Chapter 16 Greedy Algorithm

Dr. He Emil Huang
School of Computer Science and Technology
Soochow University

E-mail: huangh@suda.edu.cn

Greedy Algorithms

Main Topics in this chapter:

- Activity-selection problem
- Elements of the greedy strategy
- Fractional Knapsack
- Huffman coding
- Matroid

Instances of Greedy Algorithms

- Scheduling
  - Activity Selection 活动选择 (Chap 16.1)
  - Minimizing time in system
  - Deadline scheduling 任务调度 (Chap 16.5)
- Graph Algorithms
  - Minimum Spanning Trees (Chap 23)
  - Dijkstra's (shortest path) Algorithm (Chap 24.3)
- Other Heuristics *(what's the meaning of "Heuristics")*
  - Huffman coding (Chap 16.3, can get optimal solution)
  - Coloring a graph
  - Traveling Salesman Problem (Chap 35.2)
  - Set-covering (Chap 35.3)
  - Subset-sum problem (Chap 35.5)

Instances of Greedy Algorithms

- Heuristics algorithm definition
  - A heuristic algorithm is one that is designed to solve a problem in a faster and more efficient fashion than traditional methods by sacrificing optimality, accuracy, precision, or completeness for speed. Heuristic algorithms often times used to solve NP-complete problems, a class of decision problems.

Greedy Method

- For many optimization problem, Dynamic Programming (DP) is overkill. A greedy algorithm always make the choice that looks best at every step. That is, it makes local optimal solution in the hope that this choice will lead to a ***globally optimal one***. (See the context of the first paragraph on page 237)
  - I make the shortest path to the target at each step. Sometime I win, sometime I lose.

Change-Making Problem

Given unlimited amounts of coins of denominations $d_1 > \ldots > d_m$,

give change for amount n with the least number of coins

Example: $d_1 = 25c$, $d_2 = 10c$, $d_3 = 5c$, $d_4 = 1c$ and $n = 48c$

DP solution: https://www.youtube.com/watch?v=jgiZlGzXMBw

Greedy solution:

Greedy solution is

- Optimal for any amount and "normal" set of denominations
- May not be optimal for arbitrary coin denominations

人们一般都采取贪心算法找钱，*i.e.*, 每一步都选出满足要求的最大面值的货币

Description of the Change-Making Alg.:
找零钱算法的形式化表述：

```
Function change (n) //用尽可能少的硬币找出n个单位的零钱
  const C={100,25,10,5,1}
  S=Φ;  s=0  //S为包含解的硬币集合，s为S中硬币面值之和
  while s<n do
    x=(max in C and s+x<=n) //检查候选对象x
    if there is no such item then
       return “no solution found”
    S=S∪{a coin of x}
    s=s+x
  return S
```

Greedy Technique

Constructs a solution to an *optimization problem* piece by piece through a sequence of choices that are:

- *Feasible*
- *Locally optimal*
- *Irrevocable*

For some problems, yields an optimal solution for every instances.

For most, does not but can be useful for fast approximations.

8

General Characteristics of Greedy Alg.

一般，贪婪算法可解的问题有如下特性：

（1）优化问题，有一个候选对象的集合，如硬币，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）有一个函数(solution function)，该函数用于检查候选对象集合是否是(or 提供了)问题的解，该函数不考虑此时的解决方法是否最优;

（4）还有一个函数，检查候选对象(candidate)是否可加入到当前解的对象集合中(*i.e.*, feasible可行的)，同样这一步骤不考虑解决方法的最优性;

（5）选择函数(selection function)，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(object function)，给出解的值。如硬币个数，路径长度，顶点个数等。


```
function  greedy(C : set) : set
  {C是候选对象集合}
  S=Φ {在集合S中构造解}
  while C<>Φ and not solution(S) do
    x=select(C)
    C=C\{x}
    if feasible(S∪{x}) then S=S∪{x}
   if solution(S) then return S
   else return “No solution”
```


回到找钱问题上来，看看其一般特性：

（1）找最少硬币数，候选对象集 {100,25,10,5,1}

（2）选到的硬币集和未被选的硬币集。

（3）判断解函数（solution）,检查目前已选的硬币集中的金额是否等于要找的钱数。

（4）如果集合中硬币钱数不超过应找金额，则该集合是可行的。

（5）选择函数（selection）,从未选硬币集合中找一个面值最大的硬币。

（6）目标函数（object）：计算硬币数目。

Applications of the Greedy Strategy

- **Optimal solutions:**
  - Change-making for "normal" coin denominations
  - Minimum spanning tree (MST)
  - Single-source shortest paths (i.e. Dijkstra)
  - Simple scheduling problems
  - Huffman codes (Huffman Tree Construction)
- **Approximations:**
  - Traveling salesman problem (TSP)
  - Knapsack problem
  - Other combinatorial optimization problems

13

Ch16.贪心算法 (课本P237)

- 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之。
- 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，但有时是(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，若从v_i搜索到邻点v_k最短，未必是v_i到v_j最短。
- 但是，仍有许多问题贪心法将产生全局最优解，如*MST*，*单源最短路径*等。

14

16.1 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,…,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相容 (compatible不冲突)是指：$[s_i, f_i)$，$[s_j, f_j)$不重叠，满足$s_i \ge f_j$ or $s_j \ge f_i$。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，使相容活动集合最大。即：A⊆S，A中活动互不冲突且|A|最大。

15

16.1 活动选择问题

- 例: (*按完成时间递增序排列)

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}}

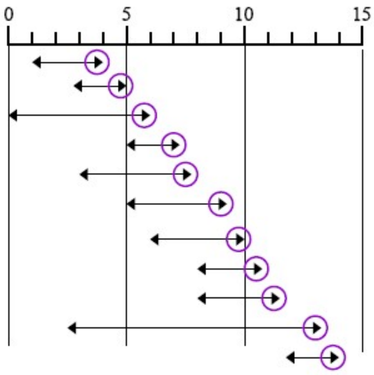
最优解： A_2和A_3

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之关系，下面先考虑动态规划解法。

16

Activity Selection

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



16.1 活动选择问题

1、活动选择问题的最优子结构

- 子问题空间可描述为：$S_{ij} = \{a_k \in S : f_i \le s_k < f_k \le s_j\}$

  S_{ij}是S的子集，它包含那些(在活动a_i完成之后开始)and(可在活动a_j开始之前完成)的活动a_k。

  亦即：S_{ij}中所有活动a_k与活动a_i、a_j相容，不妨称a_i、a_j和子集S_{ij}相容，因此a_k也与所有不迟于f_i完成的活动以及与所有不早于s_j开始的活动相容。

