第五章 贪心方法

目录

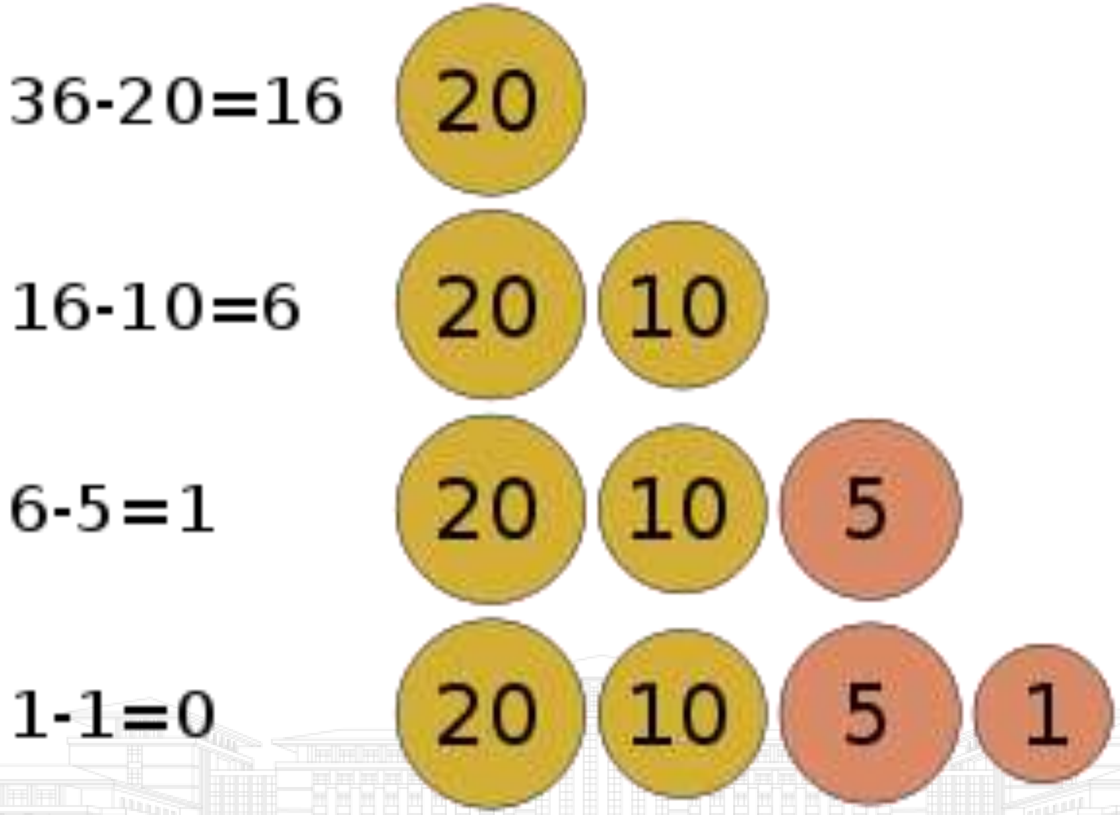
- 5.1 一般方法
- 5.2 背包问题
- 5.3 带有期限的作业调度问题
- 5.4最小生成树问题
- 5.5货郎担问题
- 5.6小结

5.1 一般方法

- 方法适用的问题特点
- 方法的基础知识
- 方法的求解步骤及核心问题
- 方法的抽象化控制
- 方法的缺点和优点

一个现实世界中的例子

- 把36元换成若干20元、10元、5元、1元的零钱，要求换得零钱的张数最少？



贪心方法适用的问题特点

- 有这样一类问题：它有n个输入, 而它的解就是这n个输入的某个子集，这些子集必须满足某些事先给定的条件。
- 约束条件：必须满足的条件
- 可行解：满足约束条件的子集
- 目标函数：为了衡量可行解的优劣，预先给定衡量标准，以函数形式给出
- 最优解：使目标函数取极值(极大值或极小值)的可行解

思考：上述术语和上一页的例子如何对应？

贪心法的求解思想

- 贪心方法是一种"只顾眼前"的分级处理方法：
  - 根据题意选取一种量度标准；
  - 按该标准一次选中一个输入；
  - 如果这个输入和当前的部分解加在一起满足约束条件，则将其加入到部分解中；否则舍弃掉。

思考：贪心法得到的可行解是否一定是问题的最优解？

取决于贪心法中的哪个环节？

量度标准

算法5.1 贪心法的抽象化控制

```
Procedure GREEDY(A, n)
  //A(1:n)包含n个输入
  solution←Φ  //解向量solution初始化为空
   for i ←1 to n do
      x← SELECT(A)
      if FEASIBLE(solution,x)
         then solution←UNION(solution,x)
      endif
    repeat
   return (solution)
end GREEDY
```

- SELECT：按某种最优量度标准从A中选择一个输入赋值给x
- FEASIBLE：判定x是否可以包含在解向量中。
- UNION：将x与解向量结合并修改约束判定。

贪心法的特点

- 贪心法设计求解的核心问题是
  - 选择能产生问题最优解的量度标准，即最优量度标准
- 缺点：
  - 不是对所有问题都能得到最优解
  - 基于目标函数制定的度量标准不一定是最优的
  - 最优量度标准需要经过证明
- 优点：
  - 一旦证明成立后，它就是一种高效的算法
  - 策略的构造简单易行
  - 对许多问题都能产生整体最优解或者近似最优解

5.2 背包问题

- 问题描述
- 背包问题实例
- 背包问题的贪心算法
- 定理5.1

问题描述

- 已知有n种物品和一个可容纳M重量的背包，每种物品i的重量为w_i，效益值为p_i ,假定将物品i的某一部分x_i放入背包就会得到$p_i x_i$的效益($0 \le x_i \le 1, p_i > 0$)，采用怎样的装包方法会使装入背包物品的总效益为最大？
- 问题的形式化描述：

极大化	$\sum_{1 \leqslant i \leqslant n} p_i x_i$	$0 \le x_i \le 1,\ p_i > 0$
约束条件	$\sum_{1 \leqslant i \leqslant n} w_i x_i \le M$	$w_i > 0,\ 1 \le i \le n$

背包问题实例

- 考虑下列情况下的背包问题：
  - $n=3, M=20, (p_1,p_2,p_3)=(25,24,15), (w_1,w_2,w_3)=(18,15,10)$

下面给出4个可行解：

	(x_1, x_2, x_3)	$\sum w_i x_i$	$\sum p_i x_i$
①	(1, 2/15, 0)	20	28.2
②	(1, 0, 1/5)	20	28
③	(0, 2/3, 1)	20	31
④	(0, 1, 1/2)	20	31.5

背包问题的量度标准选择

- 标准1：选效益值为量度标准，即背包每装入一件物品就获得最大可能的效益值增量
- 步骤1. 将物品按效益值非增次序排序: $(p_1,p_2,p_3)=(25,24,15)$

  对应重量排序为$(w_1, w_2, w_3)=(18, 15, 10)$
- 步骤2. 按该次序逐一放物品（先放效益最大的）

背包问题的量度标准选择

- 先装物品1(效益最大), 即x_1=1, w_1=18;
- 物品2不能全部放入，只能放物品2的2/15，即x_2=2/15
- 总效益值为$\sum p_i x_i$ = 25+24*2/15=28.2
- 所得到的(x_1, x_2, x_3)=(1，2/15，0)是一个次优解，因为背包承重量消耗太快！

