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)

5.6 序贯方法

有一类组合优化问题称为**划分型问题**。这类问题的一个可行解可以视为给定实例的元件集的一个划分，它满足相应问题的约束条件。

对于划分问题，我们可以使用**序贯方法**求解，其主要思想如下：

- 初始，根据某个准则，将给定实例的所有元件排序
- 然后，以序贯的方式逐步构造所有元件的一个划分

可以看得出来，序贯方法与前面讨论的贪婪算法非常相似。

5.6 序贯方法（续一）

ALGORITHM 9.1 *General Sequential Algorithm*

Input a set I of items;

Output partition P of I.

begin

sort I according to some criterion;

$(x_1, x_2, \cdots, x_n) \leftarrow$ the obtained sequence;

$P := \{\{x_1\}\}$.

for $i := 2$ **to** n **do**

if x_i can be added to a set p in P **then**

$p := x_i$;

$P := P \cup \{\{x_i\}\}$.

end-for

return P.

5.6 序贯方法（续二）

显然，对于一个具体的划分问题，当我们设计序贯方法时，我们需要考虑使用什么样的准则来

- 定义元件的顺序
- 将某个元件放人划分的集合中

当然，对于每一个划分问题，都有多个可以采用的准则。

在上面描述的序贯方法中，当我们考虑元件 x_i 时，不允许对已经放入或者未放入正在构造的划分中的某个集合内的元素元件 x_j 进行任何调整，$j < i$。换言之，如果两个元件一旦放入到划分中同一个集合内，那么在最后输出的划分中它们一定也还在同一个集合内。

5.6 排序

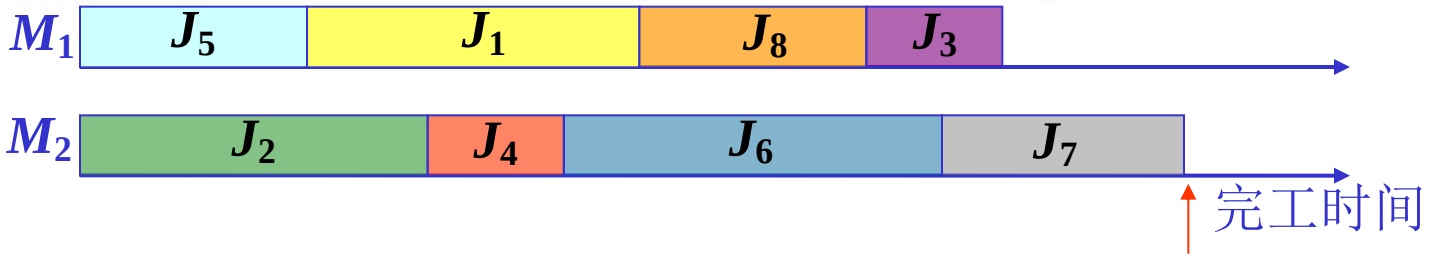
我们首先考虑多处理器的排序问题。给定 m 台相同的机器（处理器），一组工件 $J=\{J_j\}$，每一个工件 J_j 在所有机器上的加工时间都是 t_j。我们的任务是将每个工件分配给一台机器去加工，且安排每台机器上工件的加工顺序。目标是所有机器中加工完分配给它的工件的最长时间最短。

问　题: 多处理器排序

实　例: m 台机器和 n 个工件集合 J，
及每个工件 $J_j \in J$ 的加工时间 t_j

可行解: J 的一个 m-划分，即一个映射 $f: J \to \{1, 2, \ldots, m\}$.

目　标: 极小化 映射 f 的完工时间，$\max\{\sum_{f(J_j)=i} t_j \mid i=1,\ldots, m\}$



5.6 排序（续一）

下面讲述如何用序贯方法求解上述排序问题。首先假定我们考虑工件的顺序已经给定了。因而我们只需要引入一个准则来将每一个工件分配给 m 台机器中的某一台。一个非常自然的策略就是将当前考虑的工件分配给当前承担工件的加工时间之和最小的那台机器。假设前 $j-1$ 个工件已经分配给机器加工了，设 $c_i(j-1)$ 为第 i 台机器此时的承担工件的加工时间之和，亦即

$$c_i(j-1) = \sum \{ t_k \mid f(J_k) = i, 1 \le k \le j-1 \}。$$

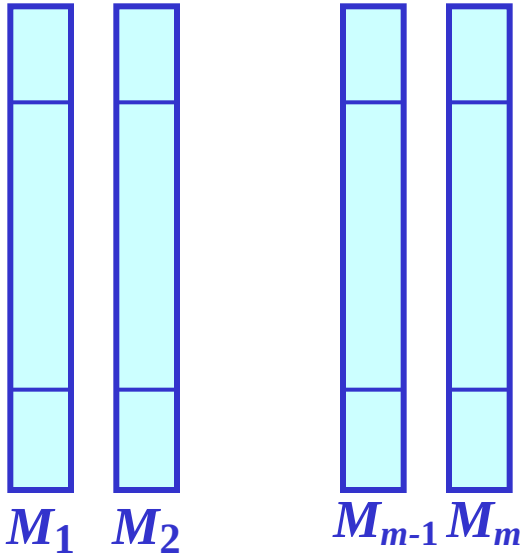
将第 j 个工件 J_j 分配给当前完工时间最早的那台机器。这样我们就可以最小化新加工一个工件所造成的完工时间增加。这个序贯方法称为 **列表排序**。