  Note：S_{ij}中可能有冲突的活动。

  为方便处理，设有两个虚拟的活动a_0和a_{n+1}，并且假定 $f_0 = 0, s_{n+1} = \infty$

  因此，$S = S_{0,n+1}, 0 \le i, j \le n+1$

18

16.1 活动选择问题

为了更严格定义i和j的范围，假定所有活动已**按完成时间的单调增有序**进行排序：

$$f_0 < f_1 \le f_2 \le \ldots \le f_n < f_{n+1} \qquad 16.1式$$

$\because S_{ij}=\phi$, 若$i \ge j$ //易于用反证法证明

$\therefore$ 只考虑S_{ij}, 当$i<j$。因此i, j的范围是: $0 \le i < j \le n+1$

16.1 活动选择问题

❖如何分解问题？

设子问题 $S_{ij} \ne \phi, a_k \in S_{ij}, f_i \le s_k < f_k \le s_j$ 。若S_{ij}的解中选择了a_k，使用a_k产生S_{ij}的两个子问题：

S_{ik}: 包括在a_i完成之后开始，在a_k开始前完成的那些活动

S_{kj}: 包括在a_k完成之后开始，在a_j开始前完成的那些活动

均是S_{ij}的子集，这两子集与a_k相容

S_{ij}的解显然是S_{ik}和S_{kj}解的并，再加上a_k。

■**注意S_{ik}和S_{kj}已去掉了那些与a_k冲突的活动，而这些活动原来可能在S_{ij}中。**

16.1 活动选择问题

❖最优子结构

设S_{ij}的最优解是A_{ij}，$a_k \in A_{ij}$，

则两子问题S_{ik}和S_{kj}的解A_{ik}和A_{kj}也必须是最优的

易用反证法证明：

因为A_{ik}和A_{kj}是独立的，可用$cut-and-paste$证明

❖用子问题的最优解构造原问题的最优解

$$A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj} \qquad (16.2式)$$

整个问题的最优解是$S_{0,n+1}$的一个最优解

16.1 活动选择问题

2、一个递归解

设**C[i,j]**表示S_{ij}的**最优解的值**，即S_{ij}中**相容活动最大子集的size**：$C[i,j]=|A_{ij}|$

1) 当$S_{ij}=\phi$时，$C[i,j]=0$，此时$i \ge j$ // $\because A_{ij} \subseteq S_{ij}, \therefore A_{ij}=\phi$

2) 当$S_{ij} \ne \Phi$时，假设 $a_k \in A_{ij}$ 则可用 S_{ik}和 S_{kj}两子问题的最优解来构造 S_{ij}的最优解，由16.2式：

$C[i,j]=C[i,k]+C[k,j]+1, i+1 \le k \le j-1$, // k有$j-i-1$选择

$|A_{ij}|=|A_{ik}|+|\{a_k\}|+|A_{kj}|=|A_{ik}|+|A_{kj}|+1$

$$C[i,j]=\begin{cases} 0 & 当S_{ij}=\Phi \\ \max\limits_{i<k<j}\{C[i,k]+C[k,j]+1\} & 当S_{ij}\ne\Phi \end{cases}$$

16.1 活动选择问题

■作业：Ex16.1-1

给出Activity Selection的DP算法

16.1 活动选择问题

3、动态规划到贪心法的转化

到此可直接写出动态规划解，留作练习

可通过下面的定理简化其解，该定理也说明**贪心算法的正确性(*i.e.*, 贪心选择的最优性)**。

Th16.1 设S_{ij}是任一非空子问题，a_m是S_{ij}中具有**最早**完成时间的活动：$f_m=\min\{f_k: a_k \in S_{ij}\}$，则：

①活动a_m必定被包含在S_{ij}的**某个**最优解中；

②子问题S_{im}是空集，使S_{mj}是唯一可能的非空集 //选a_m的目的

pf：先证第2部分：

假定S_{im}非空，则$\exists a_k \in S_{im}$，使$f_i \le s_k < f_k \le s_m$

$\because a_k$的完成时间先于a_m且$a_k \in S_{ij}$

这与a_m是S_{ij}的最早完成活动矛盾！

再证第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增排过序，a_k是其中第一个活动。

25

若$a_k = a_m$，则问题已得证，即最优解包含a_m

若$a_k \ne a_m$，则构造子集$A_{ij}' = A_{ij} - \{a_k\} \cup \{a_m\}$，

只需证A_{ij}'也是最优解即可

$\because a_k \in A_{ij} \subseteq S_{ij}$　　A_{ij}是S_{ij}的某个最优解

$\therefore f_m \le f_k$

又$\because a_k$是A_{ij}中最早完成的任务，a_m比a_k更早完成

$\therefore A_{ij}'$中的活动互不冲突，也即A_{ij}'是S_{ij}的一个解

$\because |A_{ij}'| = |A_{ij}|$

$\therefore A_{ij}'$亦是S_{ij}的一个最优解，它包含a_m

26

16.1 活动选择问题

- 该定理的价值可简化问题的解
  - 在动态规划求解时，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$j-i-1$中可能
  - 定理16.1将其简化为：
    - ①求S_{ij}的最优解时只用到一个子问题，另一个子问题为空；
    - ②只须考虑一种选择：即选S_{ij}中最早完成的活动
  - 该定理带来的另一个好处是：
    - 能以自顶向下的方式解每一个子问题：

27

16.1 活动选择问题

①对原问题$S=S_{0,n+1}$，选其中最早完成的活动a_{m1}

∵S中的活动已按完成时间单调增有序

∴m1=1。下一子问题应是$S_{m1,\ n+1}$

//已去掉与a_{m1}冲突的活动

②选$S_{m1,n+1}$中最早完成的活动a_{m2}

$\because$须将S中与a_{m1}冲突的活动删除才能得到$S_{m1,n+1}$

$\therefore m2$未必为2

③一般形式，当选择a_{mi}加到解集中之后，需解的子问题变为$S_{mi,n+1}$

显然，所选活动是按完成时间单调增有序的(m1<m2<…<mi<…)

28

16.1 活动选择问题

- 贪心选择
- 当某个a_{mi}加入解集合后，我们总是在剩余的活动中选择第一个不与a_{mi}冲突的活动加入解集合，该活动是能够最早完成且与a_{mi}相容的。
- 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相容的活动

