算法设计与分析

主讲人：吴庭芳

Email: *tfwu@suda.edu.cn*

苏州大学 计算机学院







第十二讲 贪心算法

内容提要：

- 贪心算法思想
- 活动选择问题
- 贪心算法原理

贪心算法思想

- 求解最优化问题的算法通常需要经过一系列的步骤，在每个步骤面临多种选择。使用动态规划算法是将这些选择都计算一遍，从而得到最优选择
- 实际上，对于有些问题可以使用更简单、更高效的算法：贪心算法
  - 贪心算法是这样一种策略：它在每一步都做出当时看起来最佳的选择，也就是做出局部最优的选择，并寄希望这样的选择最终能导致全局最优解
  - 贪心算法并不保证得到最优解，但对很多问题确实可以求得最优解
  - 贪心方法是一种强有力的算法设计方法，可以很好地解决很多问题，比如最小生成树算法、单源最短路径的 Dijkstra 算法等







第十二讲 贪心算法

内容提要：

- □ 贪心算法思想
- □ 活动选择问题
- □ 贪心算法原理

活动选择问题

- **调度竞争共享资源的多个活动选择问题：**
  - 假定有一个包含 n 个活动的集合 S={a_1，a_2，…，a_n}，这些活动使用同一个资源（例如同一个教室），而这个资源在某个时刻只能供一个活动使用
  - 每个活动 a_i 都有一个开始时间 s_i 和结束时间 f_i，其中 $0 \le s_i < f_i$。如果被选中，任务 a_i 发生在半开区间 $[s_i, f_i)$ 期间
  - 如果两个活动 a_i 和 a_j 满足 $[s_i, f_i)$ 和 $[s_j, f_j)$ 不重叠，则称它们是兼容的，即 $s_i \ge f_j$ 或 $s_j \ge f_i$
  - 在活动选择问题中，希望选出一个最大兼容活动集。不失一般性，设活动已经按照结束时间单调递增排序：

$$f_1 \leqslant f_2 \leqslant f_3 \leqslant \cdots \leqslant f_{n-1} \leqslant f_n$$

活动选择问题

- **例如：设有 11 个待安排的活动，它们的开始和结束时间如下，并假设活动已经按结束时间的非减序排序：**

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

按结束时间的非减序排序

- 则 $\{a_3, a_9, a_{11}\}$、 $\{a_1, a_4, a_8, a_{11}\}$、 $\{a_2, a_4, a_9, a_{11}\}$ 都是兼容活动集合
- 其中 $\{a_1, a_4, a_8, a_{11}\}$、 $\{a_2, a_4, a_9, a_{11}\}$ 是最大兼容活动集合。显然最大兼容活动集合不一定是唯一的

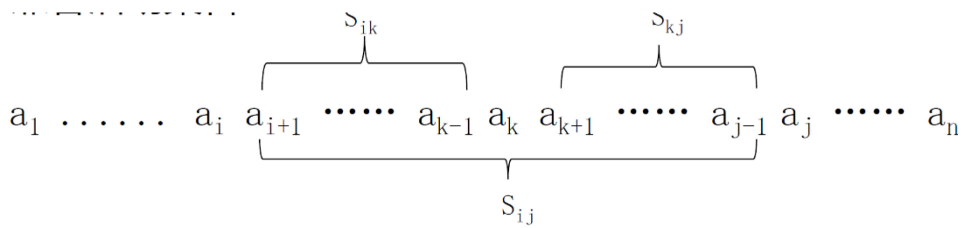
活动选择问题

- **活动选择问题分析：**
  - 可以通过动态规划方法将活动选择问题分为两个子问题，然后将两个子问题的最优解整合成原问题的一个最优解。在确定将哪些子问题用于最优解时，要考虑几种选择
  - 贪心算法更简单一些，只需要考虑一个选择（即贪心选择）。在做贪心选择时，只留下一个非空子问题