背包问题的量度标准选择

- 标准2：选物品重量为量度标准，即背包每次装入重量最小的物品，使背包承载量尽可能慢地被消耗。
- 步骤1. 将物品按重量非降次序排序（递增）:$(w_3,w_2,w_1)=(10,15,18)$

  对应效益排序为$(p_3,p_2,p_1)=(15, 24, 25)$
- 步骤2. 按该次序逐一放物品（先放重量最小的）

背包问题的量度标准选择

- 先装物品3(重量最小), 即$x_3=1$, $w_3=10$;
- 剩余背包承载量为20-10=10
- 再放物品2（重量为15），不能全部放入，只能放入物品2的10/15，即$x_2=2/3$
- 总效益值为$\sum p_i x_i=31$
- 所得到的$(x_1, x_2, x_3)=(0, 2/3, 1)$也是一个次优解，因为——容量虽然消耗地慢，但是效益值没有迅速增加。

如何兼顾标准1与标准2

- 标准1：以效益为标准，背包容积消耗快
- 标准2：以重量为标准，效益增加慢
- 有无可能二者兼顾？
- 标准3：以单位重量的效益为标准

  $(p_1,p_2,p_3)=(25,24,15)$, $(w_1,w_2,w_3)=(18,15,10)$

  物品1单位重量的效益：$p_1/w_1=25/18$

  物品2单位重量的效益：$p_2/w_2=24/15$

  物品3单位重量的效益：$p_3/w_3=15/10$

- 按p_i/w_i的非增次序排序: $(p_2/w_2, p_3/w_3, p_1/w_1) =(24/15, 15/10, 25/18)$

度量标准3

- 步骤1. 将物品按单位重量效益的非增次序排序（递减），（p_2/w_2，p_3/w_3，p_1/w_1）=(24/15，15/10，25/18)
- 步骤2. 按该次序逐一放物品
- 首先放入物品2，$x_2=1$
- 然后放入物品3，因剩余容量为$20-w_2=5$，故只能放入物品3的5/10，即1/2
- 总效益值为$\sum p_i x_i = 24+15*1/2=31.5$

度量标准3

- 所得到的可行解$(x_1,\ x_2,\ x_3)=(0,\ 1,\ 1/2)$
- 这是一个最优解！
- 为什么？
- 稍后给出的定理5.1会给出证明

算法5.2 背包问题的贪心算法

先将物品按p_i/w_i比值的非增次序排序(降序)

```
procedure GREEDY-KNAPSACK(P,W,M,X,n)
real  P(1:n), W(1:n), X(1:n), M, cu; //X表示解向量
integer   i, n;
X← 0
cu ← M //cu表示背包剩余容量
for i ← 1 to n do
     if  (W(i)>cu)  then  exit endif
     X(i) ← 1
     cu ← cu-W(i)
repeat
if (i≤n) then  X(i) ← cu/W(i) //物品i放入一部分
end  GREEDY-KNAPSACK
```

将解向量X初始化为0
cu为背包的剩余重量

若物品i的重量大于背包的剩余重量,则退出循环

若物品i的重量小于等于背包的剩余容量,则物品i可放入背包内

放入物品i 的一部分

最优量度标准证明的基本思想

- 把算法的贪心解x与假定的最优解相比较，如果这两个解不同，就去找开始不同的第一个x_i，然后设法用贪心解的x_i去替换掉最优解中的x_i，然后证明最优解在分量替换前后的目标函数值无任何变化。
- 反复进行这种代换，直到新产生的最优解与贪心解完全一样，从而证明了贪心解就是最优解。

证明步骤--构造性证明

- 分析贪心解X的形式
- 假设最优解Y
- 比较贪心解X和最优解Y
- 重新构造最优解，使之向X转化

定理5.1

如果$p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$，则算法5. 2GREEDY-KNAPSACK对于给定的背包问题实例生成一个最优解。

证明：设$X=(x_1,\ldots,x_n)$是算法所生成的贪心解。

1.如果所有的x_i等于1，显然这个解就是最优解。否则，设j是使$x_j \neq 1$的最小下标，由算法可知：

对于$1 \leq i < j$，$x_i = 1$;

对于$j < i \leq n$，$x_i = 0$;

对于$i=j$，$0 \leq x_j < 1$。

$x_1 \ldots\ldots x_{j-1}\ x_j\ x_{j+1} \ldots\ldots x_n$

$1, \ldots\ldots, 1, x_j, 0, \ldots\ldots, 0$

2.若X不是最优解，则必存在一个最优解 $Y=(y_1,\ldots,y_n)$，使得$\sum p_i y_i > \sum p_i x_i$。不失一般性，假定$\sum w_i y_i = M$，设k是使得$y_k \neq x_k$的最小下标，显然，这样的k必定存在。

3. 可以推断$y_k<x_k$成立，k与j的关系有三种可能发生的情况：

①若k<j，则$x_k=1$，因$y_k \neq x_k$，从而$y_k<x_k$；

②若k=j，如果$x_k<y_k$，因为$\sum w_i x_i =M$，则有$\sum w_i y_i>M$，这与$\sum w_i y_i =M$矛盾，如果$y_k=x_k$，与假设$y_k \neq x_k$矛盾。故k＝j时必有 $y_k<x_k$；

③若k>j，因为$\sum w_i x_i =M$，则有$\sum w_i y_i >M$，这是不可能发生的(与$\sum w_i y_i =M$矛盾)。

故不存在k>j的情况。

$x_1 \ldots\ldots x_{j-1}\ x_j\ x_{j+1} \ldots\ldots x_n$

$1, \ldots\ldots, 1, x_j, 0, \ldots\ldots, 0$

$y_1 \ldots\ldots y_{j-1}\ y_j\ y_{j+1} \ldots\ldots y_n$

4.现在把 y_k 增加到 x_k，那么必须从$(y_{k+1},\ldots,y_n)$ 中减去同样多的量，使得总容量仍为M。这构造一个新的解$Z = (z_1,\ldots,z_n)$，其中 $z_i= x_i$, $1\le i\le k$。并且

$$\sum_{k<i\le n} w_i(y_i-z_i) = w_k(z_k-y_k)$$

在k处增加的量

Y中：$y_i= x_i$, $1\le i\le k-1$

从K+1开始，后面减少的量

5.因此，对于Z有：

$$\sum_{1\le i\le n} p_iz_i = \sum_{1\le i\le n} p_iy_i + (z_k-y_k)p_k - \sum_{k<i\le n}(y_i-z_i)p_i$$

$$= \sum_{1\le i\le n} p_iy_i + (z_k-y_k)w_k\, p_k/w_k - \sum_{k<i\le n}(y_i-z_i)w_ip_i/w_i$$

$$\ge \sum_{1\le i\le n} p_iy_i + [(z_k-y_k)w_k - \sum_{k<i\le n}(y_i-z_i)w_i]p_k/w_k$$

$$= \sum_{1\le i\le n} p_iy_i$$

元素按p_i/w_i非增次序排列。

即$\sum p_iz_i \ge \sum p_iy_i$

6.若$\sum p_iz_i > \sum p_iy_i$，则Y不可能是最优解，所以$\sum p_iz_i = \sum p_iy_i$。如果Z=X，则X就是最优解；否则重复上面的讨论，每次Y中少一个和X不同的值，最终把Y转换成X，从而证明了X也是最优解，证毕。

