数据结构与算法设计

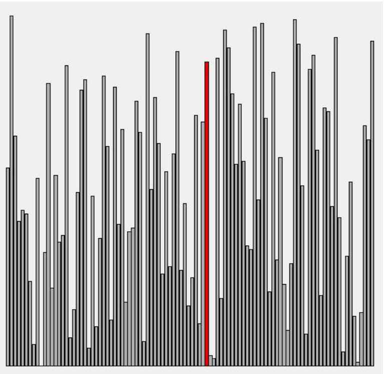
周　可

Mail：zhke@hust.edu.cn

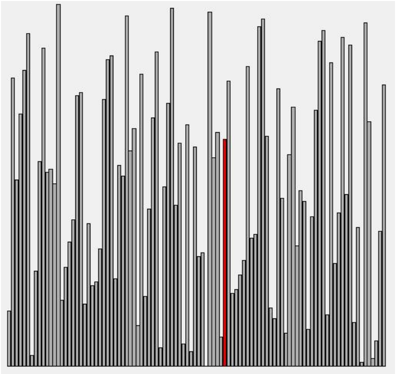
华中科技大学，武汉光电国家研究中心

Review: Algorithm & Thoughts

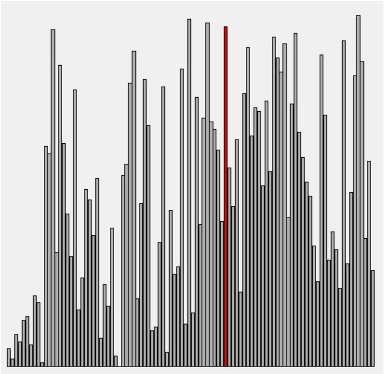
- 排序算法——使得序列有序
- 什么样的序列是一个有序序列?
  - 任意第i个元素是第i小的元素
  - 任意子串都有序
  - 任意元素的左边元素都比其小，右边的元素都比其大



插入排序



归并排序

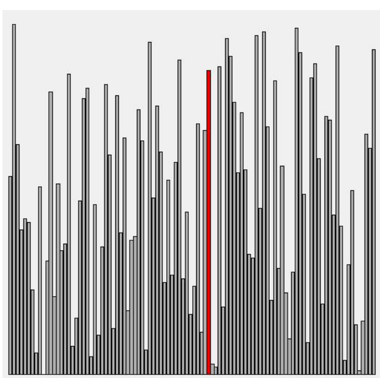


快速排序

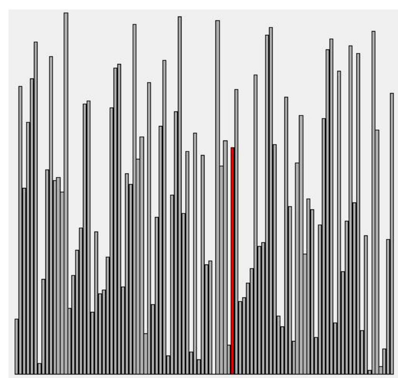
Review: Algorithm & Thoughts

1. **排序算法**——使得序列有序

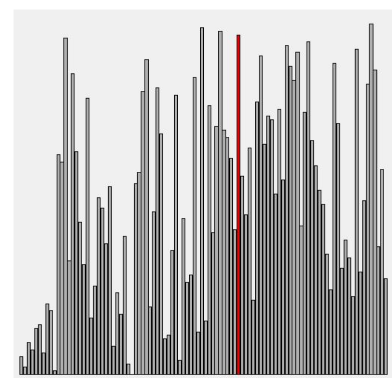
- 什么样的序列是一个有序序列?
  - 任意第i个元素是第i小的元素 贪心
  - 任意子串都有序 分治
  - 任意元素的左边元素都比其小，右边的元素都比其大 定标



插入排序



归并排序

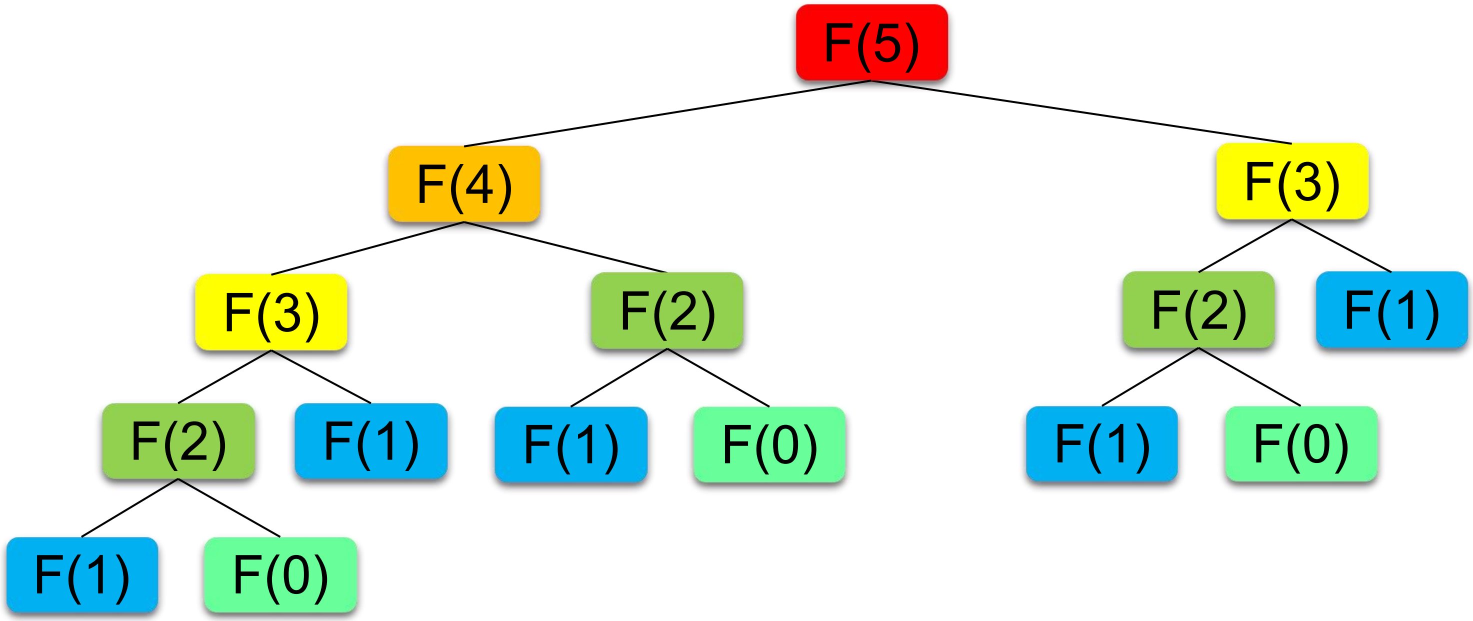


快速排序

Review: Algorithm & Thoughts

2. 动态规划

   - 每个子问题求解一次且仅一次
   - 记录子问题的结果

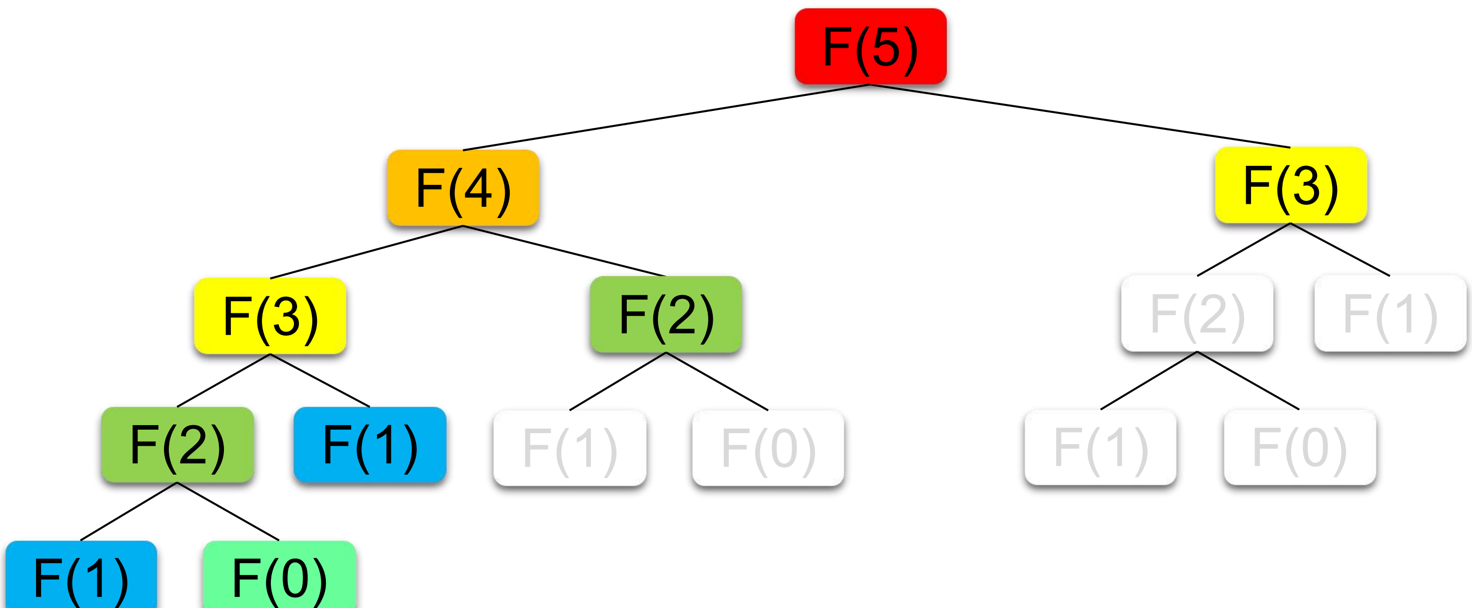


斐波那契数

Review: Algorithm & Thoughts

2. 动态规划

- 每个子问题求解一次且仅一次 尝试
- 记录子问题的结果 学习



斐波那契数

Review: Algorithm & Life

选择排序

归并排序

快速排序

以人为鉴，可明得失

以古为鉴，可知兴替

——李世民

动态规划

比较

找到差距，成就更好的自己

学习

借鉴他人智慧，减少低效率重复

Review: Algorithm & Life

算法思想

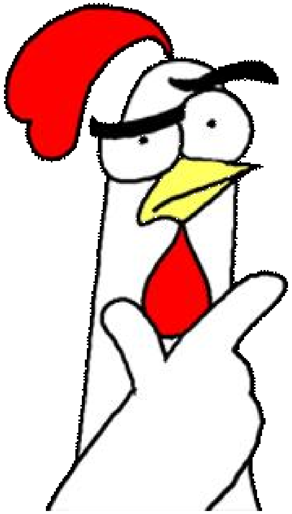
选择排序 → 贪心

归并排序 → 分治

快速排序 → 定标

动态规划 → 尝试

动态规划 → 学习



一个生活中的例子

Review: Algorithm & Life

选择排序 → 贪心

归并排序 → 分治

快速排序 → 定标

动态规划 → 尝试

动态规划 → 学习

根据属地安排，定于5月14日（周六）开展核酸扩面检测，请各学院做好学生组织工作，确保应检尽检。
现将主校区核酸检测安排通知如下：
1.检测对象
3岁以上全体在校人员
2.检测时间
15:30-21:00
3.检测地点
东边操场、西边操场、中心操场
4.注意事项
①请携带手机，登记时出示微信武汉战疫健康码，如无健康码，带身份证或户口本备用。
②为避免聚集，减少排队时间，请按照指引有序就近前往检测。

