第四章 动态规划

4.8 0/1 背包问题

- 在0/1背包问题中，物体或者被装入背包中，或者不被装入背包中，只有两种选择。
- 假设：x_i表示物体被装入背包的情况，则有当x_i=0时表示物体没有被装入背包，当x_i=1时表示物体被装入背包。
- 根据题目要求，有如下约束方程和目标函数：

$$\sum_{i=1}^{n} w_i x_i \le M, \quad optp = \max \sum_{i=1}^{n} p_i x_i$$

问题被归结为寻找一个满足上述约束方程并且使得目标函数为最大的解向量$X(x_1, x_2, \ldots, x_n)$。

假设背包的重量范围为0~m，类似于资源分配一样，令$optp_i(j)$ 表示在前i个物体中，能够装入载重量为j 的背包中的物体的最大值，j=1,2,…,m。

显然，此时在前i个物体中，有些物体可以装入背包，有些物体不能装入背包。于是可以建立下来动态规划函数：

$$optp_i(0) = optp_0(j) = 0 \qquad (4.8.1)$$

$$optp_i(j) = \begin{cases} optp_{i-1}(j) & j < w_i \\ \max\{optp_{i-1}(j), optp_{i-1}(j-w_i) + p_i\} & j \ge w_i \end{cases} \qquad (4.8.2)$$

(4.8.1)式说明，将前面i个物体装入重量为0的背包，或者把0个物体装入重量为j的背包，得到的价值都为0.

$$optp_i(j) = \begin{cases} optp_{i-1}(j) & j < w_i \\ \max\{optp_{i-1}(j), optp_{i-1}(j-w_i) + p_i\} & j \ge w_i \end{cases} \qquad (4.8.2)$$

（4.8.2）第一式说明，如果第i个物体的重量大于背包的载重量，则装入前面i个物体得到的最大值与装入前i-1个物体得到的最大值一样（第i个物体没有装入背包）。

（4.8.2）第二式中$optp_{i-1}(j-w_i)+p_i$说明，当第i个物体的重量小于背包的重量时，如果把第i个物体装入载重量为j 的背包，则背包中物体的价值等于把前面i-1个物体状态载重量为j-w_i的背包所得到的价值加上第i个物体的价值p_i。

按照这样的定义，可以把求解划分为二个阶段：

*第一个阶段，*只装入一个物体，确定在各种不同载重量的背包下能够得到的最大值。

*第二阶段，*装入前两个物体，确定在各种不同的载重量的背包下等够得到的最大值。以此类推，直到第n个阶段。最后，$Opt\,p_n(m)$便是在载重量为m的背包下，装入n个物体时能够得到的最大价值。

为了确定装入背包的具体物体，从最大价值$Opt\,p_n(m)$向前倒推。如果$Opt\,p_n(m)$大于$Opt\,p_{n-1}(m)$，表面第n个物体被装入背包，则前n-1个物体被装入重量为m-w_i的背包中。如果$Opt\,p_n(m)$小于$Opt\,p_{n-1}(m)$，表面第n个物体未被装入背包中，则前n-1个物体被装入在载重量为m的背包中。以此类推，直到确定第一个物体是否被装入背包中为止。

若 $optp_i(j) = optp_{i-1}(j)$，则$x_i = 0$

若 $optp_i(j) > optp_{i-1}(j)$，则$x_i = 1, j = j - w_i$

例题：五个物体，重量为2,2,6,5,4，价值为6,3,5,4,6，

背包的重量为10.

求解：设定一个数组，存放前面i个物体装入载重量为j的背包时所获得的最大价值。

$$optp_i(0) = optp_0(j) = 0$$

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0										
2	0										
3	0										
4	0										
5	0										

例题：五个物体，重量为2,2,6,5,4，价值为6,3,5,4,6，

背包的重量为10.

i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	**0**	**6**								

i=1;*j*=1
*w*i=2,
j<*w*i
$optp_{i-1}(1)=0$

i=1;*j*=2
*w*i=2,
j=*w*i
$optp_{i-1}(2)=0$
$optp_{i-1}(2-2)+6=6$

$$optp_i(j) = \begin{cases} optp_{i-1}(j) & j < w_i \\ \max\{optp_{i-1}(j), optp_{i-1}(j-w_i)+p_i\} & j \ge w_i \end{cases}$$

例题：五个物体，重量为2,2,6,5,4，价值为6,3,5,4,6，

背包的重量为10.

i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6	6
2	0	0	6	6	9	9	9	9	9	9	9
3	0	0	6	6	9	9	9	9	11	11	14
4	0	0	6	6	9	9	9	10	11	13	14
5	0	0	6	6	9	9	12	12	15	15	***15***

$$optp_i(j) = \begin{cases} optp_{i-1}(j) & j < w_i \\ \max\{optp_{i-1}(j), optp_{i-1}(j-w_i)+p_i\} & j \ge w_i \end{cases}$$

例题：五个物体，重量为2,2,6,5,4，价值为6,3,5,4,6，

背包的重量为10.