5.6 排序（续二）

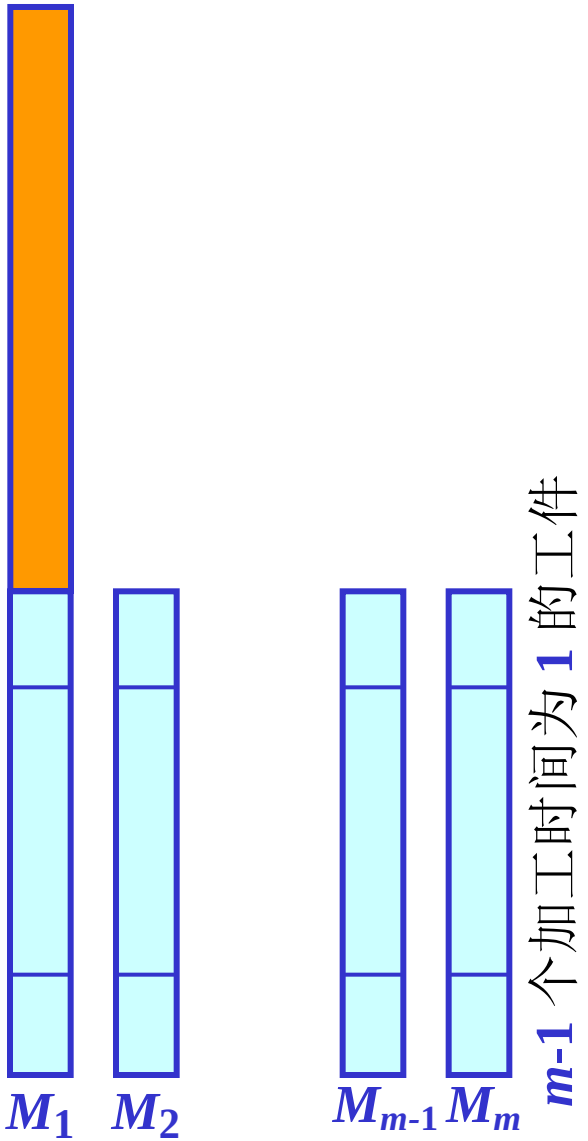
给定的工件顺序:

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

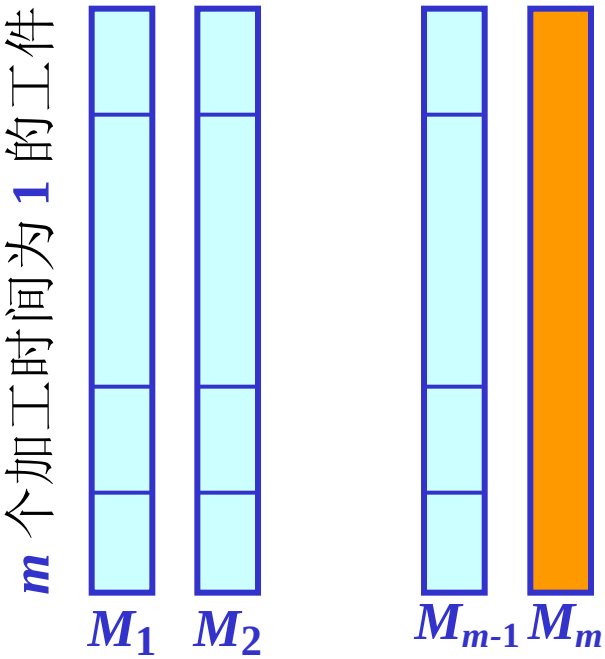
$t_m=m$



5.6 排序（续三）



列表排序



最优排序

5.6 排序（续四）

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

证明. 设 $T = t_1 + t_2 + \ldots + t_{|J|}$。则最优排序所产生的完工时间 $c_{opt}(I) \ge T/m$。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 $c_{LS}(I) - t_j$ 以后。因此有

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

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

注意：上述证明的分析与我们安排工件时它们的顺序无关。

5.6 排序（续五）

考虑到上述的证明与工件的顺序无关，而在前面的例子中加工时间最长的工件是在最后安排的，因而造成列表排序的效果不好。可以期望如果将工件的顺序做适当的安排，列表排序的效果会变好。考虑工件的一个非常自然的顺序是，**按照它们的加工时间由长到短排列**，亦即 $t_1 \geq t_2 \geq \ldots \geq t_{|J|}$。基于此顺序逐个安排所有的工件，就得到如下改进的列表排序算法。

ALGORITHM 9.3 *Improved List Scheduling Algorithm*

sort jobs in J by non-increasing order with respect to their processing time;
$(t_1, t_2, \cdots, t_n)$:= the obtained sequence;
$M := \{\{J_1\}, \emptyset, \cdots, \emptyset\}$.
for $j := 2$ **to** n **do**
    choose the machine $M_i \in M$ that has the most earliest finishing time,
    $f(J_j) := i$.
end-for
return f.

5.6 排序（续六）

定理 2 给定多处理器排序问题的一个实例 **I**，设 $c_{ILS}(I)$ 是改进的列表排序所产生的完工时间，$c_{opt}(I)$ 是最优排序所产生的完工时间。则 $c_{ILS}(I) / c_{opt}(I) \le 4/3 - 1/3m$。

证明. 设 t_n 为排在最后的工件的加工时间，它也是最短的加工时间。 考虑以下两种情形。

情形 1. $t_n > c_{opt}(I)/3$。此时， 改进的列表排序输出的一定是最优解（练习）。

情形 2. $t_n \le c_{opt}(I)/3$。此时，有 $c_{ILS}(I) \le \frac{T}{m} + \frac{(m-1)\,t_n}{m}$，

