# 运筹学通论I


**胡晓东**

应用数学研究所

中国科学院数学与系统科学研究院

**http://www.amt.ac.cn/member/huxiaodong/index.html**


# 5. 组合优化-算法设计技巧

**精确算法**

分而治之 (搜索、排大小序、旅行商)
动态规划 (最短路、三角剖分、背包)
分支定界 (整数线性规划、旅行商、工件排序)

**近似算法或启发式算法**

贪婪策略 (最小生成树、最大可满足、背包、
顶点覆盖、独立集、旅行商)
局部搜索(最大匹配、旅行商、最大割)
序贯法 (工件排序、装箱、顶点着色)
整数规划法 (顶点覆盖、最大可满足、最大割)
随机方法 (最小割、最大可满足、顶点覆盖)
在线算法 (页面调度、*k*-服务器、工件排序、装箱)
不可近似 (最大团、背包、旅行商、装箱、连通控制集等)

# 5.9 经典例子

到目前为止，当我们考虑如何设计求解一个优化问题的算法的时候，都是假设这个优化问题中的每一个（变）量或者参数都已经知道了。然而，在我们处理实际问题时，常常会发现有些问题的（变）量或者参数的具体数值，事先并不知道，但是我们还是需要即时（**online**）做出决定或者选择。

我们考虑一个经典的例子：假如这个冬天你决定去滑雪。那么去之前，你面临两种选择：

- 每次去滑雪时，花 **1$** 钱去租一套滑雪用具。
- 花***T*$** 钱去买一副滑雪用具，这样整个冬天你就可一直用它。

显然，如果你知道能会花 $L$ 次，那么当 $L < T$ 时，你就每次去租；否则 $L \geq T$，你就买一套。

# 5.9 经典例子（续一）


然而，你不太确定你能滑多少次。
是买？
是租？

一个可能的（在线）方法是：
首先，滑 ***k*** 次且租 ***k*** 次；
然后，在去滑第 ***k*+1** 次时，再买。


# 5.9 经典例子（续二）

然而，我们怎么选定这个 $k$ 呢？更一般的讲，我们如何评价在线算法的优劣呢？很自然地想法是估计一下你能滑几次，或者给出滑 $n$ 次的概率分布，然后，再算一下在线算法所产生的费用的期望值。然而，一般来讲，这个估计值或者概率分布是非常难确定的。 不过，我们可以将在线算法所产生的费用与**离线算法**（假设所有信息都知道）所产生的费用进行比较和分析，这样的方法被称为**竞争分析**。

设 **A** 是一个**在线算法**，$S$ 是**在线**(最小化)**问题**的一个变量输入序列，用 $c_A(S)$ 记算法**A** 应对$S$ 所产生的费用，用 $c_{opt}(S)$ 记**最优离线算法**应对（一次给定的）$S$ 所产生的费用。若对于任意的 $S$，都有

$$c_A(S) \le \alpha\, c_{opt}(S)$$

则称在线算法 **A** 是一个 $\alpha$-**竞争算法**。

# 5.9 经典例子（续三）

让我们再用竞争分析的方法来考虑如何处理滑雪时租-买雪橇问题。最简单的方法是：第一天就买滑雪用具（即 $k = 0$），花费 $T\$$。注意，如果（事先知道）只滑了1 次，即 $L = 1$，那么最优的策略是租，花费 1\$。竞争比是 $T$，这不是一个常数！

一个稍微复杂一点的方法是：前 $T - 1$ 天租，然后再买。

- 如果 $L < T$，那么即使事先知道此信息，最优策略也是租。
- 如果 $L \geq T$， 那么在线算法的费用是 $(2T - 1)\$$;
  当事先知道此信息，最优策略是买，费用为 $T\$$.

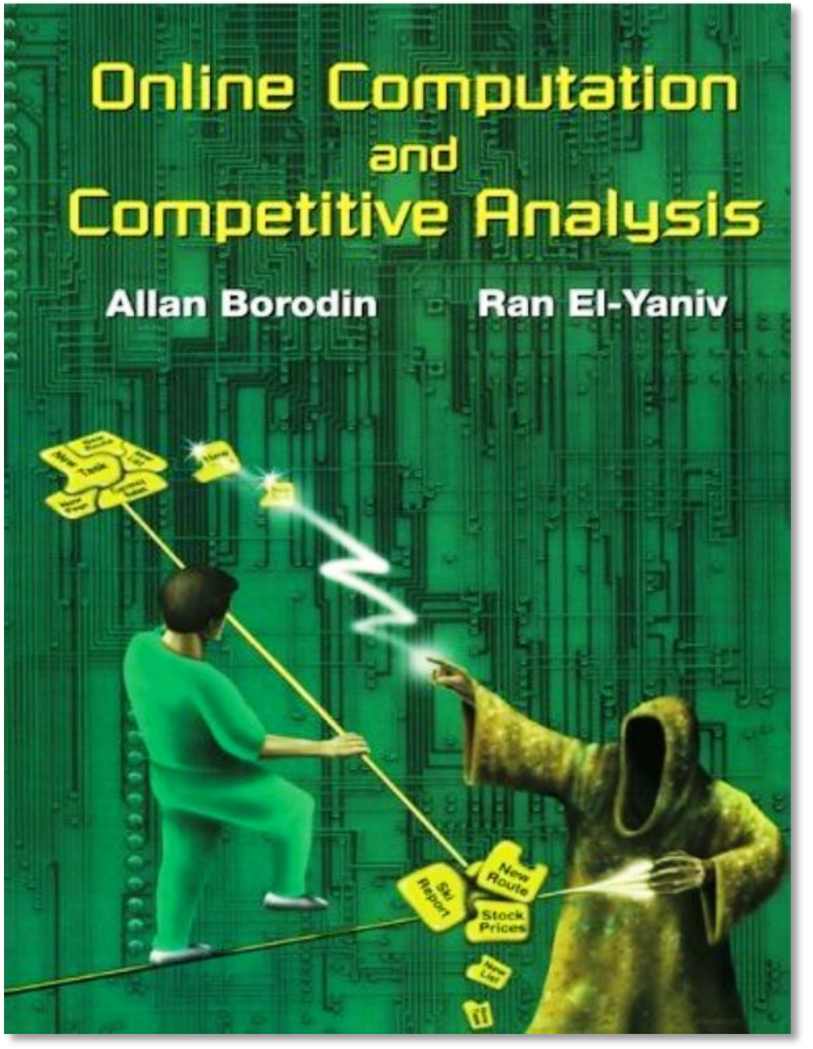
因此上述在线算法是一个 $(2 - 1/T)$-竞争算法。实际上这是最优的在线策略（练习：证明这一结论）。

**思考题：**可否用随机算法处理租-买雪橇问题？比如，以等概率决定租还是买；亦或以概率 $T/(1+T)$ 租，以概率 $1/(1+T)$ 租买。

# 5.9 经典例子（续四）

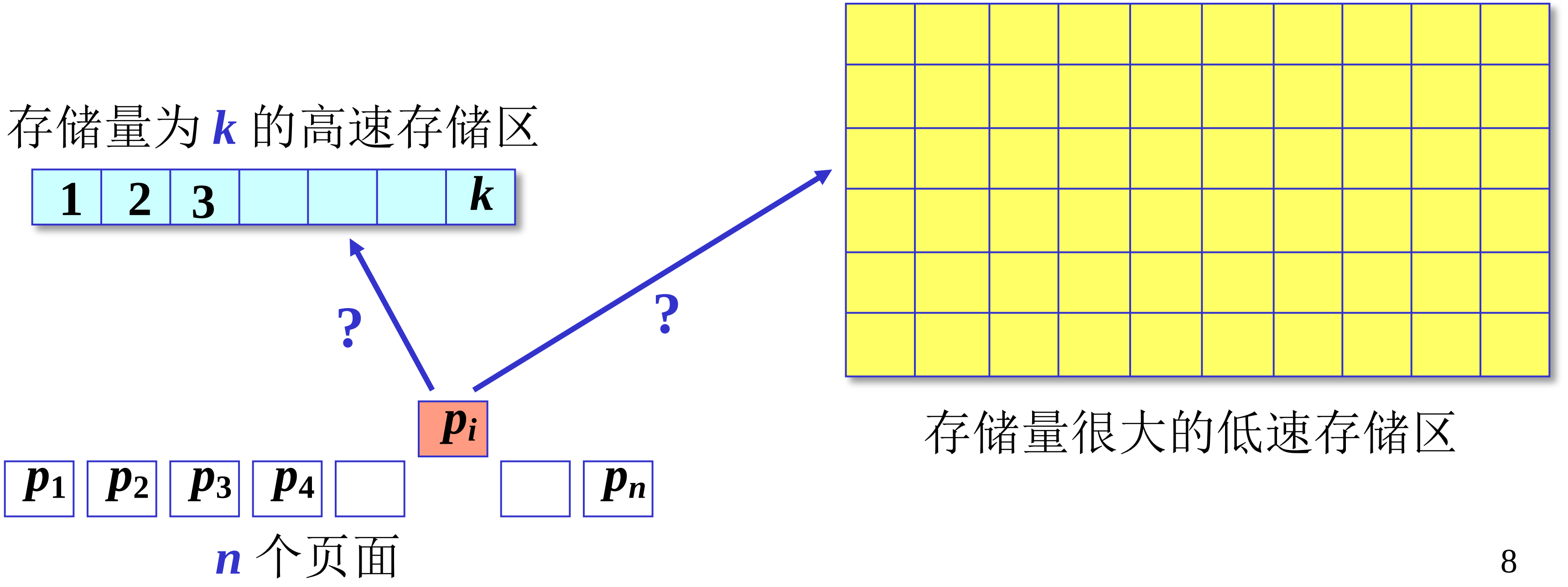
竞争分析的核心可以从博弈观点来解释：这是一个二人对局，一方是在线玩家，另一方是邪恶的庄家。庄家一次出一张“牌”，玩家看到一张牌以后，就要做出选择（他不知道庄家以后会出什么牌）。