→ 5月14日后，三岁以上在校人员核酸检测都已完成

定标思想

Review: Algorithm & Life

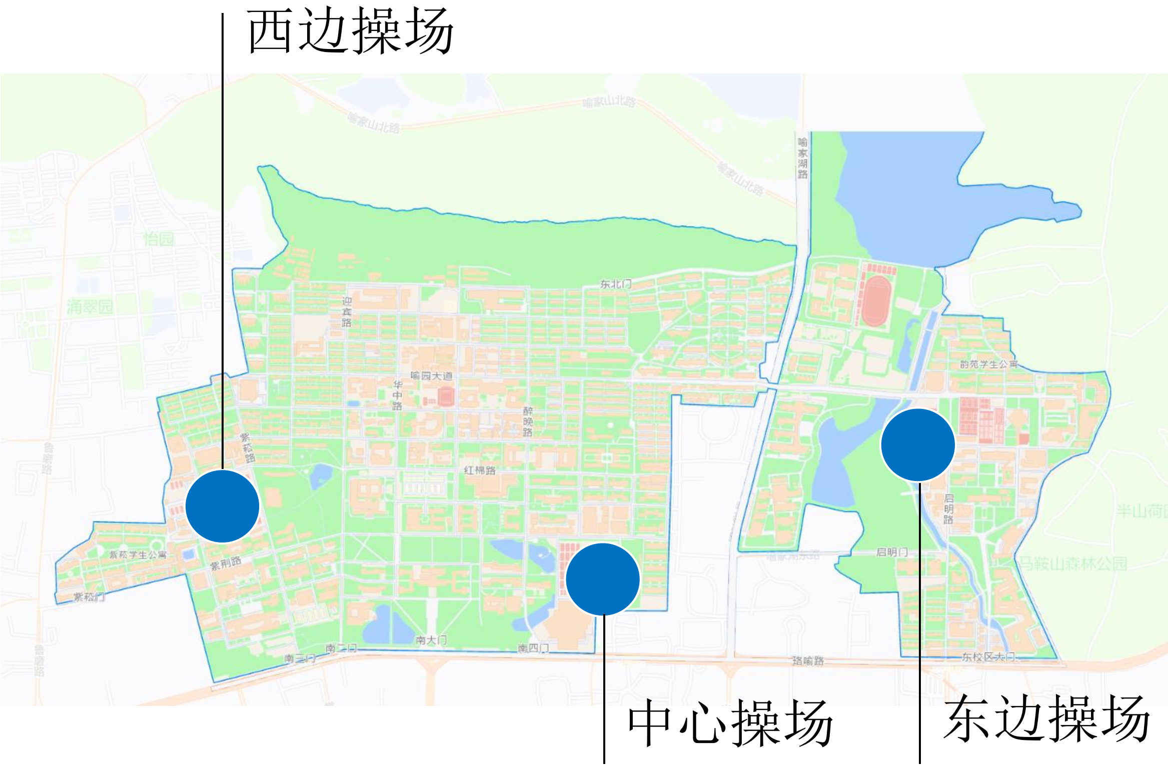
选择排序 → 贪心

归并排序 → 分治

快速排序 → 定标

动态规划 → 尝试

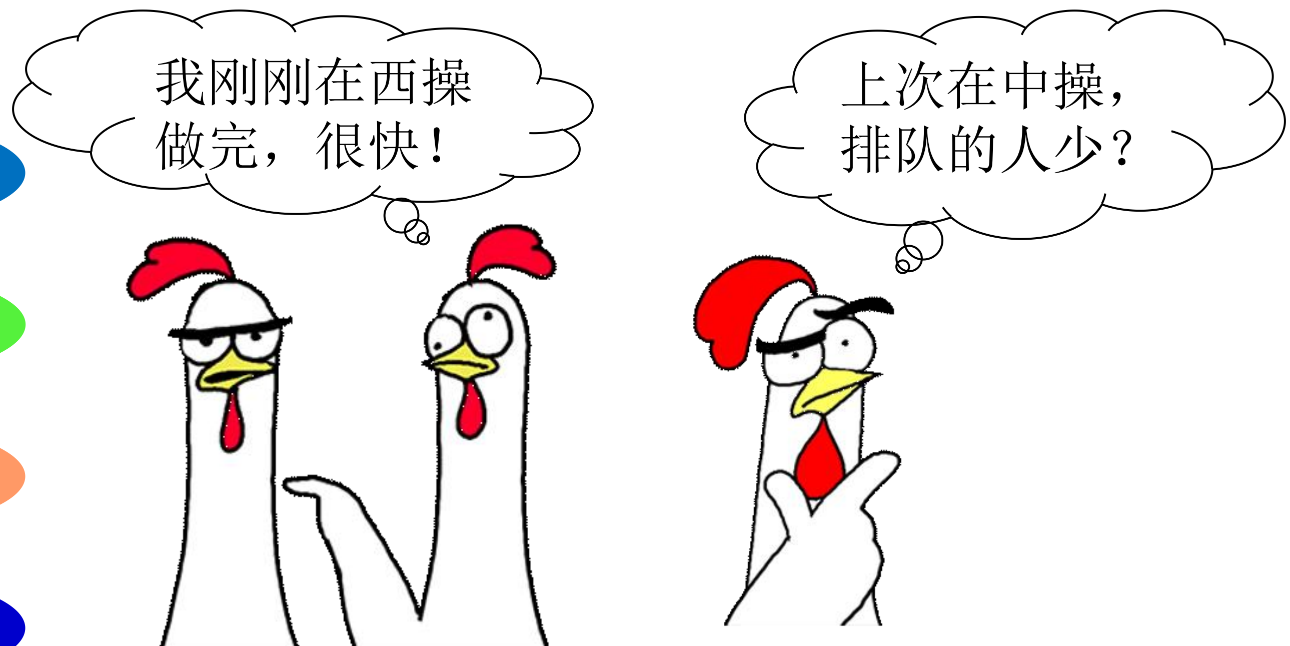
动态规划 → 学习



多个核酸检测点：*分治思想*

Review: Algorithm & Life

- 选择排序 → 贪心
- 归并排序 → 分治
- 快速排序 → 定标
- 动态规划 → 尝试
- 动态规划 → 学习



尝试思想　　*学习思想*

Review: Algorithm & Life

选择排序 → 贪心

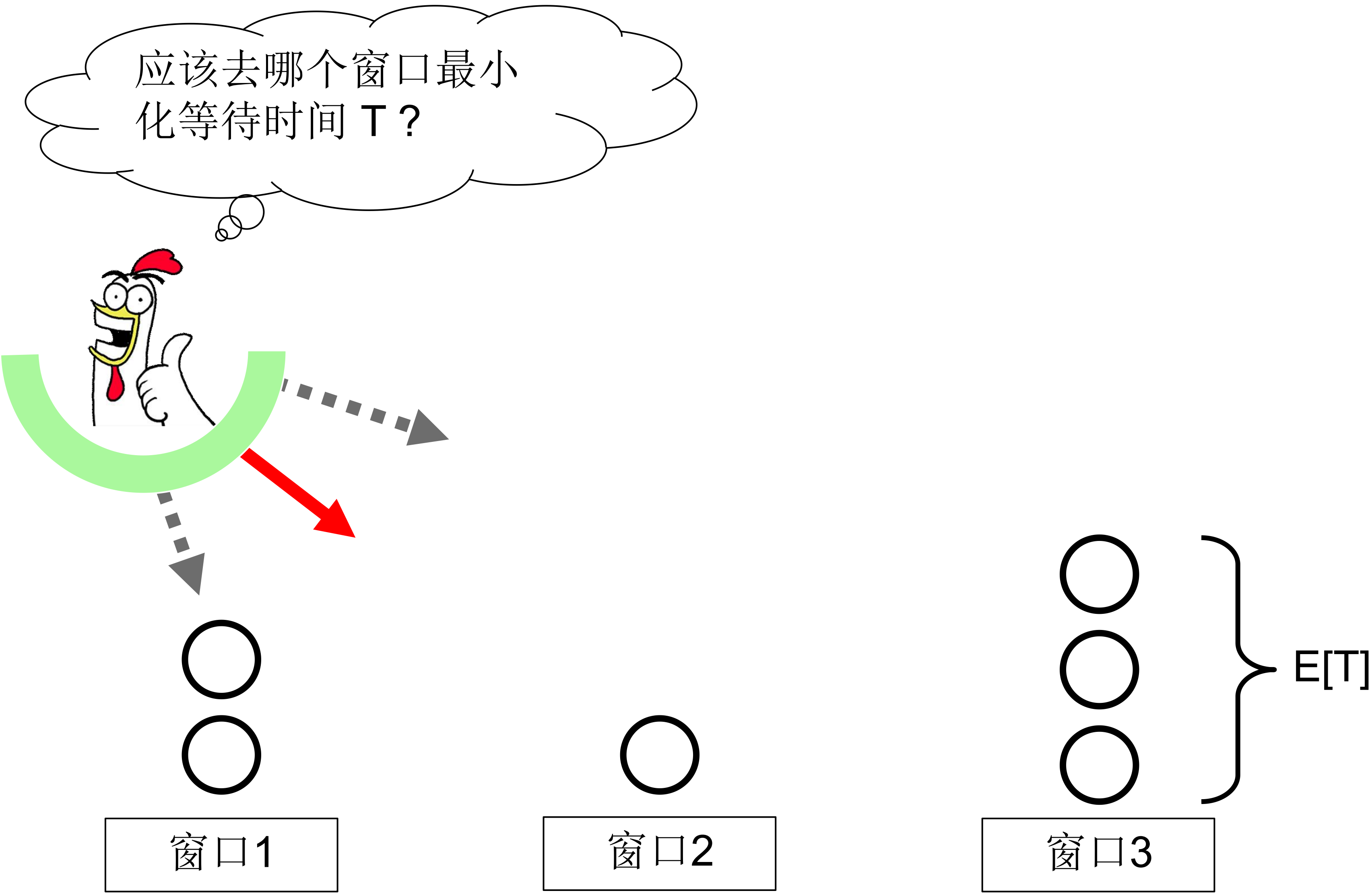
归并排序 → 分治

快速排序 → 定标

动态规划 → 尝试

动态规划 → 学习

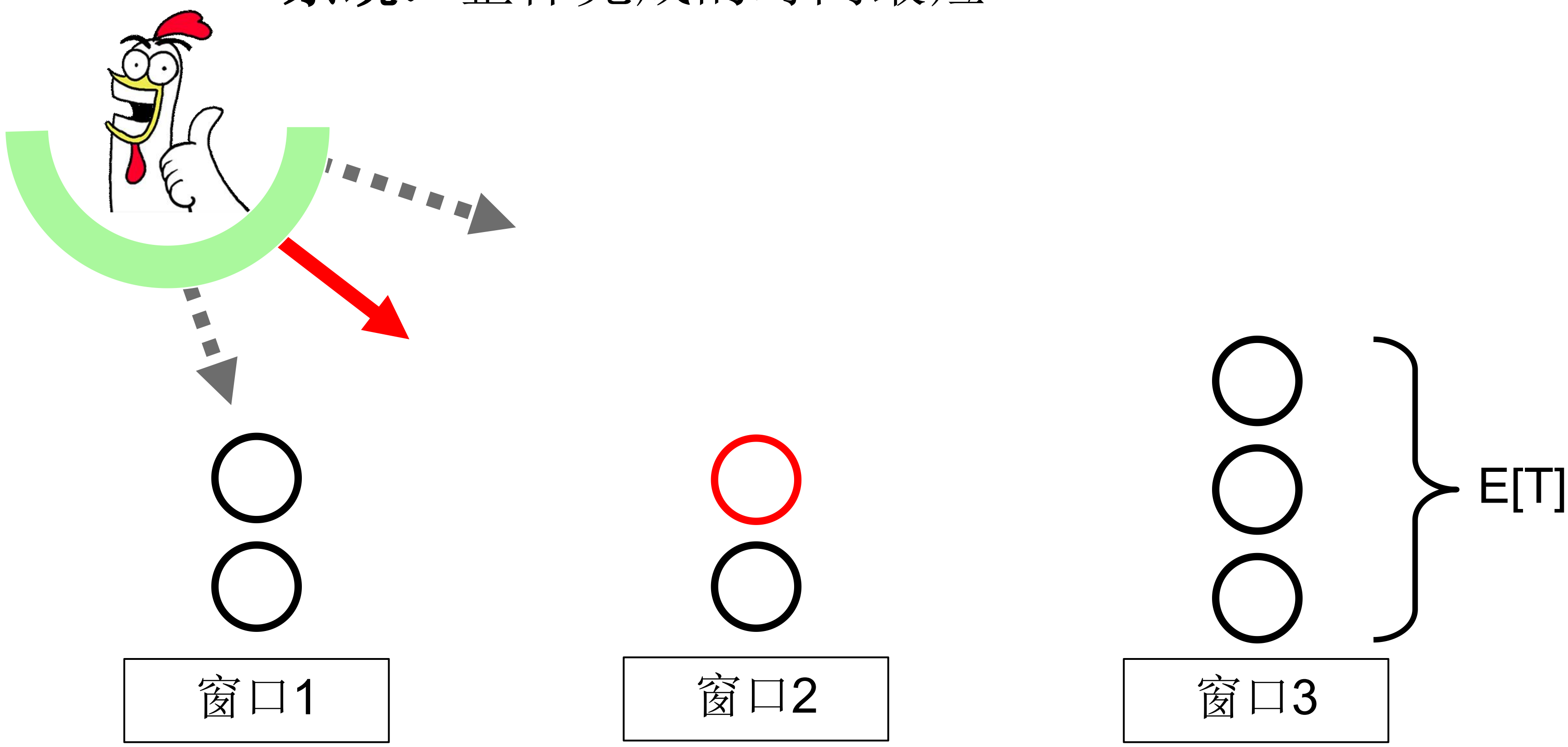
贪心思想


应该去哪个窗口最小
化等待时间 T？
E[T]
窗口1
窗口2
窗口3

贪心法： 选择队列最短的排队

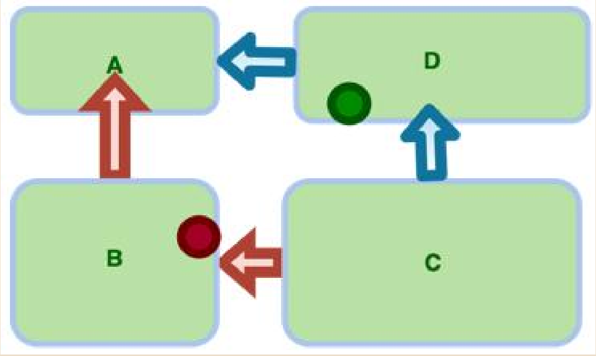
个人： 获得最短的等待时间期望E[T]

系统： 整体完成的时间最短



贪心的应用无处不在

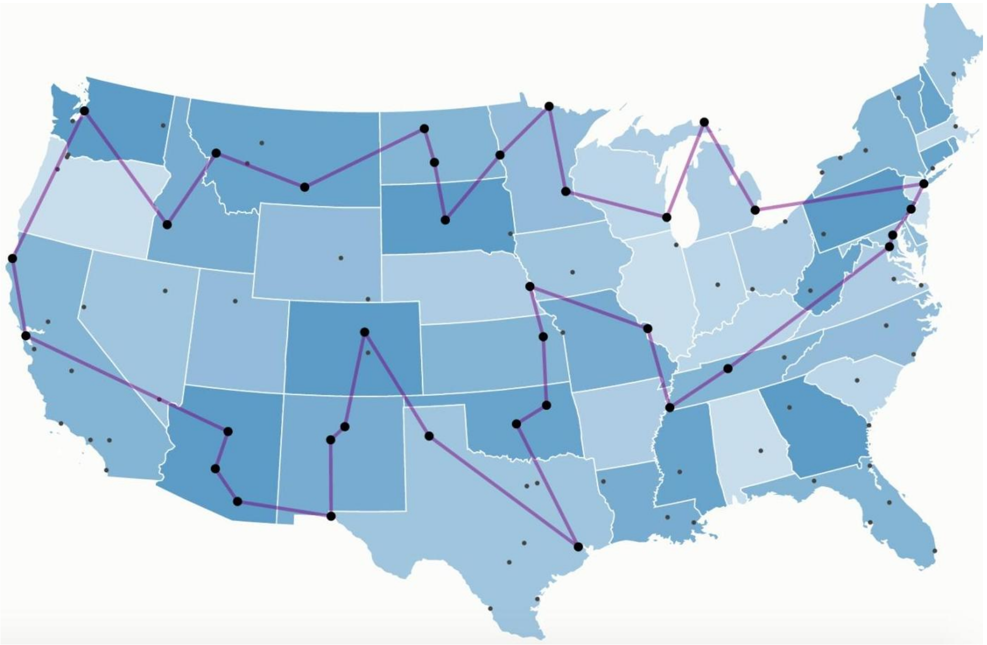
大自然中的贪心	古人的智慧
 费马原理	 田忌赛马

生活中的贪心	
 十字路口选择	 芝加哥惊魂记

千禧难题

P=NP? 是千禧年大奖难题（世界七大难题）之首。

TSP，即旅行商问题，是数学领域著名问题之一，也是NP问题。



10个城市为例：

10！=3628800

贪心算法求解

算法简单

- **贪心法示例**
- 贪心策略的求解方法
- 哈夫曼编码

Idea

You want to maximize a global function, which could be hard

(1) You always make the best local decision in the hope of getting the overall best solution.

(2) The idea is natural (and in some situation it is the best a human can do).

Idea

You want to maximize a global function, which could be hard

