课程回顾

- 动态规划问题：最长公共子序列LCS 、最优二叉搜索树、0-1背包

第4章 贪心算法

苏州大学 计算机科学与技术学院
汪笑宇
Email: xywang21@suda.edu.cn

引入

贪婪，我找不到一个更好的词来描述它，它就是好！它就是对！它就是有效！

——影片《华尔街》
美国演员迈克尔.道格拉斯

本章内容

- 贪心算法（教材Chapter 16）
  - 活动选择问题
  - 贪心算法原理
  - 分数背包问题
  - Huffman编码
  - 拟阵
  - 其他应用

贪心算法求解实例

- 最优解：
  - 调度问题
    - 活动选择问题（教材Chapter 16.1）
    - 任务调度问题（教材Chapter 16.5）
  - 图算法
    - 最小生成树（教材Chapter 23）
    - 单源点最短路径Dijkstra算法（教材Chapter 24.3）
  - 其他：Huffman编码（教材Chapter 16.3）
- 近似解：
  - 旅行商问题TSP（教材Chapter 35.2）
  - 集合覆盖问题（教材Chapter 35.3）
  - 子集和问题（教材Chapter 35.5）

启发式算法 (Heuristic algorithms)

- In mathematical programming, a heuristic algorithm is a procedure that determines near-optimal solutions to an optimization problem. However, this is achieved by trading optimality, completeness, accuracy, or precision for speed. Nevertheless, heuristics is a widely used technique for a variety of reasons:
  - Problems that do not have an exact solution or for which the formulation is unknown
  - The computation of a problem is computationally intensive
  - Calculation of bounds on the optimal solution in branch and bound solution processes

来源：https://optimization.cbe.cornell.edu/index.php?title=Heuristic_algorithms

贪心算法概述

- 求最优解的问题可看作是通过一系列步骤，每一步有一个选择的集合，对于较简单的问题，动态规划显得过于复杂，可用较简单有效的算法求解
- 贪心算法总是在当前步骤上选取最好的方案，即它是一种局部最优的选择，并希望它导致一个全局最优，但有时（或者是大部分）不可能导致全局最优
  - 例：求v_i到v_j的一条最短路径，若贪心地从v_i到最近距离点v_k，未必包含在v_i到v_j的最短路径中
- 但仍有许多问题贪心法将产生全局最优解，如最小生成树MST、单源最短路径等

贪心算法概述 (续)

■一般来说，贪心算法可解的问题有如下特性：

1. 优化问题，有一个候选对象集合，如零钱、边(Kruskal)、路径(Dijkstra)、顶点(Prim)等
2. 随着算法的进行，累积形成两个集合，一个是已经被选中的对象集合，另一个是被抛弃的对象集合
3. 函数1（solution function）：检查候选对象集合是否提供了问题的解，不考虑此时的解决方法是否最优
4. 函数2：检查候选对象是否可加入到当前解的对象集合中（可行的，feasible），不考虑解决方法的最优性
5. 选择函数：指出哪个剩余的候选对象（没有被选择过也没有被丢弃过）最有可能构成问题的解
6. 目标函数：给出解的值。如零钱个数，路径长度，顶点个数等

贪心算法概述 (续)

■贪心算法一般形式

```
GREEDY(C)   // C是候选对象集合
1    S ← ∅          // 在集合S中构造解
2    while C ≠ ∅ and not solution(S) do
3         x ← select(C)
4         C ← C \ {x}
5         if feasible(S ∪ {x})
6              S ← S ∪ {x}
7    if solution(S)
8         return S
9    else return "No solution"
```

贪心算法概述 (续)

■例：找零问题（作业3第4题）设数组$A[1..n]$ 中的元素表示n个零钱面值，需寻找可找开某个金额M的最少零钱数量，及相应找零方案

```
CHANGE_GREEDY(A, M)
1   S ← ∅; s ← 0   // S为解中包含的零钱集合，s为S中零钱面值之和
2   while s ≠ M do
3       x ← 使得s+x不超过M的最大零钱面值
4       if 找不到这样的面值
5           error “无法找开”
6       else S ← S ∪ 一个x面值的零钱
7           s ← s + x
8    return S
```

贪心算法得到的是否是最优解？

在正常的货币系统下是最优解，但零钱面值任意指定时不一定最优

贪心算法概述 (续)