活动选择问题

活动选择问题的最优子结构

- 令 S_{ij} 表示在 a_i 结束之后开始，且在 a_j 开始之前结束的那些活动的集合



- 设 A_{ij} 是 S_{ij} 的一个最大兼容活动集合，并设 A_{ij} 包含活动 a_k（做出一个选择 k），则得到两个子问题：寻找 S_{ik} 的最大兼容活动集合（在 a_i 结束之后开始且 a_k 开始之前结束的那些活动）和寻找 S_{kj} 的最大兼容活动集合（在 a_k 结束之后开始且 a_j 开始之前结束的那些活动）
- 令 $A_{ik}=A_{ij}\cap S_{ik}$，$A_{kj}=A_{ij}\cap S_{kj}$，则 A_{ik} 包含 A_{ij} 中 a_k 开始之前结束的活动子集，A_{kj} 包含 A_{ij} 中 a_k 结束之后开始的活动子集，因此有 $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$（原问题 S_{ij} 的最优解 A_{ij} 由两个子问题的解所构成）

活动选择问题

- **活动选择问题具有最优子结构性，即：**
  - 证明：用剪切-粘贴法证明最优解 $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$ 必然包含两个子问题 S_{ik} 和 S_{kj} 的最优解，即 A_{ik} 必是 S_{ik} 一个最大兼容活动子集，A_{kj} 必是 S_{kj} 一个最大兼容活动子集
    - 设 S_{kj} 存在另一个最大兼容活动集 A'_{kj}，满足 $|A'_{kj}| > |A_{kj}|$，则可以将 A'_{kj} 作为 S_{ij} 最优解的一部分。这样就构造出一个兼容活动集合，其大小

$$|A_{ik}|+|A'_{kj}|+1 > |A_{ik}|+|A_{kj}|+1 = |A_{ij}|,$$

与 A_{ij} 是最大兼容活动集合相矛盾

活动选择问题

- **动态规划方法**
  - 活动选择问题具有最优子结构性，所以可用态规划方法求解
  - 令 $c[i, j]$ 表示集合 S_{ij} 的最优解大小，即兼容活动的个数，可以得到递归式如下：

$$c[i,j] = c[i,k] + c[k,j] + 1$$

为了选择 k，有：

$$c[i,j] = \begin{cases} 0 & \text{if } S_{ij} = \varnothing \\ \max\limits_{a_k \in S_{ij}} \{c[i,k] + c[k,j] + 1\} & \text{if } S_{ij} \neq \varnothing \end{cases}$$

  - 可以设计带备忘机制的或自底向上的动态规划算法进行求解

活动选择问题

活动选择问题的贪心算法

- 假如无需求解所有子问题就可以选择出一个活动加入到最优解中，那么可以省去上述递归式中考查所有选择的过程（遍历 k）。实际上，对于活动选择问题，只需考虑一个选择：贪心选择
- **贪心选择**：在贪心算法的每一步所做的当前最优选择（局部最优选择）
- 活动选择问题的贪心选择：每次总选择具有最早结束时间的兼容活动加入到集合 A 中
- **为什么?** 直观上，按这种方法选择兼容活动可以为未安排的活动留下尽可能多的时间。也就是说，该算法的贪心选择意义是使剩余的可安排时间段最大化，以便安排尽可能多的兼容活动

活动选择问题

- 活动选择问题的贪心算法
  - 注意：选择最早结束的活动并不是本问题唯一的贪心选择方法
  - 练习16.1-3，其他贪心选择有：选择持续时间最短者、选择与其他剩余活动重叠最少者、以及选择最早开始者，但均不能得到最优解

活动选择问题

- 例：

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

结束时间递增

- 由于活动已按结束时间单调递增的顺序排序，贪心选择就是活动 a_1。当做出贪心选择后，只剩下一个子问题需要求解：寻找在 a_1 结束后开始的活动
- 为什么不需要考虑在 a_1 开始前结束的活动？因为 $s_1 < f_1$，且 f_1 是最早结束时间，所以不会有活动的结束时间早于 s_1。因此所有与 a_1 兼容的活动都是在 a_1 结束之后开始
- 令 $S_k=\{a_i \in S \mid s_i \geq f_k\}$ 为在 a_k 结束之后开始的任务集合。当做出贪心选择，选择了 a_1 后，剩下的 S_1 是唯一需要求解的子问题

活动选择问题

- **例：**

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

结束时间递增

- 已经证明活动选择问题具有最优子结构性质，根据最优子结构性质：如果 a_1 在最优解中，那么原问题的最优解由活动 a_1 及子问题 S_1 的最优子解构成
- 对 S_1 可以继续按照相同的方式递归求解