庄家出牌的目的是，使得玩家做出的一系列选择所产生的成本越高越好，同时自己所需要的成本越低越好。

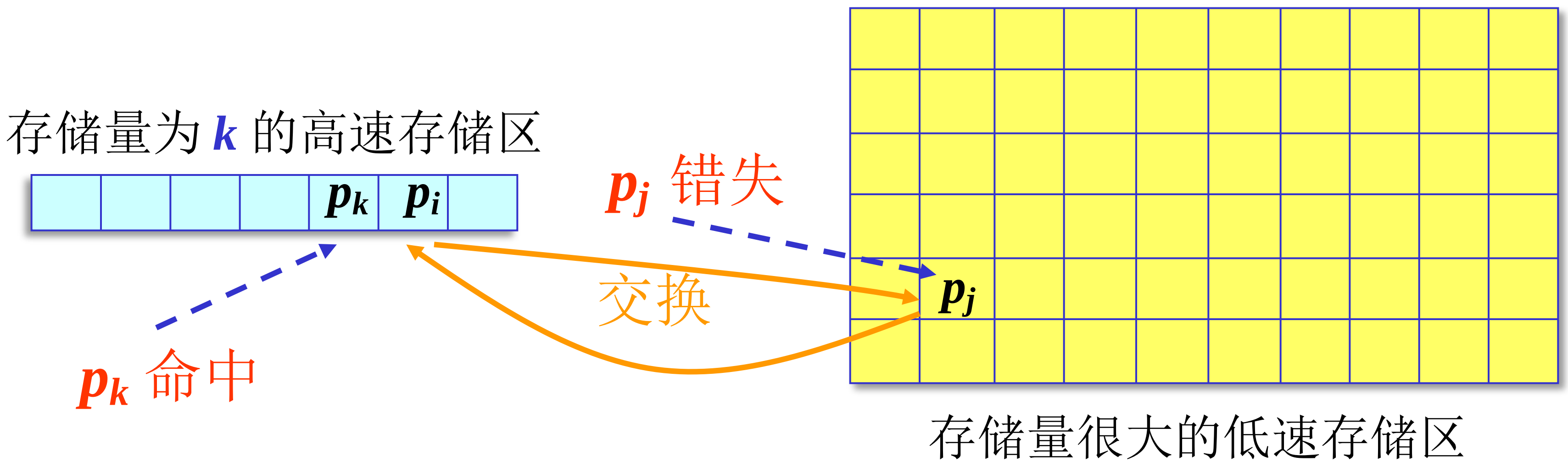


# 5.9 页面调度问题

首先考虑虚拟存储系统设计中的一个核心问题：页面调度。在系统的每一层中，都有一定量的存储单元，称为页面。第一层是低速存储区，它可以存储 $n$ 个页面 $P = \{ p_1, p_2, \ldots , p_n \}$，第二层是高速存储区，它可以存贮 $P$ 中的任意一个 $k$-子集，其中 $k < n$。



# 5.9 页面调度问题（续一）




当请求调用页面 $p_i$ 时，它必须是放在高速存储区中。若 $p_i$ 已在高速存储区中，那么就称该请求“命中”，系统不需要进行任何操作；否则称其为“错失”，系统必须将页面 $p_i$ 从低速存储区拷贝到高速存储区中的某个单元。为了存储页面 $p_i$，系统需要决定将高速存储区中的某一个页面移至低速存储区，称为页面“交换”。**页面调度问题**是，给定一系列调用页面请求 $R = \langle r_1, r_2, \ldots, \rangle$，如何在产生“错失”时，进行相应的页面“交换”，使得所有请求都得到满足，且进行交换的次数最少。

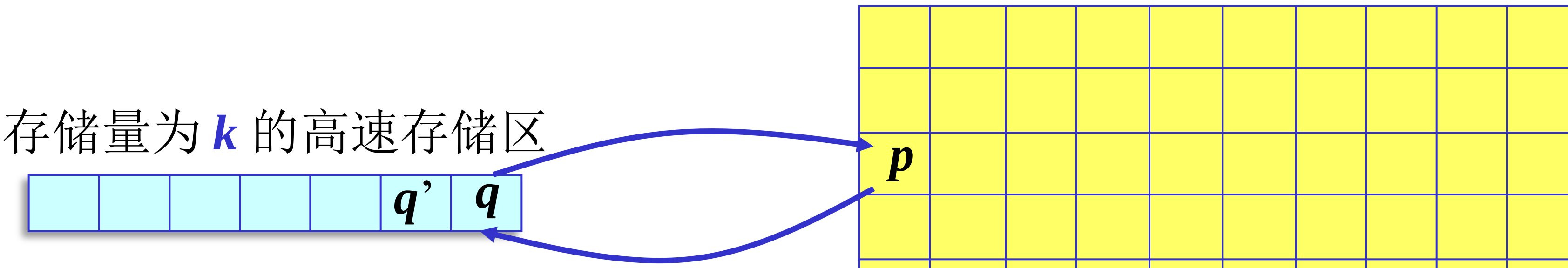
# 5.9 页面调度问题（续二）

人们提出并研究过许多非常简单的在线调度页面的策略。首先，考虑这样一个策略，“新进先出”策略 $\mathbf{A}_{\mathrm{LIFO}}$：将最近一次放人高速存储区的页面交换出去。假定交替请求调用两个不同的页面 $p$ 和 $q$，即 $R = < r_1 = p, r_2 = q, r_3 = p, r_4 = q, \ldots >$，且在初始状态，页面 $p$ 在低速存储区，而页面 $q$ 在高速存储区。

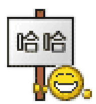
根据“新进先出”策略 $\mathbf{A}_{\mathrm{LIFO}}$，当请求调用页面 $p$ 时，系统会将它与页面 $q$ 交换，此后当请求调用页面 $q$ 时，系统又会将页面 $p$ 与其交换回来，这个过程将不断地重复。而最优的离线调度策略是，将页面 $p$ 与高速存储区中的页面 $q' \neq q$ 进行交换，此后，就不用再进行任何交换了。由此可知，对于任意一个常数 $\alpha$，“新进先出”策略 $\mathbf{A}_{\mathrm{LIFO}}$ 不是一个 $\alpha$-竞争策略。

# 5.9 页面调度问题（续三）

存储量为 $k$ 的高速存储区

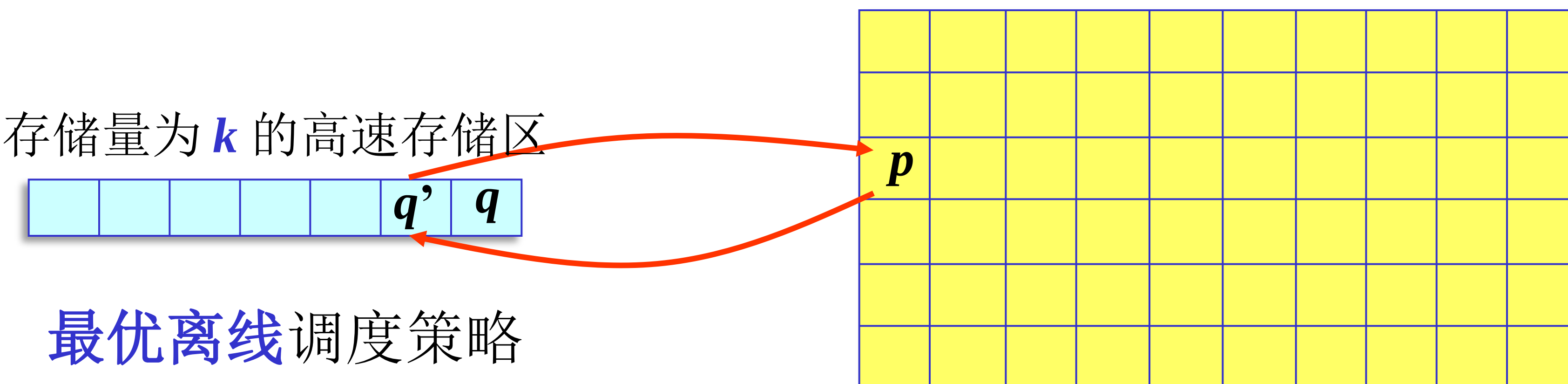



**“新进先出”** 调度策略




$$R = < r_1 = p, r_2 = q, r_3 = p, r_4 = q \; \ldots\ldots >$$

存储量为 $k$ 的高速存储区




**最优离线**调度策略

存储量很大的低速存储区

# 5.9 页面调度问题（续四）

我们下面再考虑另外一个简单的在线页面调度策略，“**最少调用**”策略 **A**$_{\text{LFU}}$：交换出去最少被调用的页面。例如，当请求调用页面 $r_{11}=d$ 时， 前面的页面调用请求如下，其中页面 $d$ 不在高速存储区中，