■找零问题特性：

1. 找最少零钱数，候选对象集为面值在 $A[1..n]$中的零钱
2. 选择的零钱集合和未被选的零钱集合
3. 判断解函数，检查目前已选零钱集合中的金额是否等于要找的钱数
4. 如果集合中零钱面额不超过应找金额，则该集合是可行的
5. 选择函数，从未选零钱集合中找面值最大的零钱
6. 目标函数，计算零钱数目

贪心算法内容

- 活动选择问题
- 贪心算法原理
- 分数背包问题
- Huffman编码
- 拟阵
- 其他应用

活动选择问题

- 多个活动竞争资源的调度问题：尽可能多地选择互不冲突的活动
- 设有n个活动（activity）$S=\{a_1, a_2, \ldots, a_n\}$，均要使用某资源（如教室），该资源使用方式为独占式，一次只供一个活动使用
  - 每个活动a_i发生的时间为$[s_i, f_i)$, $0 \le s_i < f_i < \infty$
  - 两活动a_i, a_j兼容（compatible不冲突）：$[s_i, f_i)$, $[s_j, f_j)$不重叠，满足$s_i \ge f_j$ 或$s_j \ge f_i$，即：一活动的开始时间大于等于另一活动的完成时间
  - 活动选择问题：选择最多的互不冲突的活动，使兼容活动集合最大，即求解$A \subseteq S$，A中活动互不冲突且$|A|$最大

活动选择问题 (续)

■假定活动已按结束时间单调递增顺序排序

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	8	9	10	11	12	13	14

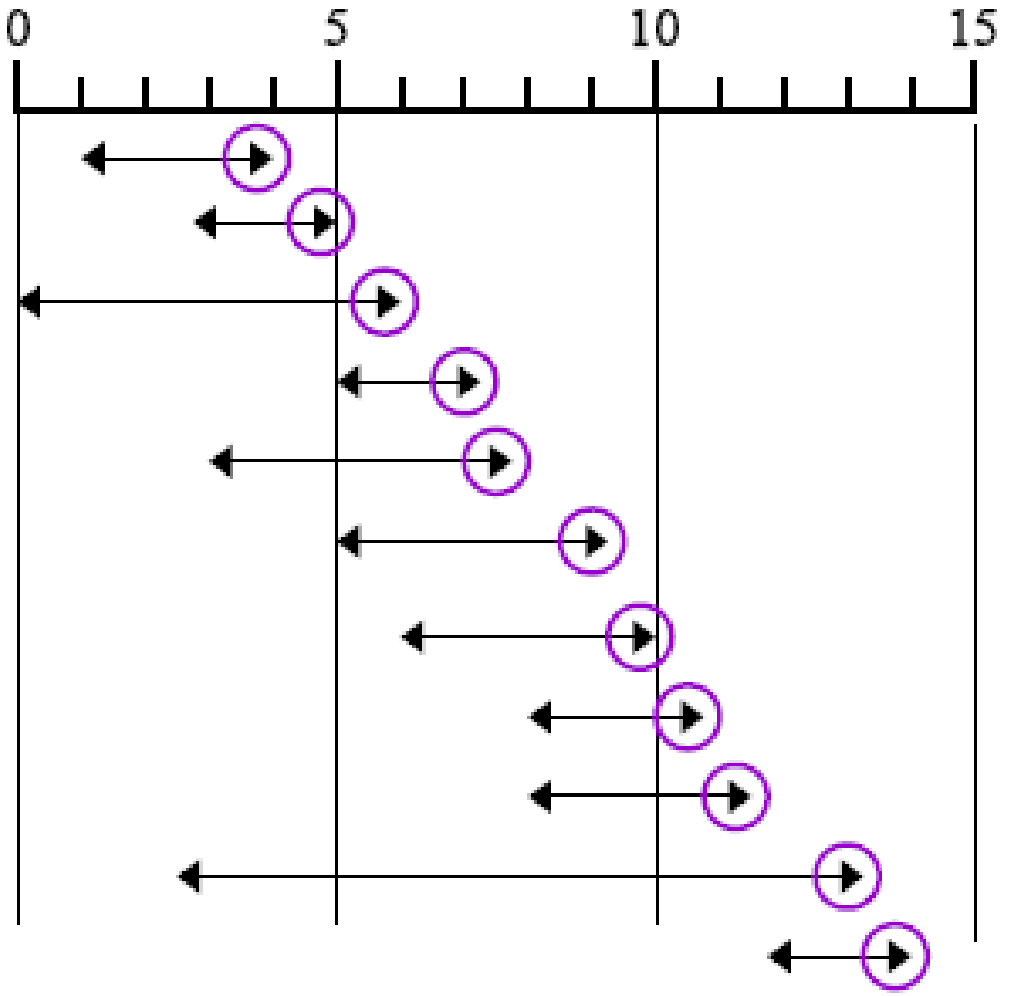
➢问题的解：$A_1=\{a_3, a_9, a_{11}\}$，$A_2=\{a_1, a_4, a_8, a_{11}\}$，$A_3=\{a_2, a_4, a_9, a_{11}\}$

