CS2045M: 高级数据结构与算法分析 **2025-2026 学年秋冬学期**

Lecture 11: 近似算法

编写人：吴一航 yhwu_is@zju.edu.cn

11.1 基本思想与概念

上一讲讨论了 NP 完全性，引入了一类 “很难解决” 的问题，这些问题至今无法找到多项式时间的算法。因此需要做一些妥协。整体而言，对一个最优化问题的算法有如下三个期望：

1. 算法要能找到确切的最优解（optimality）；
2. 算法能高效（通常是多项式时间）运行（efficiency）；
3. 算法是通用的，能够解决所有问题（all instances）。

对于 NP 完全问题而言，目前无法同时做到以上三点，除非 $\mathsf{P} = \mathsf{NP}$，因此需要做一些妥协。如果算法舍弃第二个期望，则是用例如回溯等方法在指数时间内解决问题，这对于输入不大的情况是可以接受的；如果舍弃第三个期望，相当于为问题找到一些容易解决的特例；如果舍弃第一个期望，但能保证高效找到的解是和真正的最优解 “相差不大” 的，那么称这类算法为近似算法。

本讲的目标就是介绍几个基本的近似算法的例子，从中体会近似算法的基本设计思路——因为目前的基础有限，时间也有限，因此只能接触非常基础的内容，毕竟近似算法是值得一个长学期一门大课来吸收其背后深刻内涵的主题。

首先定义近似算法的抽象的基本范式和概念：

定义 11.1 假设有某类问题 $\mathcal{I}$（例如背包问题），其中的一个具体实例记为 I（当背包问题的参数给定的时候即为一个实例），且有一个复杂度为多项式的近似算法 A。定义：

- $A(I)$ 为算法 A 在实例 I 上得到的解；
- $OPT(I)$ 为实例 I 的最优解。

考虑 $\mathcal{I}$ 是最小化问题，若存在 $r \geqslant 1$，对任意的 I 都有

$$A(I) \leqslant r \cdot OPT(I),$$

那么称 A 为该问题的 r-近似算法（即对于任何可能的问题实例，A 给出的解都不会比最优解的 r 倍更大）。我们特别关心其中可以取到的最小 r，称

$$\rho = \inf\{r : A(I) \leqslant r \cdot OPT(I), \forall I\}$$

为近似比（*approximation ratio*，即算法 A 最紧的近似界）。它可以等价定义为：

$$\rho = \sup_I \frac{A(I)}{OPT(I)},$$

即这一比值最大的实例对应的比值就是最紧的界。反之，如果是极大化问题，那么上式应该改为

$$\rho = \sup_I \frac{OPT(I)}{A(I)},$$

将两者合并起来，可以统一写作

$$\rho = \sup_I \left\{ \frac{OPT(I)}{A(I)}, \frac{A(I)}{OPT(I)} \right\}.$$

需要注意的是，上面只讨论了近似比为常数的情况。事实上有时候近似比也可能与输入规模有关，例如下面会讨论的调度问题问题等，但大部分情况下的结果都是常数的，因此一般而言定义都是常数。

给定一类问题 $\mathcal{I}$ 和算法 A，事实上很难根据定义求出 A 的近似比，因为 $OPT(I)$ 一般未知。因此，只能通过 $OPT(I)$ 的范围来确定近似比。以最小化问题为例，确定近似比需要以下两个步骤：

1. 首先寻找一个 $r \geqslant 1$，对于任何实例 I ，都有 $A(I) \leqslant r \cdot OPT(I)$（可以首先寻找到 $OPT(I)$ 的一个下界 $LB(I) \leqslant OPT(I)$，然后让 $A(I) \leqslant r \cdot LB(I)$ 即可）；
2. 接下来证明 r 是不可改进的，即对任意的 $\varepsilon \geqslant 0$，都存在一个实例 I_ε ，使得 $A(I_\varepsilon) \geqslant (r-\varepsilon) \cdot OPT(I_\varepsilon)$。

在 $\mathsf{P} \neq \mathsf{NP}$ 的假设下，没有多项式算法解决 NP 问题，因此近似比不可能为 1；不过，我们希望设计近似比尽可能小（尽可能接近 1）的近似算法。那么什么样的近似是最好可能的呢？设 $|I|$ 代表问题 I 的规模，f 是一个可计算（computable）函数，不一定为多项式函数，那么有

1. PTAS（多项式时间近似方案，Polynomial time approximation scheme）：存在算法 A，使得对每一个固定的 $\varepsilon \geqslant 0$，对任意的实例 I 都有

   $$A(I) \leqslant (1+\varepsilon) \cdot OPT(I),$$

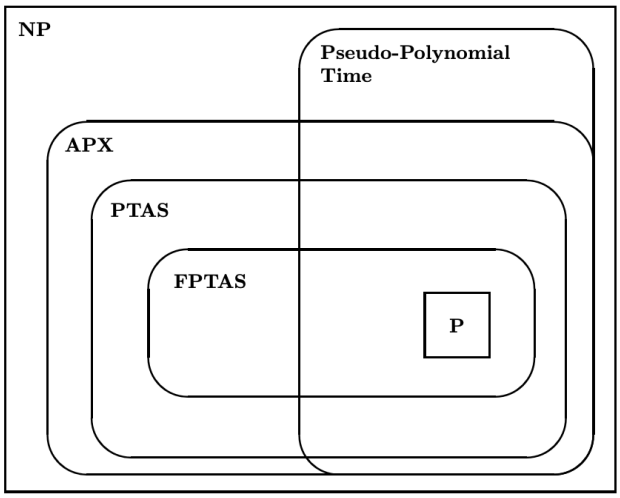
   且算法 A 的运行时间以问题规模 $|I|$ 的多项式为上界，则称 A 是该问题类的一个 PTAS。

   理论上 A 在多项式时间内可以无限近似；不过对于不同的 ε，A 的运行时间上界也可能不同，例如可以令 A 的复杂度为 $O(|I|^{1/\varepsilon})$ 甚至 $O(|I|^{\exp(1/\varepsilon)})$，这样算法的表现就会很糟糕，因为指数过大。一般可以将 PTAS 的复杂度记为 $O(|I|^{f(1/\varepsilon)})$。

2. EPTAS（Efficient PTAS）：在 PTAS 的基础上，要求算法 A 的复杂度是 $O(|I|^c)$ 的，其中 $c \geqslant 0$ 是与 ε 无关的常数。可以将 EPTAS 的复杂度记为 $|I|^{O(1)} f(1/\varepsilon)$。

3. FPTAS（Fully PTAS）：在 PTAS 的基础上，要求算法 A 的运行时间关于 $|I|$ 和 ε 都呈多项式，即可以将 FPTAS 的复杂度记为 $|I|^{O(1)}(1/\varepsilon)^{O(1)}$。

除了上面三者外，还有范围最广的 APX，代表可近似（approximable），如果某个 NP 问题存在近似比为常数的多项式时间近似算法，则称该问题属于 APX；除此之外还有 QPTAS 即 Quasi PTAS，对于每个固定的 ε，要求算法 A 的复杂度是 $O(n^{(\log n)^c})$ 的。事实上，除非 P = NP，否则都有 FPTAS $\subsetneq$ PTAS $\subsetneq$ APX，并且 APX 困难问题是没有 PTAS 的。