$r_1=a$, $r_2=b$, $r_3=a$, $r_4=c$, $r_5=b$, $r_6=a$, $r_7=a$, $r_8=b$, $r_9=c$, $r_{10}=a$

因为页面 $a$ 被调用了 **5** 次，页面 $b$ 被调用了 **3** 次，而页面 $c$ 被调用了 **2** 次，所以“最少调用”策略 **A**$_{\text{LFU}}$，系统将页面 $c$ 与页面 $d$ 进行交换。

**练习**. 证明：对于任意一个常数 $\alpha$， “最少调用”策略 **A**$_{\text{LFU}}$ 也不是一个 $\alpha$-竞争算法。

# 5.9 页面调度问题（续五）

最后，我们讨论另外一个简单的页面调度在线策略，“**最近最先**”策略 $A_{LRU}$：交换出这样一个页面它最近的一次调用是最早产生的。我们将证明这是一个最优的在线策略。假设，当请求调度页面 $r_{11} = d$ 时，已有的页面调度请求记录如下，其中页面 $d$ 不在高速存储区，

$$r_1=a,\ r_2=b,\ r_3=a,\ r_4=c,\ r_5=b,\ r_6=a,\ r_7=a,\ r_8=b,\ r_9=c,\ r_{10}=a$$

因为最近一次调用页面 $a$ 是在时刻**10**，最近一次调用页面 $b$ 和页面 $c$ 分别是在时刻 **8** 和时刻 **9**，$A_{LRU}$ 将页面 $b$ 与页面 $d$ 进行交换。

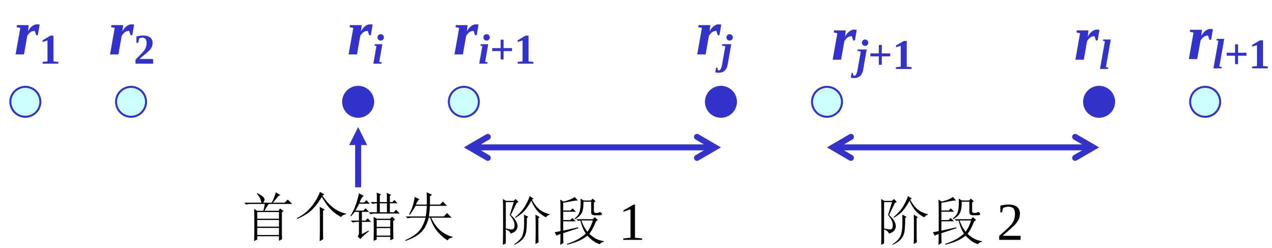
在下述定理的证明中，我们将所考虑的**在线算法**产生的费用与任意一个**离线算法**产生的费用的下界相比较。这是在分析算法性能时常用的一个典型技巧（不仅适用于在线算法，同样也适用于近似算法）。

# 5.9 页面调度问题（续六）

**定理 1** “最近最先”策略 $A_{LRU}$ 是一个 $k$-竞争算法，其中 $k$ 是高速存储区含有的单元数（一个单元可存储一个页面）。

**证明** 设 $A_{opt}$ 为最优离线策略。对于任意给定的页面调度请求系列 $R$，我们分析一下依据“最近最先”策略 $A_{LRU}$ 系统应如何调度页面。我们将这一过程分为不同的阶段，每一个阶段中恰好产生了 $k$ 次错失，且最后一次请求是个错失。或者表述为：第一阶段的最后一个请求 $r_j$ 满足

$$j = \min \{ t \mid \text{在处理 } r_{i+1}, \ldots, r_t \text{ 时，} A_{LRU} \text{ 产生了 } k \text{ 次错失} \}.$$



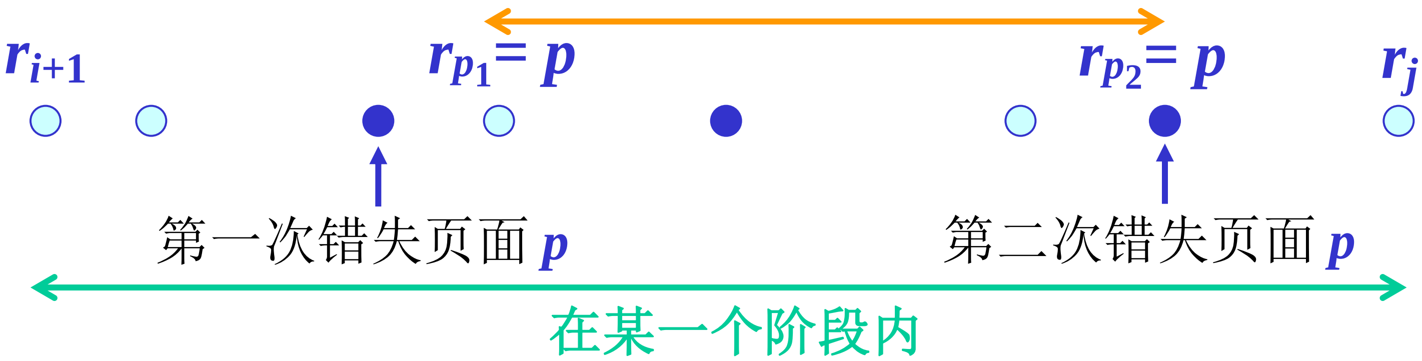
# 5.9 页面调度问题（续七）

易知，在每一个阶段，“最近最先”原则 $\mathbf{A}_{\mathrm{LRU}}$ 都产生了 $\boldsymbol{k}$ 次错失。下面我们来证明，最优离线策略 $\mathbf{A}_{\mathrm{opt}}$ 在每一个阶段至少产生一次错失。为此，我们分两种情况讨论。

**情形 1.** 在同一个阶段 $\mathbf{A}_{\mathrm{LRU}}$ 在处理请求调度页面 $\boldsymbol{p}$ 是产生了两次错失。此时，在第一次错失页面 $\boldsymbol{p}$ 后，$\boldsymbol{p}$ 被交换到高速存储区中，而后又被其他某个页面交换出了高速存储区，因为页面 $\boldsymbol{p}$ 是高速存储区中最近被调用的一个页面。因此，若 $\mathbf{A}_{\mathrm{LRU}}$ 在处理请求调度页面 $\boldsymbol{p}$ 是再次产生错失，则意味着其他 $\boldsymbol{k}$ 个页面，在调用页面 $\boldsymbol{r}_{\boldsymbol{p}_2}$ 请求发出之前，且在第一次错失页面 $\boldsymbol{p}$ 之后，就已经在高速存储区内了。因而，在这一个阶段，至少有 $\boldsymbol{k}+\mathbf{1}$ 个不同的页面被请求调用，这说明，最优离线策略 $\mathbf{A}_{\mathrm{opt}}$ 在处理这些页面调度请求时，至少会产生一次错失。

# 5.9 页面调度问题（续八）

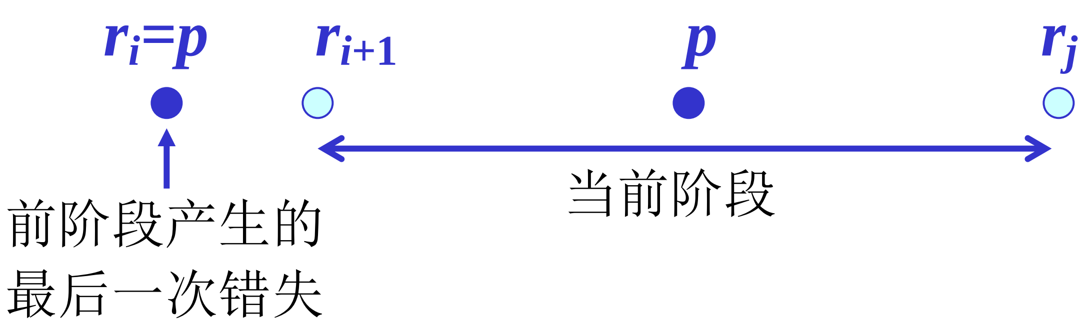
在高速存储区中的 $k$ 个页面在第二次错失页面 $p$ 时，就已经被请求调用了。




**情形 2**. 在某一个阶段内，“最近最先”策略 $A_{LRU}$ 产生了 $k$ 次不同的错失。此时，我们再针对该阶段内最后一次错失，假设在处理请求调度页面 $p$ 时产生，分两种子情形讨论。

# 5.9 页面调度问题（续九）

**情形 2.1** 在该阶段存在一次错失页面 ***p***，且在前一个阶段的最后一次错失也是在处理调用页面 ***p*** 请求时产生的。这种子情形与前一种情形十分相似。 实际上，在第二次错失页面 ***p*** 之前，除了页面 ***p***，一定还请求调用了 ***k*** 个页面，这样才能迫使交换页面 ***p***。因而，在该阶段内一定请求调用了至少 ***k*+1** 个不同的页面。



# 5.9 页面调度问题（续十）

**情形 2.2** 在前阶段不存在错失页面 $p$。此时，在此阶段开始前， 最优离线策略 $A_{opt}$ 一定置页面 $p$ 于高速存储区内（我们不必考虑其他 $k-1$ 个单元）。不过，注意，在当前阶段中，有 $k$ 个请求调用不同的页面，它们不包含页面 $p$。因而处理所有这些页面调度请求， 最优离线策略 $A_{opt}$ 必须交换 $p$。