(1) You always make the best local decision in the hope of getting the overall best solution.

(2) The idea is natural (and in some situation it is the best a human can do).

(3) Of course, sometimes it might not work.

Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

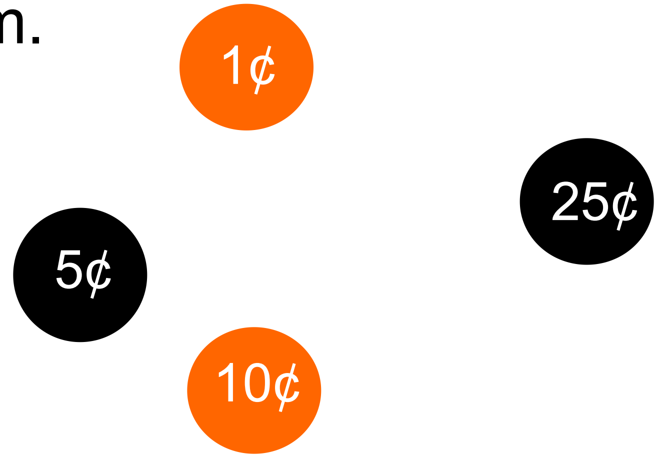
Remember that in this case a penny and a dime is considered the same (possibly in weight).

Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

Remember that in this case a penny and a dime is considered the same (possibly in weight).

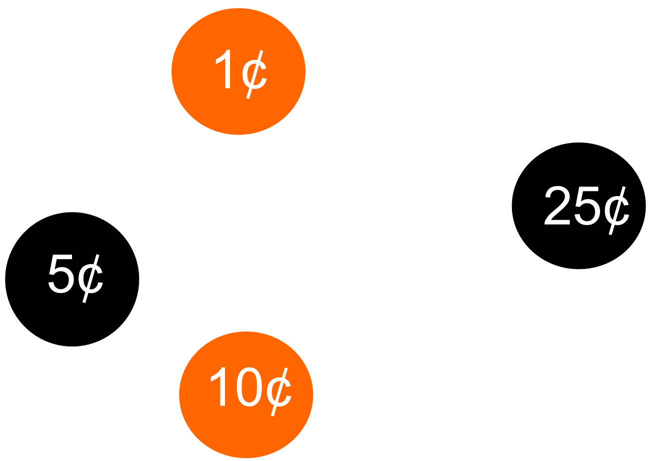
Let's first look at the US system.



Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

How to make change for \$2.17?



Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

How to make change for $2.17?

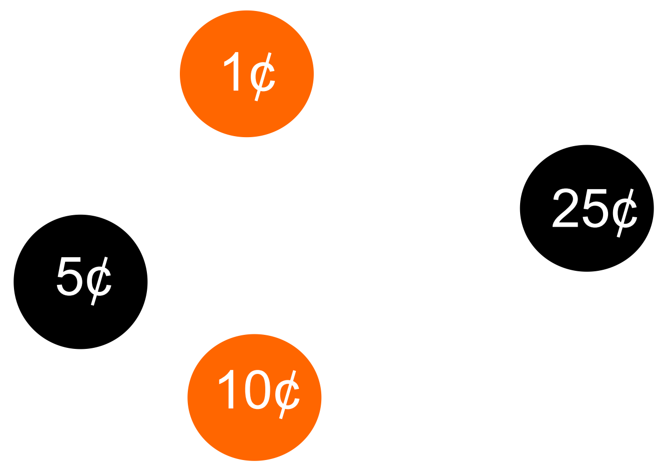
8 quarters = $2.00

1 dime = $0.10

1 nickel = $0.05

2 pennies = $0.02

So we use 12 coins!



Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

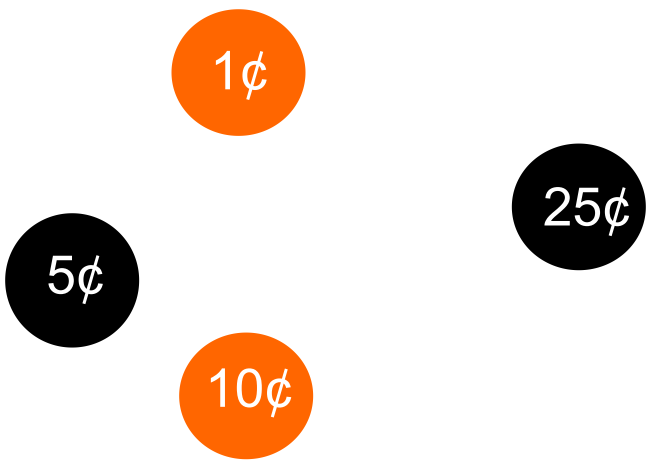
How to make change for \$2.17?

8 quarters = \$2.00

1 dime = \$0.10

1 nickel = \$0.05

2 pennies = \$0.02



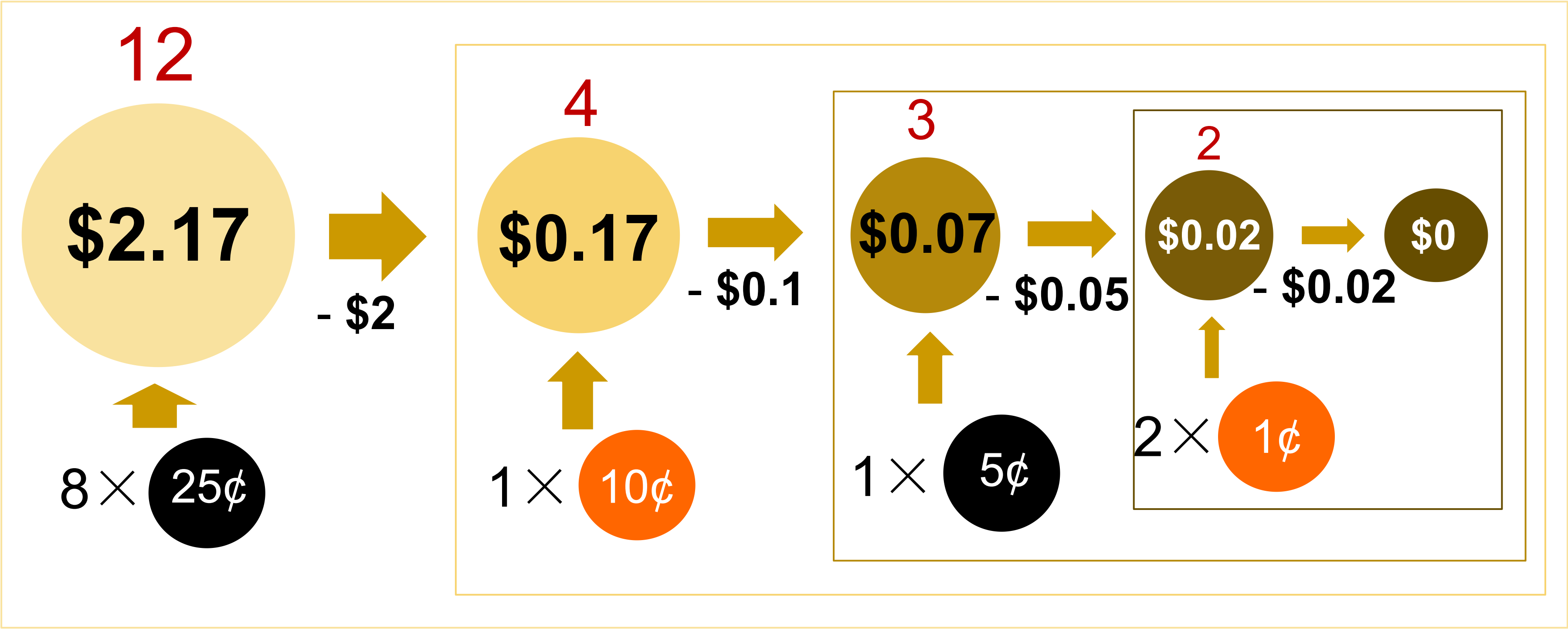
Why this works?