➢最优解：A_2和A_3

■此问题可用迭代方法直接给出贪心算法，但为比较和动态规划关系，以下先考虑动态规划解法

活动选择问题 (续)

i	s_i	f_i
1	1	4
2	3	5
3	0	6
4	5	7
5	3	8
6	5	9
7	6	10
8	8	11
9	8	12
10	2	13
11	12	14



活动选择问题 (续)

1. 活动选择问题的最优子结构

- S_{ij}：在a_i结束之后开始且在a_j开始之前结束的活动集合
  $S_{ij} = \{a_k \in S: f_i \le s_k < f_k \le s_j\}$
- A_{ij}：S_{ij}的一个最大相互兼容的活动子集，包含活动a_k
  （A_{ij}是S_{ij}问题的最优解）
- 问题分解：
  S_{ik}中最大兼容活动子集 + S_{kj}中最大兼容活动子集
  $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$ （剪切-粘贴及反证法证明）

活动选择问题 (续)

2. 递归解

- $c[i, j]$：在a_i结束之后开始且在a_j开始之前结束的活动集合S_{ij}的一个最大相互兼容的活动子集的大小，即$c[i, j] = |A_{ij}|$
- 若$a_k \in A_{ij}$，则有$c[i, j] = c[i, k] + c[k, j] + 1$
- 总体递归式：

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

动态规划求解算法留作作业（练习16.1-1）

活动选择问题 (续)

3. 贪心算法

- 直观上，我们应该选择这样一个活动，选出它后剩下的资源应能被尽量多的其他任务所用
- 每次选择候选集中最早结束的活动
- **定理16.1** 考虑任意非空子问题S_{ij}，令a_m是S_{ij}中结束时间最早的活动，即：$f_m = \min\{f_k : a_k \in S_{ij}\}$，则
  1. a_m在S_{ij}的某个最大兼容活动子集中
  2. 子问题S_{im}的解是空集

活动选择问题 (续)

3. 贪心算法

- **定理16.1** 考虑任意非空子问题S_{ij}，令a_m是S_{ij}中结束时间最早的活动，即：$f_m = \min\{f_k : a_k \in S_{ij}\}$，则
  1. a_m在S_{ij}的某个最大兼容活动子集中
  2. 子问题S_{im}的解是空集
- 证明：（第2部分，反证法）假定S_{im}的解非空，则存在$a_k \in S_{im}$，使得$f_i \le s_k < f_k \le s_m$。由此得到$a_k \in S_{ij}$的完成时间先于$a_m$，与$a_m$是$S_{ij}$最早完成的活动矛盾

活动选择问题 (续)

3. 贪心算法

- **定理16.1** 考虑任意非空子问题S_{ij}，令a_m是S_{ij}中结束时间最早的活动，即：$f_m = \min\{f_k: a_k \in S_{ij}\}$，则
  1. a_m在S_{ij}的某个最大兼容活动子集中
  2. 子问题S_{im}的解是空集
- 证明：（第1部分）设A_{ij}是S_{ij}的某个最优解，假设A_{ij}中的活动已按完成时间单调递增排序，且a_k是A_{ij}中最早结束的活动：

  1、若$a_k = a_m$，则问题已得证，即最优解包含a_m；

  2、若$a_k \neq a_m$，构造子集$A'_{ij} = (A_{ij} - \{a_k\}) \cup \{a_m\}$，即将最优解中的$a_k$替换为$a_m$，则需证明$A'_{ij}$也是最优解

  因为$f_m \leq f_k$，因此A'_{ij}中的活动也不冲突，且$|A'_{ij}| = |A_{ij}|$

  A'_{ij}也是S_{ij}的一个最优解，包含a_m

活动选择问题 (续)

3. 贪心算法

- 动态规划求解时，原问题S_{ij}可分解为两个子问题S_{ik}和S_{kj}求解，且这种分解有$|S_{ij}|$种可能
- 定理16.1可简化问题求解过程：
  - 求S_{ij}最优解时只用到一个子问题，另一个子问题为空
  - 只需考虑一种选择，即选择S_{ij}中最早完成的活动
- 定理16.1可以自顶向下的方式解每一个子问题