29

16.1 活动选择问题

4、递归的贪心算法

- 输入：向量s和f，并假定已按f单调增有序
  子问题S_{ij}的下标
- 输出：S_{ij}的最优解
- 算法：

30

$Re\,cursiveActivitySelector(s, f, i, j)\{$

//求S_{ij}的最优解，a_i刚加入解集合. 原问题的解： $i=0, j=n+1$

$m \leftarrow i+1;$ //当$i=0$时，有a_1必加入解集

$while\ (m<j) and (s_m<f_i)\ do$

//将与a_i冲突的活动去掉，才能得到S_{ij}

$m \leftarrow m+1;$ //在$a_{i+1}, a_{i+2}, ..., a_{j-1}$中找第1个与$a_i$相容的活动$a_m$

$if\ (m<j) then$ //$s_m \ge f_i, a_m$是S_{ij}中的第1个活动，

//即在a_i完成后开始且最早能完成的活动

$return\quad \{a_m\} \cup Re\,cursiveActivitySelector(s, f, m, j);$

//a_m和S_{mj}最优解并

$else$

$return\quad \Phi;$ //a_j开始前能够完成的活动均与a_i冲突

$\}$

31

16.1 活动选择问题

- 时间
  - 若要排序，则时间为 $\Theta(n\lg n)$
  - 若已排序，则RecursiveActivitySelector(s,f,0,n+1)的时间为 $\Theta(n)$

    在所有的调用里，每个活动被检查一次，请注意每次循环检查时，活动序号只增加不减少，从RAS(s,f,i,j)调用到RAS(s,f,m,j)时，必有i<m。j始终为n+1

32

16.1 活动选择问题

5、迭代的贪心算法

上述递归很接近于尾递归，只是结束前多了一个Union操作，很易转换为迭代算法：

①j始终不变，j=n+1

②当一个活动a_i加入解集合之后，我们只要从a_i依次往后找到第一个不与a_i冲突的活动a_m，由于活动事先按完成时间单调增有序，故a_m是最早完成且与a_i相容的活动，它也是S_{ij}中的第一个活动

33

$GreedyActivitySelector(s, f)\{$ //f单调增有序

$n \leftarrow length[s];$

$A \leftarrow \{a_1\};$ //A为解集合

$i \leftarrow 1;$ //i是最近加入解集合A的活动a_i的下标

$for\ m \leftarrow 2\quad to\quad n\quad do$ //找$S_{i,n+1}$中最早完成的活动a_m

$if (f_i \le s_m) then\{$ //$i<m, a_m$与a_i相容

$A \leftarrow A \cup \{a_m\};$ //a_m是与a_i相容且完成时间

//最早的活动

$i \leftarrow m;$ //i仍记录最近加入A的活动a_i的下标

$\}$//endif

$return\quad A;$

$\}$

34

16.1 活动选择问题

注意，若直接给出迭代算法，一般要证A确实为最优解。这一点由递归算法得以保证，亦可用归纳法直接证明：

①总是存在一个最优解，第一步是贪心的选择，即选择活动a_1；

②在已做的选择数目上做归纳法证明

35

16.1 活动选择问题

- 算法要点
  - i始终表示最近加入A的活动的下标

$\because$ 活动已按完成时间有序

$\therefore f_i$总是当前A中所有活动的最大完成时间：

$$f_i = \max\{f_k : a_k \in A\}$$

当a_m加入A时，$\because s_m \ge f_i, \therefore s_m$也大于$A$中所有活动的完成时间，即$a_m$与$A$中所有活动相容.

- 时间：O(n)　　//已排序

36

活动选择问题

- 作业：Ex16.1-3

16.2贪心策略要点

- 贪心算法是通过做一系列选择来获得最优解，在算法里的每一个决策点上，都力图选择最好的方案，这种启发式策略并非总能产生最优解。
- 上节介绍的贪心算法的步骤
  - 确定问题的最优子结构
  - 给出递归解(指递归方程)
  - 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。
  - 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
  - 根据贪心策略写出递归算法
  - 将递归算法转换为迭代算法

16.2 贪心策略要点

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。例如，活动选择问题可改进为(直接写出递归解)：

- 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 $S_{ij} \Rightarrow S_{i,n+1}$

  ∵n+1不变，可省略，子问题变为：

  $S_i = \{a_k \in S : f_i \le s_k\} // S_i$表示$a_i$完成后开始的任务集
- 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相容)的最优解结合，能产生S_i的最优解。

16.2 贪心策略要点

- 贪心算法的一般设计步骤

  ① 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

  ② 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

  ③ 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
- 没有一个一般的方法告诉我们一个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

16.2 贪心策略要点

1、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：

  ①对输入做预处理

  ②使用合适的数据结构(如优先队列)

16.2 贪心策略要点

2、最优子结构

- 和动态规划一样，该性质也是贪心法的一个关键要素

  例如，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

  原问题的最优解 ⇒ 贪心选择 + 子问题的最优解

  通常可使用归纳法证明在每一步上做贪心选择可产生最优解，由此可导出最优子结构

16.2 贪心策略要点

3、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：
  - ① 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
  - ② 当事实上只有动态规划才能求解时，错误地使用了贪心法

为了说明两种技术的细微区别，请看一个古典优化问题的两个变种：

背包问题：n个物品重$w_1,w_2,\dots,w_n$,背包可放入重量W，问能否从n件物品中选择若干件放入背包，使重量和正好等于W。

43

16.2 贪心策略要点

- 0-1背包问题和分数背包问题

物品件数：n；第i件物品价值$\$v_i$，重量$w_i$磅，

v_i和w_i为整数，背包载重量：W磅（整数）

怎样选物品装包使得包内含有价值最大，且总重量$\le W$

不允许一物品多次装包，可理解为n个物品互不相同

- 0-1背包：某物品拿与不拿(1,0的选择)
- 分数背包：某物品可取部分装包

想象：小偷偷东西，包容量有限，想尽可能使偷走的东西总价值最大，0-1背包偷走的是金条之类物品，分数背包偷走的是金粉之类物品。

44

16.2 贪心策略要点

- 两个背包问题都具有最优子结构!!
  - 0-1背包问题 (0-1 knapsack problem)

原问题的最优解：设背包中装有重量至多为W磅的最有价值的物品(取自n件物品)；

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。

显然，原问题分解为子问题时，原问题的最优解也要求子问题的解最优

45

16.2 贪心策略要点

- 分数背包 (fractional knapsack problem)

子问题：从背包中去掉物品j (Note：part of)，重量为w(该物品剩余w_j-w)，则包中剩余物品应是除j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