再由 $c_{opt}(I) \ge T/m$，即可得

$$c_{ILS}(I) \le c_{opt}(I) + \frac{m-1}{3m} c_{opt}(I) = \left(\frac{4}{3} - \frac{1}{3m}\right) c_{opt}(I)。$$

5.6 装箱

现在我们来考虑**装箱问题**： 给定一组尺寸不同的物品，及若干个箱子；任务是将每一个物品都放到某个箱子中，且每一个箱子所装的物品的尺寸不能超过箱子的容积；目标是所用的箱子越少越好。这里我们只考虑其特殊情形，**一维装箱**：每个物品的尺寸都小于**1**，且所有箱子的容积都是 **1**。

问　题: 装箱

实　例: 一组物品 $\mathbf{I} = \{s_1, s_2, \ldots, s_n\}$ 物品 i 的尺寸为 s_i

可行解: $\mathbf{I}$ 的一个 k-划分$\{\mathbf{B}_1, \ldots, \mathbf{B}_k\}$ 使得 $\sum_{s_i \in \mathbf{B}_j} s_i \le 1,\ j = 1, \ldots, k$

目　标: 极小化构成划分的集合个数 k

5.6 装箱（续一）

我们现在讲述如何用序贯方法设计求解该问题的近似算法。一个非常简单的算法是**邻近适装**：每一次只考虑一个物品（任意物品顺序）；将第一个物品 s_1 放入箱子 B_1 中；当考虑物品 s_i 时，假设前一个物品 s_{i-1} 放在了箱子 B_j 内。如果可以将它放入箱子 B_j 内而不超过其容积，那么就放入其中，否则将它放入一个新的（空）箱子 B_{j+1} 内。

ALGORITHM 9.4 *Next Fit Algorithm*

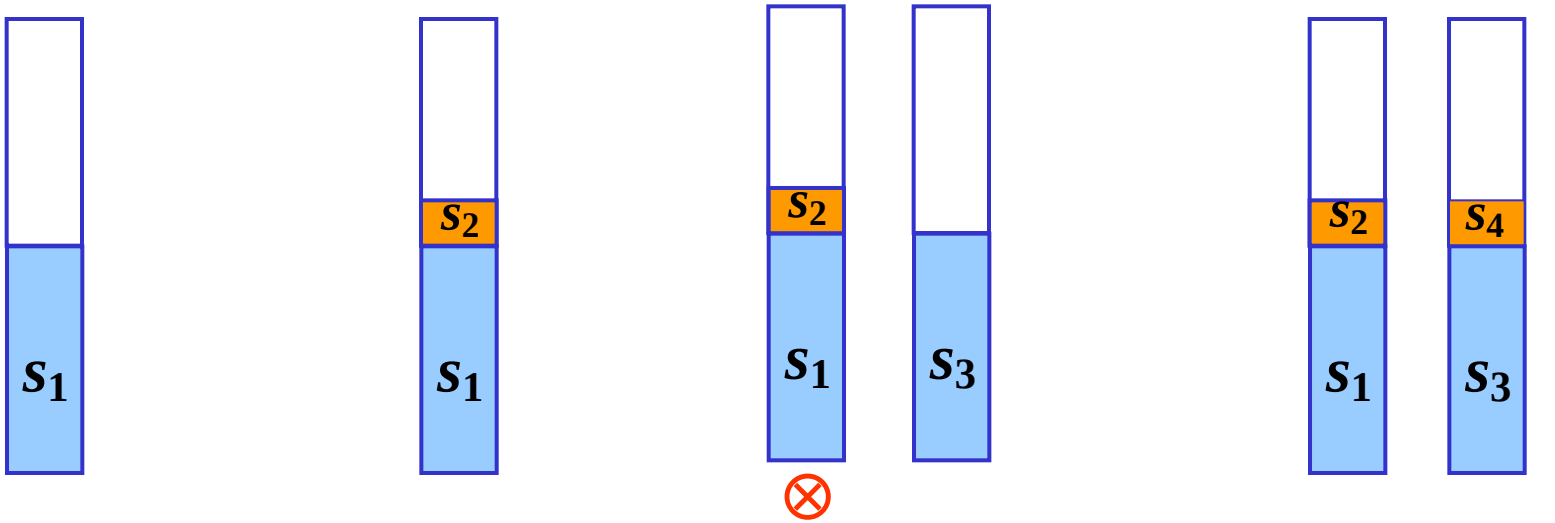
$B_1 := s_1$.
for $i := 2$ **to** n **do**
    $B_j :=$ the last used bin.
    if $s_i + \sum_{s_h \in B_j} s_h \leq 1$ **then** $B_j := s_i$;
    else $B_{j+1} := s_i$; // introduce a new bin
end-for
return $B_1, B_2, \cdots, B_k$.

5.6 装箱（续二）

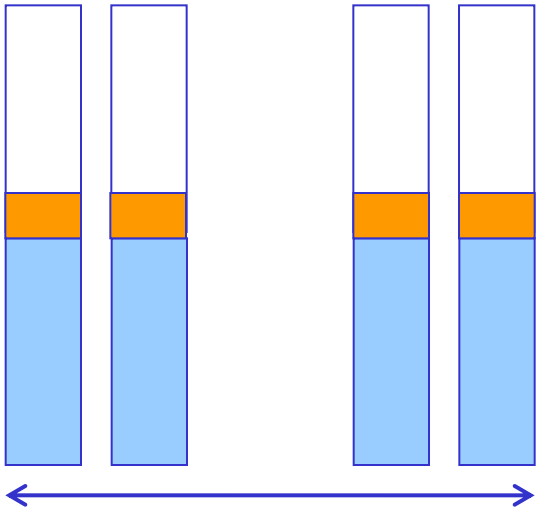
一组 $4n$ 个物品 $I=\{ s_1=1/2, s_2=1/2n, s_3=1/2, s_4=1/2n,\dots \}$