活动选择问题 (续)

3. 贪心算法

- 当某个a_m加入解集合后，我们总是在剩余活动中选择第一个不与a_m冲突的活动加入解集，该活动是能够最早完成且与a_m兼容的
- 这种选择为剩余活动的调度留下了尽可能多的机会，即：留出尽可能多的时间给剩余的尚未调度的活动，以使解集合中包含的活动最多

每次选一个最早完成并与刚加入解集元素兼容的活动

活动选择问题 (续)

3. 贪心算法

➢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	8	9	10	11	12	13	14

1. 选择a_1，则下一活动开始时间须大于等于4，排除a_2, a_3
2. 选择a_4，则下一活动开始时间须大于等于7，排除a_5, a_6, a_7
3. 选择a_8，则下一活动开始时间须大于等于11，排除a_9, a_{10}
4. 选择a_{11}，则下一活动开始时间须大于等于14
5. 无可选活动，结束，解集为$\{a_1, a_4, a_8, a_{11}\}$

活动选择问题 (续)

4. 递归的贪心算法

- 输入：$s[1..n]$：活动开始时间
  $f[0..n]$：活动结束时间
  i, j：子问题S_{ij}的下标
- 输出：S_{ij}的最优解

注：定义虚拟活动a_0
结束时间为f_0
活动已按照结束时间
单调递增排序

```
RECURSIVE_ACTIVITY_SELECTOR(s, f, i, j)
1  m ← i + 1
2  while m ≤ j and s[m] < f[i] do   // 将与a_i冲突的活动去掉
3      m ← m + 1
4  if m ≤ j
5      return {a_m} ∪ RECURSIVE_ACTIVITY_SELECTOR(s, f, m, j)
6  else return ∅
```

RECURSIVE_ACTIVITY_SELECTOR$(s, f, 0, n)$的运行时间为$\Theta(n)$

活动选择问题 (续)

5. 迭代贪心算法

- RECURSIVE_ACTIVITY_SELECTOR几乎就是尾递归：以一个对自身的递归调用再接一次并集操作结尾
- 尾递归过程改为迭代形式通常很直接，某些特定语言的编译器可以自动完成这一工作

```
GREEDY_ACTIVITY_SELECTOR(s, f)
1  n ← s.length
2  A ← {a_1}
3  k ← 1
4  for m ← 2 to n do
5      if s[m] ≥ f[k]
6          A ← A ∪ {a_m}
7          k ← m
8  return A
```

时间复杂度：$\Theta(n)$
（已排序情况）

排序：$\Theta(n\lg n)$

算法正确性证明？
循环不变式及证明→定理16.1证明→算法正确

贪心算法内容

- 活动选择问题
- 贪心算法原理
- 分数背包问题
- Huffman编码
- 拟阵
- 其他应用

贪心算法原理

- 贪心算法是通过做一系列选择来获得最优解，在每个决策点，都选择在当时看来最佳的选择，这种启发式策略并不保证总能产生最优解
- 前述贪心算法的步骤
  1. 确定问题的最优子结构
  2. 给出递归解
  3. 证明在递归的每一步，有一个最优的选择是贪心的选择，因此做出这种选择是安全的
  4. 证明除了贪心选择导出的子问题外，其余子问题都是空集合
  5. 根据贪心策略写出递归算法
  6. 将递归算法转换为迭代算法

贪心算法原理 (续)

- 上述步骤是以动态规划为基础的。实际上可改进它，重点关注贪心选择
- 例：活动选择问题可改进为：
  - 首先定义子问题S_{ij}，i, j均可变
  - 如果我们总是做贪心选择，则子问题变为：$S_i=\{a_k \in S: f_i \le s_k\}$（只需考虑$S_{ij}$中$i$或$j$一端变化情况）
  - 证明一个贪心的选择（即选S_i中第一个完成的活动a_m），和剩余子问题S_m（与a_m兼容）的最优解结合，能产生S_i的最优解

贪心算法原理 (续)

- 贪心算法的一般设计步骤
  - 将优化问题分解为做出一种选择及留下一个待解的子问题
  - 证明对于原问题总是存在一个最优解会做出贪心选择，从而保证贪心选择是安全的
  - 验证当做出贪心选择之后，它和剩余的一个子问题的最优解组合在一起，构成了原问题的最优解
- 没有一般的方法说明贪心算法是否会解决一个特殊的最优问题，但是有两个要素有助于使用贪心算法：
  - 贪心选择性质
  - 最优子结构