5.3 带有期限的作业调度问题

- 问题描述
- 算法实现思想
- 定理5.2
- 定理5.3
- 算法5.4
- 集合树
- 算法5.5

带有期限的作业调度

- 问题描述——假定只能在一台机器上处理n个作业：
  - 每个作业均可在单位时间内完成;
  - 又假定每个作业i都有一个截止期限$d_i>0$(d_i是整数), 当且仅当作业i在它的期限截止之前被完成时，方可获得$p_i>0$的效益

问题的可行解是什么？

可行解是这n个作业的一个子集合J,
J中的每个作业都能在各自的截止期限之前完成

  - 可行解的效益值是J中这些作业的效益之和
  - 问题的最优解是什么？

  具有最大效益值的可行解就是最优解

带有期限的作业调度

作业调度的形式化描述

目标函数： $$\sum_{j \in J} p_j$$

约束条件： 作业 j 的处理时间 $t_j \le d_j$, $d_j > 0$, $p_j > 0$, $1 \le j \le n$

带有期限的作业调度

- 实例

有4个作业, n=4, 其效益为$(p_1,p_2,p_3,p_4)=(100,10,15,20)$
其截止期限为$(d_1,d_2,d_3,d_4)=(2,1,2,1)$,求该问题的最优解

	可行解J	处理顺序	效益值$\sum p_j$
①	{1}	作业1	100
②	{2}	作业2	10
③	{3}	作业3	15
④	{4}	作业4	20
⑤	{1，2}	2，1	110
⑥	{1，3}	1, 3 或 3, 1	115
⑦	{1，4}	4，1	120
⑧	{2，3}	2，3	25
⑨	{3，4}	4，3	35

算法实现思想

当前作业一旦满足约束条件，
问题就能获得的最大效益增量

- 寻找最优量度标准
  - 以目标函数$\sum p_j$作为量度标准，将各作业按效益p_i降序排列：$p_1 \geq p_2 \geq \ldots \geq p_n$

```
procedure GREEDY_JOB(D, J, n)
// 作业按p1≥ p2≥ …≥ pn的次序输入；期限值D(1:n)≥1；J是最优解//
J←{1}
for i ← 2  to  n  do
     if  (J∪{i}的所有作业都能在它们的截止期限前完成)
          then   J ← J∪{i}
     endif
repeat
end GREEDY_JOB
```

思考1：算法GREEDY_JOB是否能提供一个最优解？

思考2：对于给定的作业集合J，如何确定它是可行解？

定理5.2

思考1得证

定理5.2：算法GREEDY_JOB所描述的贪心方法总是得到一个最优解。

- 证明思路：
  - J是贪心方法求出的作业集合，I是一个最优解的作业集合。可以证明J和I具有相同的效益值，从而J也是最优解。

- 证明步骤：

（1）找一个属于J不属于I的元素，替换I中对应的元素，获得I’

（2）证明I’仍然最优

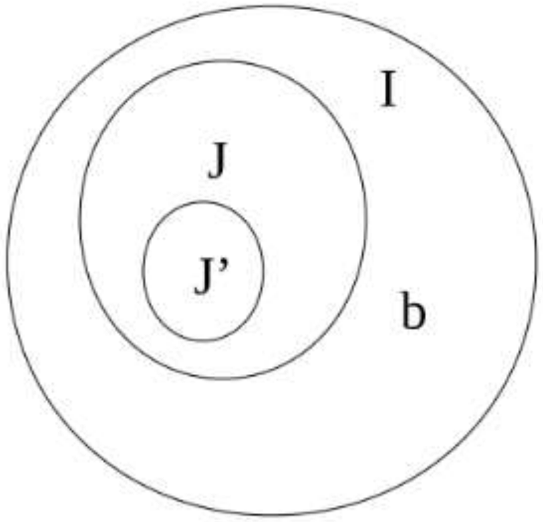
（3）重复步骤，间接证明J最优

不妨设I≠J，因为若I=J，
则J显然为最优解，无需证明

- 考虑最优解I 与贪心解J的关系

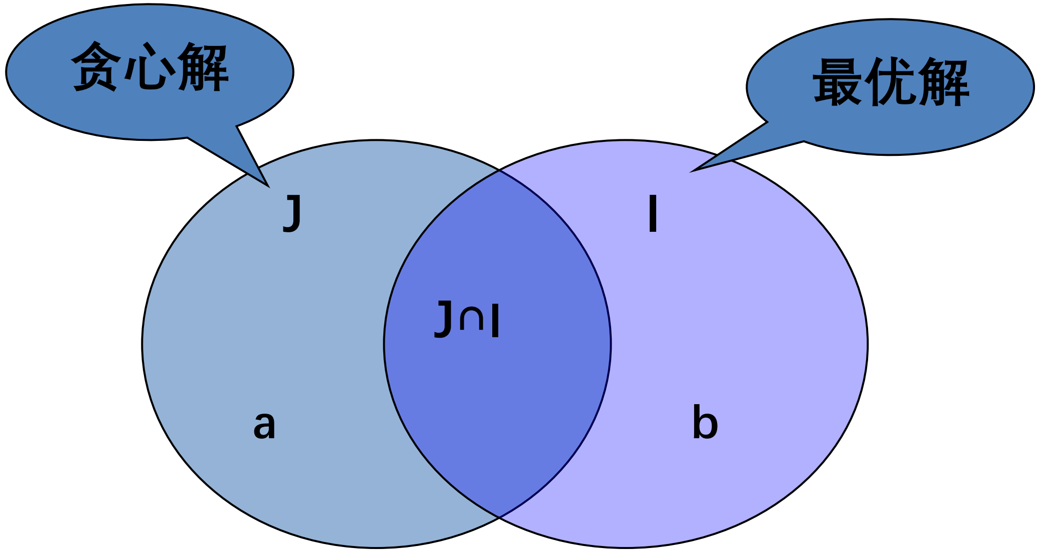
若$I \subset J$，则I的效益$<J$的效益，
这与I是最优解矛盾，故$I \subset J$不可能。

若$J \subset I$，由贪心方法的工作方式可知
$J \subset I$不可能



b是使得b∉J且b∈I的
一个的作业

因$I \subset J$与$J \subset I$均不可能，故



首先考虑J∩I，设作业i ∈J∩I

设S_J和S_I分别为J和I的可行调度表

希望作业i在S_J和S_I中的相同时间被调度

作业i在S_J中从t到t+1时刻被调用，作业i在S_I中从t'到t'+1时刻被调用，

- 情况1：t<t'，在S_J中将作业i与t'到t'+1时刻被调用的作业相置换，得到可行调度表$S_{J'}$；

t < t'

S_J: ....i....k....

S_I: ....i.....

→

t < t'

$S_{J'}$: ....k... i....

S_I: ....i....

- 情况2：t'<t，在S_I中将作业i与t到t+1时刻被调用的作业相置换，得到可行调度表$S_{I'}$；

t' < t

S_J: …………….i….

S_I: ………i….k…..

→

t' < t

S_J: …………….i….

$S_{I'}$: ………k….i…..