活动选择问题

- **直觉正确吗？即按照上述贪心选择（最早结束的活动）方法选择的活动集合是问题最优解吗？即证明贪心选择性质**
- **定理 16.1** 考虑任意非空子问题 S_k，令 a_m 是 S_k 中结束时间最早的活动，则 a_m 必在 S_k 的某个最大兼容活动子集中。
- **证明：** 令 A_k 是 S_k 中的一个最大兼容活动子集，且 a_j 是 A_k 中结束最早的活动。1）若 $a_j = a_m$，则得证

  2）否则，令 $A'_k = (A_k-\{a_j\}) \cup \{a_m\}$，即将 A_k 中的 a_j 替换为 a_m。因为 A_k 中的活动都不相交， a_j 是 A_k 中结束时间最早的活动，而 a_m 是 S_k 中结束时间最早的活动，所以 $f_m \le f_j$，即 A'_k 中的活动也是不相交的。

  由于 $|A'_k| = |A_k|$，所以 A'_k 也是 S_k 的一个最大兼容活动子集，且包含 a_m，得证

活动选择问题

- **定理 16.1 的含义：选结束时间最早的 a_m 不会错！**
  - 对于活动选择问题，虽然可以用动态规划方法进行求解，但是并不需要这么麻烦
  - 相反，从 S_0 开始，可以反复选择结束时间最早的活动，重复这一过程直至不再有剩余的兼容活动。所得子集就是最大兼容活动集合
  - 由于结束时间严格递增，故只需按照结束时间的单调递增顺序处理所有活动，每个活动只需考查一次
  - 贪心选择算法通常是自顶向下设计：作出一个选择，然后求解剩余的那个子问题。而不是像动态规划策略那样自底向上地求解出很多子问题，然后再作出选择

活动选择问题

活动选择问题的贪心算法

- 采用自顶向下的设计：首先做出一个选择，然后求解剩下的子问题。每次选择将问题转化成一个规模更小的问题

```
RECURSIVE-ACTIVITY-SELECTOR(s, f, k, n)
m = k + 1
while m ≤ n and s[m] < f[k]        // find the first activity in S_k to finish
    m = m + 1
if m ≤ n
    return {a_m} ∪ RECURSIVE-ACTIVITY-SELECTOR(s, f, m, n)
else return ∅
```

查找 S_k 中最早结束的活动，直至找到第一个与 a_k 兼容的活动 a_m，即开始时间 s_m 大于活动 a_k 的结束时间 f_k

$m > n$，意味着在 S_k 中未找到与 a_k 兼容的活动

- 数组 s、f 分别表示 n 个活动的开始时间和结束时间，下标 k 指出要求解的子问题 S_k。并假定 n 个活动已经按照结束时间单调递增排列好，算法返回 S_k 的一个最大兼容活动集
- 初次调用：RECURSIVE-ACTIVITY-SELECTOR(s, f, 0, n)