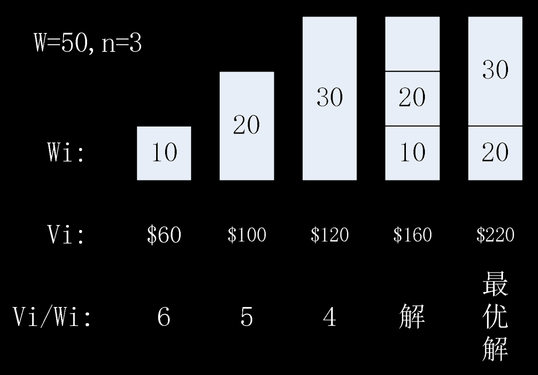
显然，原问题解最优亦要求子问题的解最优

46

16.2 贪心策略要点

- 两个背包问题的不同解法
  - 求解0-1背包最优解只能用动态规划求解
    - 按每磅价值 (v_i / w_i) 排序
    - 贪心选择策略：取单位价值最大者装包，若装不下，考虑下一单位价值最大的物品，直至包装满或所有物品都考虑过为止
    - 实际上，装入当前每磅价值最大者只能保证当前最优(局部最优)，然而放弃它可能使得后续选择更优。所以在装包前，应将某物品装包的子问题的解和放弃它的子问题的解进行比较，这将导致许多重叠子问题，这正是动态规划的特点。

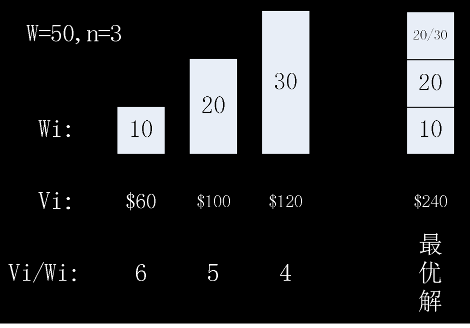
47



48

16.2 贪心策略要点

❖分数背包：可用贪心算法求出最优解



49

16.2 贪心策略要点

■三种贪心选择函数

1. 每一步选择剩余物品中价值最大的，尽可能增加总价值；
2. 每一步选择剩余物品中重量最轻的，尽可能减少总重量；
3. 单位重量价值最高

n=5, w=100

w	10	20	30	40	50
v	20	30	66	40	60
v/w	2.0	1.5	2.2	1.0	1.2

50

16.2 贪心策略要点

n=5, w=100

w	10	20	30	40	50
v	20	30	66	40	60
v/w	2.0	1.5	2.2	1.0	1.2

选择						value
最大v_i	0	0	1	0.5	1	146
最小w_i	1	1	1	1	0	156
最大v_i/w_i	1	1	1	0	0.8	164

51

16.3 Huffman编码

■略

上机题：背包问题

52

16.4 贪心算法的理论基础

■ 该理论可用来确定贪心算法何时能产生最优解，它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)，但它仍在发展。

53

16.4.1 拟阵

■ Def：一个拟阵是满足下列条件的有序对 M = (S, I)

① S是一有穷非空集

② I是S的一个非空子集(称为S的独立子集)簇，使得若B∈I且A⊆B，则A∈I (集合A,B均独立)。满足此性质的I是遗传的，即B独立(指B∈I)，则B的子集也独立。注意Φ∈I。

③ 若A∈I，B∈I且|A|<|B|，则存在某个元素x ∈B-A使得A∪{x} ∈ I。称M满足交换性质。

如何证明有序对具有拟阵性质？

54

16.4.1 拟阵

1. S满足有穷非空
2. 集合簇I满足遗传性 (某子集独立，则该独立子集的子集亦独立)
3. 拟阵M满足交换性(即可扩展)

55

16.4.1拟阵

- 例子
  - 惠特尼35年研究矩阵拟阵时引入。一个矩阵拟阵是指：S的元素是矩阵的行，若行的子集线性无关则该子集独立，易证这种结构是拟阵。
  - 图的拟阵

$M_G=(S_G,I_G)$是在无向图$G=(V,E)$基础上定义的：

• $S_G=E$

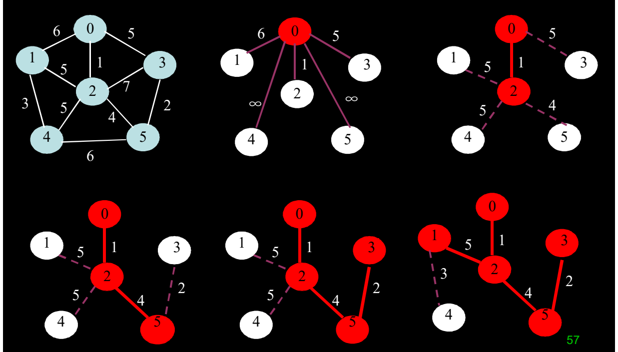
• 若$A\subseteq E$，则：$A\in I_G \Leftrightarrow A$无回路。

即：边集A独立 $\Leftrightarrow$ 子图$G_A=(V,A)$是森林

M_G与MST问题紧密相关

56

- **MST求解**



57

16.4.1拟阵

- **Th16.5 若G是无向图，则$M_G=(S_G, I_G)$是一个拟阵**

pf:

1. $S_G=E$是一有穷（非空）集
2. 证I_G是遗传的

   若$B\in I_G$，且$A\subseteq B$，意味着A是森林B的子集，它仍为森林不可能有回路，所以$A\in I_G$
3. M_G的交换性

   设$G_A=(V,A)$和$G_B=(V,B)$为森林，且$|B|>|A|$，B包含的边多于A. 现要证 $\exists e\in B-A$ 使e扩充到A后仍不产生回路

58

可利用ThB.2：具有k条边的森林包含$|V|-k$棵树

由该定理可知，G_A有$|V|-|A|$棵树，G_B有$|V|-|B|$棵树

$\because G_B$中树的数目$<G_A$中的树的数目，因为$|B|>|A|$

$\therefore G_B$中必存在某棵树T，它的顶点属于森林G_A中两棵不同的树（否则若G_B中任何树的顶点必在G_A的同一棵树中，这与G_A中树数目大于G_B矛盾）