- 情况3： t=t'，无需置换

置换后，作业i在相同时间被调用

对J∩I中所有作业执行上述操作，
得到可行调度表$S_{J'}$和$S_{I'}$，
J∩I中所有作业在相同时间被调用

考虑J∩I以外的作业

因$I \subset J$与$J \subset I$均不可能，

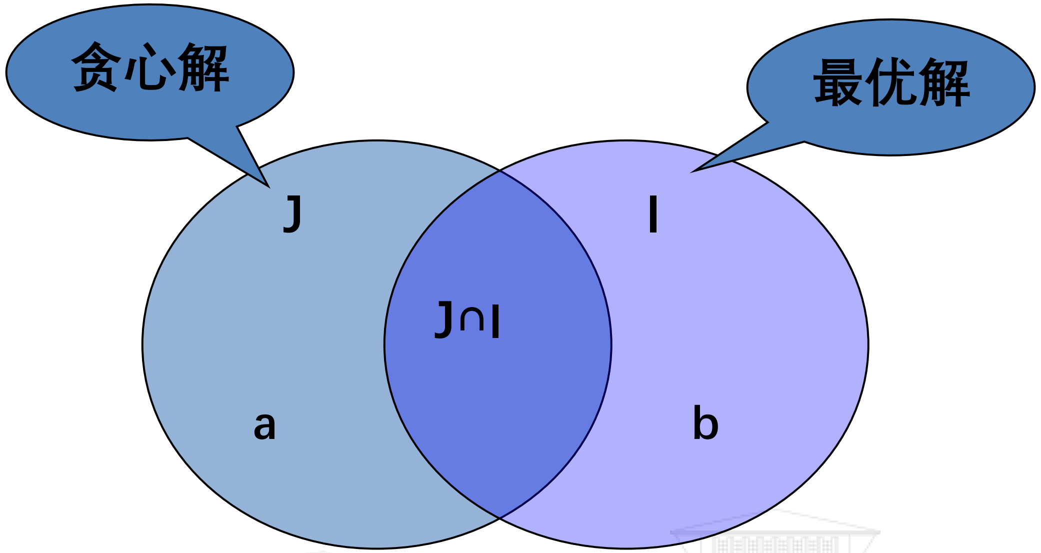
故至少存在这样的a和b：

$a \in J$且$a \notin I$

$b \in I$且$b \notin J$

设a是使得$a \in J$且$a \notin I$的效益值最大的作业

则$\forall b \in I$且$b \notin J$，有$P_a \geq P_b$



若P_a<P_b，则根据贪心策略，b会被包含到贪心解J中

- 若在调度表S_J中作业a在$[t_a,t_a+1]$时刻被调度，考虑在S_I中的$[t_a,t_a+1]$时刻被调度作业b
- 因为$P_a \geq P_b$,在S_I中将作业b替换为作业a，得到作业集合$I'=I-\{b\} \cup \{a\}$，$S_{I'}$同样是可行调度表，且效益值不小于I的效益值.
- 重复应用上述转换，使I在不减少效益值的情况下转换为J，因此J必为最优解
- 证毕

如何判断J是可行解的策略？

- 检验J的所有可能的排列：
  - б=$i_1,i_2,\ldots,i_k$是J中作业的一种排列;
  - 完成作业i_j的最早时间是j，1≤j≤k；
  - 若排列中的每个作业的d_{i_j}≥j，则б是一个允许的调度序列，J是一个可行解；否则，检验其他排列形式。

检验一种特殊的排列：
按期限的非降次序。

如果所有排列都不可行，
则本作业集合不是可行解

定理5.3

思考2得证

定理5.3：设J是k个作业的集合，б=$i_1,i_2,\dots,i_k$是J中作业的一种排列，它使得$d_{i1}\le d_{i2}\le\dots\le d_{ik}$。J是一个可行解，当且仅当J中的作业可以按照б的次序而又不违反任何一个期限的情况来处理。

实例: $(p_1,p_2,p_3,p_4)=(100,10,15,20)$, $(d_1,d_2,d_3,d_4)=(2,1,2,1)$

根据贪心方法，要将各作业按效益p降序排列来逐一考虑，因此按照1,4,3,2的顺序依次考虑作业

J={1}　　б=1；J是一个可行解且б次序是可行调度

J={1, 4}　б=4,1；J是一个可行解且б次序是可行调度

J={1,4,3}　б=4,1,3；作业3违反它的期限, 故 б次序不是可行调度；实际上因为任何次序均不可，故J不是可行解

定理5.3

定理5.3：设J是k个作业的集合，б=$i_1,i_2,\ldots,i_k$是J中作业的一种排列，它使得$d_{i1}\le d_{i2}\le\ldots\le d_{ik}$。J是一个可行解，当且仅当J中的作业可以按照б的次序而又不违反任何一个期限的情况来处理。

- 证明思路：
  - ⟵ 如果J中的作业可以按照б的次序而又不违反任何一个期限，则J是一个可行解。
  - ⟶ 若J是可行解，则必存在б'=$r_1,r_2,\ldots,r_k$，使得$d_{rj}\ge j$，$1\le j\le k$。
    - 假设б'≠б，令a是使得$r_a\ne i_a$的最小下标；设$r_b=i_a$，显然b>a。
    - 在б'中交换r_a与r_b的位置，产生新的可行排列б''，仍然是可行的。
  - 连续使用这种方法，就将б'转换成б且不违反任何一个期限。

б' : ……r_a….r_b….. ⟹ б'' : ……r_b….r_a…..

б : ……i_a ……….. б : ……i_a ………..

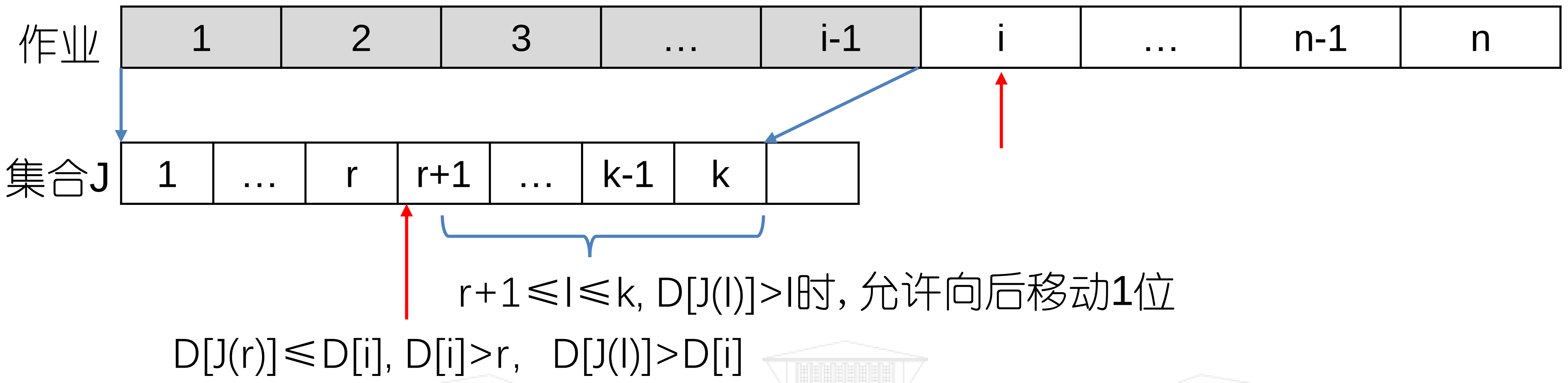
$d_{rb}=d_{ia}\leqslant d_{ra}$

思考：r_a延后执行是否可行？

基于定理5.3检验可行解

- 假设已处理了i–1个作业，有k个作业已存入J中，且D[J(1)]≤D[J(2)]≤…≤D[J(k)]
- 在J中从后向前为作业i寻找位置r+1，i插入r+1位置后，J中作业仍按照期限值从小到大排列，且不违反期限值