$2n$ 个尺寸为 ½ 的物品

$2n$ 个尺寸为 $1/2n$ 的物品

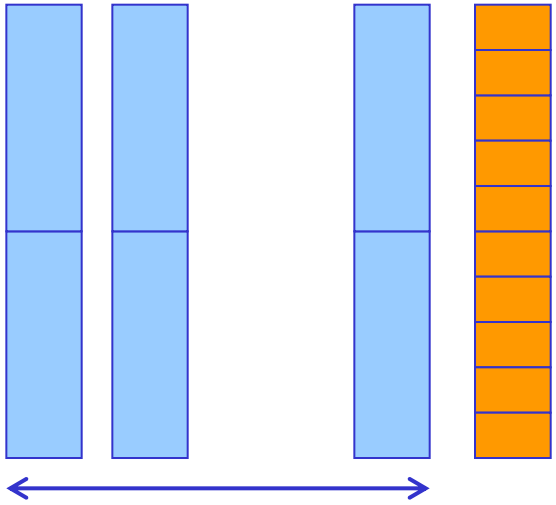


5.6 装箱（续二）



2*n* 个箱子

邻近适装法



n 个箱子

最优装箱法

5.6 装箱（续三）

定理 3 任给装箱问题的一个实例 **I**，设 $c_{NF}(I)$ 为邻近适装法所需要的箱子的个数，$c_{opt}(I)$ 为最优装箱法所需要的箱子的个数。则 $c_{NF}(I) / c_{opt}(I) \leq 2$。

证明 令 σ 表示需要装箱的所有物品的尺寸之和，即 $\sigma = s_1 + s_2 + \ldots + s_n$。易知，最优装箱法至少需要 $\lceil \sigma \rceil$ 个箱子。注意，根据邻近适装法所采取的策略可知，对任意先后考虑的两个相邻箱子，它们所装物品的尺寸之和一定大于 **1**。因而邻近适装法最多需要 $2\lceil \sigma \rceil$ 个箱子。

需要强调的是，上述分析与考虑装箱的物品的次序无关。

5.6 装箱（续四）

邻近适装法有一个非常明显的弱点：它只是考虑是否可以将要装的物品装入当前的箱子中（而不是考虑其他已经使用过的箱子）。这就提示我们采用**首适装箱**法：将要装的物品 s_i 装入第一个已经用过的可以装下该物品的箱子中；若已经装有物品的所有箱子都不能装下物品 s_i，则使用一个新的箱子。

练习 如果采用首适装箱法，将前面 $4n$ 个物品装箱，那么需要多少箱子呢？

一般来讲，首适装箱法需要的箱子个数比邻近适装法需要的箱子个数少。更严格地可以证明以下结果：对于装箱问题的任意一个实例 I，首适装箱法需要的箱子个数

$$c_{FF}(I) \le 1.7c_{opt}(I) + 2。$$

5.6 装箱（续五）

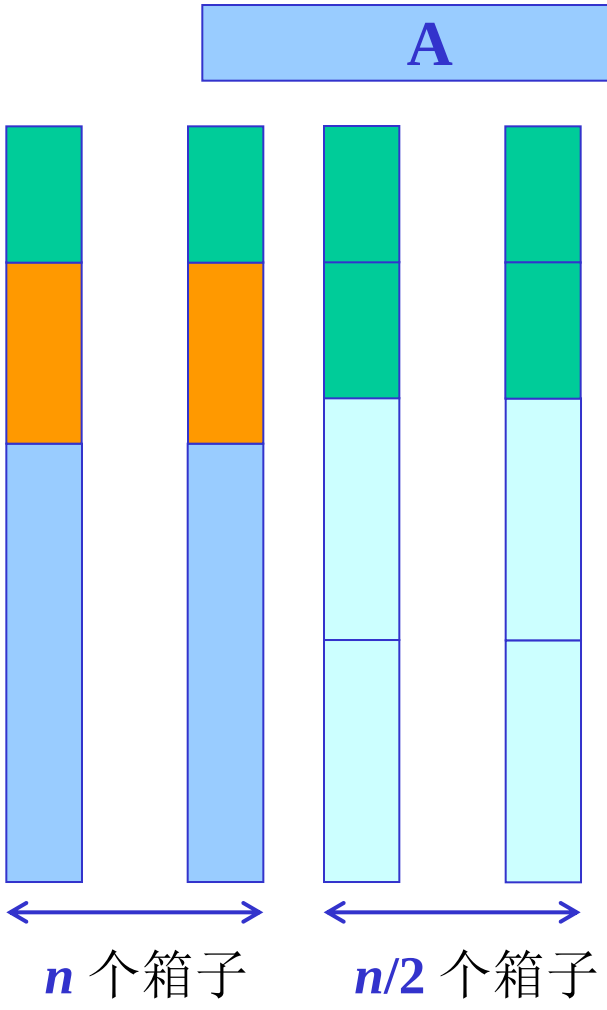
邻近适装法的另外一个弱点是，它考虑物品的次序就是问题实例中给定的实例，即它不考虑这些物品的尺寸大小。这就提示我们通过调整这些物品的次序，有可能减少所使用的箱子的个数。其中一个这样的算法就是**首适递减装箱法**：将所有的物品依它们的尺寸由大到小排序后，再以此序应用首适装箱法。