$\because T$是连通的，必存在一条边$e=(u,v)\in T$，使得u,v在G_A的两不同树中

$\therefore$将e加入G_A后不会产生回路，即$A\cup\{e\}\in I_G$，满足交换性。

综合1,2,3知M_G是一个拟阵

59

16.4.1拟阵

- **最大独立集**
  - 独立子集的扩张

    在拟阵M=(S,I)中，若A∈I，x∉A，A∪{x}∈I，则元素x称为A的一个扩张，即元素x扩充到独立子集A后仍保持独立性

    例：在图的拟阵M_G中，若A是一个独立子集，则e是A的扩张是指加入e后仍不产生环
  - 拟阵的最大独立子集

    若A是拟阵M的独立子集，且无法进行任何扩张，则A称为M的最大独立子集，即在M中没有更大的独立子集能包含A

60

16.4.1拟阵

- Th16.6 拟阵中所有最大独立子集的大小(势)相同

pf :[反证法]
设A是M的最大独立子集，且存在另一更大的最大独立子集B。
由交换性知：$\exists x \in B - A$，使$A \cup \{x\} \in I$，
即A是可扩张的，与A是最大独立子集矛盾。

例：在M_G中，每个最大独立子集是一棵生成树，有|v|-1条边

61

16.4.1拟阵

- 加权拟阵

若对∀x∈S，为x指派一个正的权值w(x)，则称M=(S,I)是加权拟阵。S的子集(独立子集)的权可定义为：

$$w(A) = \sum_{x \in A} w(x) \quad \text{for any } A \subseteq S$$

例：M_G中，可定义w(e)为边e的长度(权重),w(A)为A中所有边的长度(权重)之和

62

16.4.2 加权拟阵上的贪心算法

- 加权拟阵的最大独立子集可描述某些问题的最优解(加权拟阵的最优子集)

可用贪心算法求出最优解的许多问题(但不是全部)可形式化为一加权拟阵中找到一个权值最大的独立子集。即：给定加权拟阵M=(S, I)，希望找到I的一个独立子集A，使w(A)最大。

- 拟阵的最优子集：拟阵中权值最大的独立子集

  它也一定是拟阵中的一个最大独立子集(子集体积最大)，反之不然。

Pf：∵∀x∈S，w(x)>0，∴若权值最大的独立子集A不是最大独立子集，可将其扩张至最大独立子集，后者的权更大，使A的权非最大

63

16.4.2 加权拟阵上的贪心算法

- 例 求连通无向图$G=(V,E)$的MST问题$\Rightarrow$加权拟阵中求权最大的独立子集问题。即：求拟阵的最优子集

  设G的权函数w定义为边的长度，且$w(e)>0$

  加权拟阵M_G的权w定义为：$w'(e)=w_0-w(e)$，w_0大于各边的最大长度。在M_G中，$\forall e \in E$，$w'(e)>0$,故M_G的每个最优子集A是G的一棵MST

  pf:

  $\because A$是拟阵的最大独立子集

  $\therefore$它是一棵生成树，它的权为$w'(A)=(|V|-1)w_0-w(A)$

  $\because A$是权最大的独立子集

  $\therefore w'(A)$最大必有$w(A)$最小，即A是G的一棵MST

64

16.4.2 加权拟阵上的贪心算法

- 加权拟阵的贪心算法
  - 适用于任何加权拟阵求最优子集A
  - 贪心之处：尽可能选权值最大的元素扩充到A

65

$Greedy(M,w)\{$
$//$输入拟阵$M=(S,I)$，w表示正的权函数
  $A \leftarrow \Phi;$
  按权w单调递减对$S[M]$排序
  for 每个$x \in S[M]$，按$w(x)$单调递减 do
    $if\ (A \cup \{x\} \in I[M])\ then$ //独立性检测
      $A \leftarrow A \cup \{x\}$; //扩充$x$未破坏$A$的独立性
    //否则放弃x
  $return \quad A;$
$\}$

66

16.4.2 加权拟阵上的贪心算法

时间分析：

设$|S|=n$，排序$O(n\lg n)$

for循环n次，

每次检验$A\cup\{x\}$是否独立，

设检查时间为$O(f(n))$,

总时间为$O(n\lg n+nf(n))$

67

16.4.2 加权拟阵上的贪心算法

- Greedy算法返回1个最优子集

A是独立子集易从Φ独立开始用归纳法证明

- Lemma16.7 (拟阵呈现贪心选择性质)

设M=(S,I)是加权拟阵，权函数为w，且S已按权值的单调递减有序，设x是S的第一个使{x}独立的元素(若这样的x存在)。若x存在，则存在S的一个最优子集A包含x。(即说明第一步贪心选择正确)

Pf：1) 若无这样的x存在，则唯一的独立子集是Φ

2) 否则设B为S的任一非空最优子集，若x \in B, 则令A=B，证毕

68

16.4.2 加权拟阵上的贪心算法

若$x\notin B$，则可从B构造一个最优子集A

使其包含x，为此需证$w(A)\ge w(B)$

$i)$ 先证$\forall y\in B$，有$w(x)\ge w(y)$

$\because y\in B \quad \therefore \{y\}\subseteq B$

由于B是S的最优子集，故B也是独立子集，

由I的遗传性知$\{y\}$亦是独立子集

$\because \{x\}$是S的第一个独立子集，

而S中元素已按单调减有序，

$\therefore w(x)\ge w(y)$

69

$ii)$构造集A，使其包含x且A最优

开始令$A=\{x\}$,显然A是独立，

利用交换性质，重复地在B中找新的元素

扩充到A使A独立，直至$|A|=|B|$。于是有：

$A=B-\{y\}\cup\{x\}$ 对某个y成立

由$i)$立即知道：

$w(A)=w(B)-w(y)+w(x)\ge w(B)$

由B是最优子集即可知A亦是最优子集，

且它包括x。

70

16.4.2 加权拟阵上的贪心算法

下面的引理和推论说明：若一元素开始未被选中，则此后亦不可能被选中

- Lemma16.8 设M=(S,I)是任一拟阵，若S的一个元素x是S的某个独立子集A的一个扩张，则x亦是Φ的一个扩张

pf:

$\because x$是A的扩张

$\therefore A\cup\{x\}\in I$

由I的遗传性知,$\{x\}$是独立的，它是ϕ的一个扩张

71

16.4.2 加权拟阵上的贪心算法

- 推论16.9 设M=(S,I)是任一拟阵，若S的一个元素x不是Φ的扩张，则x也不是S的任何独立子集A的一个扩张

pf：由引理16.8易证

- 该推论告诉我们，若一开始某元素没被选中，此后亦不会选中，保证Greedy算法开始的正确性

72

16.4.2 加权拟阵上的贪心算法

- 引理16.10 (拟阵具有最优子结构性质)

