本平台 ArXiv.org 上发布了证明结果。
他拒绝去领奖，"如果我的证明是正确的，这种方式的承认是不必要的。" 2006年8月，拒绝了有着数学界诺贝尔奖之称的"菲尔兹奖"，佩雷尔曼得知菲尔兹奖将由西班牙国王颁发时，他说道："国王又不是数学家，为什么有资格颁奖？"

六、黎曼(Riemann)假设

七、杨 - 米尔斯(Yang-Mills)存在性和质量缺口

- 问题类型：
  - **P问题**：

    P is the class of decision problems that can be solved in polynomial time ($O(n^k)$, where k is a constant). Intuitively, the problems in P class are easy problems.

    P 问题：多项式时间内可解的判定问题

    如：
    - 在数组 A 中，是否存在相同的数
    - 在图 G 中，顶点 u 到 v 是否存在小于 k 的一条路径

  - **NP问题**：

    NP is the class of decision problems for which we can verify the correctness of solutions in polynomial time.

    NP 问题：多项式时间内可验证的判定问题

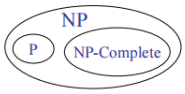
    - This doesn't say it is easy to find a solution.
    - In fact, it is often hard to find a solution!

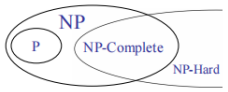
  - **NPC问题**：（对于任何的NPC问题，尚没有找到多项式算法）

    所有的**NPC问题**都可以转化为**Boolean Satisfiability**

    （CNF合取范式）（SAT可满足性定理）（穷举搜索）（回溯搜索）（局部搜索）

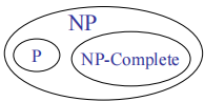
- Intuitively, NP-Complete is the class of the "most difficult" problems in NP.
- All NP-Complete problems appear to be difficult.
- No polynomial-time algorithm has been found for any NP-Complete problem. (对任何 NPC，尚没有找到多项式算法) For example,
  - Hamiltonian cycle problem
  - Boolean Satisfiability (布尔可满足性问题)
- NPC 问题：如果 NP 问题的所有可能答案都可以在多项式时间内进行正确与否的验算，就叫 NPC 问题 (完全多项式非确定问题)。
- 一个可判定性问题 C 是 NP 完全 (NPC) 的，如果：
  1. 这个问题是 NP 问题。
  2. 所有其他的 NPC 问题可以归约为 C 问题。





NP-hard: 对于判定问题 A，若 A 满足，所有的 NP 问题都可以约化到它。(NP-Hard 问题比 NPC 问题的范围广) NPC 和 NP-hard 的主要区别在于：验证一个问题 A 是否为 NP-hard 问题，无需判断 A 是否属于 NP。

- 所有的 NPC 都可以在转换为 Boolean Satisfiability
- Cook 于1971证明了 Sat 是 NPC，现在发现的 NPC 已经超过3000个?
- 如果任一 NPC 问题多项式可解，则所有 NPC 都是多项式可解!



- **RSA公钥加密系统 - NP的应用 & 算法导论作者：**

  - The concept of a public-key cryptosystem is due to Diffie and Hellman, 1976
  - The RSA cryptosystem (1977, MIT) was proposed by Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman, (Turing Award, 2002)

Rivest Shamir Adleman

Adi Shamir（2002，图灵奖；2024，沃尔夫奖），
2009年在北航讲学，报告题目“密码和安全系统是怎样被破解的”，
软件学院首任院长孙伟教授主持