贪心算法原理 (续)

■贪心选择性质

- 贪心选择性质：一个全局最优解能够通过局部最优（贪心）选择达到
- 贪心法总是在当前步骤上选择最优决策，然后解由此产生的子问题
- 贪心选择只依赖了目前所做的选择，但不依赖于将来的选择及子问题的解
- 自顶向下，每做一次贪心选择，就将子问题变得更小
- 贪心算法一般总存在相应的动态规划解，但贪心法的效率更高，原因：
  - 对输入做预处理
  - 使用合适的数据结构（如优先队列）

贪心算法原理 (续)

■最优子结构

- 和动态规划一样，该性质也是贪心算法的关键要素
  例：活动选择问题的动态规划解中：
  S_{ij}的最优解A_{ij}：若$a_k \in A_{ij}$，则A_{ij}包含S_{ik}和S_{kj}的最优解
- 对贪心算法更直接

贪心算法原理 (续)

- 贪心法与动态规划的比较
  - 相同点：两种方法都利用了最优子结构特征
  - 易错误处：
    - 当贪心算法满足全局最优时，可能我们试图使用动态规划求解，但前者更有效
    - 当事实上只有动态规划才能求解时，错误地使用了贪心算法
- 为了说明两种技术的细微区别，我们研究一个经典最优化问题的两个变形：
  - 0-1背包问题（动态规划章节已讲述；教材p243）
  - 分数背包问题（每个物品可以拿取部分）

贪心算法原理 (续)

■两个背包问题都具有最优子结构!

➢0-1背包问题：

原问题的最优解：重量至多为W的最有价值的物品（取自n件物品）的装包方案，每个物品只能选择取或不取

子问题：求解背包容量为j，可选物品编号为1, 2, ..., i时的0-1背包问题

子问题的最优解：从原问题的最优解中去掉某物品j，则包中剩余物品应是除j外、取自原n-1件物品中最有价值、且总重不超过W-w_j的若干物品

原问题的最优解也要求子问题的解最优

贪心算法原理 (续)

■两个背包问题都具有最优子结构!

➢分数背包问题：

原问题的最优解：重量至多为W的最有价值的物品（取自n件物品）的装包方案，每个物品可部分拿取

子问题：求解背包容量为j，可选物品编号为$1, 2, ..., i$时的分数背包问题

子问题的最优解：从背包中去掉包含物品j（可能是物品j的一部分），设其重量为w，则包中剩余物品应是除该部分之外、取自原n-1件物品以及物品j的w_j-w部分中最有价值、且总重至多为W-w的若干物品

原问题的最优解也要求子问题的解最优

贪心算法原理 (续)

- 求解0-1背包问题最优解只能使用动态规划
- 求解分数背包问题最优解可使用贪心算法：
  - 将物品按照单位重量价值（v_i/w_i）排序
  - 每次取单位重量价值最大物品装包，直至背包装满或所有物品都考虑过为止