对于加权拟阵M=(S,I)，设x是Greedy算法选中的第一个元素，找一个包含x的权最大的独立子集的剩余问题可归结为找加权拟阵M'=(S',I')的一个权值最大的独立子集，这里：

$S' = \{y \in S : \{x, y\} \in I\}$

$//$即S'由S中可扩张至$\{x\}$中的元素构成

$I' = \{B \subseteq S - \{x\} : B \cup \{x\} \in I\}$

$//$即I'是由S的不包含x的独立子集构成

且M'的权函数是M的权函数，但只限于S'。我们称M'是由元素x引起的M的收缩，它是M的子问题。

73

$pf:$

若A是M的任一包含x的最优子集（即权最大的独立子集），则由I的定义可知：

$A' = A - \{x\}$是M'的一个独立子集，

反之，M'的任一独立子集A'产生M的一个独立子集：$A = A' \cup \{x\}$

$\because$两种情况下，均有$w(A) = w(A') + w(x)$

$\therefore$包含x的A是M中权最大的独立子集，保证了$w(A')$须最大，即A'是M'的最优子集，反之亦然

即：A是原问题M的最优解要求A'是子问题M'的最优解；反之，$\{x\} \cup A'$构成原问题的最优解A

74

16.4.2 加权拟阵上的贪心算法

- Th16.11 (拟阵的贪心算法的正确性)

若 M=(S,I) 是 权 函 数 为 w 的 加 权 拟 阵， 则 调 用 Greedy(M,w)将返回一个最优子集

pf:

①由推论16.9知，一开始被忽略的元素可以放弃，因为它们不是Φ的扩张意味着此后也不会是任何独立子集的扩张。

②一旦一个元素x被选中，引理16.7保证了算法将其扩充到A中是正确的，因为存在一个最优子集包含x。

75

16.4.2 加权拟阵上的贪心算法

③引理16.10蕴含着剩余子问题是在M'中找一最优子集。在Greedy将A置为{x}之后，剩下的各步骤可解释为是在拟阵M'中进行的，因为∀B∈I'，B在M'中独立等价于B∪{x}在M中独立。因此，Greedy的后续操作将找出M'的一个最优子集(可用归纳法)，Greedy的全部操作可以找到M的最优子集

- 若一应用问题能抽象为加权拟阵，则一定能用贪心法求出其最优解

76

16.5 一个任务调度问题

- 问题描述

  在单处理器上对单位时间任务进行最优调度

  - 输入
    - 任务集：$S=\{a_1,a_2,\ldots,a_n\}$，每个任务在单位时间内完成，总时间为n
    - 截止期：$\forall a_i \in S$，a_i的截止期d_i为整数，且$1 \le d_i \le n$，即a_i须在d_i或d_i时刻之前完成
    - 权(罚款)：$\forall a_i \in S$，a_i的权$w_i \ge 0$，表示a_i没有在d_i或d_i之前完成所导致的罚款，但提前完成或按时完成的任务没有罚款
  - 输出：求出S的一个调度，使总罚款最小

显然S的任意枚举都是一个调度方案，在总时间n内肯定完成，但我们力图使延期完成的任务权值和最小

77

16.5 一个任务调度问题

- 将问题抽象为加权拟阵
  - 早任务：在给定的调度中，能按期或提前完成的任务(即在deadline之前完成任务)
  - 迟任务：在给定的调度中，在截止期之后完成的任务
  - 早任务优先形式：将早任务排在迟任务之前的调度
  - 任何调度均可安排成早任务优先形式

    例：……a_i……a_j…… → ……a_j……a_i……

    迟　早　　　　早　迟

    交换位置不影响a_i和a_j的早迟特性，即总罚款不变

78

16.5 一个任务调度问题

- 调度的规范形式

早任务在前、迟任务在后(即先将其按早任务优先形式调度)；并且将所有早任务按deadline单调增次序调度(迟任务可按任意次序调度)。

- 规范形式的调度不改变任务的早迟特性，因此任何调度均可安排成该形式而保持总罚款不变

例：

79

16.5 一个任务调度问题

任务顺序	早任务	迟任务
原调度次序	$\ldots a_i \ldots a_j \ldots$	…
完成时刻	…k…k+s…	…
deadlines	$d_i>d_j$	…

这里s≥1。在规范形式下，a_i应和a_j交换：

1.a_j交换后，其完成时间由k+s提前至k，仍是早任务

2.交换前a_j是早任务，即$k+s\le d_j$，由$d_j<d_i$知$k+s<d_i$，故a_i交换到k+s时刻完成，它仍是早任务

80

16.5 一个任务调度问题

- 最优调度的规范次序

因为任一调度都有一规范形式，所以最优调度也存在规范形式

①在最优调度S中找出早任务集A

②将A中任务按deadlines递增序排列

③迟任务集S-A可按任意序排列

注意：

因为早任务不导致罚款，迟任务的次序也不改变罚款的多少(权的大小)，所以上述安排不改变最优调度的权值

81

16.5 一个任务调度问题

- 独立任务集

若存在一个调度使得任务集A中没有迟任务，则称A是独立的。

下面的引理可判定一给定任务集A是否独立，为此先给出一个定义。

- Def：对t=1,2,…,n, 设 $N_t(A)$ 表示集合A中deadline小于等于t的任务数目，$N_0(A)=0$ for any A。

即$N_t(A)$表示A中想要在t及t时刻之前完成的任务个数，显然$N_t(A)\le|A|$，$0\le t\le n$

82

16.5 一个任务调度问题

- 引理16.12 对任意的任务集，下列命题等价：

(1)集合A独立

(2)对t=0,1,…,n，有$N_t(A)\le t$

(3)若对A中的任务按deadline单调增序进行调度，则A中无迟任务

Pf：(1)→(2) (反证法)

若(1)成立，则存在一个调度，使A中无迟任务

若存在t使$N_t(A)>t$，则A中有多于t个任务要在t及t之前完成，因此找不到一个调度使A中任务均在t及t之前完成，即任何调度均使A中至少有一个迟任务，与A独立矛盾。

83

16.5 一个任务调度问题

(2) →(3)

实际上只要 “$1\le t\le|A|$，有$N_t(A)\le t$” 即可

因为t>|A|时，$N_t(A)\le|A|<t$

若(2)成立，说明A是某个调度的早任务集，他的规范次序不会改变A的早任务性质。

∵∀i∈[0,n]，在i及i之前要完成的任务至多为i个

∴对A中任务按deadline增序调度时A中无迟任务

(3) →(1)

按(3)调度A中无迟任务，所以A独立

84

16.5 一个任务调度问题

- 使迟任务罚款和最小等价于使早任务罚款和最大(即权最大的独立子集)

下面的定理保证贪心法可求出最大总罚款的独立任务集A，即把该问题抽象为加权拟阵，从而由Th16.11保证贪心法可求出具有最大权值的独立任务集A

- Th16.13 若S是一个具有deadlines的单位时间任务的集合，且I为所有独立任务集构成的集合，则相应系统(S,I)是一个拟阵。

Pf:(1) S有穷，非空

(2) I的遗传性：∵独立的任务集是指对某调度而言没有迟任务的集合，∴独立任务集的任一子集必是独立的

85

16.5 一个任务调度问题

(3)M的交换性

设A、B∈I且|B|>|A|，设k是使$N_t(B) \le N_t(A)$成立的最大的t.