当面临一个具体的问题类时，有时能得到乐观的结果，例如：

- 0-1 背包问题存在 FPTAS；
- 欧氏空间中的 TSP 存在 PTAS。

另一方面，在 P $\neq$ NP 或者 NP $\neq$ ZPP 的假设下，也可能得到悲观的结果，例如顶点覆盖（Vertex Cover）不存在 PTAS（这是一个 APX 困难问题）。

当然上面主要是通过近似比定义近似算法的优良程度，事实上还有一种更自然却不常用的定义方式如下：

定义 11.2 *对一个最优化问题 $\mathcal{I}$ 的一个绝对近似算法是关于问题规模 $|I|$ 的多项式时间算法 A，使得对任意实例 I 都有*

$$|A(I) - OPT(I)| \leqslant k,$$

其中 k 是一个常数。

尽管这一定义看起来非常合理，但只对极少数经典的 NP 困难优化问题才已知存在绝对近似算法，例如着色问题。

漫长的定义结束后，将开始对几个经典例子的讨论。在开启正式旅程之前，容许我感谢知乎用户金鱼马（也是 20 级图灵班的同学）对讲义的支持，前面的定义部分以及装箱问题都参考了他的知乎专栏，也很推荐对优化等领域感兴趣的读者关注他的文章。

11.2 最小化工时调度问题

第一个例子来源于之前贪心算法中的调度问题的变种：假设有 n 个作业，分别具有正的长度 $l_1, \ldots, l_n$，有 m 台相同的机器。需要求出作业与机器的一种分配方案，使得耗时最长的机器的运行时间最短。

很显然的，假设一个作业能被拆分到不同机器上执行，那最优解就是在每个机器上平分，最短时间也就是 $\sum_{i=1}^{n} l_i/n$。然而要求每个作业不能被拆分，举个简单的例子，假如有两台一样的机器，作业长度分别为 $[1,2,2,3]$，那么可以将作业分配为 $[1,2],[2,3]$，这样完成时间最小为 5。如果将作业分配为 $[1,3],[2,2]$，那么完成时间最小为 4。这一问题直接使用贪心算法无法保证最优解，所以考虑能不能有近似解。

在开始正式的讨论之前，先来看一个动态规划算法，问题的设置略有不同。将问题场景设置为只有两台不同的机器 A 和 B，第 i 个任务在处理处理机 A 上处理需要的时间为 a_i，在处理机 B 上处理的时间为 b_i，需要求耗时最长的机器的最短运行时间（称最小化的最长机器运行时间为最小工时）。

很显然的，这是原始版本 $m = 2$ 的更一般化的情况。动态规划需要找到最优子结构并写出递推式。对于这个问题，可以考虑当完成第 k 个任务时，有两种可能：

1. A 处理机完成了第 k 个任务，那么 B 处理机完成 k 个任务的最短时间就与 B 处理机完成 $k-1$ 个任务所需的最短时间是相同的；
2. B 处理机完成了第 k 个任务，那么 B 处理机完成 k 个任务的最短时间就等于 B 处理机完成 $k-1$ 个任务的最短时间加上 B 处理机完成第 k 个任务所需要的时间。

设 $F[k][x]$ 表示完成第 k 个任务时 A 耗费的时间为 x 的情况下，B 所花费的最短时间，其中 $0 \leqslant k \leqslant n$，$0 \leqslant x \leqslant \sum_{i=1}^{n} a_i$，那么，状态转移方程为

$$F[k][x] = \min\{F[k-1][x-a_k], F[k-1][x] + b_k\},$$

最终的结果即是完成 n 个任务时 A 和 B 所需时间的较大值，即 $\min_x \max\{F[n][x], x\}$。实际上这和 0-1 背包问题的动态规划设计思路是类似的，只是思路上绕了一些。

看到有动态规划算法，似乎找到了多项式时间的解决方案。然而只要仔细观察 x 的取值范围，就会发现这里其实出现了和 0-1 背包同样的问题，即 $\sum_{i=1}^{n} a_i$ 其实是输入的指数函数，因为输入只需要 $\log \sum_{i=1}^{n} a_i$

级别的二进制编码长度，所以也是伪多项式的算法。事实上，即使是 A 和 B 是同样的机器（即所有作业在其上运行时间一致），也有如下结论：

定理 11.3 给定若干个作业，两台一样的机器，判定问题：是否存在一种分配方案使得最小工时不超过 T 是 NP 完全的。

证明: 可由划分问题（Partition problem）规约。划分问题是 Karp 的 21 个 NP 完全问题之一，该问题表述为：给定若干个正整数，问可否将其划分成和相等的两个部分；换言之，给定 $c_1, \ldots, c_n \in \mathbb{Z}^+$，划分问题问是否存在一个 $S \subseteq \{1, \ldots, n\}$ 使得 $\sum_{i \in S} c_i = \sum_{i \notin S} c_i$。

为了将划分问题归约到最小化工时调度问题，构造一个最小化工时调度问题的实例。设输入的作业长度为

$$\frac{2Tc_i}{\sum_{j=1}^{n} c_j},$$

那么这些作业的总时长为 $2T$，显然这些作业能用两台机器在 T 时间内完成当且仅当两台机器上安排的任务时长一致，这也等价于划分问题存在解。由此实现了从划分问题到最小化工时调度问题的多项式归约，因此定理中的判定问题是 NP 完全的，这也表明最小化工时调度问题是 NP 困难的。 ■

在证明了困难性后，就可以名正言顺地讨论近似算法了——除非你有自信解决 P = NP。最简单的设计近似算法的思路就是用一些看起来很有道理但不能在所有解上返回最优解的贪心算法。在这里的目标显然是 m 个机器上尽可能做到负载均衡，所以会自然地有如下算法：

定理 11.4 (Graham,1969) 对每个作业进行调度的时候，选择当前工作量最小的机器进行调度。这样的算法近似比为 $2 - 1/m$。

下面就来证明这一结论。在证明近似比的时候，需要做的就是利用贪心算法的特点。

证明: 设 i 表示贪心算法的调度最后得到的具有最大负载的那台机器，设其工时为 M，j 是分配给这个机器的最后一个作业。我们知道算法每次找到的机器的负载都是当时最小的，现在退回到选择 j 之前，则当时全体机器的负载之和为 $\sum_{k=1}^{j-1} l_k$，因此当时最小负载的机器的负载应当是小于等于 $\frac{1}{m}\sum_{k=1}^{j-1} l_k$ 的。因此有

$$M \leqslant l_j + \frac{1}{m}\sum_{k=1}^{j-1} l_k \leqslant l_j + \frac{1}{m}\sum_{k \neq j} l_k,$$