若　$optp_i(j) = optp_{i-1}(j)$，则$x_i = 0$

若　$optp_i(j) > optp_{i-1}(j)$，则$x_i = 1, j = j - w_i$

i

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6	6
2	0	0	6	6	9	9	9	9	9	9	9
3	0	0	6	6	9	9	9	9	11	11	14
4	0	0	6	6	9	9	9	10	11	13	14
5	0	0	6	6	9	9	12	12	15	15	***15***

i=5;*j*=10
$optp_i(10)=15$
$optp_{i-1}(10)=14$
*x*5=1;
*w*5=4,
j=10-4=6

i=4;*j*=6
$optp_i(6)=9$
$optp_{i-1}(6)=9$
*x*4=0;

i=3;*j*=6
$optp_i(6)=9$
$optp_{i-1}(6)=9$
*x*3=0;

i=2;*j*=6
$optp_i(6)=9$
$optp_{i-1}(6)=6$
*x*2=1;
*w*2=2,
j=6-2=4

i=1;*j*=4
$optp_i(4)=6$
$optp_{i-1}(4)=0$
*x*1=1;

4.9工程施工问题

问题的提出：

- 某住宅小区施工阶段。业主要求在春节前完成浇捣箱基墙板混凝土的施工。
- 浇捣箱基墙板混凝土不得留有施工缝隙，应该一次浇完。
- 天气条件可能变化。所以不能确定哪天开机浇捣混凝土。
- 在时间上距离业主的要求还有几天的选择余地。因此，对浇捣混凝土的具体日期不做轻率的决定，力求做到满足业主的要求，又做到科学经济合理。

工程施工问题

时间的安排：

- 把开机浇捣混凝土的时间安排在**5**天内加以选择。

开机最早时间定在**1**月**4**号，最迟开工时间为**1**月**8**号。

工程施工问题

不同天气条件下施工费用：

- 天无雨 ➡ •￥150000
- 天降小雨 ➡ •￥160000
- 天降大雨 ➡ •￥180000

工程施工问题

	January 4th	January 5th	January 6th	January 7th	January 8th
	0.2	0.2	0.2	0.4	0.4
	0.2	0.2	0.2	0.4	0.4
	0.6	0.6	0.6	0.2	0.2

Weather Condition

工程施工问题

把问题按天分为5个阶段,目标是使得施工费用最少

是否施工？ 是否施工？ ➡ 是否施工？➡ 是否施工？➡ 是否施工？

day1 — day2 — day3 — day4 — day5

工程施工问题

解题思路：

➢如果这一天碰上大雨，也很容易做出决策“暂不施工”，除非是第5天，即使遇到大雨也只能冒雨施工。

➢如果遇到小雨，则不能凭借主观做出决策。因为这个时候无法知道以后几天的天气情况，只能知道他们的概率分布。

工程施工问题

解题：

➢这是一个五阶段的一维随机动态变量的动态规划问题。对第 *i*（*i*=1,2,3,4,5)天碰上第 *j* 种天气情况是（*j*=1表示大雨，*j*=2表示小雨, *j*=3表示无雨），决策的标准是以当天施工的施工费和当天不施工延后施工的施工费相比较，取其中较小者为最优决策。

工程施工问题

- P_{ij} 表示第*i*天的天气情况
- $R(S_{ij})$ 表示与S_{ij} 对应的概率
- S_{ij} 表示第*i*天遇到第*j*种天气的施工费用
- $r(S_{ij})$ 表示第*i*天遇到第*j*种天气的最优施工费用