用直接插入法比较期限值大小



算法5.4带有限期的作业排序的贪心算法

```
procedure JS(D, J, n, k)
 integer   D(0:n), J(0:n), i, k, n, r;
 D(0) ←J(0) ←0;  k←1;  J(1) ← 1;
 for i←2 to n do
       r ← k;
       while ( D(J(r))>D(i) and D(J(r)) ≠r ) do
             r←r – 1
       repeat
       if  ( D(J(r))≤D(i) and D(i)>r ) then
           for  l←k to r+1  by –1 do
                   J(l+1)←J(l);
           repeat
            J(r+1)← i ;   k←k+1;
       endif
 repeat
end JS
```

从D(J(k))到 D(J(1))依次与D(i)比较来寻找插入位置r的过程

满足条件表示找到插入位置r

实现作业r+1到作业k依次往后移动一个位置

将作业i插入到r+1位置

- 带限期的作业排序贪心算法的时间复杂度

对于JS有两个赖以测量其时间复杂度的参数:

作业数 n 和包含在解J中的作业数 s

内层的while循环至多循环k次

插入作业 i 要执行时间为O(k–r)

外层for循环共执行(n–1)次

如果s是k的终值，即s是最后所得解的作业数，

则JS算法所需要的总时间为O(sn)

由于s≤n，所以JS算法的最坏时间复杂度为$O(n^2)$

43

一种更快的作业排序算法

- 改进思想：对作业i分配时间时，尽可能推迟对作业i的处理。(在其截止期前最靠后的空时间片)

- 算法思想：

如果J是作业的可行子集，可使用下述规则来确定J中每个作业的处理时间：若还没给作业i分配处理时间，则分给它时间片[α-1, α],其中α 应尽量取大,且该时间片是空的。若正在被考虑的作业i不存在像上面定义的α,则这个作业就不能计入J中。算法的复杂度由O(n^2)降低到接近于O(n).

一种更快的作业排序算法

- 快速作业排序实例

i	1	2	3	4	5
p_i	20	15	10	5	1
d_i	2	2	1	3	3

可行解J	已分配的时间片	考虑的作业	分配动作
∅	无	1	分配[1, 2]
{1}	[1, 2]	2	分配[0, 1]
{1, 2}	[0, 1] [1, 2]	3	舍弃
{1, 2}	[0, 1] [1, 2]	4	分配[2, 3]
{1, 2, 4}	[0, 1] [1, 2] [2, 3]	5	舍弃

问题：对于作业i，如何得到α？

集合树(补充)

- 集合树表示一个集合
  - 树中每个结点只设置PARENT信息段。
  - 对于非根结点i，PARENT(i) 存放着i的父结点下标。
  - 对于根结点i，PARENT(i)= -k，k>0表示树中结点个数。
- 集合树的操作
  - 查找：FIND(i)表示基于压缩规则查找结点i的根结点。
  - 合并：UNION(i, j)表示基于加权规则合并两个集合树i和j。

集合的合并和查找

Procedure U(i,j)

//将以i和j为根的两棵树合并为一棵树

PARENT(i)←j

合并后树根为j

End U

Procedure Find(i)//找包含元素i的树的根

j←i

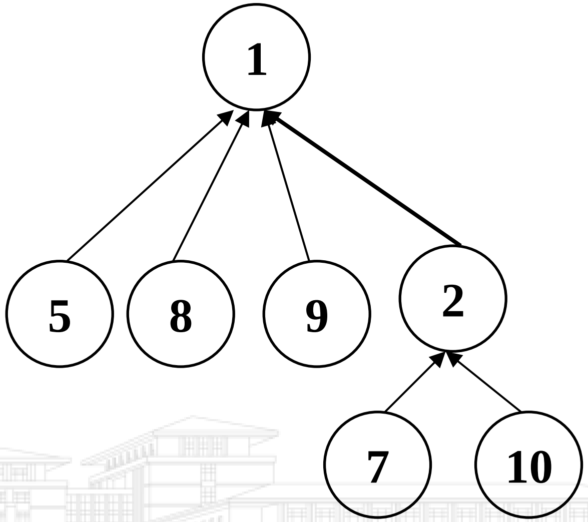
While PARENT(j)>0 do

j←PARENT(j)

repeat

return(j)

end F

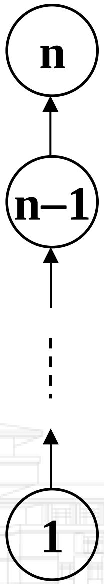


集合的合并和查找

假设有n个元素，开始时各自属于一个单独的集合，即Si={i}；

要执行如下运算

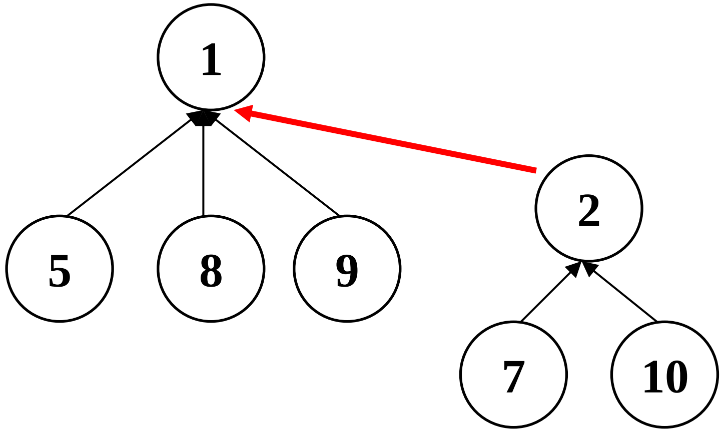
- U(1,2),Find(1),
- U(2,3),Find(1),
- U(3,4),Find(1),
- …,
- U(n–1,n), Find(1)



集合的合并和查找

- 加权规则：

集合树合并时，如果树i的结点数少于树j的结点数，则把树i链接到树j上，令j为i的父节点，反之，令i为j的父节点。



集合的合并和查找

```
Procedure  UNION(i,j)
integer i,j,x
x←PARENT(i)+ PARENT(j)
if PARENT(i)>PARENT(j)
    then  PARENT(i) ←j
          PARENT(j) ←x
    else   PARENT(j) ←i
          PARENT(i)←x
endif
end UNION
```

合并后集合中元素个数

若树j的结点个数多，以j为根

若树i的结点个数多或两棵树结点个数相同，以i为根

集合的合并和查找

- 在查找时使用压缩规则：

  如果k是由i到其根的路径上的一个结点，则置PARENT(k)←root(i)

集合的合并和查找

//查找含有元素i的树根j，并使用压缩规则压缩i到根j的所有节点

```
procedure FIND(i)

j←i
while  PARENT(j)>0 do
   j←PARENT(j)
repeat
k←i
while k≠j do
   t←PARENT(k)
   PARENT(k) ←j
   k←t
repeat
return(j)
end FIND
```

j：树根结点

找到结点i所在树的树根j

k：由i到j的路径上经过的结点

从结点i到树根j的路径上所有结点的PARENT(k)都变为j

集合树的操作总结

- FIND(i)：
  - 找到结点i所在树的树根j，并且将从结点i到树根j的路径上所有结点k的PARENT(k)都变为j，并返回根节点j
- UNION(i, j)：
  - 将以i和j为根的两棵树合并为一棵树，根据加权规则，如果树i的结点数少于树j的结点数，则j为i的父亲，反之，i为j的父亲

算法5.5更快的作业排序算法 设计思想

1. n个作业，每个作业花费一个单位时间，因此只需考虑这样一些时间片 $[i-1,i]$，$1\le i\le b$, $b=\min\{n,\max\{d_i\}\}$ 。为简便，用i表示时间片$[i-1,i]$