将上式变形有

$$M \leqslant (1 - \frac{1}{m})l_j + \frac{1}{m}\sum_{k=1}^{n} l_k,$$

我们知道 $l_j \leqslant M^*$（最优解一定比每个作业单独的长度更长），并且 $\frac{1}{m}\sum_{k=1}^{n} l_k \leqslant M^*$，因为这是代表了所有作业均匀分配到每台机器上的情况，一定是最优解的下界。将这两个结果带入上面的不等式有

$$M \leqslant (1 - \frac{1}{m})M^* + M^* = (2 - \frac{1}{m})M^*,$$

然后作为近似比一定要是下界，即要对任意的 m 找到一个例子是满足这一近似比的。考虑 m 台机器，$m(m-1)$ 个长度为 1 的作业和 1 个长度为 m 的作业作为输入，那么显然最优解是 m，但如果用 Graham 算法解为 $2m-1$，恰好符合上述近似比。因此贪心算法的近似比为 $2-1/m$。 ■

或许这个近似比还有改进的余地：如果观察恰好符合近似比的例子，我们发现问题出在最后才把最长的的作业进行调度。如果先对输入的作业根据长度从大到小排序，然后再调用 Graham 算法，对于这个例子是能得到最优解的。事实上这样先排序后贪心的算法也的确能给出更紧的近似比：

定理 11.5 *上述先排序后调用 Graham 算法的算法得到的近似比为* $\frac{4}{3}-\frac{1}{3m}$。

对这一结果感兴趣的读者可以阅读蒂姆 · 拉夫加登的《算法详解》第四卷，这里因为篇幅和编写时间限制无法给出证明。

11.3 装箱问题

提到近似算法，不可能不提到装箱这一经典的问题。经典的（一维）装箱问题具体描述为：给定若干个带有尺寸的物品，要求将所有物品放入容量给定的箱子中，使得每个箱子中的物品尺寸之和不超过箱子容量并使所用的箱子数目最少。简单起见，一般将上述模型标准化：给定 n 尺寸在 $(0,1]$ 内的物品 $a_1,a_2,\ldots,a_n$，目标是使用数量尽可能少的单位容量箱子装下所有物品，每个箱子中物品尺寸和都不超过 1。该问题是 NP 完全的；例如，给定若干个物品，问两个箱子是否能够装下，这个看起来简单的事情在多项式时间内不可解。

定理 11.6 *给定若干个物品，判断它们是否可由两个箱子装下是 NP 完全的。*

证明: 再次使用划分问题。回顾一下划分问题的表述：给定若干个正整数，问可否将其划分成和相等的两个部分；换言之，给定 $c_1,\ldots,c_n\in\mathbb{Z}^+$，划分问题问是否存在一个 $S\subseteq\{1,\ldots,n\}$ 使得 $\sum_{i\in S}c_i=\sum_{i\notin S}c_i$。

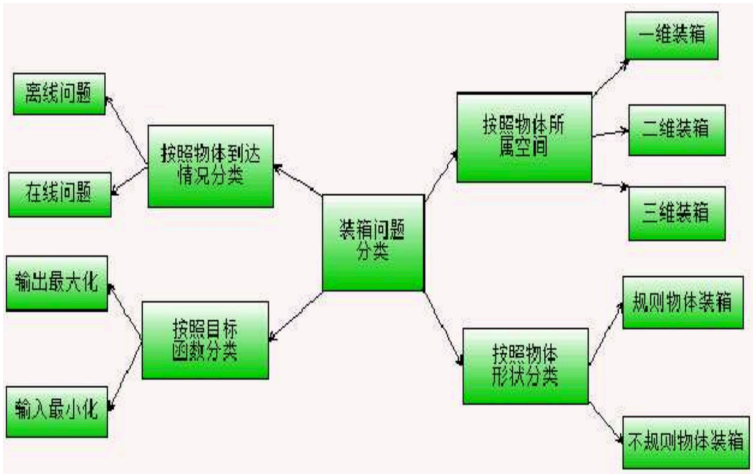
根据一个划分问题实例，可以构造一个装箱问题，令物品的尺寸为

$$a_i=\frac{2c_i}{\sum_{j=1}^{n}c_j},$$

同时假设这些 $a_i\in(0,1]$，由于 $\sum_{i=1}^{n}a_i=2$，那么显然这些物品能用两个大小为 1 的箱子装下当且仅当两个箱子都正好装满，这也同时解答了划分问题。这样就把一个已知的 NP 完全问题规约为了两个箱子的装箱问题。而显然一个给定的装箱问题的解可在多项式时间内验证，因此该问题是 NP 完全的。■

事实上上面的证明和最小化工时调度问题是类似的，其实最小化工时调度问题可以视为装箱个数固定最小化最大箱子中的物品大小的问题，因此和这里固定箱子最大的容积求最小化箱子个数的问题是对偶的。

装箱问题有极多变体，例如多维的装箱，包括几何装箱和向量装箱，以及变尺度装箱等。本节并不考虑这些变体，只介绍经典一维装箱的近似算法。一维装箱问题发展至今，其研究已经相对比较成熟。



之前衡量近似算法采用的都是“近似比（approximation ratio）”，也被称作“绝对近似比”，定义为最坏情况下算法的求解值与问题的最优值之间的比率。对任意实例 I ，用 $A(I)$ 表示针对实例 I 运行算法 A 后所用箱子数；$OPT(I)$ 表示装完实例 I 中所有物品所用的最少的箱子数。若存在常数 $\alpha \geqslant 1$，对任意实例 I 有 $A(I) \leqslant \alpha \cdot OPT(I)$，则 A 的近似比至多是 α。关于一维装箱问题，有如下结论：

定理 11.7 *除非 $\mathsf{P} = \mathsf{NP}$，否则装箱算法不存在多项式算法有小于 3/2 的绝对近似比。*

证明： 反证法。如果存在近似比小于 $\dfrac{3}{2}$ 的多项式时间近似算法 A，那么直接用它判断物品是否可由两个箱子装下：

1. 如果确实可以由两个箱子装下，即 $OPT(I) = 2$，那么近似算法根据近似比要求返回的解必定是 $A(I) < 3$，但这一近似算法返回的值必定是整数，于是 $A(I) = 2$，于是近似算法直接就返回了正确答案；
2. 如果 $OPT(I) \geqslant 3$，那么 $A(I) \geqslant 3$，此时因为近似比低于 3/2，表明最优解不可能是 2，所以通过近似算法的解也能判断出物品是否可由两个箱子装下。

这样就回答了两个箱子的装箱问题，是通过多项式时间的归约（因为这里甚至不需要归约）到一个多项式时间的算法实现的。而它是一个 NP 完全问题，因此只可能有 $\mathsf{P} = \mathsf{NP}$。 ■