工程施工问题

➢第5天是规定的最后一天，如果前4天墙板混凝土尚未浇捣，则第5天无论遇到什么天气，其最优决策均为当天施工。

$$R_5(S_{5j})=r_5(S_{5j})$$

$$R_5(S_{51})=r_5(S_{51})=180000\text{元}，\ P_{51}=0.4$$

$$R_5(S_{52})=r_5(S_{52})=160000\text{元}，\ P_{52}=0.4$$

$$R_5(S_{53})=r_5(S_{53})=150000\text{元}，\ P_{53}=0.2$$

j=1表示大雨，*j*=2表示小雨，*j*=3表示无雨

➢应该用第4天的施工费 $r_4(S_{4j})$和第5天可能的施工费来比较。因为第5天遇到不同的天气有不同的施工费。可根据各种天气出现的概率 P_{5j} ,求出第5天的施工期望为：

$$\sum_{j=1}^{3}R_5(S_{5j})P_{5j}=$$
$$=R_5(S_{51})P_{51}+R_5(S_{52})P_{52}+R_5(S_{53})P_{53}$$
$$=180000\times0.4+160000\times0.4+150000\times0.2$$
$$=166000\text{元}$$

工程施工问题

➢第4天遇到第*j*种天气，最优决策的目标函数为：

$$R_4(S_{4j})=\min\{\ r_4(S_{4j}),\sum_{j=1}^{3}R_5(S_{5j})P_{5j}\}$$

$$\text{即}R_4(S_{41})=\min\{\ 180000,166000\}=166000\text{元}$$

由上式可推出第4天如果天降大雨，则不施工为好

$$R_4(S_{42})=\min\{160000,166000\}=160000\text{元}$$
$$R_4(S_{43})=\min\{150000,166000\}=150000\text{元}$$

由上式可推出第4天如果天降小雨或者无雨，则施工为好

工程施工问题

➢第3天遇到第*j*种天气，最优决策的目标函数为：

$$R_3(S_{3j})=\min\{\ r_3(S_{3j}),\sum_{j=1}^{3}R_4(S_{4j})P_{4j}\}$$

$$\text{其中}\sum_{j=1}^{3}R_4(S_{4j})P_{4j}=160400\text{元}$$

$$\text{得}R_3(S_{31})=\min\{\ 180000,160400\}=160400\text{元}$$

如果第3天天降大雨，则不施工为好

$$R_3(S_{32})=\min\{160000,160400\}=160000\text{元}$$

$$R_3(S_{33})=\min\{150000,166000\}=150000\text{元}$$

如果第3天天降小雨或者无雨，则施工为好

工程施工问题

➢第2天遇到第*j*种天气，最优决策的目标函数为：

$$R_2(S_{2j})=\min\{\ r_2(S_{2j}),\sum_{j=1}^{3}R_3(S_{3j})P_{3j}\}$$

$$\text{其中}\sum_{j=1}^{3}R_3(S_{3j})P_{3j}=154080\text{元}$$

$$R_2(S_{21})=\min\{\ 180000,154080\}=154080\text{元}$$
$$R_2(S_{21})=\min\{\ 160000,154080\}=154080\text{元}$$

如果第2天天降大雨或者天降小雨，都不施工为好

$$R_2(S_{23})=\min\{150000,154080\}=150000\text{元}$$

如果第2天无雨，则施工为好

工程施工问题

➢第1天遇到第*j*种天气，最优决策的目标函数为：

$$R_1(S_{1j})=\min\{\ r_1(S_{1j}),\sum_{j=1}^{3}R_2(S_{2j})P_{2j}\}$$

$$\text{其中}\sum_{j=1}^{3}R_2(S_{2j})P_{2j}=151632\text{元}$$

$$R_2(S_{21})=\min\{\ 180000,151632\}=151632\text{元}$$
$$R_2(S_{21})=\min\{\ 160000,151632\}=151632\text{元}$$

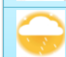
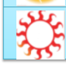
如果第1天天降大雨或者天降小雨，都不施工为好

$$R_2(S_{23})=\min\{150000,151632\}=151632\text{元}$$

如果第1天无雨，则施工为好

工程施工问题

根据以上的计算得出在这5天中浇捣箱基墙板混凝土的最优策略为：

	day1	day2	day3	day4	day5
	N	N	N	N	Y
	N	N	Y	Y	Y
	Y	Y	Y	Y	Y

4.10 信贷投资的应用

问题描述：银行倾向于投资信贷还款能力大的企业。

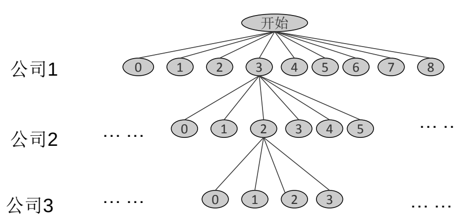
某银行准备将8000万资金用于投资给编号1、2、3三家公司。以下是银行投资金额与企业信贷还款能力的相关系数。