例如：10个作业,作业的截止期中最大值是5，只需考虑时间段[0,5]

5个作业,作业的截止期中最大值是10，同样只需考虑时间段[0,5]

所以，n个作业的期限值只需要是{1,2,…,b}中的某些（或全部）元素

2. 调度方法是：把这b个期限值分成一些 **集合**

5.3带有限期的作业排序

3. 对于任一期限i，设n_i是满足 $n_i \le i$ 且$[n_i-1, n_i]$为空时间片的最大整数

4.当且仅当$n_i=n_j$(期限值$i \ne j$)时,期限值i和j在同一个集合中（即所要处理作业的期限值如果是i或j，则当前可分配的最接近的时间片是n_i）。此时，若$i<j$, 表明时间[i,j]期间的时间片都被占用了, 此时i, i+1, i+2,……, j 都在同一个集合中。

上述方法就是作出一些以期限值为元素的集合，且使同一集合中所有元素具有相同的最大空时间片

5.对于每个期限值i，用F(i)表示当前最大空时间片，即 $F(i)=n_i$ 。

注意，F(i)不同于Find(i)

6. 使用集合的树表示法，把每个集合表示成一棵树。P(i)把期限值i链接到它的父节点。
7. 判断作业h的可用空时间片，即找min{n,d_h}的根j，若F(j)≠0,说明有时间片可以分配，则 F(j)是最接近期限值的时间片，把F(j)时间片分配给作业h并做标记。
8. 在F(j)时间片被分配以前，以j为根的集合树中所有元素的最近空时间片都是F(j)；现在F(j)时间片被分配，以j为根的集合树中元素的最近空时间片只能变为期限F(j)−1的最近空时间片，因此——以j为根的集合树必须与包含期限F(j)−1的集合树合并。

算法5.5设计思想

- 初始
  - $p_1 \geq p_2 \ldots \geq p_n$
  - $b=\min\{n,\max\{d_j\}\}$
  - $F(i) \leftarrow i$ //时间片都为空，所以F(i)=i
  - $P(i) \leftarrow -1$ // i自己是一棵树，Parent(i)的缩写
- 依次检验每个作业w
  - 寻找D(w)所在树的根节点j
  - 如果F(j) ≠0， F(j)时间片可以分配给作业w， 因此，将w并入到解集合J中，
  - 寻找F(j)-1所在树的根节点l，将l和j所在的两个树合并到一起。
  - $F(j) \leftarrow F(l)$

$j \leftarrow$FIND(min(n, D(w)))

$k \leftarrow k+1$;J(k) $\leftarrow w$;

$l \leftarrow$FIND(F(j)-1);call UNION(i,j);

算法5.5 一个更快算法

```
procedure FJS(D,n,b,J,k)
// 找最优解J=J(1),…,J(k)，已知p1≥ p2… ≥ pn, b=min{n,max{dj}}
integer b, D(n), J(n), F(0:b), P(0:b)
for i ←0 to b do
   F(i) ← i; P(i) ← -1
repeat
k← 0
for i ←1 to n do
   j← FIND(min(n, D(i)))
   if F(j) ≠0
     then k ←k+1;J(k) ←i;
          l←FIND(F(j)-1);call UNION(l,j);
          F(j) ← F(l)
   endif
repeat
End FJS
```

作业i的期限值所在的集合的根

可以给作业i分配时间片

需更新其最大可用时间片

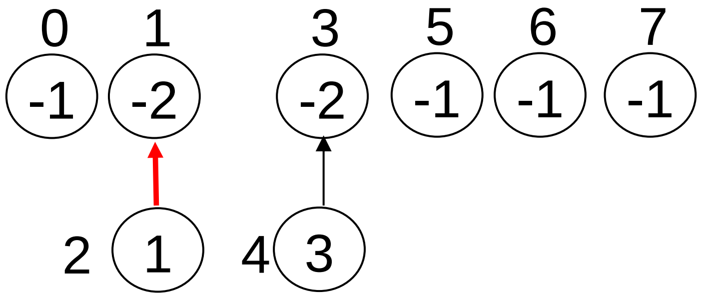
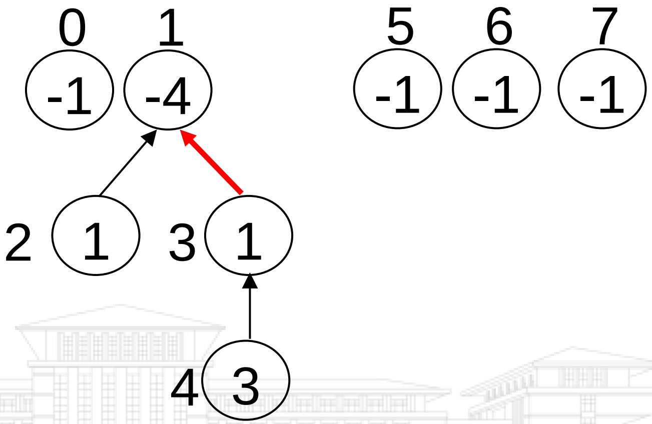
时间复杂度是0(n×ackermann函数的逆函数)

算法示例

- 设n=7,
- $(p_1, p_2, \ldots, p_7)=(35,30,25,20,15,10,5)$
- $(d_1, d_2, \ldots, d_7)=(4,2,4,3,4,8,3)$ **b= ?**

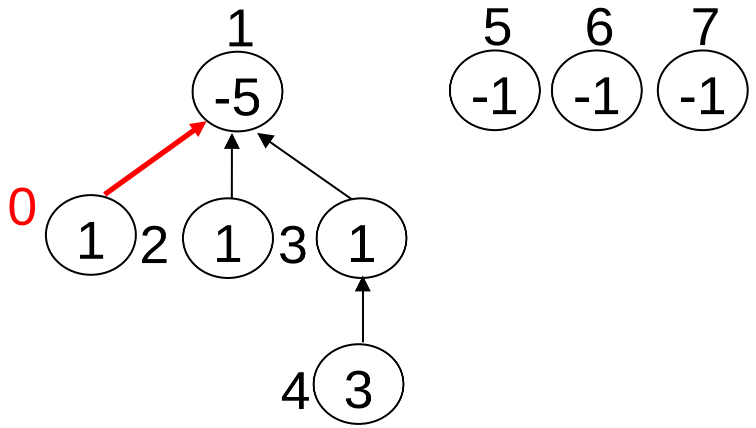
考虑作业	F(0)(1)(2)(3)(4)(5)(6)(7)	树P	J
无	0 1 2 3 4 5 6 7	0: -1, 1: -1, 2: -1, 3: -1, 4: -1, 5: -1, 6: -1, 7: -1	ϕ
1	0 1 2 3 3 5 6 7	0: -1, 1: -1, 2: -1, 3: -2, 5: -1, 6: -1, 7: -1; 4: 3	{1}

- 设n=7,
- $(p_1, p_2, \ldots, p_7)=(35,30,25,20,15,10,5)$
- $(d_1, d_2, \ldots, d_7)=(4,2,4,3,4,8,3)$