这里证明用到的 trick 在之后的不可近似的证明中也是常用的，其实就是通过在解一定是整数这一条件下，近似的答案只能是标准答案这样的观察来实现。当然上述结论的 3/2 是有算法可以恰好压线的，后面会看到，FFD（First Fit Decreasing）算法的绝对近似比就是 3/2，不能再改进。

在装箱问题中，一般不关心全部的实例，而关心 $OPT(I)$ 较大的那些实例。因此定义“渐近近似比（asymptotic approximation ratio）”如下：

定义 11.8 如果对任意常数 $\alpha \geqslant 1$，对任意实例 I，存在一个常数 k，满足

$$A(I) \leqslant \alpha \cdot OPT(I) + k$$

称所有满足上式的 α 的下确界为 A 的渐近近似比。

可以看出 α 决定了当 $OPT(I)$ 充分大时 $A(I)$ 与 $OPT(I)$ 的比值。k 除了可以是某些固定的常数，也可以是 $o(OPT(I))$，只需要在 $OPT(I)$ 充分大时 $k/OPT(I) \to 0$ 即可。当 $k = 0$ 时渐近近似比就成了绝对近似比。

还有一个概念需要强调，装箱问题中，若所有的物品信息在开始装箱前已知，则它是离线（offline）问题；若初始时物品信息并不全部给出，例如物品在传送带上逐个到达，需要即时安排，且对未到达物品信息一无所知，同时做出的决定无法更改，此时称为在线（online）问题。“近似比” 通常用来描述离线问题近似算法的性能。而对于在线问题，一般用 “竞争比（competitive ratio）” 的概念。近似比的存在来自于计算资源有限，而竞争比的存在来自于对问题信息所知有限。在线算法领域里，装箱问题也是一个十分重要的问题，不过本讲不多做介绍。PPT 第 9 页给出了关于在线算法的最差情况的一个 5/3 的结论（第 8 页给出了对应的构造），第 10 页则讨论了离线的 FFD 算法，这一点后面会展开介绍。

这里主要讨论贪心的 Fit 算法。这是一类按照某种简单规则依次将物品放入箱子的启发式算法，其中包括

1. Nest Fit（NF）：打开一个新箱子，将第一个物品装进箱子。当后续物品到达时，如果当前打开的箱子有足够的空间，就直接放入；否则关闭该箱子（不再接受任何新的物品），再打开一个新的箱子，将该物品放进去。

   - 不难看出对任意实例 I，$NF(I) \leqslant 2OPT(I) - 1$。PPT 第 6 页给出了证明，实际上就是利用 Next Fit 的特点：相邻两个箱子内物品尺寸之和大于 1（否则它们应当放到同一个箱子里），因此如果用到了 $2OPT(I)$ 个箱子，那一定表明所有物品总的大小超过了 $OPT(I)$，然而每个箱子大小为 1，如果所有物品总的大小超过了 $OPT(I)$，那么即使每个箱子都恰好装满也要用多于 $OPT(I)$ 个箱子，从而 $OPT(I)$ 不再是最优解，矛盾。
   - 事实上装箱的渐近近似比就是 2。上界来自于上面的推导；下界来自这样的一个实例：所有物品的大小为 $\frac{1}{2}, \varepsilon, \frac{1}{2}, \varepsilon, \ldots$，共有 m 组 $\frac{1}{2}, \varepsilon$，其中 $\varepsilon > 0$ 充分小，m 充分大，那么 NF 解为 m，最优解为 $m/2 + 1$。

     需要注意的是，这种取 ε 来控制近似比的样例在近似算法中非常常见，之后还会见到。
   - NF 的近似效果很差，因为其仅保持一个打开的箱子，提前关闭的箱子之空间没有得到利用。

2. Any Fit：这是一类 Fit 方案，满足如下性质：当物品到达时，除非所有目前打开的箱子都无法装下该物品，才允许打开一个新箱子。包括下面几种：

   - First Fit（FF）：选择最早打开的箱子优先填入；
   - Best Fit（BF）：选择最满的箱子（剩余空间最小）优先填入；

- Worst Fit（WF）：选择最空的箱子（剩余空间最大）有限填入。

前面的所有 Fit 算法都是在线算法。此外，Any Fit 的三种算法都满足相邻两个箱子物品尺寸之和大于 1，因此它们都不会比 NF 差。而前面 NF 的下界实例也适用于 WF，因此 WF 和 NF 一样差。

从定义来看 BF 算法似乎比 FF 算法有更高的箱子利用率，但并非总是如此。

例 11.9 考虑下面 5 个物品的实例：0.5、0.7、0.1、0.4、0.3。此实例 FF 需要 2 个箱子，而 BF 则需要 3 个；考虑下面 4 个物品的实例：0.5、0.7、0.3、0.5，此实例用 FF 需要 3 个箱子，而 BF 只需要 2 个。这两个例子说明不会存在 "BF 总比 FF 好" 或者 "FF 总比 BF" 好的结论。

与 WF 不同，FF 和 BF 算法满足一个更强的性质：设第 k 个箱子是当前打开箱子中剩余空间最大且最晚打开的一个。亦即，第 k 个打开的箱子剩余空间最大，和它有相同剩余空间的箱子打开得都比它早；那么除非当前物品无法装进前 $k-1$ 个箱子里，否则它不会装进第 k 个箱子里。满足此性质的算法称为 Almost Any Fit（AAF）算法。FF 和 BF 是 AAF 算法，而 NF 和 WF 不是。

可以很容易修正 WF 将其变成 AAF 算法。修正后的算法记为 Almost Worst Fit（AWF）：将当前物品放入能装下它的剩余空间第二大的箱子中；若这样的箱子不存在，便将其放入能装它的剩余空间最大的箱子中；若上述两种情况都不存在，则新打开一个箱子将其装入。AWF 算法看起来只是对 WF 算法的微小修正，但可观察到 AWF 属于 AAF。

有如下渐近比事实：NF 和 WF 的渐近比都是 2，而任意的 AAF （FF、BF、AWF）都可以达到相同的渐近比：1.7。特别地，对于 FF 算法，设 $\mathrm{FF}(I)$ 表示对 I 运用 FF 得到的装箱数。可以证明，对于任意实例 I，有

$$FF(I) \leqslant 1.7OPT(I) + \frac{4}{5},$$

这可以说明 First Fit 有 1.7 的渐进比。对于这一结论的证明感兴趣的读者可以阅读这篇知乎文章。

如果允许首先将所有物品从大到小排序，再使用 First Fit，这种方法叫做 First Fit Decreasing（FFD）。显然这是一种离线算法，因为要在知道所有的输入的前提下才能排序，有两个结论：

- 绝对近似比 $FFD(I) \leqslant \frac{3}{2}OPT(I)$；
- 渐近近似比 $FFD(I) \leqslant \frac{11}{9}OPT(I) + \frac{6}{9}$。