$N_n(B)=|B|$，$N_n(A)=|A|$(注意|S|=n，|A| ≤ n，|B| ≤ n ,1 ≤ d_i ≤ n，在n前所有任务可完成)

但是|B|>|A|⇨$N_n(B)>N_n(A)$，所以k≠n，即k<n

①当t≤k时有：$N_t(B) \le N_t(A)$(这样t一定存在，因为$N_0(B)=N_0(A)=0$)

②当k+1≤t≤n有$N_t(B)>N_t(A)$(注意$N_t(B)$和$N_t(A)$均为单调增for t)

86

16.5 一个任务调度问题

∴B中deadline为k+1的任务多于A，故可设a_i∈B-A且d_i=k+1.

设A'=A∪{a_i}，只要证A'独立即可

下面用引理16.12性质2来证A'独立，即要证

$$\forall t \in [0,n], 有 N_t(A') \le t$$

①对1≤t≤k

显然$N_t(A')=N_t(A)$(因为A' 中的a_i截止期为k+1)

又因为A独立，由引理16.12性质2知：

$N_t(A')= N_t(A) \le t$，1 ≤t ≤k <式1>

87

16.5 一个任务调度问题

②对k+1≤t≤n

∵ $N_t(A)<N_t(B)$, |A'|=|A|+1

∴ $N_t(A') \le N_t(B)$

又∵ B独立，由引理16.12性质2知：

$N_t(A') \le N_t(B) \le t$，k+1≤t≤n。 <式2>

综合<式1>和<式2>可得：

对1≤t≤n，有$N_t(A') \le t$

由性质2知A'独立

88

16.5 一个任务调度问题

- 利用加权拟阵的Greedy算法求最优调度
  - 通过Th16.11，可使用Greedy算法找权最大的独立任务集A，A是早任务集。
  - 时间

$$T(n) = O(n\lg n + nf(n))$$
$$= O(n\lg n + n\cdot n) = O(n^2)$$

注意到独立性检查时间为O(n)(实际上可减小到O(|A|))

89

16.5 一个任务调度问题

- 例子

a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

贪心法求出的最优子集A={1,2,3,4,7},S-A={5,6}(迟任务集)

规范形式的最优调度<$a_2,a_4,a_1,a_3,a_7,a_5,a_6$>

总罚款$w_5+w_6=50$

90

EX.

- 写出任务调度的贪心算法

16.6 最优装载

有一批集装箱要装上一艘载重量为c的轮船。其中集装箱i的重量为W_i。最优装载问题要求确定在装载体积不受限制的情况下，将尽可能多的集装箱装上轮船。

- 算法描述

最优装载问题可用贪心算法求解。采用重量最轻者先装的贪心选择策略，可产生最优装载问题的最优解。具体算法描述如下页。

16.6 最优装载

```
template<class Type>
void Loading(int x[],  Type w[], Type c, int n)
{
    int *t = new int [n+1];
    Sort(w, t, n);
    for (int i = 1; i <= n; i++) x[i] = 0;
    for (int i = 1; i <= n && w[t[i]] <= c; i++) {x[t[i]] = 1;
  c -= w[t[i]];}
}
```


16.6 最优装载

- 贪心选择性质

可以证明最优装载问题具有贪心选择性质。

- 最优子结构性质

最优装载问题具有最优子结构性质。

由最优装载问题的贪心选择性质和最优子结构性质，容易证明算法loading的正确性。

算法loading的主要计算量在于将集装箱依其重量从小到大排序，故算法所需的计算时间为O(nlogn)。

16.7单源最短路径

给定带权有向图G =(V,E)，其中每条边的权是非负实数。另外，还给定V中的一个顶点，称为源。现在要计算从源到所有其它各顶点的最短路长度。这里路的长度是指路上各边权之和。这个问题通常称为单源最短路径问题。

- 算法基本思想

Dijkstra算法是解单源最短路径问题的贪心算法。

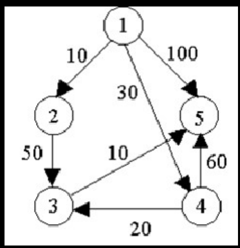
16.7单源最短路径

其基本思想是，设置顶点集合S并不断地作贪心选择来扩充这个集合。一个顶点属于集合S当且仅当从源到该顶点的最短路径长度已知。

初始时，S中仅含有源。设u是G的某一个顶点，把从源到u且中间只经过S中顶点的路称为从源到u的特殊路径，并用数组dist记录当前每个顶点所对应的最短特殊路径长度。Dijkstra算法每次从V-S中取出具有最短特殊路长度的顶点u，将u添加到S中，同时对数组dist作必要的修改。一旦S包含了所有V中顶点，dist就记录了从源到所有其它顶点之间的最短路径长度。

16.7单源最短路径

例如，对右图中的有向图，应用Dijkstra算法计算从源顶点1到其它顶点间最短路径的过程列在下页的表中。



16.7单源最短路径

Dijkstra算法的迭代过程：

迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1, 2}	2	10	60	30	100
2	{1, 2, 4}	4	10	50	30	90
3	{1, 2, 4, 3}	3	10	50	30	60
4	{1, 2, 4, 3, 5}	5	10	50	30	60

16.7单源最短路径

- 算法的正确性和计算复杂性
  - 贪心选择性质
  - 最优子结构性质
  - 计算复杂性

对于具有n个顶点和e条边的带权有向图，如果用带权邻接矩阵表示这个图，那么Dijkstra算法的主循环体需要$O(n)$时间。这个循环需要执行n-1次，所以完成循环需要$O(n^2)$时间。算法的其余部分所需要时间不超过 $O(n^2)$ 。