Example 1. Coin Change

How to make change for $2.17 using the minimum number of coins ?

(A: 8 quarters = $2.00; 1 dime = $0.10; 1 nickel = $0.05; 2 pennies = $0.02)

Why this works?



Example 1. Coin Change

Q：贪心策略的核心思想？

去掉贪心选择结果

贪心求解**当前问题**，
记录贪心选择结果

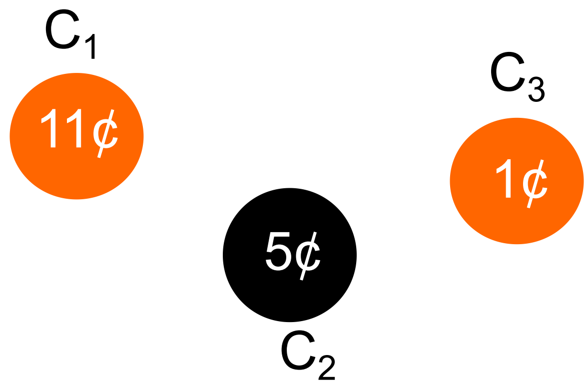
获得**子问题**

当前问题 = 子问题

Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

How to make change for $0.15 under a new system with the minimum coins?



Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

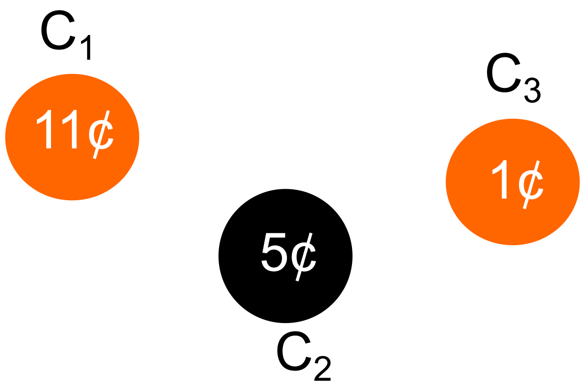
How to make change for \$0.15 under a new system with the minimum coins?

Greedy:

1 C_1 = 11¢

4 C_3= 4¢

So we will have to use 5 coins.



Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

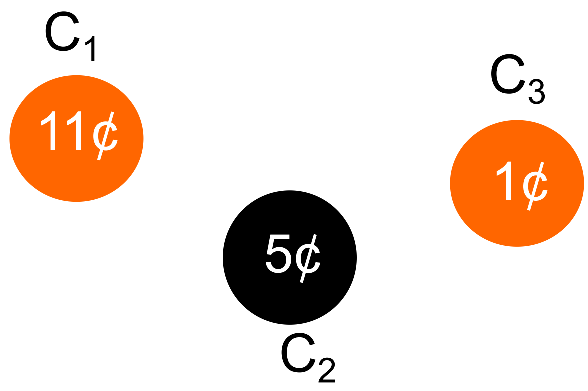
How to make change for $0.15 under a new system with the minimum coins?

Greedy:

1 C_1 = 11¢

4 C_3= 4¢

1+4=5



A better way:

3 C_2, only 3 coins!

Example 1. Coin Change

Making changes for *n* cents using the minimum number of coins.

How to make change for $0.15 under a new system with the minimum coins?

Greedy:

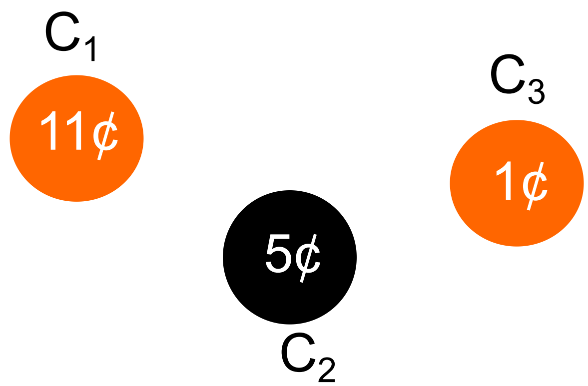
1 C_1 = 11¢

4 C_3= 4¢

1+4=5

A better way:

3 C_2, only 3 coins!



<u>While greedy methods do not always work (in fact, nothing always works), they are still useful in some situations.</u>

Example 2. The knapsack problem

You have a knapsack which can only contain certain weight C of goods.

With this weight constraint, you want to maximize the values of the goods you can put in the knapsack.

G1=candy, Total value=$1.0, Total weight=10 pounds

G2=chocolate, Total value=$2.0, Total weight=1 pounds

G3=ice cream, Total value=$2.5, Total weight=4 pounds

If C=4 pounds, what would you do?

Example 2. The knapsack problem

G1=candy, Total value=$1.0, Total weight=10 pounds

G2=chocolate, Total value=$2.0, Total weight=1 pounds

G3=ice cream, Total value=$2.5, Total weight=4 pounds

If C=4 pounds, what would you do?

Greedy 1: by maximum value → 4 pounds of ice cream, profit=$2.5

Greedy 2: by maximum weight → 4 pounds of candy, profit=$0.4

Greedy 3: by maximum unit value → 1 pound of chocolate followed with 3 pounds of ice cream, profit=$3.875

Example 2. The knapsack problem

G1=candy, Total value=$1.0, Total weight=10 pounds

G2=chocolate, Total value=$2.0, Total weight=1 pounds

G3=ice cream, Total value=$2.5, Total weight=4 pounds

In general, you have $G_1, G_2, \ldots, G_n$, each G_i with weight w_i and value v_i,

and you want to maximize the profit out of the goods you can put in the knapsack with capacity C.

How do we formulate this as a mathematical programming problem?

Example 2. The knapsack problem

In general, you have $G_1, G_2, \ldots, G_n$, each G_i with weight w_i and value v_i,

and you want to maximize the profit out of the goods you can put in the knapsack with capacity C.

How do we formulate this as a mathematical programming problem?

Let f_i be the fractional of G_i one would put in the knapsack.

Example 2. The knapsack problem

In general, you have $G_1, G_2, \ldots, G_n$, each G_i with weight w_i and value v_i,

and you want to maximize the profit out of the goods you can put in the knapsack with capacity C.

How do we formulate this as a mathematical programming problem?

Let f_i be the fractional of G_i one would put in the knapsack.

Maximize $\sum_{i=1..n} f_i v_i$

Subject to $\sum_{i=1..n} f_i w_i \leq C$,

$0 \leq f_i \leq 1, i=1..n$

Fractional Knapsack Problem

Example 2. The knapsack problem

In general, you have $G_1, G_2, \ldots, G_n$, each G_i with weight w_i and value v_i,

and you want to maximize the profit out of the goods you can put in the knapsack with capacity C.

How do we formulate this as a mathematical programming problem?

Let f_i be the fractional of G_i one would put in the knapsack.

Maximize $\sum_{i=1..n} f_i v_i$

Subject to $\sum_{i=1..n} f_i w_i \le C$,

$0 \le f_i \le 1, i=1..n$

Fractional Knapsack Problem

Maximize $\sum_{i=1..n} f_i v_i$

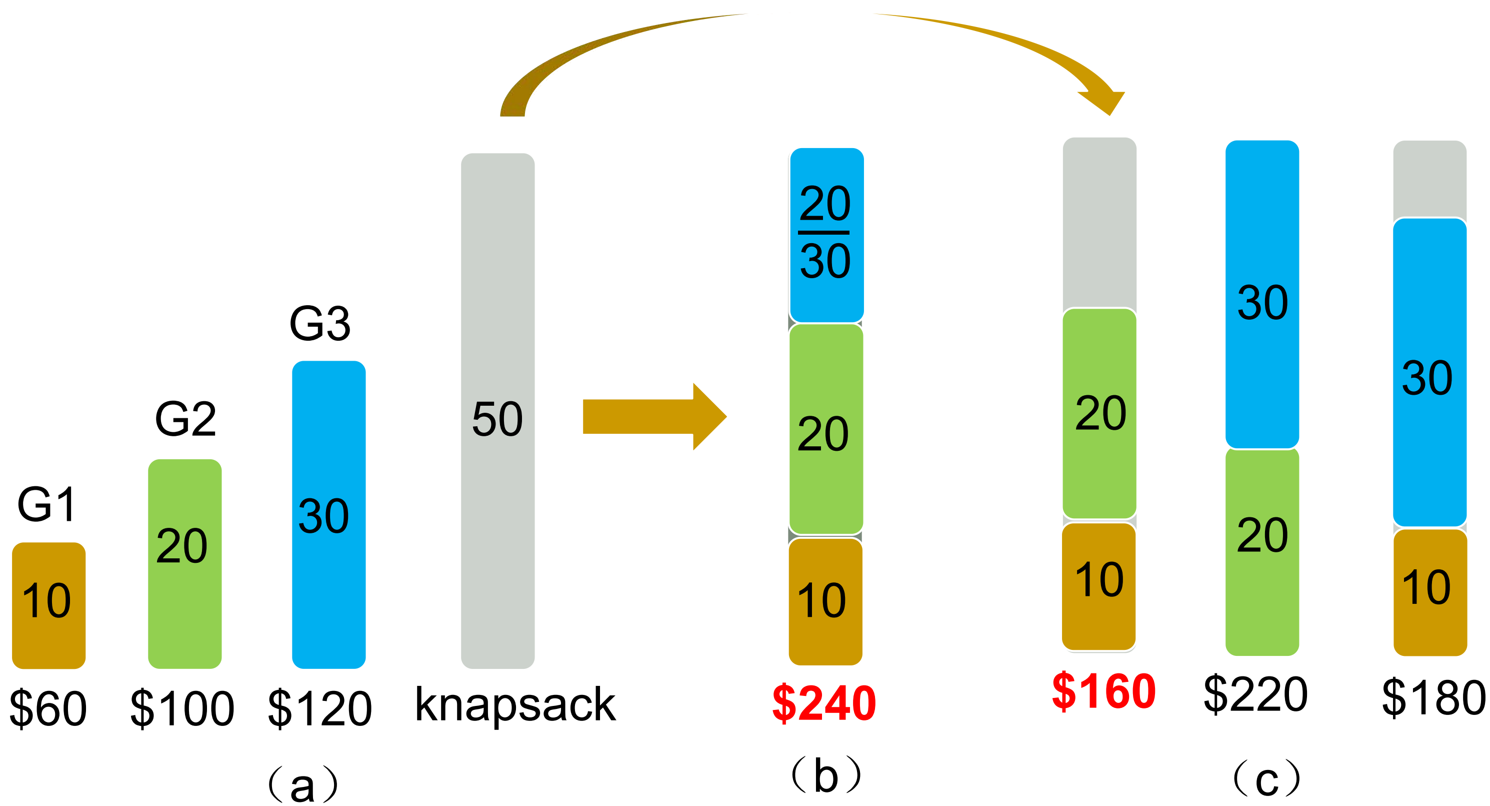
Subject to $\sum_{i=1..n} f_i w_i \le C$,

$f_i \in \{0,1\}$, $i=1..n$

Integer Knapsack Problem

Example 2. The knapsack problem

The Fractional Knapsack Problem can be solved optimally using Greedy method. But, what about the Integer Knapsack Problem?