第一个证明比较简单：

证明： 设所有的物品为 $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_n > 0$，考虑第 $j = \left\lceil \frac{2}{3}FFD(I) \right\rceil$ 个箱子 B_j，如果它包含了一个 $a_i > 1/2$ 的物体，那么 B_j 前面的箱子中物品体积都超过 $1/2$，由于排在 a_i 前面的物体体积都超过 $1/2$，因此至少有 j 个体积超过 $1/2$ 的物品，这些物品都得放在不同的箱子里，于是

$$OPT(I) \geqslant j \geqslant \frac{2}{3}FFD(I),$$

若不然，若 B_j 里面没有体积超过 1/2 的物品，那么除了最后一个箱子 $B_{FFD(I)}$，B_j 及其之后的箱子 $B_j, B_{j+1}, \ldots, B_{FFD(I)-1}$ 内至少都有两个体积不超过 1/2 的物品。于是至少有 $2(FFD(I)-j)+1$ 个物品都无法放入 $B_1, B_2, \ldots, B_{j-1}$，所以

$$\begin{aligned} OPT(I) &> \min\left\{j-1, 2\big(FFD(I)-j\big)+1\right\} \\ &\geqslant \min\left\{\left\lceil\frac{2}{3}FFD(I)\right\rceil - 1, 2\left(FFD(I)-\left(\frac{2}{3}FFD(I)+\frac{2}{3}\right)\right)+1\right\} \\ &= \left\lceil\frac{2}{3}FFD(I)\right\rceil - 1 \end{aligned}$$

由于 $OPT(I)$ 是整数，所以 $OPT(I) > \left\lceil\frac{2}{3}FFD(I)\right\rceil - 1$ 意味着 $OPT(I) \geqslant \left\lceil\frac{2}{3}FFD(I)\right\rceil \geqslant \frac{2}{3}FFD(I)$，即 $FFD(I) \leqslant \frac{3}{2}OPT(I)$。

另一方面，3/2 是紧的，考虑实例 $\left\{\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right\}$ 即可。 ■

这说明 FFD 是装箱问题的最优绝对近似比算法。

第二个结论证明较为复杂，见：Dósa, G. (2007). The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is $FFD(I) \leqslant 11/9OPT(I) + 6/9$. *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*。

除此之外，一维装箱问题还有渐进 PTAS，感兴趣的读者同样可以阅读这篇知乎文章进行初步的了解。

11.4 0-1 背包问题

11.4.1 基础的 2-近似算法

在贪心算法一节中已经看到了贪心算法解决分数背包问题的方法，实际上思想非常简单，就是让背包的每一单位重量的价值最大化。于是有一个自然的问题：对于 0-1 背包问题而言，这样的贪心算法是最优的吗?

这个问题很好回答，只需要构造一个简单的反例即可。考虑一个 0-1 背包问题，有两个物品，第一个物品的价值为 1，重量为 1，第二个物品的价值为 2，重量为 3，背包的容量为 3。贪心策略是让每单位重量的价值最大化，因此贪心算法的选择结果是选择第一个物品，但实际上最优解显然是选择第二个物品，这样背包的价值为 2，而贪心算法的解为 1。因此贪心算法并不是最优的。

事实上这是整数规划转线性规划的应用，原先 0-1 背包对应于整数规划，因为每个物品只能取 0 或 1 这样的整数个，但分数背包问题的贪心方法则可能选取分数个物品，而分数背包问题的线性规划最优解实际上也就是贪心算法对应的解。当然这只是一些思想上的说明，感兴趣的同学可以进一步了解。

为了得到近似比，需要进一步改进贪心算法：有两个贪心策略，其一是根据这里的单位重量的价值（即 PPT 上的 profit density），其二是直接贪心选择最大价值的物品，运行两个算法选择最优解，可以证明，这样结合后的近似比为 2。设分数背包问题的最优解为 P_{frac}，0 - 1 背包问题的最优解为 P_{OPT}，0-1 背包问题贪心解（即分数背包舍去选分数个的物品）为 P_{greedy}，所有装得下的物品中的最大价值为 $p_{\max}$，那么有

$$p_{\max} \leqslant P_{greedy} \leqslant P_{OPT} \leqslant P_{frac}.$$

第一个不等式来源于算法是取两种贪心策略的最优值，其中之一就是按照价值从大到小贪心选择的，因此价值最大的（当然前提是能放下）物品一定会被选择，所以贪心算法至少比这个价值大；第二个不等式来源于贪心策略是一个可行解，可行解一定小于等于最优解。于是近似比有上界

$$\frac{P_{OPT}}{P_{greedy}} \leqslant \frac{P_{frac}}{P_{greedy}} \leqslant \frac{P_{greedy} + p_{\max}}{P_{greedy}} = 1 + \frac{p_{\max}}{P_{greedy}} \leqslant 2.$$

其中第二个不等式是因为贪心策略是把分数背包的解的那个取了分数个的物品舍去了，而舍去的这一物品的价这部分值不可能比最大价值还大，因此有 $P_{greedy} + p_{\max} \geqslant P_{frac}$。

为了说明 2 是近似比，需要构造一个例子。实际上并不困难，假设背包容量为 4，有三个物品，价值分别为 $2 + 2\varepsilon, 2, 2$，重量分别为 $2 + \varepsilon, 2, 2$，其中 ε 是一个很小很小的正数。可以发现，贪心策略会选择第一个物品，而最优解是选择后两个物品，因此有近似比为 2，不可能进一步优化。

11.4.2 FPTAS 算法

现在讨论一个更美好的算法，即一个可以无限近似的算法。利用动态规划一讲的算法可知 0-1 背包问题是有伪多项式算法的，即其复杂度是 $O(nC)$ 的，其中 n 是物品的数量，C 是背包的容量。但现在可以换个角度进行动态规划，数组第二个维度不再是背包的容量，而是价值，即原先的数组 $A[i][c]$ 表达的是前 i 个物品放入容量为 c 的背包的最大价值，现在的数组是 $A[i][v]$ 表达的是前 i 个物品放入价值为 v 的背包的最小重量。PPT 第 13 页给出了转移方程，事实上和原先非常类似。显然，这一动态规划的复杂度是 $O(nV)$ 的，其中 V 是所有物品的价值之和。做个简单的变换，设 $v_{\max}$ 是所有物品的价值的最大值，那么显然有 $V \leqslant nv_{\max}$，因此复杂度是 $O(n^2 v_{\max})$ 的。

化成这一形式的目的非常明确，因为可以非常容易地看出，如果 $v_{\max}$ 的大小是 n 的多项式级别的，就可以得到一个多项式时间的算法。然而并非所有实例都能满足这一条件，因此需要对输入的价值做一些技术性的处理，即对输入的所有价值做同比例的缩小，使得 $v_{\max}$ 能缩小到 n 的多项式级别。