16.8 最小生成树

设G =(V,E)是无向连通带权图，即一个网络。E中每条边(v,w)的权为c[v][w]。如果G的子图G’是一棵包含G的所有顶点的树，则称G’为G的生成树。生成树上各边权的总和称为该生成树的耗费。在G的所有生成树中，耗费最小的生成树称为G的最小生成树。

网络的最小生成树在实际中有广泛应用。例如，在设计通信网络时，用图的顶点表示城市，用边(v,w)的权c[v][w]表示建立城市v和城市w之间的通信线路所需的费用，则最小生成树就给出了建立通信网络的最经济的方案。

16.8 最小生成树

- 最小生成树性质

用贪心算法设计策略可以设计出构造最小生成树的有效算法。本节介绍的构造最小生成树的Prim算法和Kruskal算法都可以看作是应用贪心算法设计策略的例子。尽管这2个算法做贪心选择的方式不同，它们都利用了下面的最小生成树性质：

设G=(V,E)是连通带权图，U是V的真子集。如果(u,v)∈E，且u∈U，v∈V-U，且在所有这样的边中，(u,v)的权c[u][v]最小，那么一定存在G的一棵最小生成树，它以(u,v)为其中一条边。这个性质有时也称为MST性质。

16.8 最小生成树

- Prim算法

设G=(V,E)是连通带权图，V={1,2,…,n}。

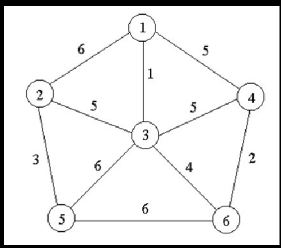
构造G的最小生成树的Prim算法的基本思想是：首先置S={1}，然后，只要S是V的真子集，就作如下的贪心选择：选取满足条件i∈S，j∈V-S，且c[i][j]最小的边，将顶点j添加到S中。这个过程一直进行到S=V时为止。

在这个过程中选取到的所有边恰好构成G的一棵最小生成树。

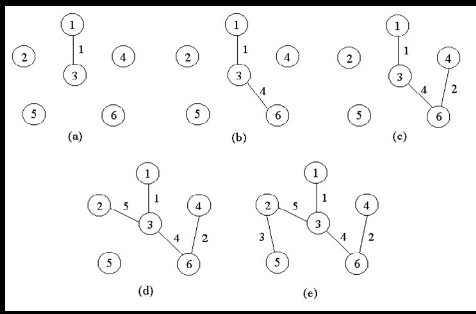
16.8 最小生成树

利用最小生成树性质和数学归纳法容易证明，上述算法中的边集合T始终包含G的某棵最小生成树中的边。因此，在算法结束时，T中的所有边构成G的一棵最小生成树。

例如，对于右图中的带权图，按Prim算法选取边的过程如下页图所示。



16.8 最小生成树



16.8 最小生成树

■ 最小生成树性质

在上述Prim算法中，还应当考虑如何有效地找出满足条件i∈S,j∈V-S，且权c[i][j]最小的边(i,j)。实现这个目的的较简单的办法是设置2个数组closest和lowcost。

在Prim算法执行过程中，先找出V-S中使lowcost值最小的顶点j，然后根据数组closest选取边(j,closest[j]），最后将j添加到S中，并对closest和lowcost作必要的修改。

用这个办法实现的Prim算法所需的计算时间为$O(n^2)$

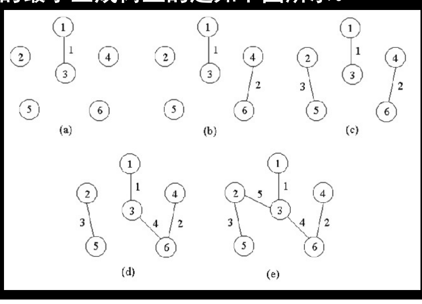
16.8 最小生成树

■ Kruskal算法

Kruskal算法构造G的最小生成树的基本思想是，首先将G的n个顶点看成n个孤立的连通分支。将所有的边按权从小到大排序。然后从第一条边开始，依边权递增的顺序查看每一条边，并按下述方法连接2个不同的连通分支：当查看到第k条边(v,w)时，如果端点v和w分别是当前2个不同的连通分支T1和T2中的顶点时，就用边(v,w)将T1和T2连接成一个连通分支，然后继续查看第k+1条边；如果端点v和w在当前的同一个连通分支中，就直接再查看第k+1条边。这个过程一直进行到只剩下一个连通分支时为止。

16.8 最小生成树

例如，对前面的连通带权图，按Kruskal算法顺序得到的最小生成树上的边如下图所示。



16.8 最小生成树

关于集合的一些基本运算可用于实现Kruskal算法。

按权的递增顺序查看等价于对优先队列执行removeMin运算。可以用堆实现这个优先队列。

对一个由连通分支组成的集合不断进行修改，需要用到抽象数据类型并查集UnionFind所支持的基本运算。

当图的边数为e时，Kruskal算法所需的计算时间是$O(eloge)$。当$e=\Omega(n^2)$时，Kruskal算法比Prim算法差，但当$e=o(n^2)$时，Kruskal算法却比Prim算法好得多。

16.9 多机调度问题

多机调度问题要求给出一种作业调度方案，使所给的n个作业在尽可能短的时间内由m台机器加工处理完成。

约定，每个作业均可在任何一台机器上加工处理，但未完工前不允许中断处理。作业不能拆分成更小的子作业。

这个问题是NP完全问题，到目前为止还没有有效的解法。对于这一类问题,用贪心选择策略有时可以设计出较好的近似算法。

109

16.9 多机调度问题

采用最长处理时间作业优先的贪心选择策略可以设计出解多机调度问题的较好的近似算法。

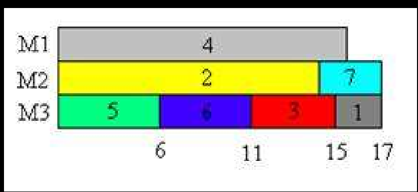
按此策略，当$n \le m$时，只要将机器i的[0, ti]时间区间分配给作业i即可，算法只需要O(1)时间。

当$n>m$时，首先将n个作业依其所需的处理时间从大到小排序。然后依此顺序将作业分配给空闲的处理机。算法所需的计算时间为O(nlogn)。

110

16.9 多机调度问题

例如，设7个独立作业{1,2,3,4,5,6,7}由3台机器M1，M2和M3加工处理。各作业所需的处理时间分别为{2,14,4,16,6,5,3}。按算法greedy产生的作业调度如下图所示，所需的加工时间为17。



111