相关系数表

工厂	投资（单位：千万元）								
	0	1	2	3	4	5	6	7	8
1	0	0.5	1.5	4	8	9	9.5	9.8	10
2	0	0.5	1.5	4	6	7	7.3	7.4	7.5
3	0	0.4	2.6	4	4.5	5	5.1	5.2	5.3

在信贷投资的应用

问题分析：把所有可能性枚举出来，可以求得结果。



在信贷投资的应用

动态规划解题：

设$F(i, j)$表示到第i个企业时，投资了j千万的最大还贷系数($1\le i\le 3$, $0\le j\le 8$)。$F(3, 8)$为3个企业的最大还贷系数和。

Data(i,j) 表示投资i企业j千万时的还贷系数（即相关系数表）。k为在已知$F(i, j)$的状态下，可以投资第i+1个企业的可能取值

状态转移方程：

F(i,j+k)= Max{F(i-1,j+k) , F(i,j+k) , F(i-1,j)+Data(i,k)};

在信贷投资的应用

解题过程分析

初始状态：F(1,j) (0≤j≤8)为

	0	1	2	3	4	5	6	7	8
阶段1	0	0.5	1.5	4	8	9	9.5	9.8	10

在信贷投资的应用

状态重合了怎么办？
使用状态方程！

解题过程分析

状态变化：从F(1,j) 递推F(2,j)

	0	1	2	3	4	5	6	7	8
阶段1	0	0.5	1.5	4	8	9	9.5	9.8	10

阶段2	0	0.5	1.5	4	8	9	9.5	12	14

状态转移方程：

F(i,j+k)= Max{F(i-1,j+k) , F(i,j+k) , F(i-1,j)+Data(i,k)};

在信贷投资的应用

状态变化：从F(2,j) 递推F(3,j)

	0	1	2	3	4	5	6	7	8
阶段2	0	0.5	1.5	4	8	9	9.5	12	14
阶段3	0	0.5	2	4	8	9	10	12	14

状态转移方程：

F(i,j+k)= Max{F(i-1,j+k) , F(i,j+k) , F(i-1,j)+Data(i,k)};

在信贷投资的应用

结果：最大信贷还款能力F(3,8)=14

	0	1	2	3	4	5	6	7	8
阶段1	0	0.5	1.5	4	8	9	9.5	9.8	10
阶段2	0	0.5	1.5	4	8	9	9.5	12	14
阶段3	0	0.5	2	4	8	9	10	12	14

银行为确保总的信贷还款能力最大，应贷给3个企业分别4(千万)、4 (千万) 、0 (千万) 。

在信贷投资的应用

小结动态规划特点：

1、记忆化搜索

2、前一状态通过某种规则递推出下一状态。这个规则应该保证每个局部解都是最优的。

4.11生产存储问题

问题描述

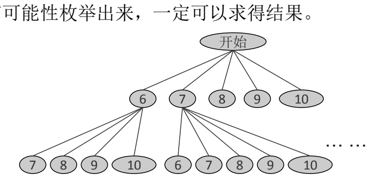
某厂根据合同，在一至四月份为客户生产某种机床。工厂每月的生产能力为10台，机床可以库存，存储费用为每台每月0.2万元，每月需要的数量及每台机床的生产成本如下表。

月份	1	2	3	4
需求（台）	6	7	12	6
生产成本（万元/台）	7	7.2	8	7.6

问每个月生产几台，才能使费用（成本和存储费用）最小？

生产存储问题

思路：把所有可能性枚举出来，一定可以求得结果。



1月份

2月份

3月份 … …

4月份 … …

生产存储问题

动态规划解题

设F(i , j) 表示当第 i 月份生产了 j 台机床时的最小费用，

F(4 , 31) 为总的最小费用。

库存 K(j)= j-sum(i),sum(i)为到第i月份的需求之和。

u表示第i月生产u台 (0≤u≤10)。

C(i)为第i月生产成本。

状态转移方程：

F(i+1,j+u)= Min{F(i+1,j+u)，F(i , j)+ K(j)+C(i+1)*u}

生产存储问题

1月到2月状态的变化

	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1月	42	49	56	63	70										
2月								92.4	99.6	106.8	114	121.2	128.4	135.6	142.8