接下来需要分析这样缩小的合理性。先不管这个同比例缩小的比例是多少，姑且设为 b。设输入的价值为 $v_1, \ldots, v_n$，缩小后的价值为 $v_1/b, \ldots, v_n/b$，但请注意这些值并不能直接放到动态规划中运行，因为这些值不一定是整数，但动态规划算法会用到这些价值作为数组的下标。因此需要将这些价值统一向上取整（向下取整也可以），则得到的价值集合为 $\lceil v_1/b \rceil, \ldots, \lceil v_n/b \rceil$，这样就可以将这些值放入动态规划中运行。

显而易见的，对于全体价值同时放缩一个比例 b，是不会影响动态规划选择的最优物品集合的（可以

看递推式，实际上就是里面出现的所有数都被放缩了一个比例，因此其中的大小关系不会变化），因此上面的动态规划算法选出的最优物品集合和当价值为 $\lceil v_1/b\rceil b, \ldots, \lceil v_n/b\rceil b$ 时的最优物品集合是一样的，只是后者的最优价值扩大了 b 倍。记 $\lceil v_i/b\rceil b$ 为 v_i'，那么显然 v_i' 和原问题的 v_i 差距应当不大（后面会形式化这一差距），因此有理由相信 $v_1', \ldots, v_n'$ 的最优解是 $v_1, \ldots, v_n$ 的最优解的一个近似。

总结一下思路：首先将全体价值缩小一个比例 b，使得最大价值是 n 的多项式级别的，然后做一个向上取整后调用动态规划算法得到准确的最优解 v 就是多项式级别的了，然后把向上取整后的价值放大回来，利用价值放缩不改变最优物品选择的特点，可知这样只是把前面动态规划得到的最优值扩大了 b 倍，因此 bv 就是放大回来后的最优解，然后因为放大回来的价值和原先的价值应当是接近的，因此有理由相信 bv 是原问题最优值的一个近似。

接下来就要确定比例 b，然后形式化算法并证明它的确是 FPTAS：

1. 给定想要达到的近似比率 ε（为了分析方便，假设 $1/\varepsilon$ 是整数，如果不是整数也可以向下找一个满足条件的数替代），令放缩比例 $b = \dfrac{\varepsilon v_{\max}}{n}$；
2. 将所有价值放缩为 $\lceil v_i/b\rceil$，然后运行动态规划算法得到最优解 v；
3. 然后将所有价值放大为 $v_i' = \lceil v_i/b\rceil b$，此时最优解为 bv，然后 bv 就是近似最优解了。

定理 11.10 *上述算法是 $0-1$ 背包问题的一个 FPTAS。*

证明: 首先时间复杂度是显然的，缩小价值后的的动态规划算法是 $O(n^2 v_{\max}/b) = O(n^3/\varepsilon)$ 的，因此是符合 FPTAS 的。接下来需要证明这一算法的近似比是符合 FPTAS 的。

首先需要将前面的一个直观形式化，即 $v_i' = \lceil v_i/b\rceil b$ 和 v_i 是接近的，事实上因为在向上取整时最多只会加上 1，因此有

$$v_i \leqslant v_i' \leqslant v_i + b.$$

设对于 $v_1, \ldots, v_n$ 而言的最优物品集合为 S^*，而对于 $v_1', \ldots, v_n'$ 而言的最优物品集合为 S，那么有

$$\sum_{i\in S} v_i' \geqslant \sum_{i\in S^*} v_i',$$

这是因为 S 是 $v_1', \ldots, v_n'$ 的最优解，而 S^* 是一个可行解。然后结合上述舍入的不等式有如下不等式链：

$$\sum_{i\in S^*} v_i \leqslant \sum_{i\in S^*} v_i' \leqslant \sum_{i\in S} v_i' \leqslant \sum_{i\in S} (v_i + b) \leqslant nb + \sum_{i\in S} v_i = \varepsilon v_{\max} + \sum_{i\in S} v_i,$$

比较链首尾，事实上和目标只差一步（首是真正的最优解，尾加号后的部分是近似的解），只要能估计出 $v_{\max}$ 和 $\sum\limits_{i\in S} v_i$ 的关系，就能得到近似比的上界。显然 $v_{\max} \leqslant \sum\limits_{i\in S} v_i$，因为根据 $1/\varepsilon$ 是整数的假设，$v_{\max}/b$ 实际上就是一个整数，向上取整对其没有影响，因此在动态规划算法中，选取 $v_{\max}/b$ 当然是一个可行解，所以最优解的价值应当大于等于 $v_{\max}/b$，再放大 b 倍回来其实就是不等式 $v_{\max} \leqslant \sum\limits_{i\in S} v_i$，综

上有

$$\sum_{i\in S^*} v_i \leqslant \varepsilon v_{\max} + \sum_{i\in S} v_i \leqslant (1+\varepsilon)\sum_{i\in S} v_i,$$

因此近似比满足 FPTAS 的要求。 ■

FPTAS 的存在性与所谓强 NP 困难的概念相关，所谓强 NP 困难，通俗（不够严谨）而言就是存在多项式函数使得即使把输入限制到这一多项式长度也是 NP 困难的（因此 0-1 背包显然不是强 NP 困难的）。关于强 NP 困难和 FPTAS，有如下结论：

定理 11.11 (Garey, Johnson, 1978) *一个具有整目标函数的强 NP 困难问题，若对某一多项式和所有实例 I 都满足*

$$OPT(I) \leqslant p(\text{size}(I)), \text{largest}(I),$$

则只有当 P = NP 时才会存在 FPTAS。

11.5 聚类问题

接下来讨论的问题也是一个已经见过的问题的变种（如果读者阅读了贪心算法一讲中的聚类问题）。当时的目标是让不同聚类之间越远越好，且直接使用 Kruskal 算法来解决问题，最后剩下几个联通分支一定是相距最远的可能。但是如果转换目标，希望同一个聚类之间越近越好，那么就需要使用不同的算法来解决问题。

首先给出问题的准确描述：输入 n 个点的位置 $s_1, \ldots, s_n$，以及一个整数 k，要求选择 K 个中心，使得每个点到最近的中心的距离最大值 $r(C)$ 最小化。这个问题被称为 k-中心问题（k-center problem）。PPT 的 15 页给出了清晰的图示，16 页给出了一些距离的定义，包括 $\text{dist}(s, C)$ 和 $r(C)$，相信理解起来还是不困难的。

直觉告诉我们，可以有一个非常简单的贪心算法（PPT 第 17 页），即把第一个点放在最合适的位置上（即使得第一个点到所有点的最大距离最小化）。PPT 上给出了这一算法合理的例子，但同时也给出了可以使得算法结果任意差的例子，因此还需要转换方向。

11.5.1 思维实验：如果我们已知最优解?