在上面的分析中，我们已经证明：在“最近最先”策略 $A_{LRU}$ 产生 $k$ 次错失的时候，最优离线策略 $A_{opt}$ 至少会产生一次错失。最后，我们需要考虑最初的那次错失。回忆一下，策略 $A_{LRU}$ 和 $A_{opt}$ 开始面临和处理的请求调用页面是一样的。因此，当“最近最先”策略$A_{LRU}$ 产生第一次错失的时候，调用的页面不在高速存储区内，而在应用离线最优策略 $A_{opt}$ 时，这个页面也不会在高速存储区内。换言之，这两个策略都会产生一次错失。

# 5.9 页面调度问题（续十一）

**练习**** 考虑这样一个在线策略，“**先进先出**”策略$A_{FIFO}$：交换出在高速存储区时间最长的页面。应用与证明**定理1**的类似方法，证明“先进先出”策略$A_{FIFO}$也是一个$k$-竞争算法。

**练习 **** 考虑这样一个离线策略，“**最后到达**”策略$A_{LFD}$：当产生一个错失以后，将高速存储区中的这样一个页面交换出来，再次调用该页面的请求最后到达。证明这是一个最优离线算法。

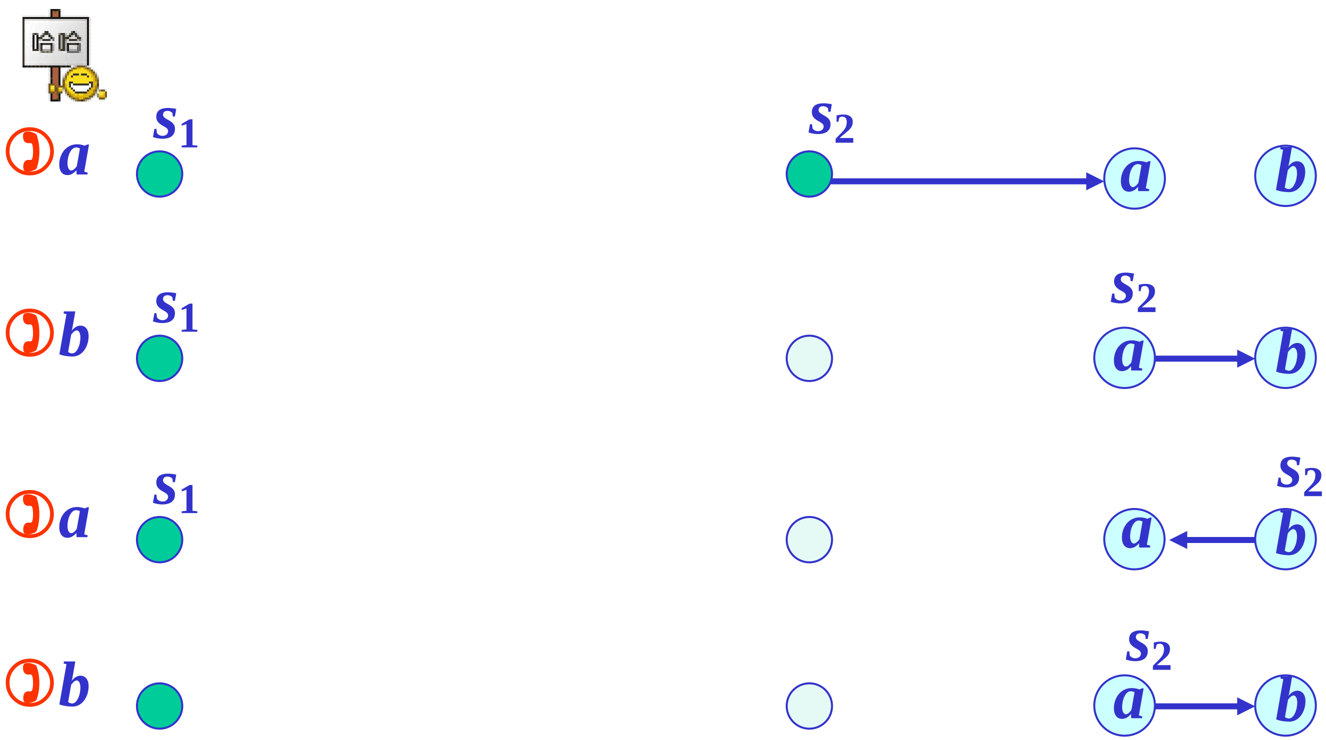
通过与最优离线策略“最后到达”$A_{LFD}$进行比较分析，我们还可以证明，“最近最先”策略$A_{LRU}$和“先进先出”策略$A_{FIFO}$的竞争比是最好可能的。

# 5.9 *k*-车问题

2008年北京奥组委交给我的一项任务：调度北京的 ***k*** 辆**110**警车。这些车停放在北京的街道上。当我接到一个报警电话，就要指派一辆警车去现场处理情况。在我指派警车时，我不知道（也无法知道）下一次哪里需要警车过去处理紧急情况。我该怎么调度使得这 ***k*** 辆警车行驶的总里程最小？


# 5.9 *k*-车问题（续一）

最简单的在线算法是**贪婪策略**：让距离事发地最近的警车去。我们用**竞争分析**法看看这是不是一个好的在线算法呢？不是！



# 5.9 $k$-车问题（续二）

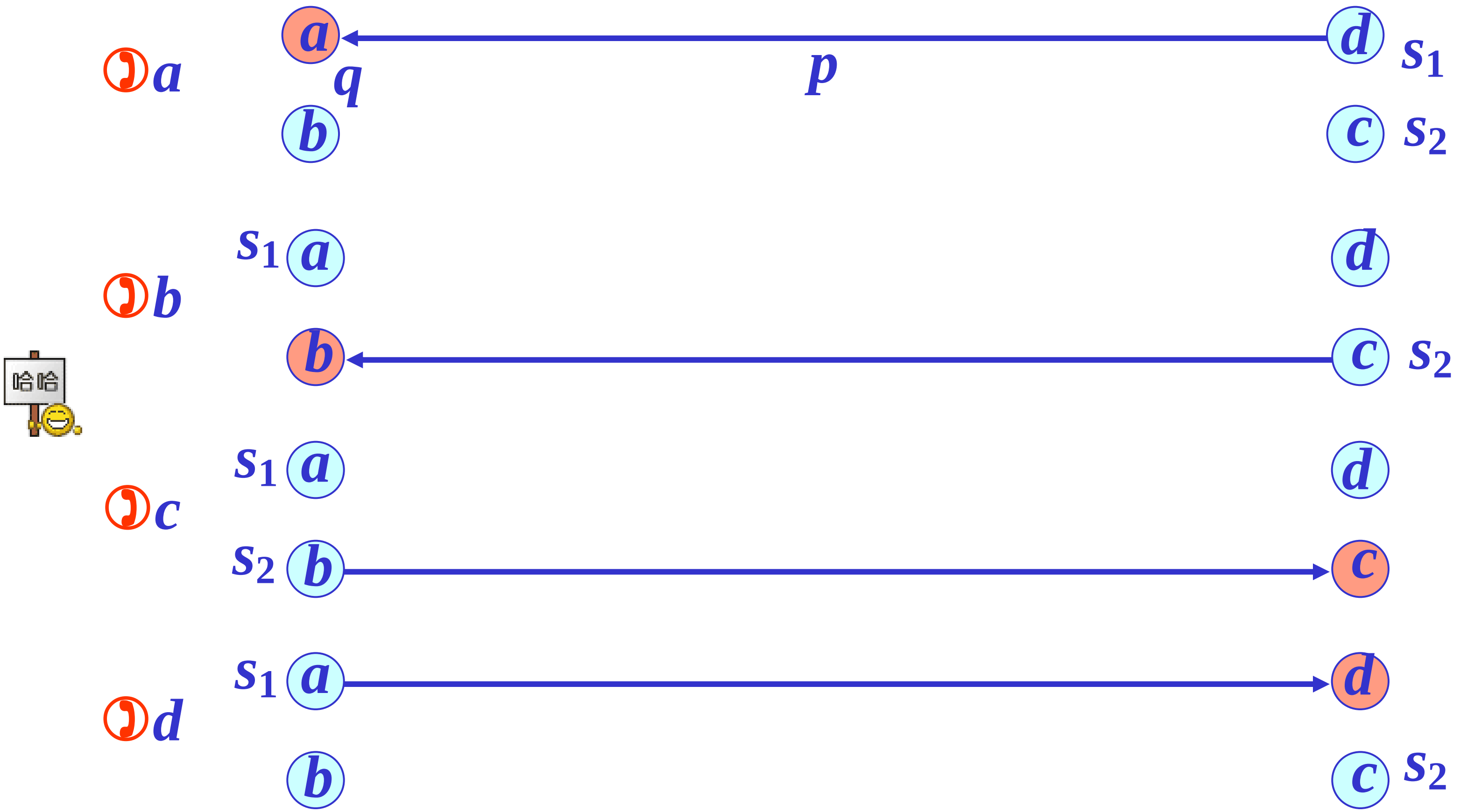
注意这个贪婪策略的最大缺点是： 总是调度一辆警车出现场，而另一辆警车总是休息。这就促使我们采用**公平的策略**，**平衡**每一辆警车的行驶距离：考虑每一辆车 $x$ 已经行驶的里程 $D(x)$，和执行本次任务 $r_i$ 需行驶的里程 $d(x, r_i)$，使得两项之和最小的警车 $s$ 去执行任务，亦即