考虑如下实例：$I = A \cup B \cup C \cup D$ 包含 $5n$ 个物品，其中

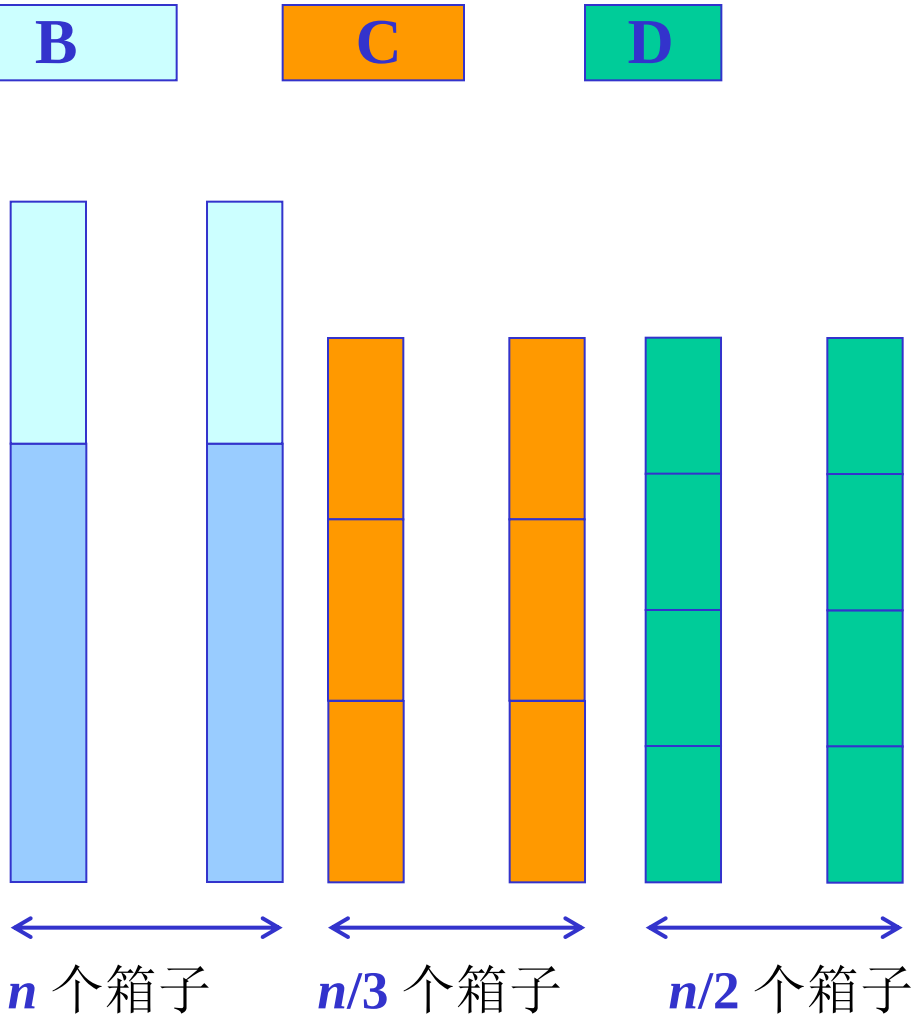
A 含有 n 个尺寸为 $\frac{1}{2} + \varepsilon$ 的物品，
B 含有 n 个尺寸为 $\frac{1}{4} + 2\varepsilon$ 的物品，
C 含有 n 个尺寸为 $\frac{1}{4} + \varepsilon$ 的物品，
D 含有 $2n$ 个尺寸为 $\frac{1}{4} - 2\varepsilon$ 的物品。

对于这个实例，首适递减装箱法需要 $11n/6$ 个箱子，而最优装箱法需要 $3n/2$ 个箱子。

5.6 装箱（续六）



最优装箱法



首适递减装箱法

5.6 装箱（续七）

定理 5 给定装箱问题的一个实例 **I**，设 $c_{FFD}(I)$ 为首适递减装箱法所需要的箱子个数，$c_{opt}(I)$ 为最优装箱法所需要的箱子的个数。则 $c_{FFD}(I) \le 1.5\, c_{opt}(I) + 1$。

证明 首先将物品集合 $I = \{ s_1, s_2, \ldots, s_n \}$ 划分为如下四个集合

$$A = \{ s_i \mid s_i > 2/3 \}, \quad B = \{ s_i \mid 2/3 \ge s_i > 1/2 \},$$
$$C = \{ s_i \mid 1/2 \ge s_i > 1/3 \}, \quad D = \{ s_i \mid 1/3 \ge s_i \}.$$

然后考虑首适递减装箱法所需要的箱子个数。

情形 1. 至少有一个箱子装的物品都属于集合 **D**。注意，这样的箱子中至多有一个箱子，它剩余的空间不小于**1/3**。因此有 $c_{opt}(I) \ge 2(c_{FFD}(I) - 1) / 3$。

情形 2. 不存在只装了 **D** 中物品的箱子。我们断言，此时首适递减装箱法是一个最优装箱策略。

5.6 装箱（续八）

为此，我们考虑另一个实例 **I’ =A∪B∪C** ，即从实例 **I** 中去除所有集合 **D** 中的物品。注意，若采用首适递减装箱法，这两个实例 **I** 和 **I’** 所需要的箱子个数是完全一样的。 因而我们仅需要证明，首适递减装箱法对于实例 **I’** 来讲，它也是一个最优装箱策略。

我们首先注意到，所有物品是依照它们的尺寸由大到小排序和装箱的。首适递减装箱法将**C** 中的每一个物品与**B** 中最大可能的一个物品装入一个箱子，且这个箱子不可能还装了其他物品。因此，**A** 中的物品一定是单独装在一个箱子中，而每一个箱子最多装了两个物品（只有一个物品可能属于**B**）。这也就意味着，首适递减装箱法所用的箱子个数与最优装箱法所用的箱子个数是一样的。