- 贪心法示例
- **贪心策略的求解方法**
- 哈夫曼编码

最优化问题：

{ 多个可行解 } 使目标函数取极值 → { 最优解 }

最优化问题的数学模型，可以用数学符号表示成：

Min F(X) 或 Max F(X)

目标函数

满足约束条件的**自变量**

<u>**贪心方法**：是求解最优化问题的一种方法。贪心算法总是作出在当前看来最好的选择，即“**局部最优**”</u>。

贪心方法求解的一般步骤：

1）**初始化**：已知问题有n个输入，置问题的解集合J为空；

2）**选度量标准**：根据题意，选取一种度量标准，按照这种度量标准对n个输入排序；

3）**考察输入**：按序一次输入一个量，看该量能否和J中已选出来的元素（称为该度量意义下的部分最优解）加在一起构成新的可行解：如果可以，则该量并入J集合，从而得到一个新的部分解集合；如果不可以，则丢弃该量，J集合保持不变。之后，继续上述过程，考察下一输入量，直到所有输入都考察完毕。

4）**获得贪心解**：当所有的输入都被考虑完毕，被记入到集合J中的元素构成了这种量度意义下的问题的最优解。

The knapsack problem

1. 问题的描述

已知n种物品，各具有重量$(w_1,w_2,\ldots,w_n)$和价值$(p_1,p_2,\ldots,p_n)$，及一个可容纳M重量的背包。

问：怎样装包才能使装入背包的物品的总价值最大？

这里：1）所有的$w_i>0$, $p_i>0$，$1\le i\le n$；

2）问题的解用向量$(x_1,x_2,\ldots,x_n)$表示，每个x_i表示物品i被放入背包的比例，$0\le x_i\le 1$。

3）当物品i的一部分x_i放入背包，可得到x_ip_i的价值，同时会占用x_iw_i的重量。

问题分析：

① 装入背包的总重量不能超过M，即 $\sum_{1\le i\le n} w_i x_i \le M$ 。

② 如果所有物品的总重量不超过M，即 $\sum_{1\le i\le n} w_i \le M$, 则显然把所有的物品都装入背包中才可获得最大的价值，此时所有的 $x_i=1$，$1\le i\le n$。

③ 如果物品的总重量 $\sum_{1\le i\le n} w_i \ge M$ ，则将有物品可能无法装入背包。此时，由于$0\le x_i\le 1$，所以可以把物品的全部或部分装入背包，最终背包中刚好装入重量为M的若干物品（整体或部分）。

问题的形式化描述

约束条件： $\sum_{1\le i\le n} w_i x_i \le M$

$$0 \le x_i \le 1, p_i > 0, w_i > 0, 1 \le i \le n$$

目标函数： $Max \sum_{1\le i\le n} p_i x_i$

可 行 解：满足上述约束条件的任一$(x_1,x_2,\ldots,x_n)$ 都是问题的一个可行解。 $(x_1,x_2,\ldots,x_n)$称为问题的一个解向量。

最 优 解：能够使目标函数取最大值的可行解是问题的最优解。最优解可能有多个。

例 设有三件物品和一个背包，物品价值 (p_1,p_2,p_3) = (25,24,15)， 重量(w_1,w_2,w_3) = (18,15,10)；背包容量M=20。求该背包问题的解。

可行解如下：

	(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$	
①	(1/2,1/3, 1/4)	16.5	24.25	//没有装满背包//
②	(1, 2/15,0)	20	28.2	
③	(0, 2/3, 1)	20	31	
④	(0, 1, 1/2)	20	31.5	

2. 贪心策略求解

① 以目标函数作为度量

解题思路：每装入一件物品，就使背包获得最大的价值增量。

处理规则：以目标函数作为度量，考虑到贪心策略的基本处理流程，则有：

- 按价值的非增次序将物品一件件地放入到背包；
- **如果正在考虑的物品放不进去**，则只取其一部分装满背包。此时，如果该物品的一部分不满足获得最大价值增量的度量标准，则在剩下的物品中选择**可以获得最大价值增量的其它物品**，将它或其一部分装入背包。如下例，

（接上）

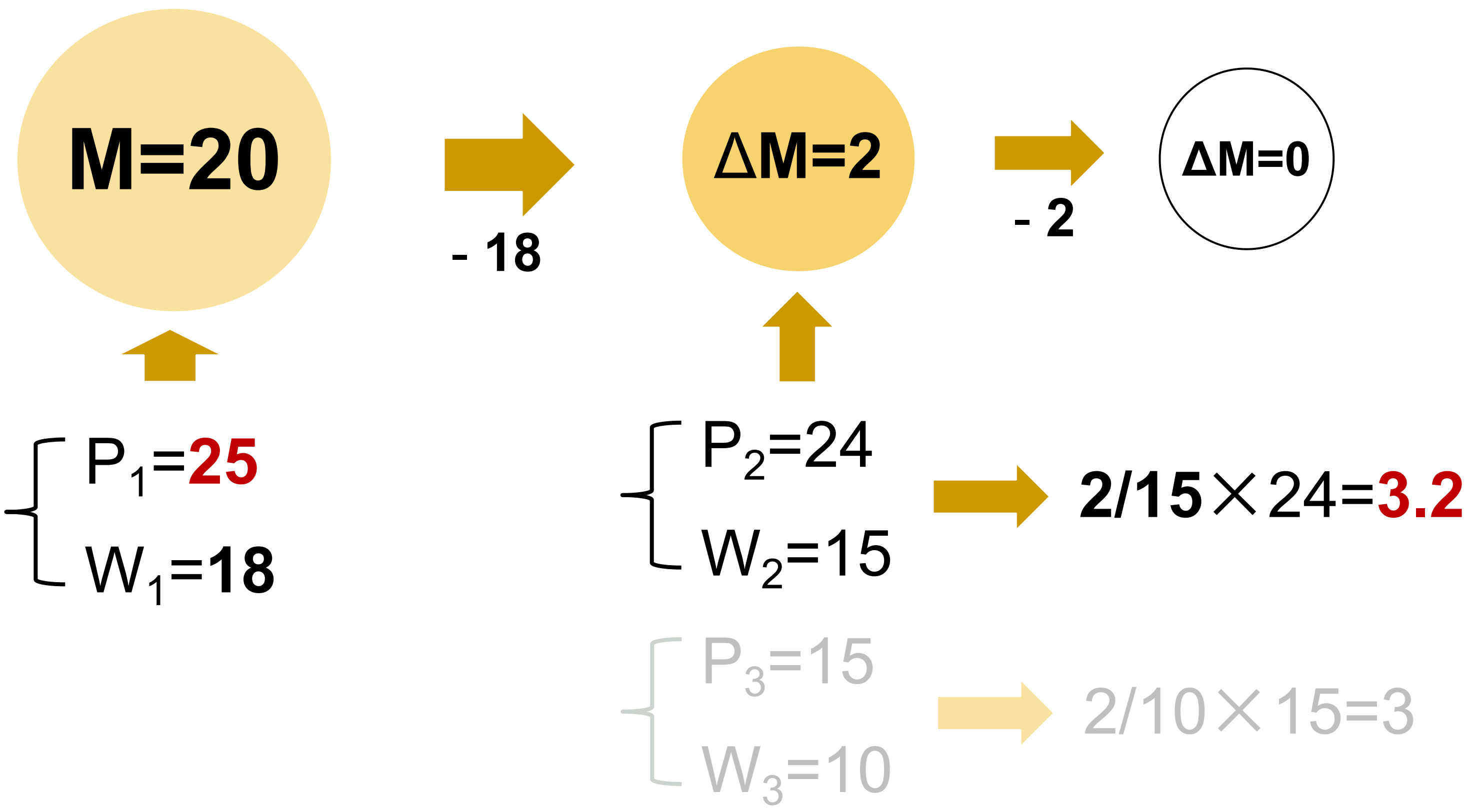
如：若背包剩余容量ΔM=2,而此时背包外还剩两件物品i,j，且有($p_i = 4$，$w_i = 4$) 和($p_j = 3$，$w_j = 2$)，则下一步应选择j而非i放入背包，因为

$$p_i/2 = 2 < p_j = 3$$

即虽然$p_i > p_j$，但物品j可以全部放入并带来3的价值，而物品i只能放1/2，带来2的价值。

实例分析（M=20，$(p_1,p_2,p_3) = (25,24,15)$，$(w_1,w_2,w_3) = (18,15,10)$）

$p_1 > p_2 > p_3$



得到的解：$(x_1, x_2, x_3)=（1, 2/15, 0）$

$\sum p_i x_i = 28.2$ ，仅为次优解，非最优解。**Why？**

分析：为什么以目标函数作为度量标准没能获得最优解？

尽管背包的价值每次得到了最大的增加，但背包容量也过快地被消耗掉了，从而不能装入“更多”的物品。

（2）以重量作为度量

解题思路：让背包容量尽可能慢地被消耗，从而可以尽可能多地装入一些物品。

处理规则：以重量作为度量，

- 按物品重量的非降次序将物品装入到背包；
- 如果正在考虑的物品放不进去，则只取其一部分装满背包即可；

实例分析（M=20，$(p_1,p_2,p_3) = (25,24,15)$，$(w_1,w_2,w_3) = (18,15,10)$）

$w_3 < w_2 < w_1$

M=20 $\xrightarrow{-10}$ ΔM=10 $\xrightarrow{-10}$ ΔM=0

$\begin{cases} P_3=15 \\ W_3=10 \end{cases}$

$\begin{cases} P_2=24 \\ W_2=15 \end{cases}$ $\Rightarrow$ $10/15 \times 24 = 16$

得到的解：(x_1, x_2, x_3)=（0, 2/3, 1）

$\sum p_i x_i = 31$，仅为**次优解**，非最优解。**Why？**

分析， 为什么以重量作为度量也没能获得最优解？

尽管背包的容量每次消耗得最少，装入物品的“个数”多了，但价值没能“最大程度”地增加。

（3）最优度量标准的选择

解题思路：片面地考虑背包的价值增量和容量消耗都是不行的，应在背包价值的增长速率和背包容量消耗速率之间取得平衡。

进一步的考虑是，让背包发挥“最大的作用”，亦即，<u>让其每一单位容量都尽可能地装进最大可能价值的物品</u>。

处理策略：以已装入的物品的累计**价值与所用容量之比**为度量。

- 按物品单位价值（即 p_i/w_i 值）的非增次序将物品装入到背包；
- 如果正在考虑的物品放不进去，则只取其部分装满背包即可。

实例分析（M=20，$(p_1,p_2,p_3) = (25,24,15)$，$(w_1,w_2,w_3) = (18,15,10)$）

$p_2/w_2 > p_3/w_3 > p_1/w_1$

M=20 → -15 → **ΔM=5** → -5 → **ΔM=0**

$P_2=$**24**

$W_2=$**15**

$P_3=15$

$W_3=10$

→ **5/10**×15=**7.5**

得到的解：$(x_1, x_2, x_3)=(0, 1, 1/2)$

$\sum p_i x_i = 31.5$ ，为**最优解**。**Why?**

例 设有三件物品和一个背包，物品价值 (p_1,p_2,p_3) = (25,24,15)， 重量(w_1,w_2,w_3) = (18,15,10)；背包容量M=20。求该背包问题的解。

可行解如下：

	(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$	
①	(1/2,1/3, 1/4)	16.5	24.25	//没有装满背包
②	(1, 2/15,0)	20	28.2	//以**价值**作为度量标准
③	(0, 2/3, 1)	20	31	//以**重量**作为度量标准
④	(0, 1, 1/2)	20	31.5	//以**单位重量价值**作为度量标准

贪心策略的基本要素

贪心算法总是作出在当前看来最好的选择。也就是说，贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的**局部最优**选择。