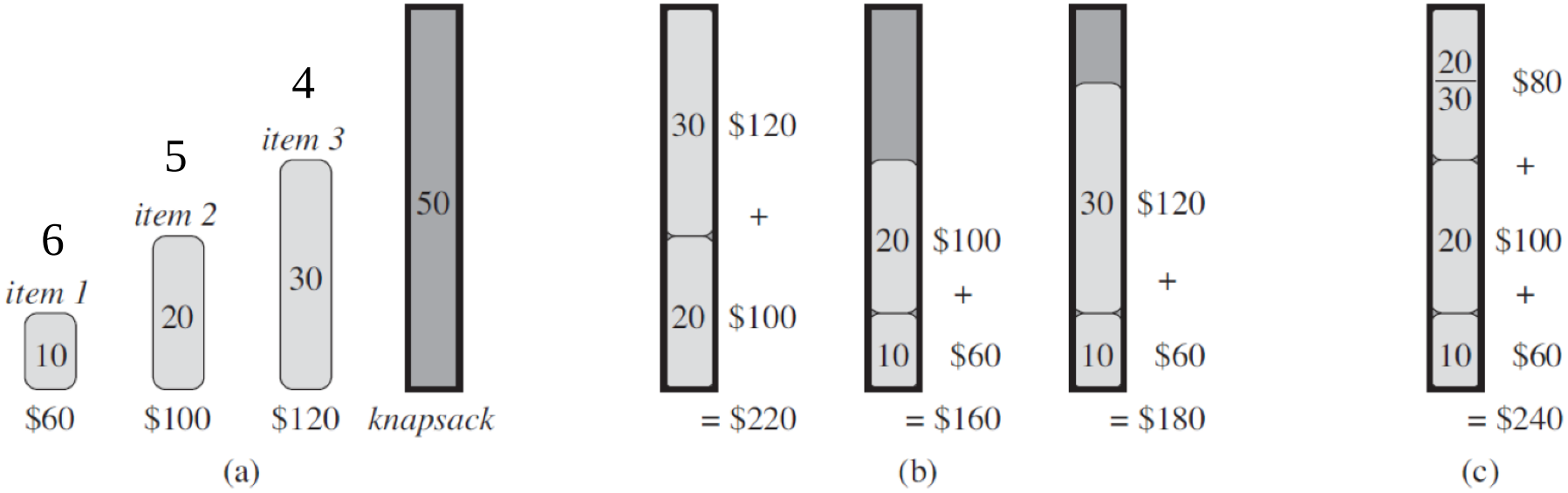
该朴素贪心算法若用于0-1背包问题，所得的解任意差！

例：

	物品1	物品2
重量w_i	1	100
价值v_i	2	199
单位重量价值v_i/w_i	2	1.99
背包容量W	100	

只装入物品1？

贪心算法原理 (续)


6
item 1
10
$60
5
item 2
20
$100
4
item 3
30
$120
50
knapsack
(a)
30 $120
+
20 $100
= $220
20 $100
+
10 $60
= $160
30 $120
+
10 $60
= $180
(b)
20/30 $80
+
20 $100
+
10 $60
= $240
(c)

贪心算法内容

- 活动选择问题
- 贪心算法原理
- 分数背包问题
- Huffman编码
- 拟阵
- 其他应用

贪心算法理论

- 该理论可用来确定贪心算法生成最优解的情形
- 用到了一种组合结构：Matroid（拟阵）
  - 该结构是Whitney于1935年在研究矩阵中线性无关结构时抽象出来的，由Korte于80年代初将该理论发展为贪心算法理论
- 该理论覆盖了许多贪心算法的应用实例（Kruskal、匈牙利算法等），但并未覆盖所有情况（如活动选择问题、Huffman编码等），仍在发展中

贪心算法理论 (续)

- 一个拟阵是满足下列条件的有序对$M = (S, \mathcal{I})$：
  - S是一个有限集
  - $\mathcal{I}$是S的子集的一个非空族，这些子集称为S的独立子集，使得若$B \in \mathcal{I}$且$A \subseteq B$，则$A \in \mathcal{I}$。若$\mathcal{I}$满足此性质，则称之为遗传的，注意空集必然是$\mathcal{I}$的成员，即$\emptyset \in \mathcal{I}$
  - 若$A \in \mathcal{I}$、$B \in \mathcal{I}$且$|A| < |B|$，那么存在某个元素$x \in B - A$，使得$A \cup \{x\} \in \mathcal{I}$，则称$M$满足交换性质

贪心算法理论 (续)

■拟阵 $M = (S, \mathcal{I})$：

- S有穷非空
- 集合族$\mathcal{I}$满足遗传性，即某子集是独立子集，则该独立子集的子集也是独立子集
- M满足交换性质，即可扩展

贪心算法理论 (续)

■例：$M = (S, \mathcal{I})$

➢矩阵拟阵：S的元素是矩阵的行，若行的子集线性无关则该子集独立

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 6 & 8 & 10 \\ 1 & 3 & 5 & 7 & 9 \end{bmatrix}$$

$$S = \{[1,2,3,4,5], [2,4,6,8,10], [1,3,5,7,9]\}$$

$$\mathcal{I} = \{\emptyset, \{[1,2,3,4,5]\}, \{[2,4,6,8,10]\}, \{[1,3,5,7,9]\}, \{[1,2,3,4,5], [1,3,5,7,9]\}, \{[2,4,6,8,10], [1,3,5,7,9]\}\}$$

贪心算法理论 (续)

■例：$M = (S, \mathcal{I})$

➢无向图$G = (V, E)$的图拟阵：$M_G = (S_G, \mathcal{I}_G)$

- S_G是G的边集，即$S_G = E$
- 设A是E的子集，$A \in \mathcal{I}_G$ 当且仅当 A无回路
  即：边集A独立 当且仅当 子图$G_A = (V, A)$形成森林

M_G与最小生成树问题MST紧密相关