5.6 装箱（续九）

注意，如果存在不至一个已经装有物品的箱子可以装下当前考虑的物品时，首适递减装箱法仅将该物品放人第一个这样的箱子中，而不是在这些箱子中选择一个最适合这个物品的箱子放入该物品。这可能会造成有些箱子装得不是太满，因而需要更多的箱子。

基于上述的观察，为了改进首适递减装箱法的性能，很自然地会想到**最适递减装箱法**：初始时，将所有的物品依照它们的尺寸由大到小递减排序。然后，按照这个顺序，逐个考虑物品。当考虑物品 s_i 时，将其放入剩余空间最小的且可装它的箱子中。采用这样的方法装箱，就可以减少将箱子的剩余空间分割成许多零散的小空间的机会，也就增加了有很大剩余空间的箱子的个数。

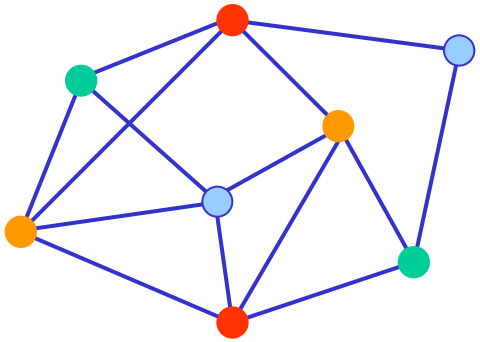
5.6 装箱（续十）

可以证明，对于装箱问题的任意一个实例，采用最适递减装箱法不会比首适递减装箱法需要更多的箱子。而且，存在装箱问题的一个实例，最适递减装箱法是最优装箱方法，而首适递减装箱法不是。不过遗憾的是，最适递减装箱法的性能比与首适递减装箱法的性能比是一样的。

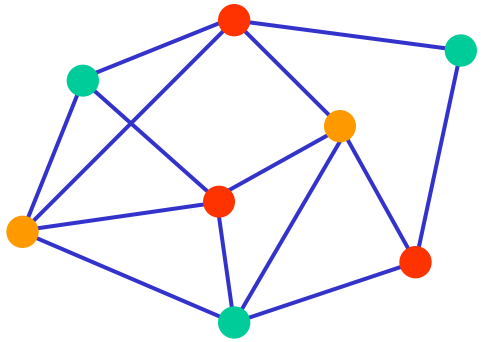
最后，我们需要强调的是，首适递减装箱法和最适递减装箱法都属于离线算法，邻近适装法和首适装箱法都属于在线算法。更严格地说，后两个算法在考虑如何将物品 s_i 装箱时，并不需要知道它的尺寸 s_j，对任意 $j > i$，而前两种算法在考虑如何将物品 s_i 装箱时， 所有物品的尺寸都已经（假定）事先知道了。

5.6 顶点着色

最后我们讨论图的**顶点着色问题**：给定一个图，任务是给图中的每个顶点着一种颜色使得如果两个顶点之间存在一条边那么应给它们着不同的颜色，目标是使用的颜色种类最少（**图的色数**）。



4-着色方法



最优着色方法

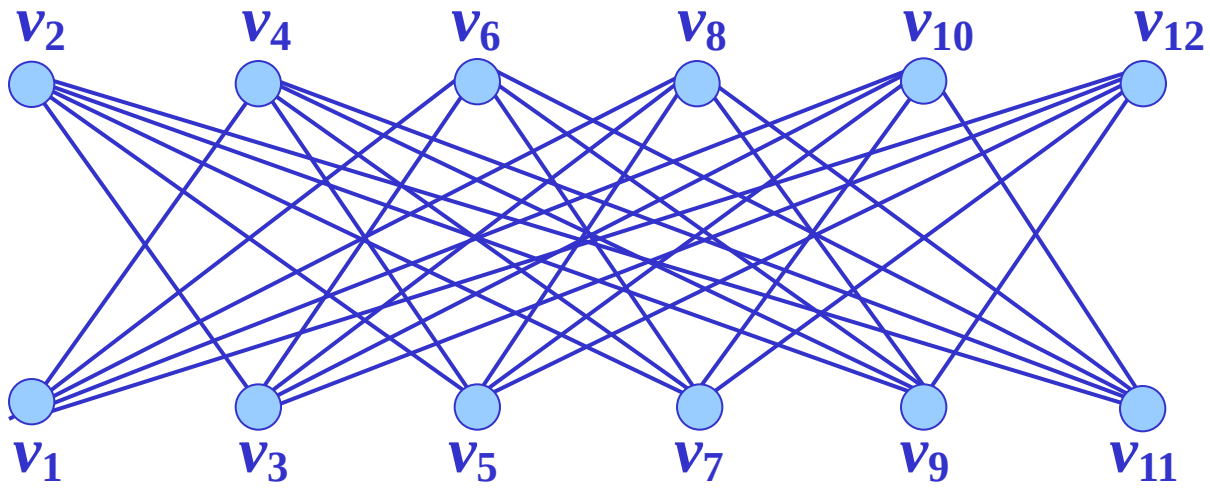
这个问题不仅仅是一个 **NP-难**问题。实际上，与最大独立集问题类似，它属于**NP-难问题类**中一个最难的子类。