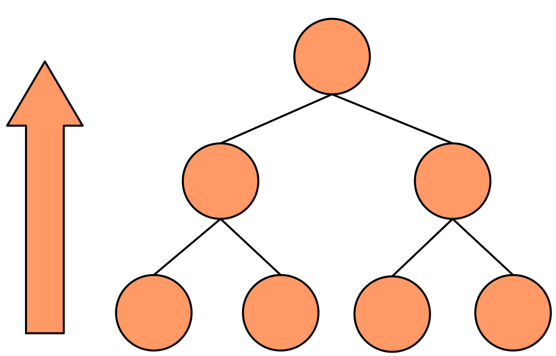
可以用贪心算法求解的问题一般具有2个重要性质：**贪心选择性质**、**最优子结构性质**。

1、贪心选择性质

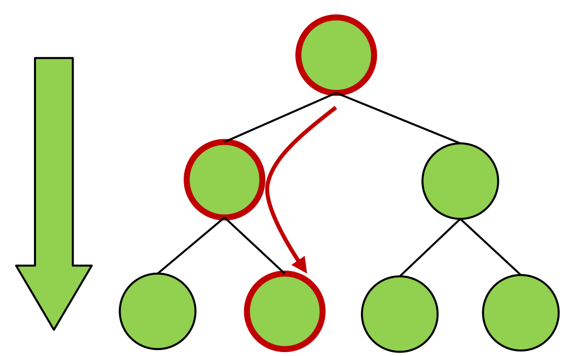
贪心选择性质是指所求问题的**整体最优解**可以通过一系列**局部最优的选择**，即贪心选择，来达到。

动态规划算法通常以自底向上的方式解各子问题，而**贪心算法**则通常以自顶向下的方式进行，以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为规模更小的子问题。

动态规划：



贪心算法：



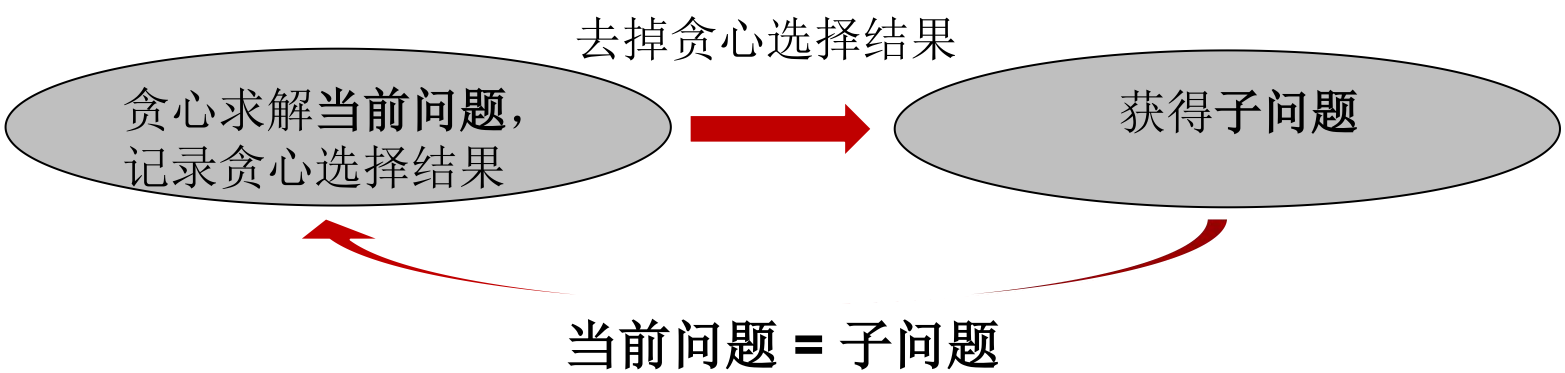
2、最优子结构性质

当一个问题的最优解包含其子问题的最优解时，称此问题具有**最优子结构性质**。

贪心策略的基本要素

1. **贪心选择性质，**整体最优解可通过一系列贪心选择来达到。

2. **最优子结构，**问题最优解包含子问题的最优解。



- 贪心法示例
- 贪心策略的求解方法
- **哈夫曼编码**

Example 3. Huffman codes

Motivation: You have a 100,000-character data file F, with only 6 characters {a, b, c, d, e, f}. You want to have a way to encode them to save space (remember that at the bottom-most level, everything is binary).

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000
Fixed-length	000	001	010	011	100	101

Example 3. Huffman codes

Motivation: You have a 100,000-character data file F, with only 6 characters {a, b, c, d, e, f}. You want to have a way to encode them to save space (remember that at the bottom-most level, everything is binary).

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000
Fixed-length	000	001	010	011	100	101
Variable-length	0	101	100	111	1101	1100

Example 3. Huffman codes

Motivation: You have a 100,000-character data file F, with only 6 characters {a, b, c, d, e, f}. You want to have a way to encode them to save space (remember that at the bottom-most level, everything is binary).

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000
Fixed-length	000	001	010	011	100	101

Cost: 100,000 x 3 = 300,000 bits

	a	b	c	d	e	f
Variable-length	0	101	100	111	1101	1100

Example 3. Huffman codes

Motivation: You have a 100,000-character data file F, with only 6 characters {a, b, c, d, e, f}. You want to have a way to encode them to save space (remember that at the bottom-most level, everything is binary).

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000
Fixed-length	000	001	010	011	100	101

Cost: 100,000 x 3 = 300,000 bits

	a	b	c	d	e	f
Variable-length	0	101	100	111	1101	1100

Cost: 45000x1+13000x3+12000x3+16000x3+9000x4+5000x4

= 224,000 bits **How much space been saved? 25%!**

Example 3. Huffman codes

Motivation: You have a 100,000-character data file F, with only 6 characters {a, b, c, d, e, f}. You want to have a way to encode them to save space (remember that at the bottom-most level, everything is binary).

So we want to design an optimal variable-length codes.

Example 3. Huffman codes

Motivation: You have a 100,000-character data file F, with only 6 characters {a,b,c,d,e,f}. You want to have a way to encode them to save space (remember at the bottom-most level, everything is binary).

So we want to design an optimal variable-length codes.

Prefix codes: no codeword is a prefix of some other codeword. Easy to encode and decode, no ambiguity.

Example. c · d · f · a=100 · 111 · 1100 · 0=10011111000

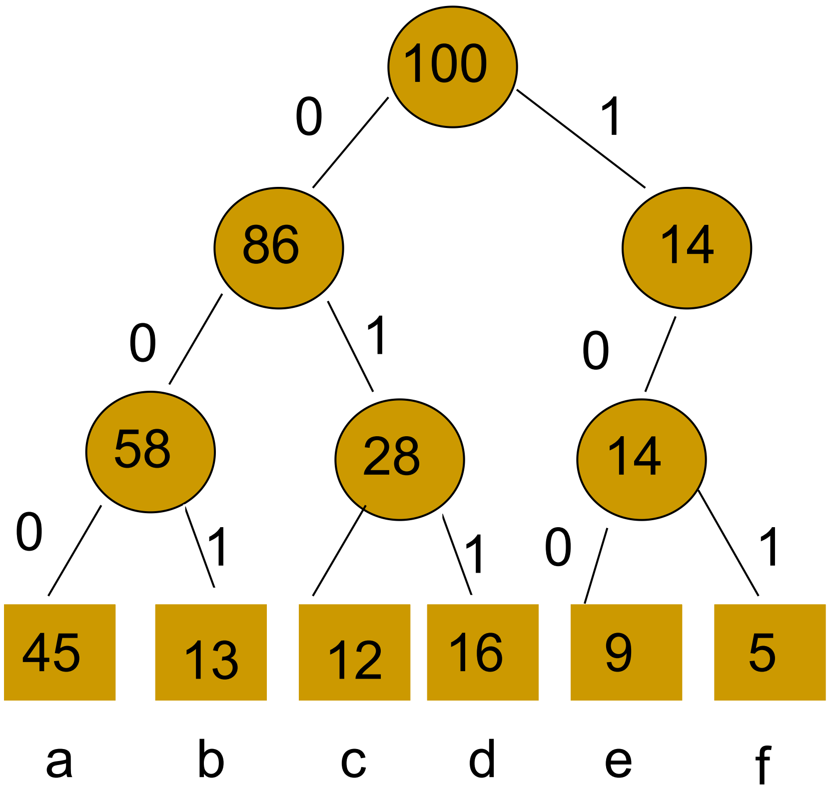
Example 3. Huffman codes

Prefix codes: no codeword is a prefix of some other codeword. Easy to encode and decode, no ambiguity.

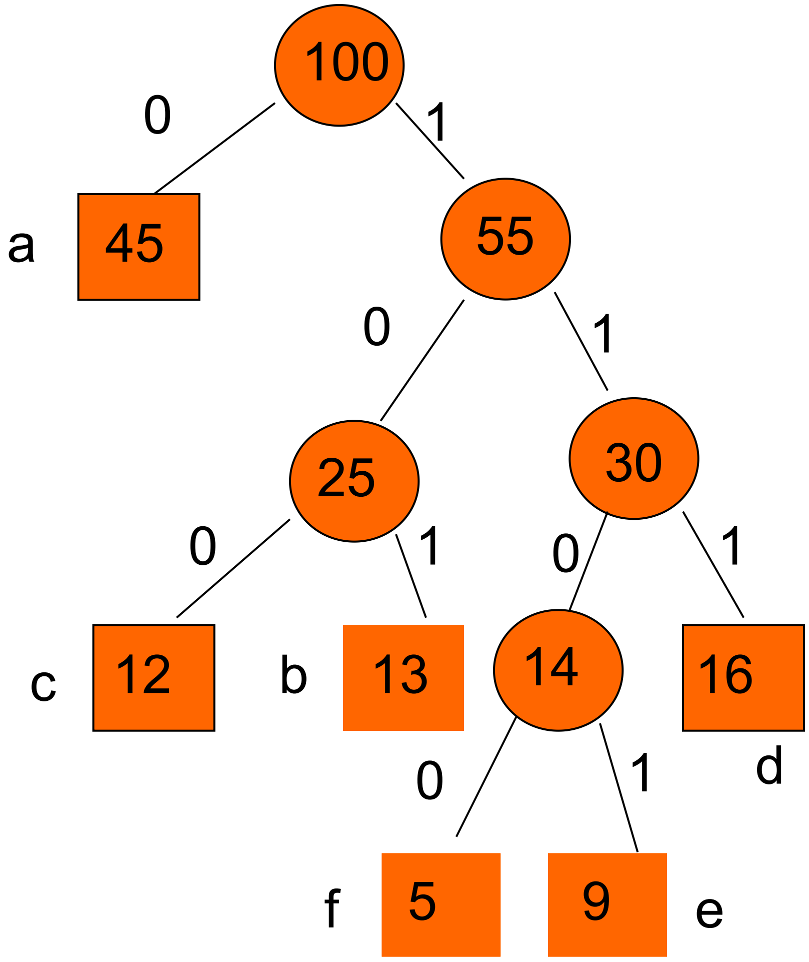
Example. c · d · f · a=100 · 111 · 1100 · 0=10011111000

We usually use a **binary tree** to represent the prefix codes, its leaves are the given characters. The binary codeword for a character is the path from the root to it, where 0 means go to left child and 1 means go to right child.