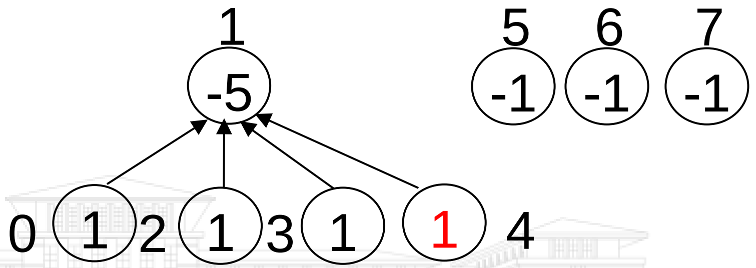
考虑作业	F(0)(1)(2)(3)(4)(5)(6)(7)		
2	0 1 1 3 3 5 6 7		{1,2}
3	0 1 1 1 3 5 6 7		{1,2,3}

- 设n=7,
- $(p_1, p_2, \ldots, p_7)=(35,30,25,20,15,10,5)$
- $(d_1, d_2, \ldots, d_7)=(4,2,4,3,4,8,3)$

考虑作业	F(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
4	0	0	1	1	3	5	6	7
5 (舍弃)	0	0	1	1	3	5	6	7

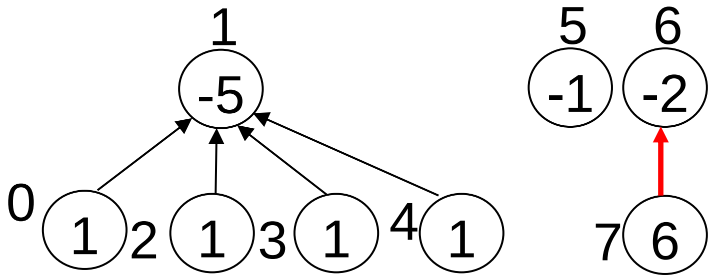


{1,2,3,4}



{1,2,3,4}

- 设n=7,
- $(p_1, p_2, \ldots, p_7)=(35,30,25,20,15,10,5)$
- $(d_1, d_2, \ldots, d_7)=(4,2,4,3,4,8,3)$

考虑作业	F(0)(1)(2)(3)(4)(5)(6)(7)		
6	0 0 1 1 3 5 6 6		{1,2,3,4,6}
7（舍弃）	0 0 1 1 3 5 6 6	同上	

最优解J＝{1,2,3,4,6}，处理次序4,2,3,1,6，效益值120

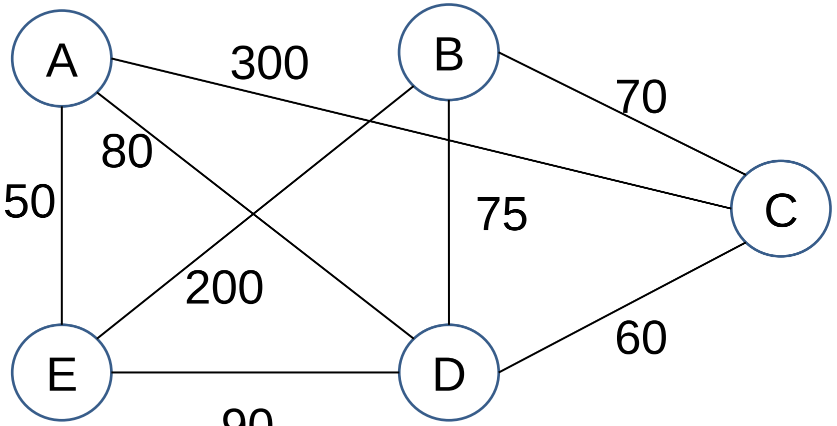
5.4 最小生成树问题

- 问题回顾
- Kruskal算法描述
- 贪心策略分析
- 实例运行
- 证明贪心解最优

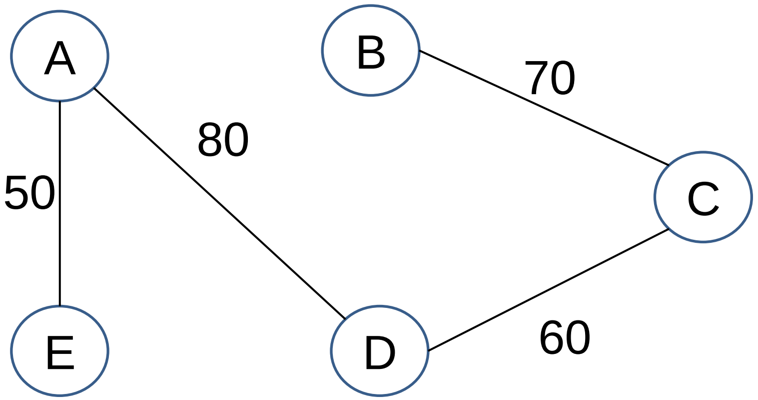
贪心法解决的一个有名问题是最小生成树问题，本节介绍最小生成树的Kruskal方法

问题回顾

- 最小生成树的定义：
  - 设G=(V,E)是一个加权无向连通图。V表示顶点集合，E表示边集合。G的一棵生成树是一棵无向树T=(V, E’)，其中E’是E的子集。生成树的权是E’的所有权之和。G的最小生成树是G的具有最小权值的生成树。



加权连通无向图



最小生成树

Kruskal算法描述

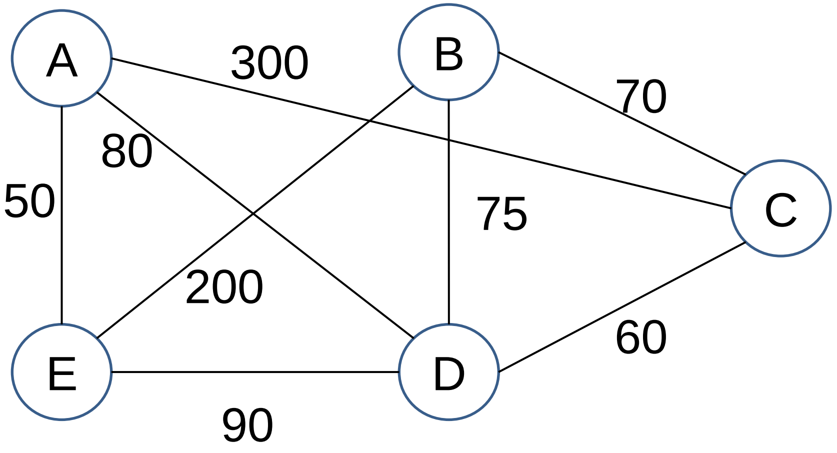
- input：一个加权连通无向图G=(V,E)
- output：图G的一棵最小生成树

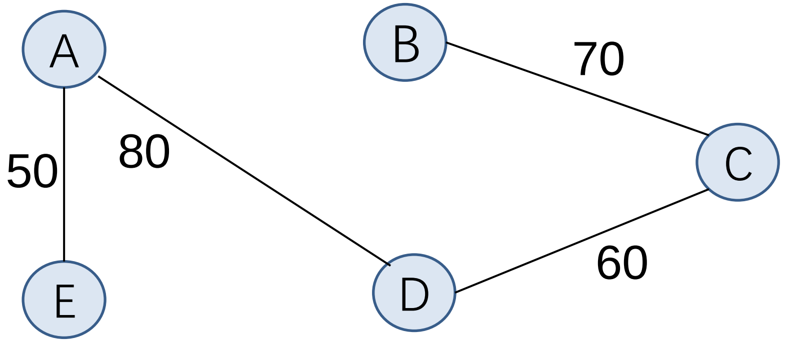
考虑预排序，从边的个数入手分析，时间复杂度为O(eloge)。

```
T←∅ //存储选中的边
while T所含的边数少于n-1 do
    从E中选择一条最小权值的边(v,w)；从E中删除该边
    if (将(v,w)加入T中没有形成一个回路)
        then 将(v,w)加入T中
        else 放弃(v,w)
    endif
repeat
```