$$D(s) + d(s, r_i) \equiv \min\{\, D(x) + d(x, r_i) \mid x \,\}。$$

这个策略可以应付前面的情况，但是它是否可以应付所有的情况吗？很遗憾，不能！

**练习** 证明：若要使得所有车辆行驶的路程之和最小，每一次只需调度一辆车。

# 5.9 *k*-车问题（续三）




**平衡策略**使得每一辆车都是水平行驶，每一次行驶的里程是 $p$。而**最优的 (离线) 策略**是：每一辆车都是垂直行驶，每一次行驶的里程是 $q$。 $q$ 可以比 $p$ 任意小。

# 5.9 $k$-车问题（续四）

非常有趣的是，若把平衡策略做出如下修改：

$$D(s) + 2d(s, r_i) \equiv \min\{\, D(x) + 2d(x, r_i) \mid x \,\}。$$

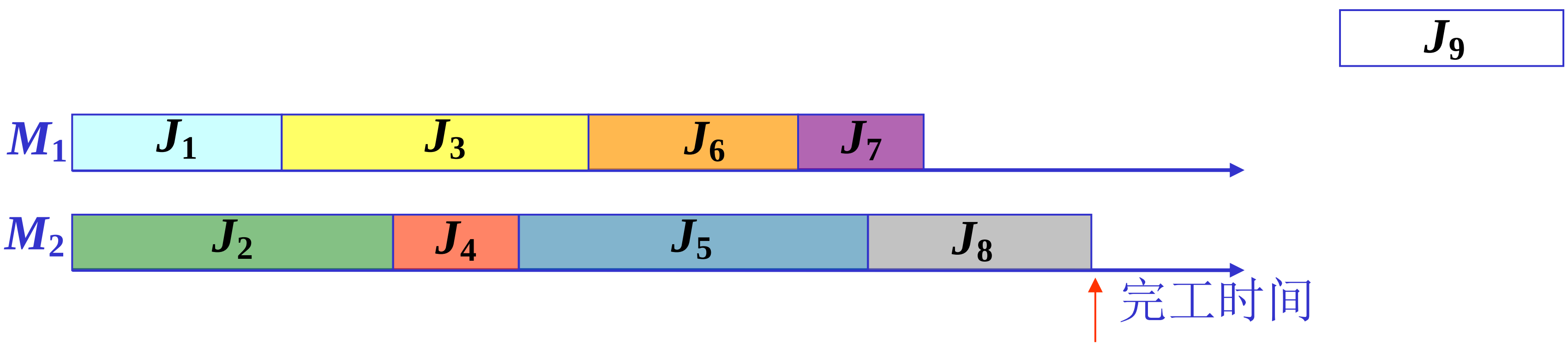
那么，就得到处理两辆警车调度问题的一个**10**-竞争在线算法。

- 已经证明在距离空间上不存在 $\alpha$-竞争算法，其中 $\alpha < k$。
- 已经证明在基于树结构的距离空间上，存在 $k$-竞争算法。
- 已经证明存在 $(2k - 1)$-竞争算法。
- 人们猜测存在 $k$-竞争算法。

# 5.9 排序问题

2008年北京奥组委交给我的另一项任务是：在奥运村**调度**来注册的运动员。每当一个运动员来注册，需要给他/她在 $k$ 个注册台中指定一个，为其办理相关手续。可不知道运动员是什么时候来，在某段时间内有多少位运动员来注册；且每位运动员注册需要的时间可能不一样；目标是在最短的时间内帮助所有运动员注册完毕，即最后一个运动员注册完的时间最早。