活动选择问题

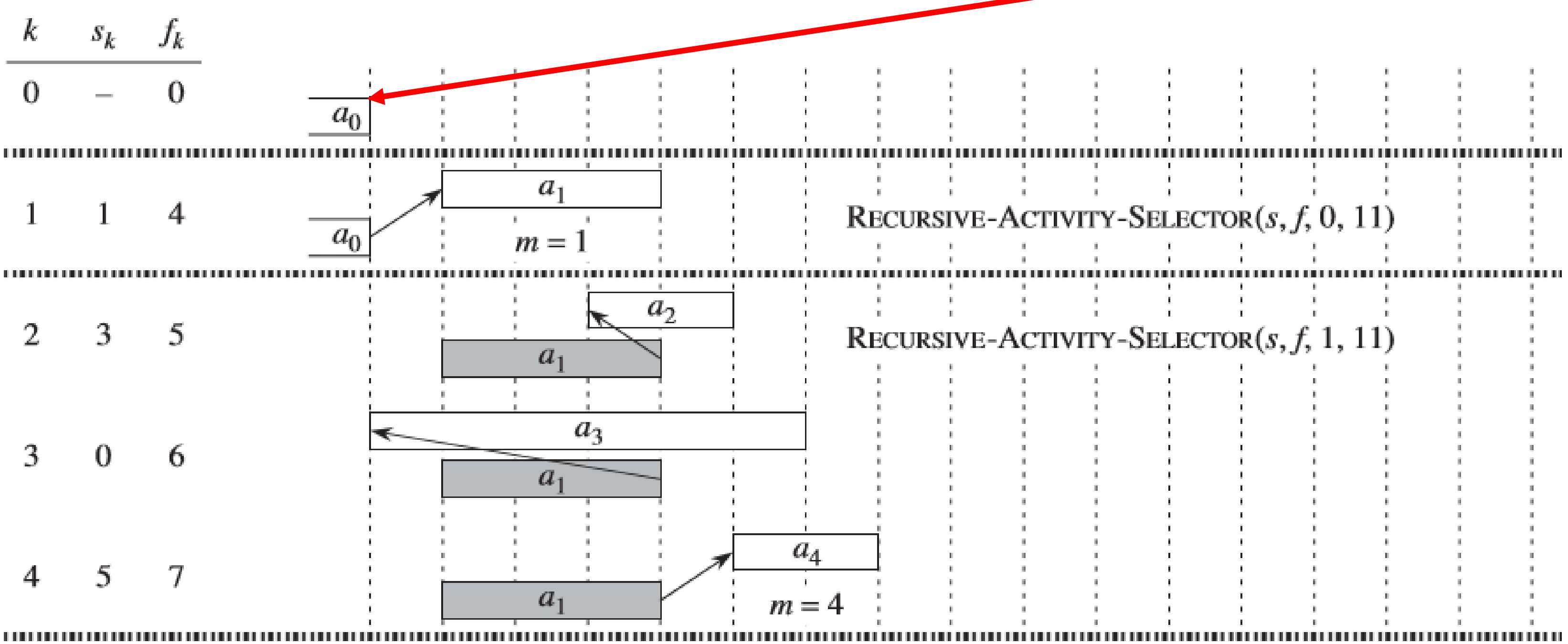
例：

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

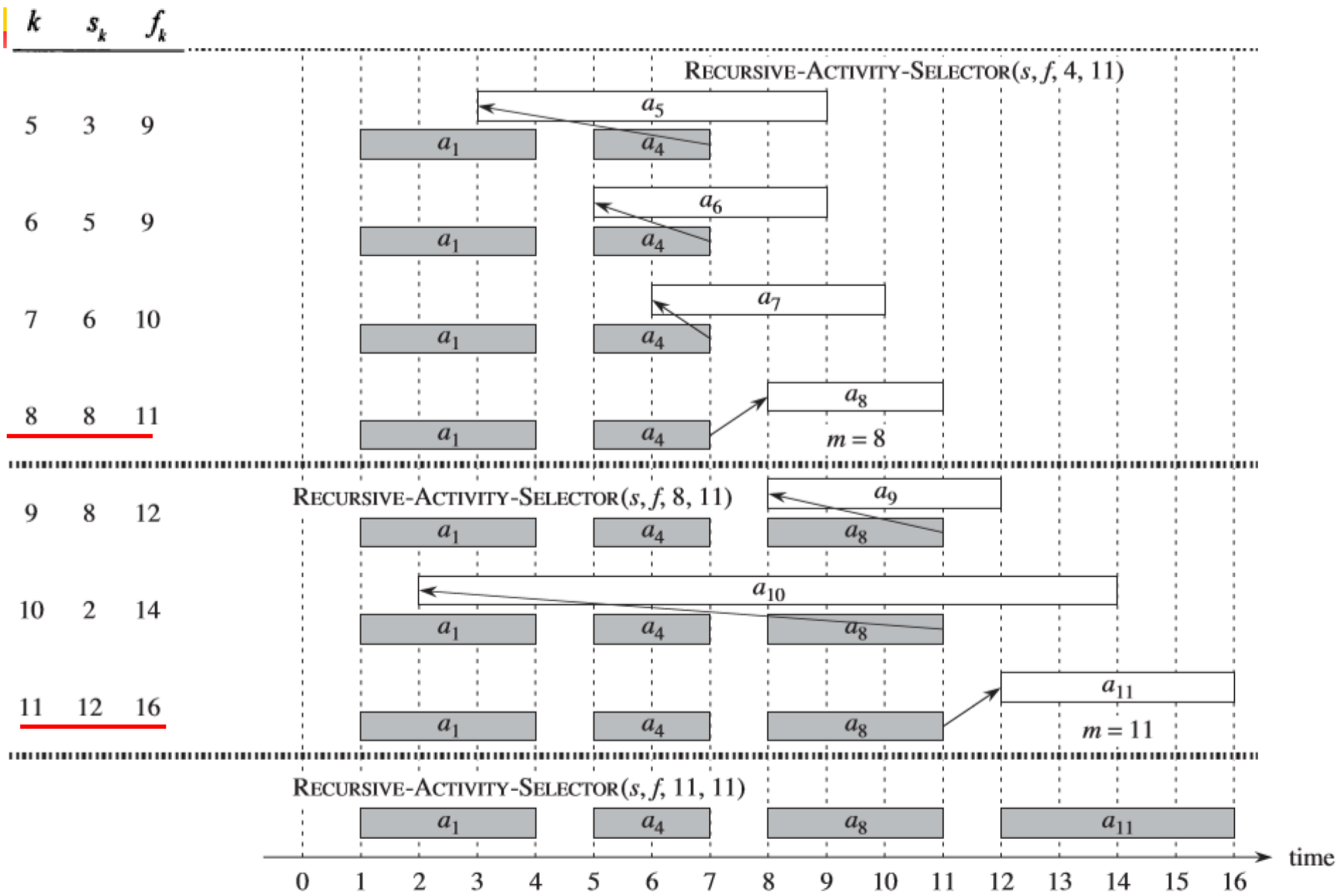
结束时间递增

执行过程如图所示：

注：为了处理的方便，这里引入一个虚拟活动 a_0，其结束时间 $f_0 = 0$



活动选择问题



k s_k f_k
5 3 9
6 5 9
7 6 10
8 8 11
9 8 12
10 2 14
11 12 16
RECURSIVE-ACTIVITY-SELECTOR(s, f, 4, 11)
RECURSIVE-ACTIVITY-SELECTOR(s, f, 8, 11)
RECURSIVE-ACTIVITY-SELECTOR(s, f, 11, 11)
a_1
a_4
a_5
a_6
a_7
a_8
a_9
a_{10}
a_{11}
$m = 8$
$m = 11$
time
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

活动选择问题

- 例：

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

结束时间递增

- 假定输入的活动已按结束时间的递增顺序排列，贪心算法只需 $O(n)$ 的时间即可选择出来 n 个活动的最大兼容活动集合。在整个递归调用过程中，每个活动被且只被第 2 行的 while 循环检查一次
- 如果所给出的活动未按非减序排列，可以用 $O(n\lg n)$ 的时间进行排序

活动选择问题

▫ 迭代实现的贪心算法

- 上述 RECURSIVE-ACTIVITY-SELECTOR 是一个“尾递归”过程：以一个对自身的递归调用再接一次并集操作结尾，可以很容易地转化为迭代形式