实例运行

- 边的权值排序：
  - 边(AE) 50
  - 边(CD) 60
  - 边(BC) 70
  - 边(BD) 75
  - 边(AD) 80
  - 边(ED) 90
  - 边(BE) 200
  - 边(AC) 300





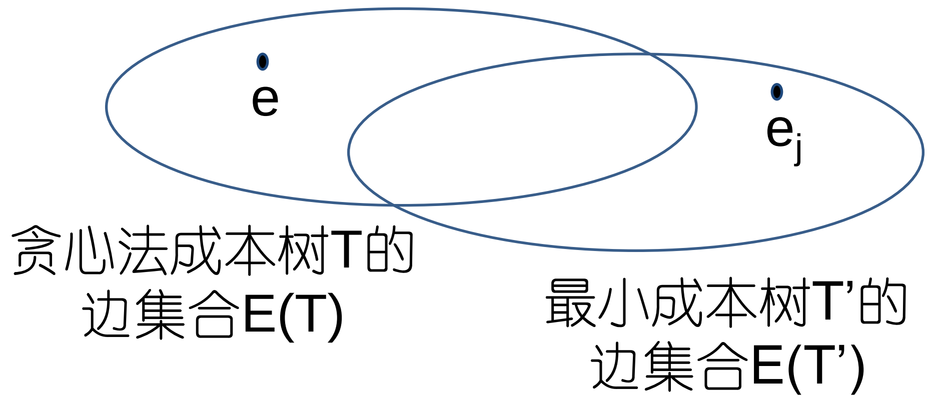
贪心策略分析

- 问题原型
  - 构造加权无向连通图G=(V,E)的最小生成树T=(V, E’)， |V|=n
- 约束条件：
  - T中|V|=n；T中|E’|= n-1，E’⊆E；T中没有环
- 目标函数：
  - 权值和最小
- 量度标准：
  - 权值从小到大排列

把目标函数作为度量标准

思考： 该量度标准是否是最优量度标准？

证明：Kruskal算法对于每一个无向连通图G产生一棵最小生成树。



- 证明路线：
  - 用e替换掉e_j，获得新的可行解T ",证明新解也是最小生成树。反复替换，从而命题得证。
- 证明难点：
  - 怎样选择e和e_j？
  - 怎样确定e和e_j的成本关系？

1. 设T是Kruskal算法产生的G的生成树，T’是G的最小成本生成树。现在要证明T和T’具有相同的成本。
2. 设E(T)和E(T’)分别是T和T’的边集。若n是G中的节点数，则T和T’都有n-1条边。
3. 若E(T)=E(T’)，则T显然就是最小成本生成树。
4. 若E(T)≠E(T’)，则假设e是一条使得e∈E(T)但e ∉E(T’)的最小成本的边。
5. 把e加入到T’中，则一定会产生一个环。假设$e,e_1,e_2,\ldots e_k$是这个环。
6. 可知$e_1,e_2,\ldots e_k$中至少有一条不在E(T)中，如若不然，则T也包含这个环，与算法矛盾。设e_j是这个环中使得$e_j \notin E(T)$的一条边。

4-6：选择e和e_j

e_j和e的关系

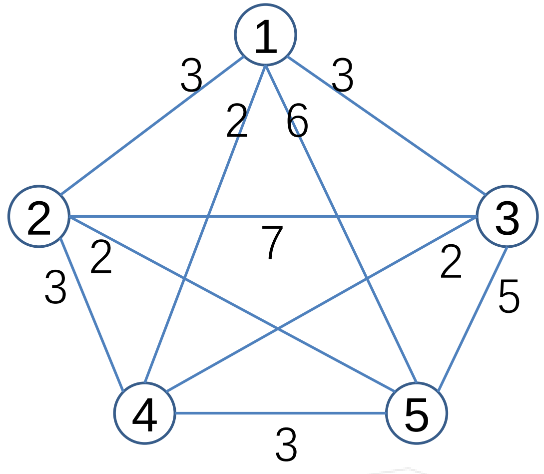
7. 如果e_j比e有更小的成本，则Kruskal算法会在e之前考虑e_j，并把e_j计入T中。与假设矛盾，因此e_j不比e有更小的成本，记作$c(e_j) \geq c(e)$。
8. 重新考虑E(T') U {e}的图。删去边e_j，产生新树T",由于$c(e_j) \geq c(e)$，因此T"的成本不比T'大。因此T"也是一棵最小成本树。
9. 反复上述转换，树T'转换成T而在成本上没有任何增加，故T是一棵最小成本生成树，证毕。

5.5 货郎担问题

- 货郎担问题也叫旅行商问题，即TSP问题（Traveling Salesman Problem），是数学领域中著名问题之一。
- 问题描述：某售货员要到若干个城市销售货物，已知各城市之间的距离，要求售货员从某一城市出发并选择旅行路线，使每个城市经过一次，最后回到原出发城市，而总路程最短。

贪心策略

- 根据目标函数，选择距离当前城市最短距离的城市
  - 从某一个城市开始，每次选择一个新城市，直到所有的城市被走完。
  - 在选择下一个城市时，只考虑当前情况，保证当前选择的城市距离最小且不形成回路。



	1	2	3	4	5
1	∞	3	3	2	6
2	3	∞	7	3	2
3	3	7	∞	2	5
4	2	3	2	∞	3
5	6	2	5	3	∞

直接法求最优解：算法的时间复杂度？

基于贪心策略的近似解：n条回路中的最小距离。复杂度？

从城市1出发

	1	2	3	4	5
1	∞	3	3	2	6
2	3	∞	7	3	$\boxed{2}$
3	3	7	∞	$\boxed{2}$	5
4	$\boxed{2}$	3	2	∞	3
5	6	2	$\boxed{5}$	3	∞

1->4->3->5->2->1，总距离14

从城市2出发

	1	2	3	4	5
1	∞	3	3	$\boxed{2}$	6
2	3	∞	7	3	2
3	$\boxed{3}$	7	∞	2	5
4	2	3	2	∞	$\boxed{3}$
5	6	$\boxed{2}$	5	3	∞

2->5->4->1->3->2，总距离17

5.6 小结

- 贪心法特点：
  - 多步判断，只顾眼前
    - 不考虑子问题的计算结果的优劣
    - 只考虑子问题当前决策的优劣
  - 有时只能获得局部最优解
  - 全局最优解需要经过正确性证明
- 对于许多NP难的组合优化问题，近似算法是一种比较可行的途径
  - 贪心法常用于这些近似算法的设计中。

5.6 小结

- 5.1 一般方法
  - 掌握约束条件、可行解、最优解、目标函数的定义；理解量度标准的意义；掌握贪心方法适用的问题特点和求解思想等基本知识。能掌握贪心法求解问题的一般过程。
- 5.2背包问题
- 5.3 带有期限的作业调度问题
  - 掌握贪心法最优解证明的一般方法和贪心算法设计的一般思想。深入理解贪心法时间复杂度的影响因素，掌握贪心法优化思路。

5.6 小结

- 5.5 最小生成树问题
  - 深入理解贪心法适用的问题特征，掌握复杂问题求解办法。掌握贪心法求解复杂问题时最优解的证明方法。
- 5.6 货郎担问题
  - 理解贪心法近似解和实际最优解间的差异。

能够识别出适合贪心法的可计算性问题、独立设计算法和分析算法复杂度。

本 章 结 束