怎么调度呢？

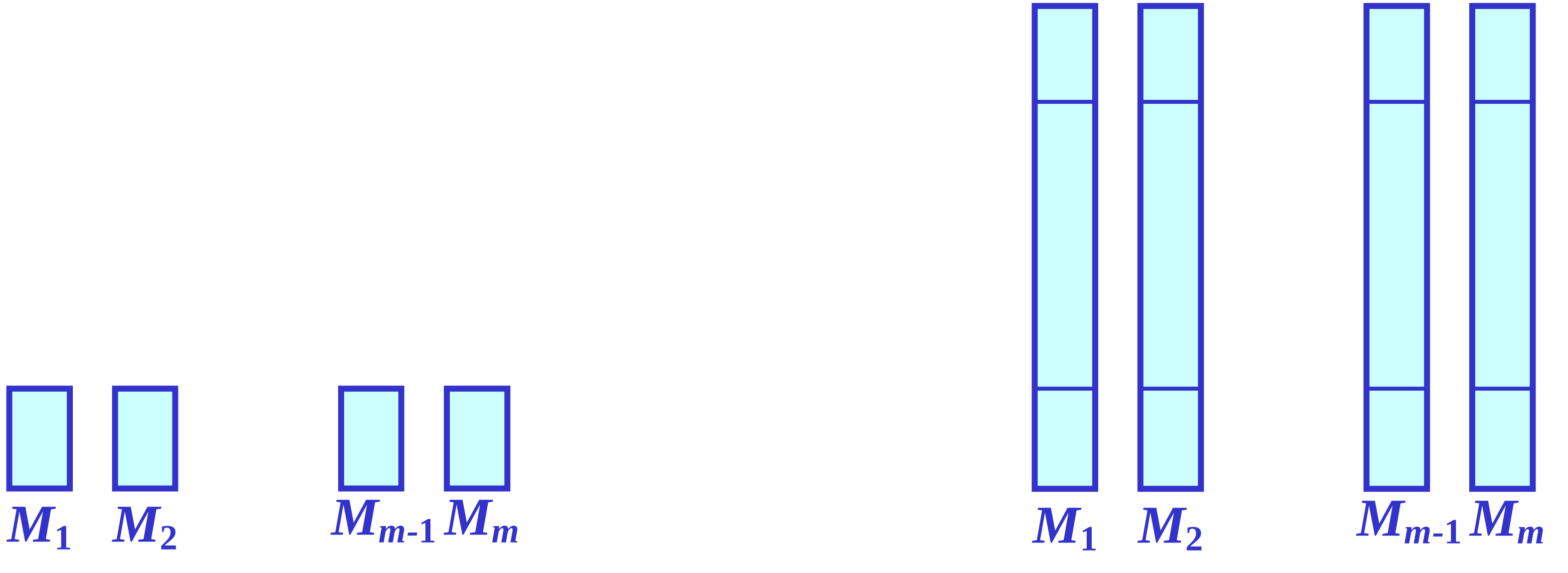


# 5.9 排序问题（续一）

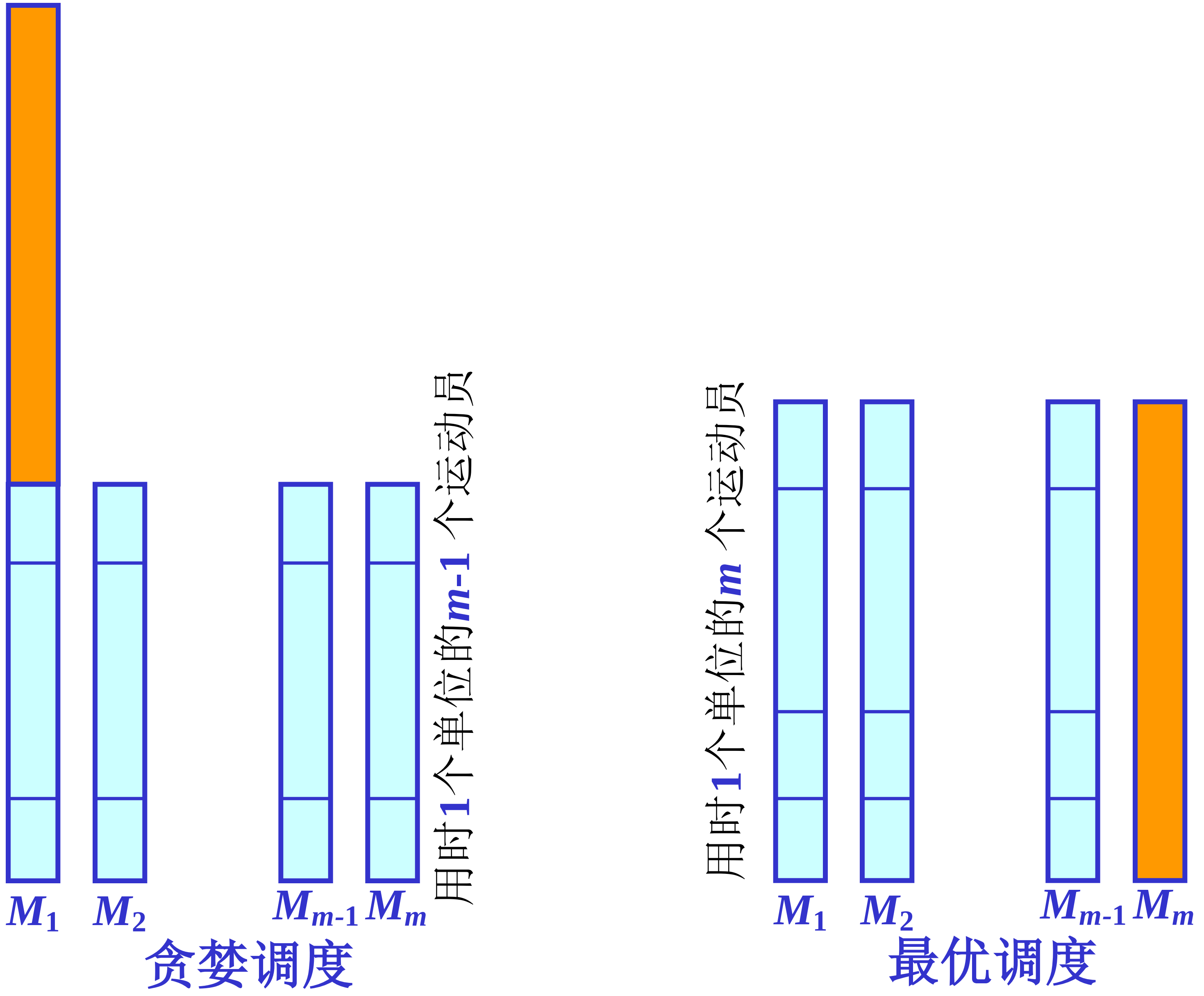
$m\,(m-1)+1$ 位运动员先后到达，用时分别为

$t_1=1,\ t_2=1,\ \ldots\ldots,\ t_{m(m-1)}=1$

$t_m=m$



# 5.9 排序问题（续二）


用时1个单位的m-1 个运动员
M1
M2
Mm-1
Mm
贪婪调度
用时1个单位的m 个运动员
M1
M2
Mm-1
Mm
最优调度

# 5.9 排序问题（续三）

**定理 2** “列表排序” 贪婪算法是一个 **2-竞争算法**。给定多处理器排序问题的一个实例 **I**，设 $c_{\mathrm{LS}}(\mathrm{I})$ 为列表排序方法所产生的完工时间，$c_{\mathrm{opt}}(I)$ 为最优排序所产生的完工时间。则 $c_{\mathrm{LS}}(\mathrm{I})/c_{\mathrm{opt}}(\mathrm{I}) \le (2 - 1/m)$.

**证明**. 设 $T= t_1 + t_2 + \ldots + t_{|J|}$。则最优排序所产生的完工时间 $c_{\mathrm{opt}}(I) \ge T/m$。现假定按照列表排序，机器 $M_i$ 的完工时间是最晚的，并设 $J_j$ 是最后分配到这台机器上加工的工件。因为列表排序是将工件 $J_j$ 分配给完工时间最早的那台机器，所以其他机器的完工时间至少在时间 $c_{\mathrm{LS}}(\mathrm{I}) - t_j$ 以后。因此有

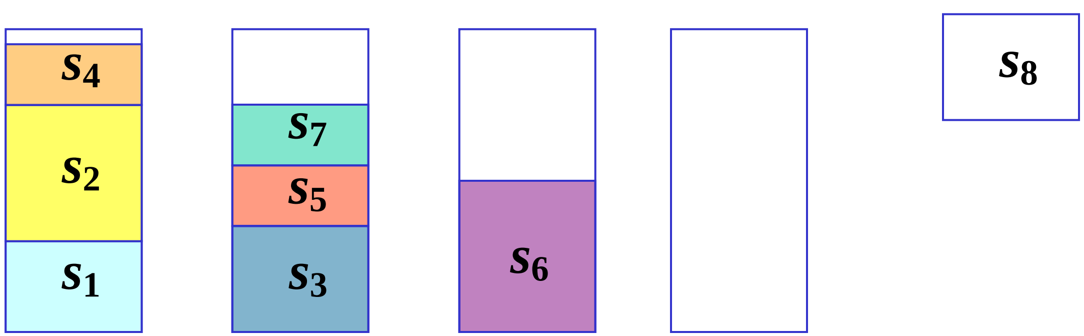
$T \ge m(c_{\mathrm{LS}}(\mathrm{I}) - t_j) + t_j$，即 $c_{\mathrm{LS}}(\mathrm{I}) \le \dfrac{T}{m} + \dfrac{(m-1)t_j}{m}$。又因为

$t_j \le c_{\mathrm{opt}}(\mathrm{I})$。所以有 $c_{\mathrm{LS}}(\mathrm{I}) \le c_{\mathrm{opt}}(\mathrm{I}) + \dfrac{m-1}{m} c_{\mathrm{opt}}(\mathrm{I}) = (2 - \dfrac{1}{m})\, c_{\mathrm{opt}}(\mathrm{I})$.

# 5.9 装箱问题

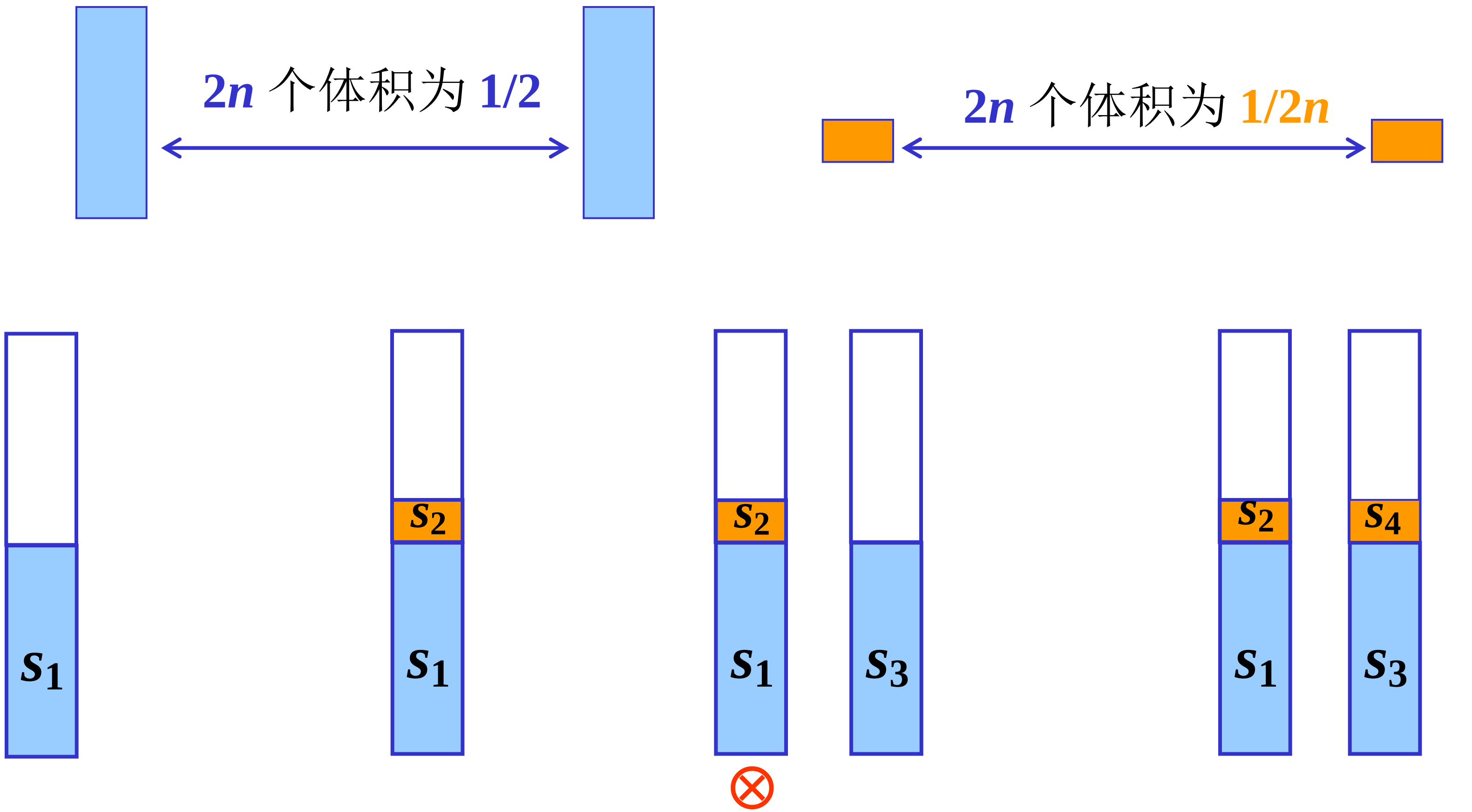
2008年北京奥组委交给我的最后一项任务是：在奥运村的邮局**装箱**。每当客人来邮寄包裹，需要把包裹装到一个规格一定的箱子里，装满了以后，就可以运往目的地了。可不知道什么时候客人会来寄包裹，每一个包裹都有多大；目标是用最少的箱子把客人的包裹都装下（然后寄走）。

一个简单的贪婪策略是"**邻近试装**"：每收到一个包裹，试试将其装入当前的箱子中；如果"装得下"，那么就装；如果"装不下"，那么就将这个箱子打包（运走）；并取一个新的箱子，把它装入。