Example 3. Huffman codes

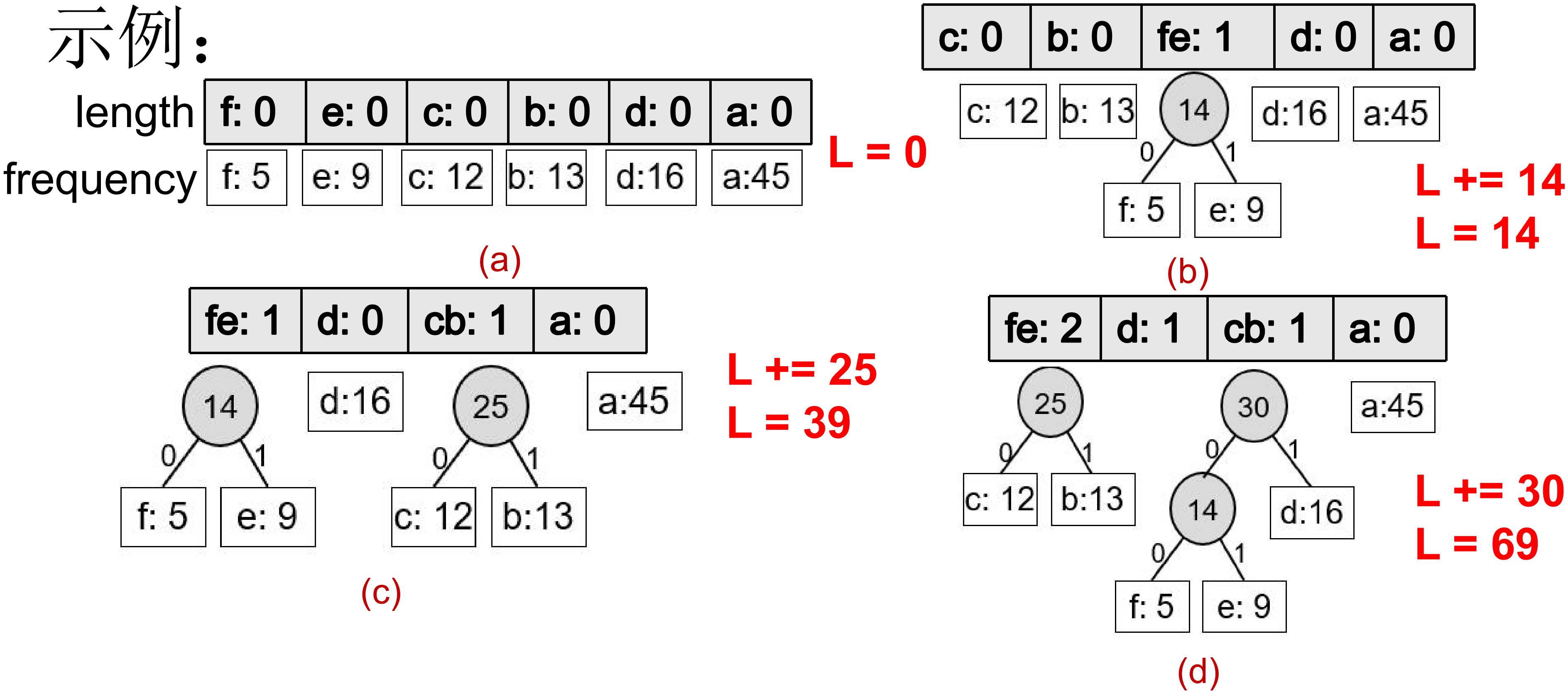


Fixed-length codeword

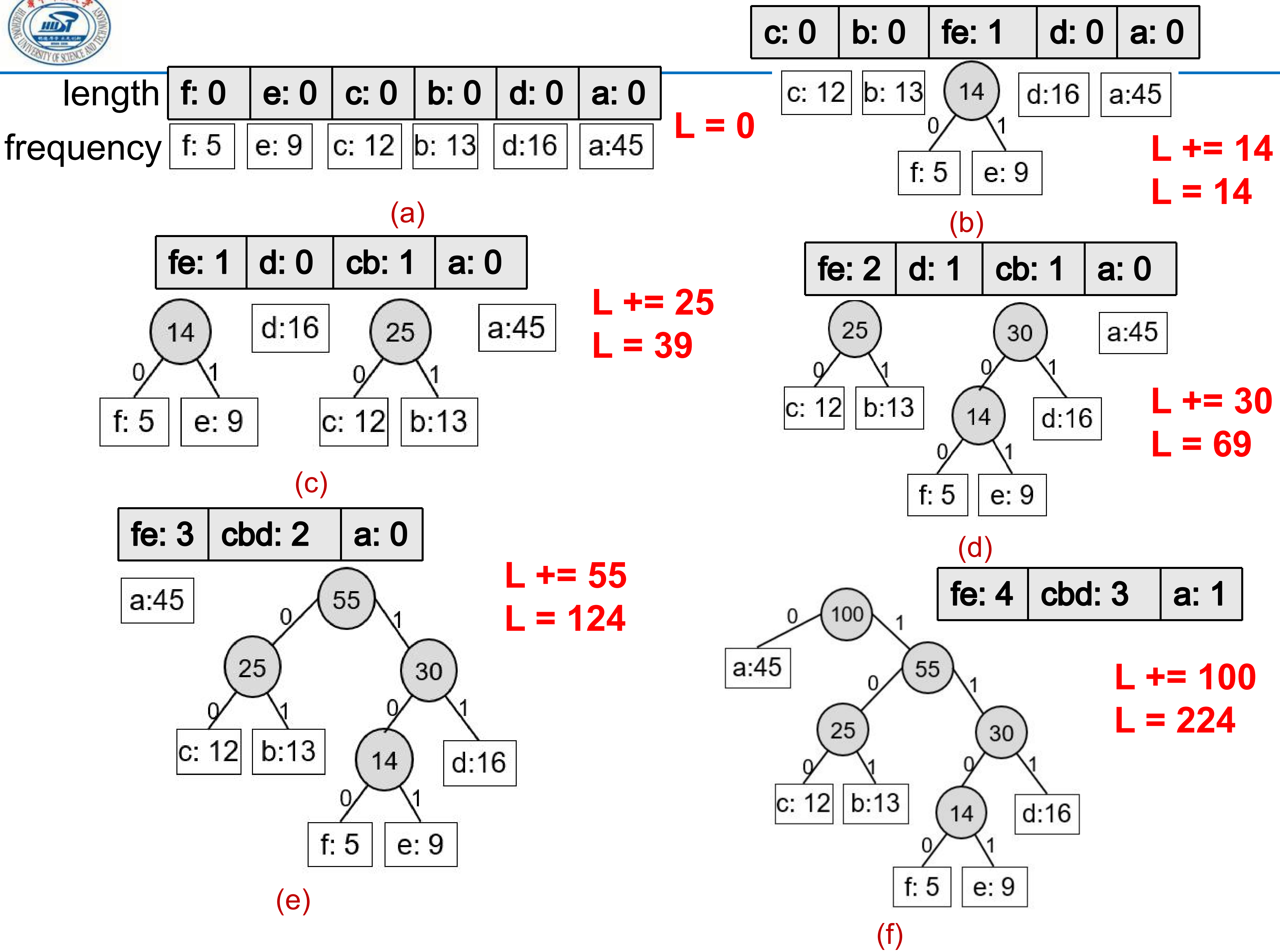


Variable-length codeword

示例：

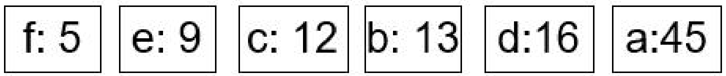


length
f: 0 | e: 0 | c: 0 | b: 0 | d: 0 | a: 0
frequency
f: 5 | e: 9 | c: 12 | b: 13 | d:16 | a:45
L = 0
(a)
c: 0 | b: 0 | fe: 1 | d: 0 | a: 0
c: 12
b: 13
14
0
1
f: 5
e: 9
d:16
a:45
L += 14
L = 14
(b)
fe: 1 | d: 0 | cb: 1 | a: 0
14
0
1
f: 5
e: 9
d:16
25
0
1
c: 12
b:13
a:45
L += 25
L = 39
(c)
fe: 2 | d: 1 | cb: 1 | a: 0
25
0
1
c: 12
b:13
30
0
1
14
0
1
f: 5
e: 9
d:16
a:45
L += 30
L = 69
(d)

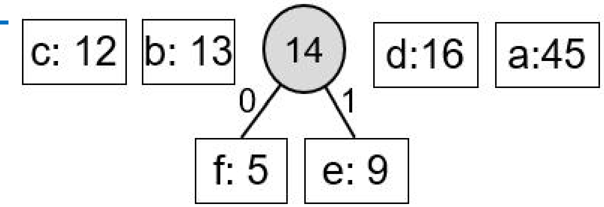


length
f: 0 | e: 0 | c: 0 | b: 0 | d: 0 | a: 0
frequency
f: 5 | e: 9 | c: 12 | b: 13 | d:16 | a:45
L = 0
(a)
c: 0 | b: 0 | fe: 1 | d: 0 | a: 0
c: 12 | b: 13 | 14 | d:16 | a:45
f: 5 | e: 9
L += 14
L = 14
(b)
fe: 1 | d: 0 | cb: 1 | a: 0
14 | d:16 | 25 | a:45
f: 5 | e: 9 | c: 12 | b:13
L += 25
L = 39
(c)
fe: 2 | d: 1 | cb: 1 | a: 0
25 | 30 | a:45
c: 12 | b:13 | 14 | d:16
f: 5 | e: 9
L += 30
L = 69
(d)
fe: 3 | cbd: 2 | a: 0
a:45 | 55 | 25 | 30
c: 12 | b:13 | 14 | d:16
f: 5 | e: 9
L += 55
L = 124
(e)
fe: 4 | cbd: 3 | a: 1
100 | a:45 | 55 | 25 | 30
c: 12 | b:13 | 14 | d:16
f: 5 | e: 9
L += 100
L = 224
(f)

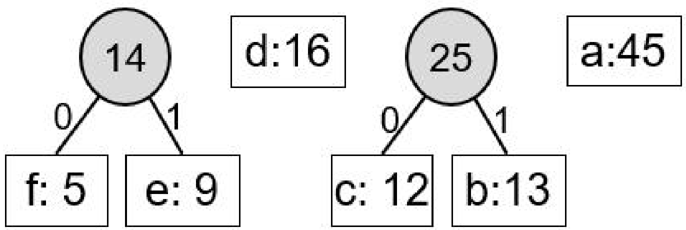
Q：如何体现贪心算法的两大性质？



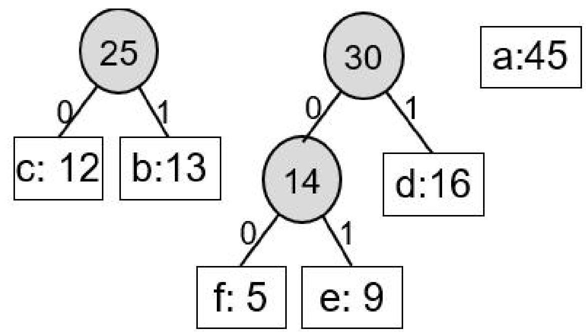
(a)



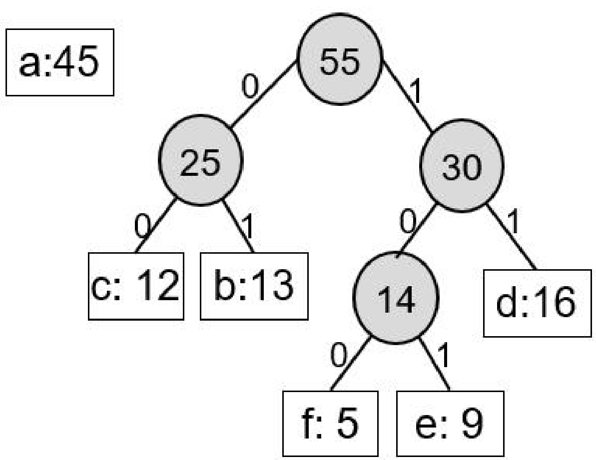
(b)



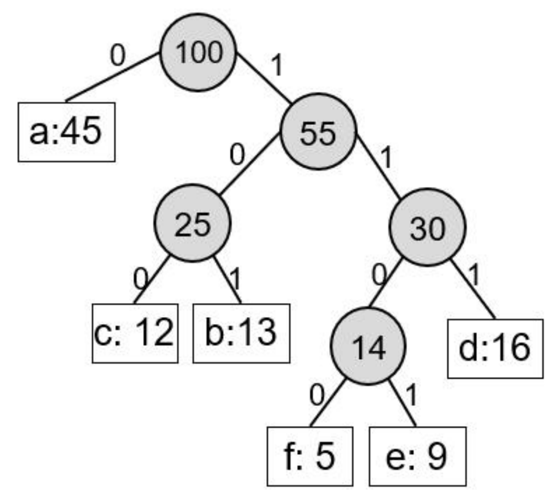
(c)



(d)



(e)



(f)

Example 3. Huffman codes

Huffman(C)

1. n ← |C|
2. Q ← C //Q is a priority queue, keyed on frequency f
3. for i=1 to n-1
4. z ← Allocate-node()
5. left[z]←x←Extract-Min(Q)
6. right[z]←y←Extract-Min(Q)
7. f[z] ← f[x]+f[y]
8. Insert(Q,z)
9. Return Extract-Min(Q) //now we have the binary tree

Example 3. Huffman codes

```
Huffman(C)
n ← |C|
Q ← C  //Q is a priority queue, keyed on frequency f
for i=1 to n-1
    z ← Allocate-node()
    left[z]←x←Extract-Min(Q)
    right[z]←y←Extract-Min(Q)
    f[z] ← f[x]+f[y]
    Insert(Q,z)
Return Extract-Min(Q) //now we have the binary tree
```

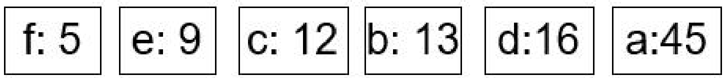
What is the running time?