- 假定活动已经按照结束时间单调递增的顺序排列好

```
GREEDY-ACTIVITY-SELECTOR(s, f)
n = s.length
A = {a_1}
k = 1
for m = 2 to n
    if s[m] ≥ f[k]
        A = A ∪ {a_m}
        k = m
return A
```

集合 A 用于存入选出的活动

变量 k 对应最后一个加入 A 的活动的下标

f_k 是 A 中活动的最大结束时间

for 循环查找 S_k 中最早结束的活动，若 a_m 与之前选出的活动兼容，即开始时间 s_m 大于 f_k，则将 a_m 加入 A

算法的运行时间是 O(n)







第十二讲 贪心算法

内容提要：

- 贪心算法思想
- 活动选择问题
- 贪心算法原理

贪心算法原理

- **贪心算法通过做出一系列选择来求问题的最优解**

——即**贪心选择**：在每个决策点，它做出在当时看来是最佳的选择

  - 这种**启发式策略**并不保证总能找到最优解，但对有些问题确实有效，相比动态规划算法，贪心算法简单和直接得多

- **贪心算法通常采用自顶向下的设计，做出一个选择，然后求解剩下的子问题。每次选择将问题转化成一个更小规模的问题**

贪心算法原理

- **贪心算法求解的一般步骤：**
  1. 确定问题的最优子结构
  2. 基于得到的递归式设计一个递归算法
  3. 证明如果做出一个贪心选择，只剩下一个子问题需要求解
  4. 证明贪心选择总是安全的
  5. 设计一个递归算法实现贪心策略
  6. 将递归算法转换为迭代算法
- 贪心算法以动态规划方法为基础：对于活动选择问题，首先定义子问题 S_{ij}，其中 i 和 j 都可变。如果总是做出贪心选择，则可以将子问题限定为 S_k 的形式