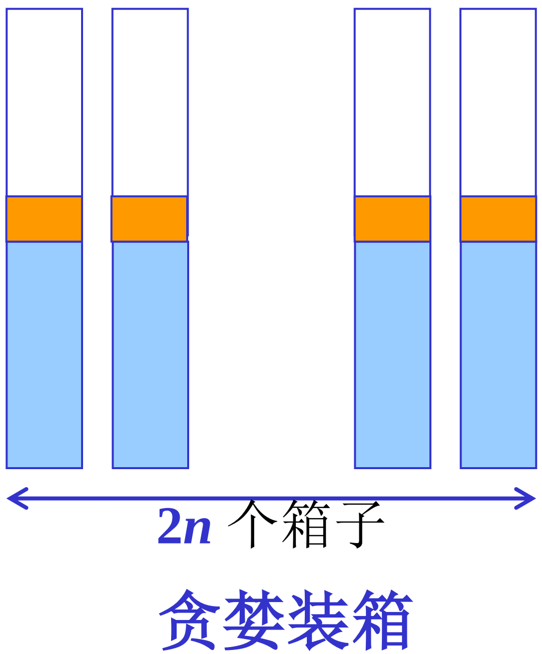


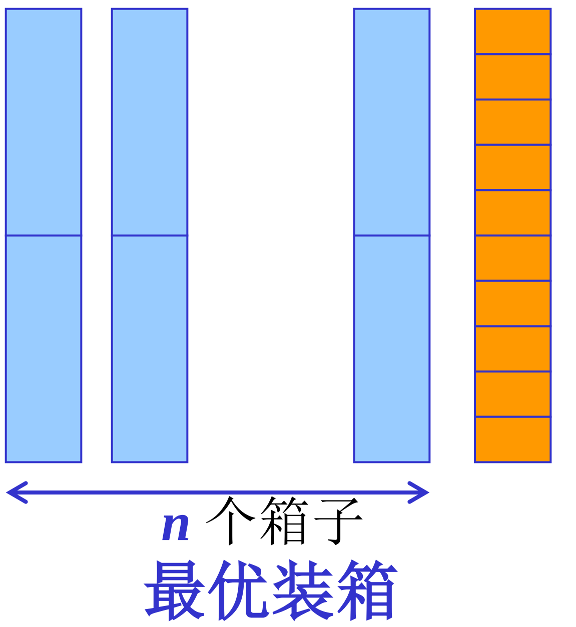
# 5.9 装箱问题（续一）

先后收到 $4n$ 个包裹 $I=\{\, s_1=1/2,\ s_2=1/2n,\ s_3=1/2,\ s_4=1/2n, \ldots \,\}$



# 5.9 装箱问题（续二）




**定理 3** “邻近试装”贪婪算法是一个2-竞争算法。

注意，我们可以将装箱问题看作排序问题的对偶问题。

# 5.9 竞争群组检测问题

1943年 **R. Dorfman** 为美国公共健康服务和筛选服务系统设计了一个方案，用来筛查出应征参军的报名者中携带淋病病毒的年青人。通常的筛查方法是采集血样以后，对它们一个一个地进行检测， 而 **R. Dorfman** 的建议是采用**群组检测**技巧。


# 5.9 竞争群组检测问题（续一）

应用这个新技巧便产生了一个非常有趣的数学问题，**组合群组检测**。假设在给定的 $n$ 个样本中，（在做任何检测以前）已知其中含有 $d$ 个“**坏**”样本。用尽可能少次的测试检测出所有 $d$ 个坏样本。

假设对（可含有任意多个样本的）一个群组的样本进行检测，可能出现两种结果：“纯净的”，即**阴性**（群组中的所有样本都是好的），和“污染的”，即**阳性**（群组中的样本中至少有一个是坏的）。

# 5.9 竞争群组检测问题（续二）

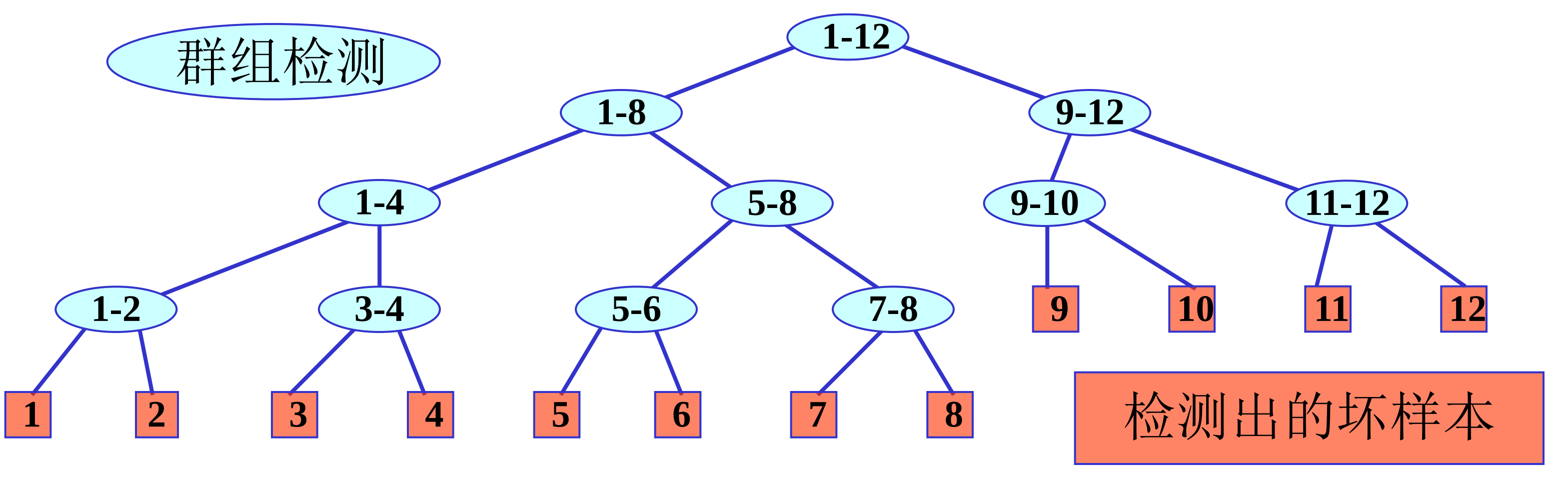
一个经典的结果是这样的：若 $n \le \frac{21}{8}d$，则**逐个检测**是最好的方法。换言之，如果事先知道存在很多的坏样本，那么采用群组检测技巧并不能节省检测的次数。另外，当事先知道没有很多坏样本的时候，采用**二分检测法**是非常有效率的。

我们这里要讨论的是组合群组检测问题的在线形式，**竞争合群组检测**：只有在检测出所有的 $d$ 个坏样本以后，我们才能知道 $n$ 个样本中有多少个坏样本（即知道 $d$ 的确切数值）。很显然，要为竞争组合群组检测问题设计出有效的测试方案比（离线）组合群组检测问题要更加困难。

# 5.9 竞争群组检测问题（续三）

下面我们看一个简单的例子，它可以说明即使在开始检测之前不知道坏样本的确切个数，群组检测法仍然是有效的。首先我们构造一棵二叉树 ***T*** 如下：

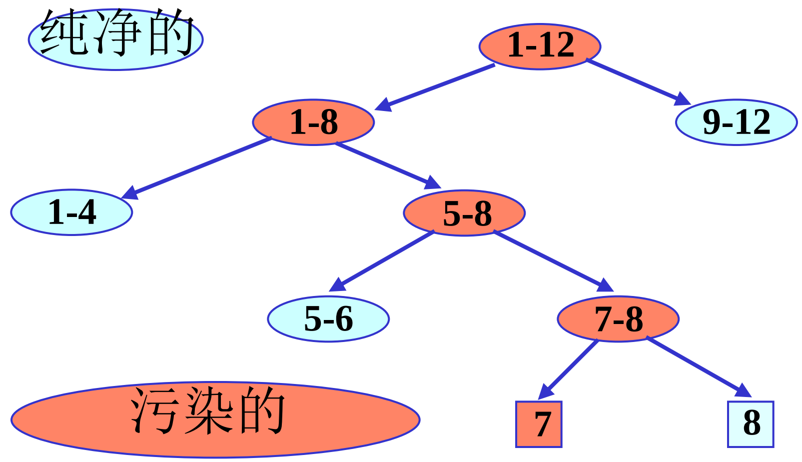
- 根节点在第一层，下面是第二层、第三层，等等。
- 在同一层的节点由左到右排序。



# 5.9 竞争群组检测问题（续四）

根据这种**广度优先搜索序**，我们可以设计如下**二分检测法**：在每一步，若检测出一个被污染的群组 $X$，则将其进行二分，得到两个子群组 $X'$ 和 $X''$，然后分别检测它们，其中$X'$ 含有 $2^{\lceil \log|X| \rceil - 1}$ 个样本，而 $X'' = X \setminus X'$。

右图给出了将二分检测法应用于前面的二叉树 $T_B$ 时可能发生的一种情况。检测的最后结果表明，经过 **9** 次检测，仅有第 **7** 个样本是坏的。




**练习** 当仅有第 **9** 个样本和第**10** 个样本是坏的时候，二分检测法需要做 **7** 次检测。

# 5.9 竞争群组检测问题（续五）

ALGORITHM 14.1 *Bisecting Algorithm*

$G := \emptyset$ (a bin for good items)
$D := \emptyset$ (a bin for defectives)
test all items in $S$.
**if** $S$ is a pure set **then** $G := S$ and $Q := \emptyset$
**else** $Q := \{S\}$ (a queue of contaminated subsets to be tested)
**repeat** take the first element $X$ of $Q$,
    bisect $X$ into $X'$ and $X''$,
    test $X'$.
    **if** $X'$ is contaminated **then** test $X''$
    (if $X'$ is pure, then $X''$ must be contaminated. No need to test $X''$.)
    **for** $Y := X'$ and $X''$ **do**
        **if** $Y$ is pure **then** $G := G \cup Y$.
        **if** $Y$ is contaminated and singleton **then** $D := D \cup Y$.
        **else** push $Y$ into $Q$.
    **end-for**
**until** $Q = \emptyset$