一个可行的解法来源于一个思维实验：假设已经知道最优解 $r(C^*)$（也许是神谕），这是否会有帮助呢? 这一利用思维实验，假设已知最优解并基于此构造算法的方式在近似算法设计中也是常见的。事实上，在知道了最优解 $r(C^*)$ 后，还是无法直接得到最优的中心放置的位置，所以现在如果任意放 K 个中心，以 $r(C^*)$ 为半径，很有可能做不到将所有点覆盖。但可以考虑这样一个事实：在知道了最优解 $r(C^*)$ 后，某个点 s 到最优解中最近的中心 C_s 的距离是不超过 $r(C^*)$ 的，既然任意选取 s 为中心时用 $r(C^*)$

为半径无法覆盖，那么如果放大半径，将真正最优解 C_s 为中心，$r(C^*)$ 为半径的圆内所有点都囊括进来，那不就还是可以实现覆盖吗?

接下来的问题是，半径需要放到多大。事实上结论是显然的：在半径设置为 $2r(C^*)$ 时就可以实现。这是因为 s 到 C_s 的距离为 $r(C^*)$，而 C_s 到最远的点的距离也不会超过 $r(C^*)$，因此 s 到 C_s 为中心，$r(C^*)$ 为半径的任意点的距离都不会超过 $2r(C^*)$。因此可以得到 PPT 第 19 页给出的 2-近似算法（Greedy-2r）：首先设置一个最优解 $r(C^*)$，然后从输入点集中随机选取第一个点作为第一个中心，然后删除该点为中心，$2r(C^*)$ 为半径的所有点，然后在剩余点中随机选择第二个中心，以此类推。根据前面的讨论，如果的确有神谕告诉我们最优解，那么这一算法在 K 步之内必然停止，且得到的解是最优解的 2 倍：

定理 11.12 *Greedy-2r 算法在给定最优解 $r(C^*)$ 的情况下是一个 2-近似算法。*

证明: 因为每次都是删除选取的中心 $2r(C^*)$ 为半径的圆内所有点，所以如果算法能在 K 步之内停止，那么得到的解一定是小于等于最优解的 2 倍，因此只需要证明算法能在 K 步之内停止即可。根据前面的讨论，如果最优解是 $r(C^*)$，那么在上面的算法中，每次随机选择一个剩余的点作为中心，$2r(C^*)$ 为半径的圆至少会带走一个真正的最优解中的点为中心，$r(C^*)$ 为半径的圆内所有点，因此 K 步之后必然最优解覆盖的所有点都被算法覆盖，因此必然停止。 ■

考虑逆否命题，这一定理及其证明表明：如果 Greedy-2r 算法在 K 步之内不停止，那么最优解一定不是 $r(C^*)$（即会大于 $r(C^*)$）。为什么要讨论这一逆否命题呢？因为事实上并没有神谕，因此 Greedy-2r 算法输入的最优解需要自己猜，PPT 的 20 页给出了二分查找的思路，根据逆否命题不断地判断选择的 $r(C^*)$ 是否合适即可。

11.5.2 不使用思维实验的近似算法设计

在尝到神谕的甜头后，可以来讨论一个抛弃神谕的方案：如果不知道最优解 $r(C^*)$，能否设计一个近似算法呢？事实上也是可以的，PPT 第 21 页给出了这一算法（Greedy-Kcenter），其基本思路是：首先从输入点集中随机选取一个点作为第一个中心，加入中心点集 C。然后每轮循环在剩余的点中找到一个点 s 的 $\mathrm{dist}(s, C)$ 最大，即 s 是到现有中心最短距离最大的点，将其加入中心点集 C，直到 C 中有 K 个点。这一算法的近似比是 2，需要依赖于前面 Greedy-2r 算法的正确性：

定理 11.13 *Greedy-Kcenter 算法是一个 2 -近似算法。*

证明: 反证法，设最优解为 $r(C^*)$，并假设 Greedy-Kcenter 算法给出的最优解大于 $2r(C^*)$，这说明 Greedy-Kcenter 算法结束后，一定存在一个点 s 距离所有的中心的距离大于 $2r(C^*)$，否则所有点都落在某个中心的 $2r(C^*)$ 范围之内，最优解一定不会大于 $2r(C^*)$。

回顾 Greedy-Kcenter 算法的步骤，每一步都在选择一个距离现有中心 C' 最远的点，既然 s 每一步都

没有被选到，这就说明每一步选取的点 c 都有（假设最终的中心为 C）

$$\mathrm{dist}(c, C') \geqslant \mathrm{dist}(s, C') \geqslant \mathrm{dist}(s, C) > 2r(C^*),$$

回顾 Greedy-2r 算法，在 Greedy-2r 算法中，每一步删除选择的中心的 $2r(C^*)$ 范围内所有点，然后随机挑选一个剩余点作为中心。而前面得到的 $\mathrm{dist}(c, C') > 2r(C^*)$ 表明在假设的情况（存在一个点 s 距离所有的中心的距离大于 $2r(C^*)$）下，Greedy-Kcenter 算法中每一步选择的点也是符合 Greedy-2r 算法的选择条件的，因为每一步都选择的是当前选过的中心 $2r(C^*)$ 开外的点，所以 Greedy-Kcenter 算法选择 K 个点就相当于 Greedy-2r 算法中选择 K 个点，而根据 Greedy-2r 算法的性质，K 步之后还剩下点 s 没有被覆盖，说明真正最优解一定大于 $r(C^*)$，与假设矛盾。 ■

11.5.3 聚类问题的最优近似比

有同学可能很感兴趣，前面我们给出的算法近似比都是 2，那么这个 2 是最优的吗：

定理 11.14 *除非 $\mathsf{P} = \mathsf{NP}$，否则 K-center 问题不存在 ρ-近似算法（$\rho < 2$）。*

为了证明这一命题，需要首先引入一个 NP 完全问题：dominating set 问题：给定一个图 $G = (V, E)$ 和一个整数 k，需要判断是否存在一个大小为 k 的集合 $S \subset V$，使得每个点要么在 S 中，要么与 S 中的某个点相邻。现在通过将 dominating set 问题归约到寻找 ρ-近似 k-center 问题（$\rho < 2$）来证明上述定理。

【讨论：】dominating set 问题和 vertex cover 问题是一样的吗？如果不一样，能给出例子说明吗？

证明: 为了归约，给定一个 dominating set 问题的实例 $G = (V, E)$ 和整数 k，构造一个 k-center 问题的实例：将 G 中相邻的点之间的距离设为 1，不相邻的点之间的距离设为 2，这一距离不难检验符合距离的三条定义。

显然的，此时 k-center 问题的解只可能是 1 或 2（因为所有的边长度只有 1 和 2 两种情况），并且存在半径为 1 的解当且仅当 dominating set 问题存在大小为 k 的解（由此看出 k-center 问题是 NP 困难的）。现在假设存在一个 ρ-近似算法 A（$\rho < 2$）来解决 k-center 问题，则有：