贪心算法原理

- **贪心算法求解的一般步骤：**
  - 通过贪心选择来改进最优子结构，使得选择后只留下一个子问题。在活动选择问题中，将子问题定义为 S_k 的形式；然后，证明贪心选择（S_k 中最早结束的活动 a_m）与剩余兼容活动集的最优解组合在一起，就会得到 S_k 的最优解
  - 更一般地，贪心算法设计步骤：（1）将最优化问题转化这样的形式：每次对其作出选择后，只剩下一个子问题需要求解

    （2）证明作出贪心选择后，原问题总存在最优解，即贪心选择总是安全的

    （3）证明作出贪心选择后，剩余的子问题满足：其最优子解与前面的贪心选择组合即可得到原问题最优解 (具有最优子结构)

贪心算法原理

- **对应每个贪心算法，都有一个动态规划算法，但动态规划算法要繁琐的多**
- **如何证明一个最优化问题适合用贪心算法求解?**
  - 没有适合所有情况的方法
  - **贪心选择性质**和**最优子结构性**是两个关键要素
  - 如果能够证明问题具有这两个性质，则向贪心算法迈出了重要一步

贪心算法原理

- **贪心选择性质：**
  - 贪心选择性质：可以通过做出局部最优（贪心）选择来构造全局最优解的性质
  - 贪心选择性质使得我们进行选择时，只需做出当前看起来最优的选择，而不用考虑子问题解

贪心算法原理

- **贪心策略 VS 动态规划策略：**
  - 在动态规划方法中，每个步骤也都要进行一次选择，但这种选择通常依赖于子问题的解。因此，通常以自底向上地方式求解动态规划问题，先求解较小的子问题，然后才能求解较大的子问题
  - 在贪心算法中，我们总是做出当前看来最佳的选择，然后求解剩下的唯一一个子问题。贪心算法进行选择时可能依赖之前做出的选择，但不依赖任何将来的选择或子问题的解
  - 动态规划要先求解子问题才能进行第一次选择，贪心算法在进行第一次选择之前不需要求解任何子问题
  - 动态规划算法通常采用自底向上的方式完成计算，而贪心算法通常是自顶向下的，每一次选择，将给定问题实例转换成更小的问题

贪心算法原理

- **如何证明每次贪心选择能生成全局最优解?**
  - 必须证明每个步骤做出贪心选择能生成全局最优解
  - 通常首先考查某个子问题的最优解，然后**用贪心选择替换某个其它选择**来修改此解，从而得到一个相似但更小的子问题
  - 如果进行贪心选择时不得不考虑众多选择，通常意味着可以改进贪心选择，使其更为高效。例如，活动选择问题中，假定已经将活动按结束时间单调递增顺序排好序，则对每个活动能够只需要处理一次。通过对输入进行预处理或者使用适合的数据结构，通常可以使贪心选择更快速、更高效

贪心算法原理

- **最优子结构性**
  - 含义：一个问题的最优解包含其子问题的最优解
  - 最优子结构性质是能否应用动态规划和贪心方法的关键要素
- **贪心算法更为直接地使用最优子结构：**
  - 通过对原问题应用贪心选择后即可得到子问题
  - 需要证明：将子问题的最优解与贪心选择组合在一起就能生成原问题的最优解

贪心算法原理

- **对比动态规划算法和贪心算法：**
  - 0-1 背包问题和分数背包问题：都具有最优子结构性质
    - 0-1 背包问题：动态规划算法
    - 分数背包问题：贪心算法，按 p_i / w_i 的降序考虑问题

贪心算法原理

- **0-1 背包问题问题描述：**
  - 一个正在抢劫商店的小偷发现了 n 个商品，第 i 个商品的重量是 w_i 磅，其价值为 v_i 美元， w_i 和 v_i 都是整数。小偷的背包最多容纳 W 磅重的商品， W 是一个整数
  - 小偷应如何选择装入背包的商品，使得装入背包中商品的总价值最大?
  - 小偷在选择装入背包的商品时，对每种商品 i 只有 2 种选择，要么完整拿走，要么把它留下；不能将商品 i 装入背包多次，也不能只装入部分的商品 i