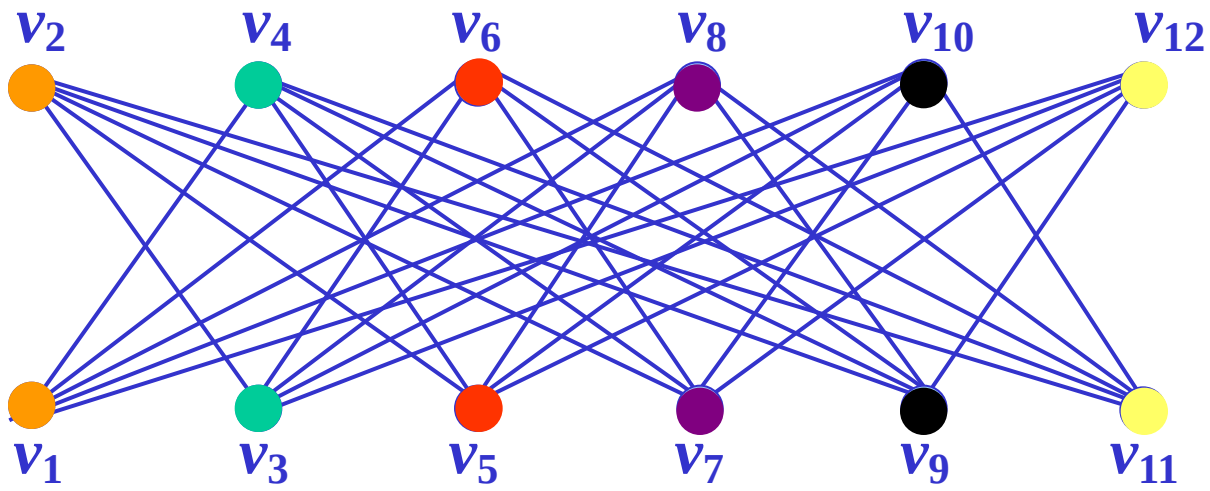
5.6 顶点着色（续一）

我们下面说明如何用序贯方法求解此问题。首先，假设所有的顶点按照一个序排列，$v_1, v_2, \ldots, v_n$。由此我们就可以设计一个非常简单的序贯算法：

- 给顶点 v_1 着色 c_1。
- 假设已经用 k 种颜色给顶点 $v_1, v_2, \ldots, v_{i-1}$ 分别着色。
  如果可以用已经使用过的某种颜色给顶点 v_i 着色，
  那么就用该颜色给其着色。
- 否则顶点 v_i 一定与 k 个顶点相邻，且它们已经被用不同的颜色着色了，用一种新颜色 c_{k+1} 给顶点 v_i 着色。
- 重复此着色过程直到所有的顶点都被着色。

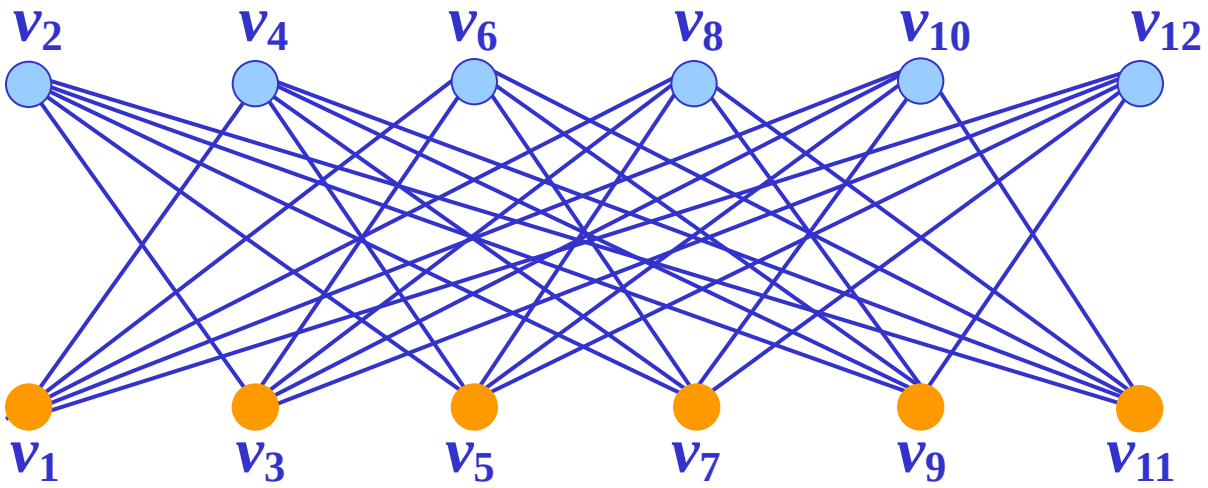
5.6 顶点着色（续二）





序贯着色

5.6 顶点着色（续三）



最优着色

注意，在应用上述序贯着色方法时，我们并没有利用顶点的序的可能的性质 $v_1, v_2, \ldots, v_n$。现在让我们考虑任意一个特定的顶点序列，设 G_i 是由顶点集合 $\{v_1, \ldots, v_i\}$ 诱导出的子图（特别地，G_n 是给定的图 G），另设 k_i 为用序贯着色方法给图 G_i 着色所需要的颜色种数（特别地，k_n 表示用序贯着色方法给图 G 所需要的颜色种数）。再设 $d_i(v)$ 是图 G_i 中顶点 v 的度数（特别地，$d_n(v)$ 是图 G 中顶点 v 的度数）。