1. 如果 A 返回的解 r 在 $1 \leqslant r \leqslant \rho$，则说明存在一个半径为 1 的解（记住解只有 1 和 2 两种可能，此时不可能是 2），根据前面的分析有 dominating set 问题存在大小为 k 的解；
2. 如果 A 返回的解 r 在 $2 \leqslant r \leqslant 2\rho$，则说明不存在一个半径为 1 的解，即 dominating set 问题不存在大小为 k 的解。

由于归约是多项式时间的（只需要设置一些距离），然后可以通过 A 来在多项式时间解决 dominating set 问题，这就说明可以在多项式时间内解决 NP 完全的 dominating set 问题，这就说明了 $\mathsf{P} = \mathsf{NP}$。■

有的同学可能会疑惑，如果证明不存在 3-近似算法（甚至任意近似比的算法），好像所有的构造都是一样的，那为什么还能得到 2-近似算法呢？这里特别要注意距离的三角不等式条件，3-近似的构造就不再满足了。

11.6 旅行商问题

回顾旅行商问题：给定一个完全图 $G=(V,E)$，每条边 $e\in E$ 有一个权重 $w(e)$，求一条经过所有点的最短路径，即一条权重之和最小的哈密顿回路。

上一讲已经说明了旅行商问题的判定版本是一个 NP 完全问题，那么对应的旅行商问题是一个 NP 困难问题。自然地，我们希望找到一个多项式时间的近似算法来解决这个问题。然而有如下定理：

定理 11.15 *除非 $\mathsf{P}=\mathsf{NP}$，否则对于任意的 $k\geqslant 1$，TSP 不存在 k-近似算法。*

类似于聚类问题，这里将哈密顿回路问题归约到 TSP 的 k-近似算法：

证明: 反证法：假设存在一个 k-近似算法 A，则可以推出哈密顿回路问题有多项式时间的算法，根据哈密顿回路问题是 NP 完全问题可知 $\mathsf{P}=\mathsf{NP}$。

给定一个哈密顿回路问题的输入图 $G=(V,E)$，将其转化为一个 TSP 问题的实例，即一个完全图 G'，将 G' 的边权重定义如下：

$$w(e)=\begin{cases}1 & e\in E\\ 2+(k-1)n & e\notin E\end{cases},$$

其中 $n=|V|$。可以看到，如果 G 中存在哈密顿回路，则其对应的 TSP 问题的最优解为 n，否则回路长度至少为 $(n-1)+(2+(k-1)n)=kn+1$。而我们知道存在 A 是一个 k-近似算法，使用 A 进行计算，则有：

1. 如果算法多项式时间内返回的解为 n，则说明 G 存在哈密顿回路，这一点显然；
2. 如果 G 存在哈密顿回路，那么 A 在多项式时间内也必定返回 n，这是因为 A 是一个 k -近似算法，因此 A 的解至多为 kn，而其它任意解都是 $kn+1>kn$。

因此完成了从哈密顿回路到 TSP 的多项式时间归约（因为归约过程只需要对边赋权重），而用算法 A 可以在多项式时间内返回正确结果，这就说明了哈密顿回路问题可以在多项式时间内解决，这就说明了 $\mathsf{P}=\mathsf{NP}$。 ■

尽管这一定理的结果十分悲观，但实际上这里给出的归约构造在现实问题中并不合理：现实中的 TSP 边的权重（也就是旅行商在两点之间通行的距离）应当满足下述三角不等式：

$$w(u,v)\leqslant w(u,x)+w(x,v),$$

其中 u, v, x 是图中的任意三个点。将权重满足三角不等式的 TSP 问题称为度量 TSP 问题（因为三角不等式是度量空间的性质）。这让我们想起了上面的聚类问题，在那里也是通过度量空间的三角不等式限制从而允许了 2-近似算法的存在。那么在旅行商问题中，在添加度量空间的条件后，能得到怎样的结果呢?

定理 11.16 *旅行商问题存在如下 2-近似算法：首先求出图 G 的最小生成树 T，然后在 T 上按深度优先搜索（先序遍历）的顶点遍历顺序得到一条哈密顿回路，其权重不超过最小生成树的权重的两倍。*

证明: 设 H^* 表示在给定顶点集合上的一个最优旅行路。显然最小生成树的权值是最优旅行路线代价的一个下界:

$$c(T) \leqslant c(H^*).$$

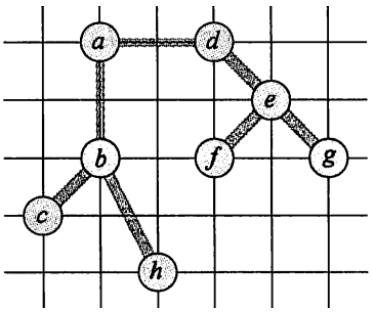
可以对 T 进行完全遍历，实际上就是深度优先搜索（记为 W），显然深度优先搜索恰好会经过 T 的每条边两次，所以有

$$c(W) = 2c(T),$$

从而有

$$c(W) \leqslant 2c(H^*),$$

即 W 的代价在最优旅行路线代价的 2 倍之内。但是，W 一般来说不是一个旅行路线，因为它对于某些顶点的访问次数超过一次。然而可以通过如下变换得到一条哈密顿回路（参考下面的图）：深度优先搜索（也就是先序遍历）的顺序是 $a \to b \to c \to b \to h \to b \to a \to d \cdots$，而如果将所有顶点除了第一次访问外的访问都去掉，得到的路线为 $a \to b \to c \to h \to d \to \cdots$，显然这样能得到哈密顿回路。并且这样的去除是不会增加原先深度优先搜索得到的解的代价的，这是由三角不等式保证的。此时不是按先序遍历进行遍历，而是按先序遍历的取点顺序每个点只取第一次出现，这样遍历的代价不超过最优解的二倍，并且得到的一定是一个哈密顿回路，所以的确得到了一个 2-近似算法。 ■



事实上上面证明的关键就是原先的先序遍历的路线就是一个不超过最优解两倍的路线，只是不是哈密顿回路罢了，然后通过三角不等式说明对路线做点的删除变成哈密顿回路是不会增加代价的即可。

事实上旅行商问题还有 3/2-近似算法，需要一定的其它知识背景故不在此介绍。此外，之前讨论的都是离散图上的旅行商问题，事实上还可以定义欧氏空间中的旅行商问题，在这一问题中点与点之间的距离是由欧氏距离给出的。限于难度以及背景知识需要，只能在这里陈述结论，即欧氏空间中的旅行商问题存在多项式时间近似方案，由 Arora（1998）和 Mitchell（1999）给出。

11.7 其它问题

由于时间和篇幅限制，无法在讲义中讨论所有经典的近似算法问题。顶点覆盖和集合覆盖问题是两个同样非常经典的例子，可以参考《算法导论》或《Algorithm Design》。《算法导论》的章末习题中也有很多经典的例子（包括作业题中的近似最大生成树），当然在前面已经讨论了一些。当然，这里给出的都是在目前知识范围内能接受的例子，如果希望进一步了解更深入、复杂的近似算法设计，可以选择相关的课程或教材。