# 5.9 竞争群组检测问题（续六）

注意，如果给定的样本中有很多的坏样本，那么二分检测法需要的检测次数要多于逐一检测法。比如，在前面的例子中，当所有标号为奇数的样本都是坏的时候，二分检测法需要对所有 **23** 个节点对应的群组进行检测。

对于一个检测算法 **A**，我们用 $N_A(s)$ 来表示它检测样本实例 $s$ 所需要的检测次数，并定义

$$M_A(\, d \mid n \,) = \max \{N_A(s) \mid s \in S(n, d)\},$$

其中$S(n, d)$表示 $n$ 个样本中含有 $d$ 个坏样本的所有实例的集合。

称算法 **A** 是 ***α*-竞争算法** 如果存在一个常数 $c$ 使得对任意 $0 < d < n$,

$$M_A(\, d \mid n \,) \le \alpha M(d, n) + c,$$

其中 $M(d, n)$ 表示针对（未检测就）已知 $n$ 个样本中含有 $d$ 个坏样本的所有实例，**最优检测算法**所需要做的最多检测次数。

# 5.9 竞争群组检测问题（续六）

注意，我们在上述定义中排除了 $d = 0$ 和 $d = n$ 这两种情形，这是因为 $M(0, n) = M(n, n) = 0$，而任意一个合理的算法都需要进行至少一次检测。此外，$M(1, n) = \lceil \log n \rceil$。

**练习*** 对任意正整数 $1 \le d \le n-1$，$M_B(d \mid n) \le 2n-1$。

**引理 1** 假设 $n$ 是 2 的一个次幂。则对任意 $1 \le d \le n$，$M_B(d \mid n) \le 2d(\log n/d + 1) - 1$。

**引理 2** 对任意 $0 \le d \le n$，$M_B(d \mid n) \le 2d(\log n/d + 1) + 1$。

**引理 3** 对任意 $0 < d < 8n/21$，$M(d, n) \ge d(\log n/d + 1.096) - 0.5\log d - 1.222$。

**定理 4** 对任意 $1 \le d \le n-1$，$M_B(d \mid n) \le 2M(d, n) + 5$。

# 5.9 随机在线算法

我们前面讨论的在线算法都属于确定性的。这类算法有一个缺点，"**敌手算法**"总可以清楚地知道在线算法遇到某个实例时所会采取的步骤，从而它可以邪恶地预设一个实例，使得在线算法陷入窘境。因而，人们提出了一种随机在线算法，它可以更好地对付"敌手算法"。

一个随机**在线算法 A** 是确定性在线算法空间 $\{ \mathbf{A}_x \}$ 的一个概率分布。或者更加简单地说，一个随机在线算法就是一个可以掷硬币的在线算法。

# 5.9 随机在线算法（续一）

不过请注意，当我们设计一个随机在线算法时，必须说明允许敌手知道在线算法的什么信息。我们假设，敌手是无法预测出一个在线算法掷硬币的结果，或者等价地，敌手必须事先预设好一系列“陷阱”。严格地说，我们可以假定敌手是知道随机在线算法 **A** 所采用的确定性在线算法空间的概率分布，但是他无法预知掷硬币的结果是“正面”还是“背面”。

在设计在线算法时引入随机性对应对“**邪恶敌手**”是非常有用的，因为“敌手”无法预选知晓在线算法的所有信息和运行时的具体状态。

# 5.9 随机在线算法（续二）

假设一个在线算法 **A** 是确定性在线算法空间 $\{\mathbf{A}_x\}$ 的一个概率分布。称其为一个应对任意邪恶敌手的$\alpha$-**竞争算法**，如果存在一个常数 $\alpha$ 使得

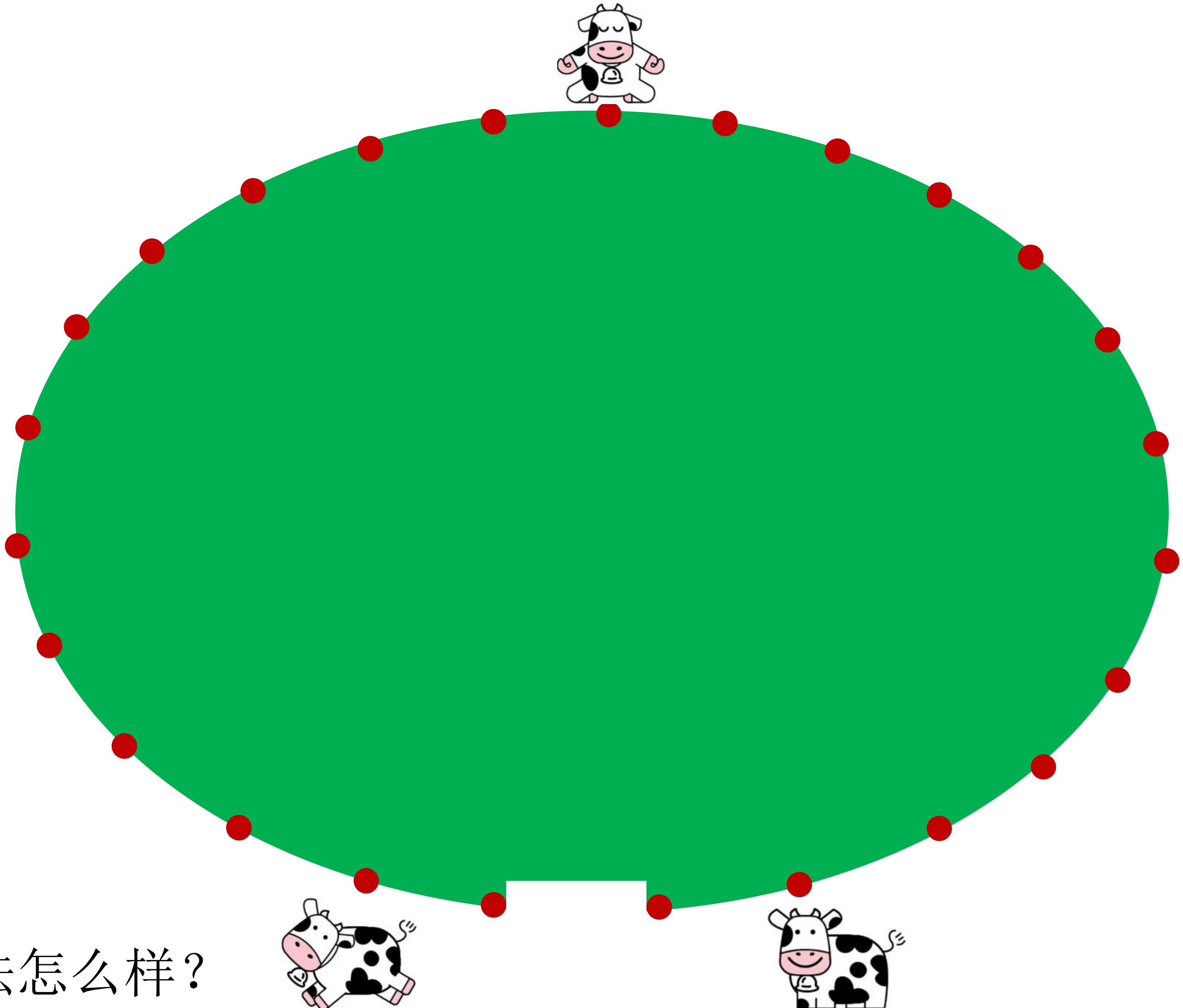
$$\mathrm{EXP}[c_{\mathrm{A}_x}(S)] \le \alpha\, c_{\mathrm{opt}}(S),\ \forall S。$$

我们称一个应对在线算法 **A** 的在线敌手 **Q** 是**自适应的**，如果 **A** 和 **Q** 都必须在 **Q** 给定下一个“请求”前，对当前面临的“请求”做出回应；亦即，敌手 **Q** 可以从在线算法 **A** 对他的请求 $r_i$ 所做出的回应 $a_i$ 中了解在线算法 **A** 的信息，但这只能是在他已经做出了他自己的回应 $q_i$ 以后，并且他给出下一个请求 $r_{i+1}$ 之前。

# 5.9 练习题

假设一个奶牛场只有一个出入口，有一只奶牛想回到奶牛场中，但是它不知道沿着木栏逆时针走距离入口近，还是顺时针走距离入口近。请问：它应该怎么走，才能确保在任何情况下它走的距离都不会超过最短距离的常数倍？

# 5.9 练习题（续一）



**思考题：**

随机算法怎么样？

# 5.9 练习题（续二）

假设你有机会与 *n* 个女/男孩逐个约会。在每一次约会时，若你选择接受她/他，则没有机会见到后面的；若你选择放弃她/他，则没有机会再接受她/他。问你怎么做出选择呢？

# 5.9 练习题（续三）

生活中类似情形很多：选投资项目，选秀投票，或者有几个工作的机会。现实的情况都不容你脚踩几条船来比较，只能是一个个的考虑抉择，错过了，就无法再重新考虑。注意，对于任何一种选择策略，都可以给出一个具体实例，使得该策略得到最差的结果。