5.6 顶点着色（续四）

定理 6 给定一个简单图 $G = (V, E)$， 及其顶点的一个序列 $v_1, v_2, \ldots, v_n$，设用序贯着色法给它的顶点进行正常着色需要 k_n 种颜色。则有

$$k_n \le 1 + \max\{\ \min\{d_n(v_i), i-1\} \mid i = 1, 2, \ldots, n\}。$$

证明. 注意，当用序贯着色法给顶点 v_i 着色时，我们或者用一种已经使用过的颜色，此时有 $k_i = k_{i-1}$，或者用一种新颜色，此时有 $k_i = k_{i-1} + 1$ 且 $d_i(v_i) \ge k_{i-1}$。根据对 i 进行的数学归纳法，可得

$$k_i \le 1 + \max\{\ d_i(v_i) \mid i = 1, 2, \ldots, n\}。$$

显然， $d_i(v_i)$ 不会大于 $d_n(v_i)$，也不会大于 $i-1$ （图 G_i 中顶点的个数），由此即得定理中的不等式。

5.6 顶点着色（续五）

练习 给定一个图 $G=(V, E)$，及其顶点的任意一个序列。若用序贯着色法给 V 中的所有顶点进行正常着色，则最多需要 $\Delta(G)+1$ 种颜色，其中 $\Delta(G)$ 是图 G 中最大顶点度数。

练习 给定一个图 $G=(V, E)$。则对顶点的任意一个序列，都有

$$k_n \le 1 + \max\{d_i(v_i) \mid i = 1, 2, \ldots, n\}。$$

易知，若将所有顶点依照它们的度数由大到小排列，**定理 6** 中的不等式的上界达到最小。因此，应用序贯着色法的时候，我们就可以采用这个顶点序。不过，遗憾的是，可以构造出顶点着色问题的一个实例，应用这样的序贯着色法所需要的颜色种数比最优着色法所需要的颜色种数要多。

5.6 顶点着色（续六）

为了改进顶点依照度数递减排列序的性能，人们提出了许多启发式的方法。其中之一称为**最小最后**序贯着色法，我们可以借助递归的方式描述该方法：

- 给定有个顶点的图 G，选择其中最小度数的顶点做为最后一个考虑的顶点 v_n。
- 假设已经确定顶点序列 $v_{i+1}, \ldots, v_n$，在子集 $V \setminus \{v_{i+1}, \ldots, v_n\}$ 诱导出的子图中选择一个顶点度数最小的顶点做为 v_i。（若这样的顶点不惟一，则可随机选择其中一个。）

注意，采用最小最后序的序贯着色法可以使得上述的上界达到最小，这是因为 $d_i(v_i) = \min\{ d_i(v_j) \mid v_j \in G_j \}$。然而，对于这个顶点序，存在顶点着色问题的一个实例， 序贯着色法所需要的颜色种数比最优着色法所需要的颜色种数要任意地大。

5.6 顶点着色（续七）

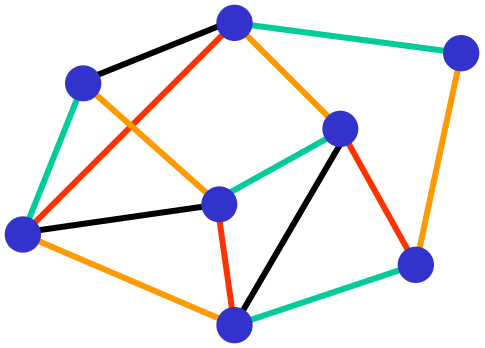
练习 ** 用最小最后序贯着色法给任意一个平面图着色最多需要六种颜色。

需要指出的是：用序贯着色法给任意图的顶点着色，还无法在理论上得到该算法所需要的颜色种数的一个好的估计值，这是因为图的顶点着色问题本身确实是太难了。实际上，人们已经证明不存在求解图的顶点着色问题的任何具有比较好的近似性的多项式时间算法（不仅仅是序贯算法），除非 **P =NP**。

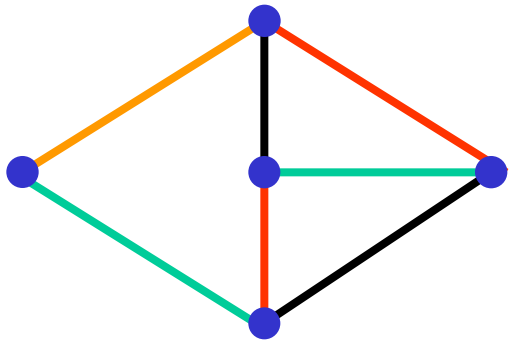
可以说，图的顶点着色问题不仅是一个 **NP-难**的问题，实际上，它是其中最难的一类，因为不太可能存在求解该问题的近似算法其近似比是一个常数。

5.6 边着色

在结束之前，我们要提一下图的**边着色问题**：给定一个图，任务是给图中的每条边着一个颜色使得如果两条边有共同的端点那么应给它们着不同的颜色，目标是使用的颜色种类最少。



$\Delta(G)$ =4; 4 种颜色



$\Delta(G)$ =3; 4 种颜色

尽管图的边着色问题是**NP-难**的，但是我们可以认为它是其中最简单的一类问题，因为存在求解该问题的一个近似算法其**绝对近似差值**是一个常数 **1**（**Vizing 定理**，1964）。

5.6 总结

练习 ** 证明：存在一个多项式时间算法，任给一个简单图 G，它可以用最多$\Delta(G)+1$ 种颜色对图 G 的所有边进行正常边着色，其中 $\Delta(G)$ 是图 G 的最大顶点度数。
（因为任意一个正常边着色方法至少需要 $\Delta(G)$ 种颜色，所以这样的算法称为 **1-绝对近似算法**。）

最后我们需要强调的是，一个序贯算法性能的优劣取决于

- 如何将给定的元件排序
- 如何选择放入划分集合的元件