贪心算法原理

- **分数背包问题问题描述：**
  - 设定与 0-1 背包问题类似，但对每个商品，小偷可以只拿走商品 i 的一部分，而不是只能做出二元（0-1）选择
  - 比如，可以将 0-1 背包问题中的商品想象为金锭，而分数背包问题中的商品更像金砂

贪心算法原理

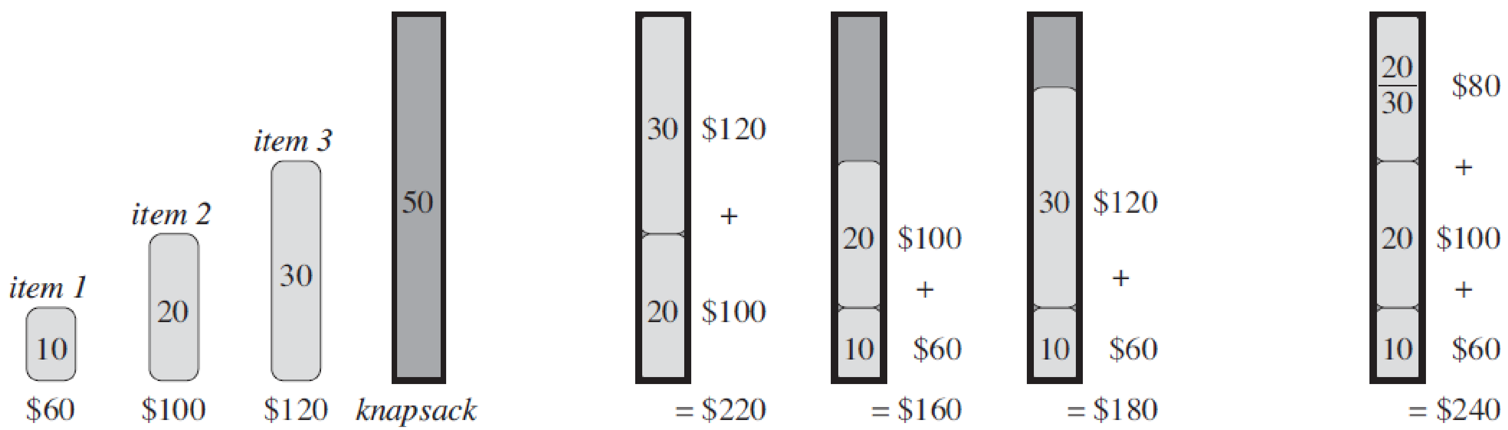
- **0-1 背包问题问题：**
  - 这 2 类背包问题都具有最优子结构性质
  - 对于 0-1 背包问题，考虑重量不超过 W 而价值最高的装包方案，如果将商品 j 从此方案中删除，则剩余商品必须是重量不超过 $W-w_j$ 的价值最高的方案，即小偷只能从不包括商品 j 的 n-1 个商品中选择拿走哪些

贪心算法原理

- **用贪心算法解分数背包问题的基本步骤：**
  - 可以用贪心策略求解分数背包问题，而不能求解 0-1 背包问题
  - 首先计算每种商品每磅的价值 v_i/w_i。然后，遵循贪心策略，小偷首先尽可能多地拿走每磅价值最高的商品。如果该商品已全部拿走而背包尚未满，继续尽可能多地拿走每磅价值第二高的商品，依此类推，直至达到重量上限 W
  - 因此，通过将商品按每磅价值排序，贪心算法的运行时间为 $O(n\lg n)$

贪心算法原理

- 对于 0-1 背包问题，贪心策略是无效的
- 下图所给出的问题实例：商品 1 的每磅价值为 6 美元，商品 2 的每磅价值为 5 美元，商品 3 的每磅价值为 4 美元。对于 0-1 背包问题，按照上述贪心策略，首先拿走商品 1；而最优解为拿走商品 2 和商品 3，而留下商品 1
- 因此，贪心策略对于 0-1 背包问题之所以无效是因为在这种情况下，它无法保证最终能将背包装满，部分闲置背包空间使得每磅背包空间的价值降低了



贪心算法原理

- 事实上，在考虑 0-1 背包问题时，应比较选择该商品和不选择该商品所导致的最终方案，然后再作出最好选择，由此就导出许多互相重叠的子问题—这正是该问题可用动态规划算法求解的另一重要特征，实际上动态规划算法的确可以有效地解 0-1背包问题







谢谢！

Q & A

作业：16.1-3

16.2-1, 16.2-2