Example 3. Huffman codes

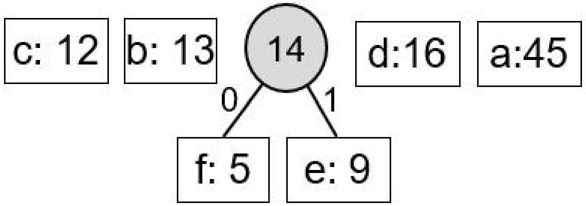
Huffman(C)

1. n ← |C|
2. Q ← C //Q is a priority queue, keyed on frequency f
3. for i=1 to n-1
4. z ← Allocate-node()
5. left[z]←x←Extract-Min(Q)
6. right[z]←y←Extract-Min(Q)
7. f[z] ← f[x]+f[y]
8. Insert(Q,z)
9. Return Extract-Min(Q) //now we have the binary tree

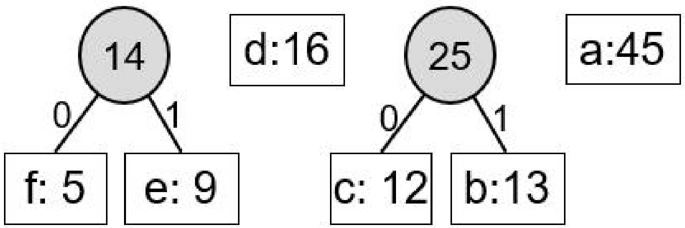
What is the running time? **O(nlgn)**


f: 5
e: 9
c: 12
b: 13
d:16
a:45

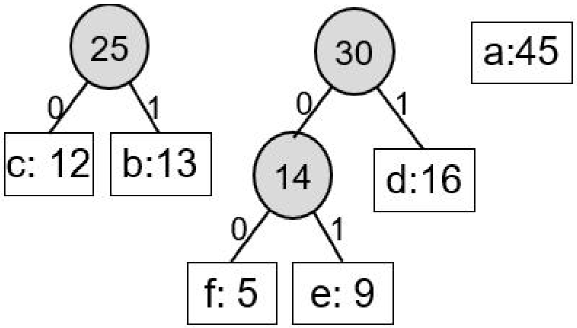
(a)


c: 12
b: 13
14
d:16
a:45
0
1
f: 5
e: 9

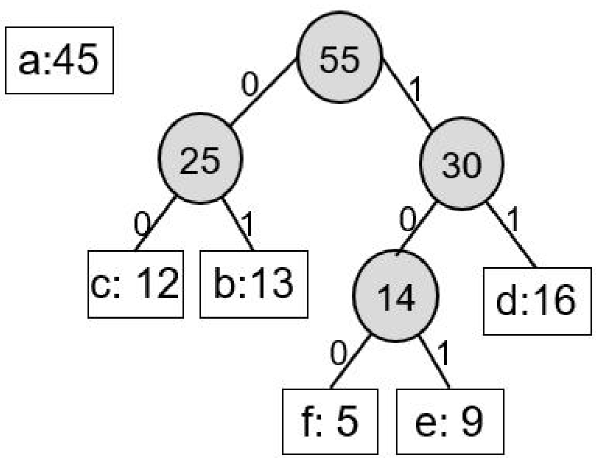
(b)


14
d:16
25
a:45
0
1
0
1
f: 5
e: 9
c: 12
b:13

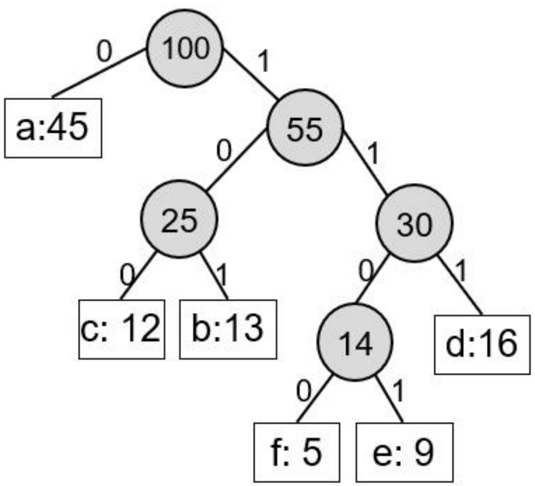
(c)


25
30
a:45
0
1
0
1
c: 12
b:13
14
d:16
0
1
f: 5
e: 9

(d)


a:45
55
0
1
25
30
0
1
0
1
c: 12
b:13
14
d:16
0
1
f: 5
e: 9

(e)


0
100
1
a:45
55
0
1
25
30
0
1
0
1
c: 12
b:13
14
d:16
0
1
f: 5
e: 9

(f)

关于哈夫曼编码的讨论：

- **几个关键词**

  无损编码，可变长编码，前缀码

- **相关约束**

  预处理，扫描得到频次集

  编码不唯一，但长度相同

  解码时，需要压缩后的结果，以及码表

几个需要思考的问题：

- 贪心解一定是问题的最优解吗？

答案：不一定！

- 度量标准怎么选？

答案：具体问题具体分析。直接将目标函数作为度量标准不一定能够得到问题的最优解。

- 贪心方法求解问题的关键?

答案：选取能够得到问题最优解的度量标准。

- ***如何求得人生最优解?***

拓展——技术角度

围棋每一步可能走法约250种，平均下一盘棋要走150步，$250^{150} \approx 10^{360}$

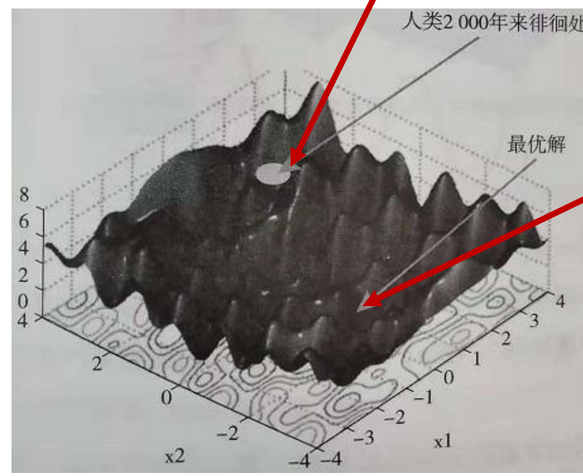
天才的哽咽

2016年，AlphaGo 4:1 战胜李世石

2017年，柯洁 0:3 完败AlphaGo

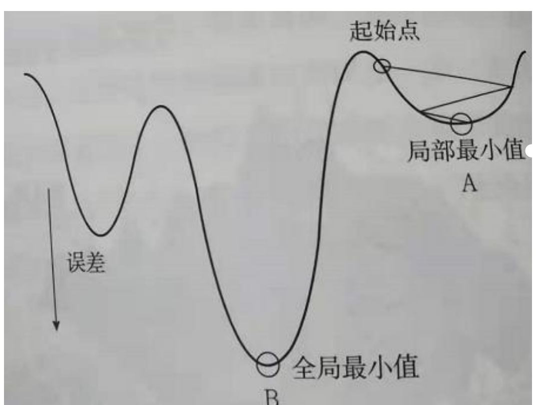
AlphaGo的“上帝视角”

人类2000年来徘徊处

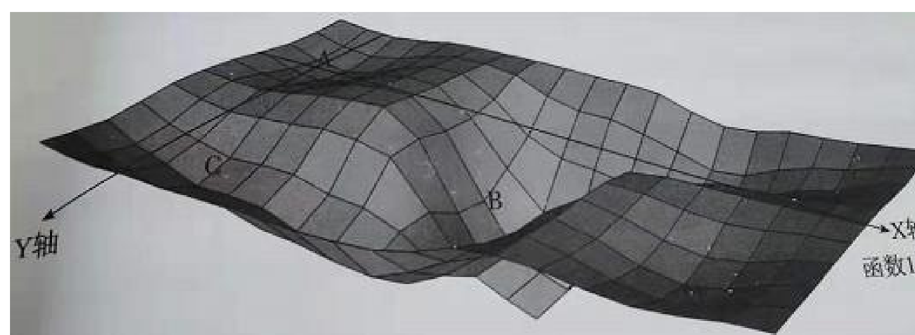


最优解

局部最优：没到山底怎么办



维度扩展



拓展——国家层面

America first

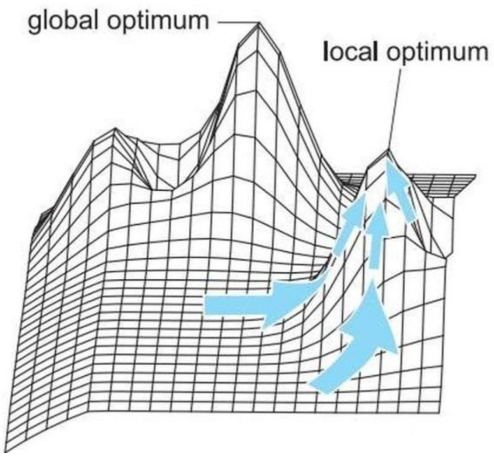
VS

人类命运共同体

贪心策略追求局部最优，却损害全局，可取吗？

拓展——个人发展

警惕局部最优陷阱

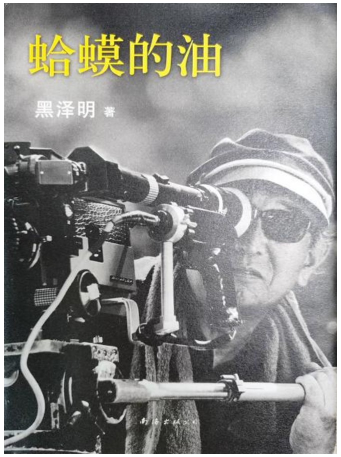


所有积累都有用

推荐：

《蛤蟆的油》

黑泽明（著）



迷路

“像个无头苍蝇，到处乱撞，想找到一条出路。”

“我贪婪地往头脑里灌输美术、文学、戏剧、音乐和电影方面的知识，为了自己有个用武之地，我一直彷徨不已。”

（黑泽明：第一位获奥斯卡终身成就奖的亚洲电影人）

本章作业

Question1:

Suppose there are characters to be encoded: a, b, c, d, e, f, g, h. Their corresponding frequencies are shown in the following table.

Character	a	b	c	d	e	f	g	h
Frequency	1	1	2	3	4	5	13	10

a) Please draw the Huffman coding tree (the left child is the smaller one, encoded as '0') and write out the code for each character.

b) Decode a sequence 001010011.

2. 阅读内容：《算法导论》16.1～16.3

Thank You!

Q&A