状态转移方程：
F(i+1,j+u)= Min{F(i+1,j+u)，F(i , j)+ K(j)+C(i+1)*u};

生产存储问题

2月到3月状态的变化

	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
2月	92.4	99.6	106.8	114	121.2	128.4	135.6	142.8										
3月													184.2	192.2	200.2	208.2	216.2	224.2

状态转移方程：
F(i+1,j+u)= Min{F(i+1,j+u)，F(i , j)+ K(j)+C(i+1)*u};

生产存储问题

- 最后从3月推出4月，F(4,31)= 229.8(万元)
- 如何求每个月的生产量？
- 还要结合刚才的递推过程。
  plan(i , j) 表示生产j台时，第i月生产的数量。

生产存储问题

- **当F(i+1,j+u)> F(i , j)+ K(j)+C(i+1)*u时， plan(i+1,j+u)=u;**

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1月						6	7	8	9	10																					
2月													7	8	9	10	9	10	10	10											
3月																									5	6	7	8	9	10	
4月																															6

最优化安排是：
第1月至4月分别生产：10，10，5，6。
生产与存储最小费用为229.8万元。

4.12 最长上升子序列

1、问题描述

一个数的序列b_i，当$b_1 < b_2 < ... < b_S$ 的时候，我们称这个序列是上升的。对于给定的一个序列(a_1, a_2, ..., a_N)，我们可以得到一些上升的子序列(a_{i1}, a_{i2}, ..., a_{iK})，这里1 <= i1 < i2 < ... <iK <= N。比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7), (3, 4, 8)等等。这些子序列中最长的长度是4，比如子序列(1, 3, 5, 8).

你的任务，就是对于给定的序列，求出最长上升子序列的长度。

问题描述

输入数据

输入的第一行是序列的长度N (1 <= N <= 1000)。第二行给出序列中的N 个整数，这些整数的取值范围都在0 到10000。

输出要求

最长上升子序列的长度。

输入样例

7

1 7 3 5 9 4 8

输出样例

4

2、解题思路

■ 如何把这个问题分解成子问题呢？经过分析，发现 “求以ak（k=1, 2, 3…N）为终点的最长上升子序列的长度”是个好的子问题——这里把一个上升子序列中最右边的那个数，称为该子序列的“终点”。虽然这个子问题和原问题形式上并不完全一样，但是只要这N 个子问题都解决了，那么这N 个子问题的解中，最大的那个就是整个问题的解。

■ 由上所述的子问题只和一个变量相关，就是数字的位置。因此序列中数的位置k 就是“状态”，而状态 k 对应的“值”，就是以ak 做为“终点”的最长上升子序列的长度。这个问题的状态一共有N 个。状态定义出来后，转移方程就不难想了。

■假定MaxLen (k)表示以a_k 做为“终点”的最长上升子序列的长度，那么：

■MaxLen (1) = 1

■MaxLen (k) = Max { MaxLen (i)：1<i < k 且 a_i < a_k 且 k≠1 } + 1

■这个状态转移方程的意思就是，MaxLen(k)的值，就是在a_k左边，“终点”数值小于a_k，且长度最大的那个上升子序列的长度再加1。因为a_k 左边任何“终点”小于a_k 的子序列，加上a_k 后就能形成一个更长的上升子序列。

■实际实现的时候，可以不必编写递归函数，因为从MaxLen(1)就能推算出MaxLen(2)，有了MaxLen(1)和MaxLen(2)就能推算出MaxLen(3)……

4.13 Two Ends

1、问题描述

Two Ends游戏：数量为偶数的卡片被排成一行，每一张卡片上写有一个正数。玩家轮流抽取一张卡片，但是只能从这行卡片的最左边或最右边抽。到最后，卡片的点数加起来多的玩家赢。

如果第二名玩家一直采用贪心策略（即优先选择数字较大的卡片，如果相同则选择最左边的），求出第一名玩家与第二名玩家之间的点数差最多是多少。

例如：3 2 10 4

问题描述

输入数据

每行的第一个数字表示卡片的总数n，之后的n个数字安顺序表示卡片上的数字。

输出要求

输出两个玩家最大的点数差。

问题描述

输入样例

4 3 2 10 4
8 1 2 3 4 5 6 7 8
8 2 2 1 5 3 8 7 3
0

输出样例

In game 1, the greedy strategy might lose by as many as 7 points.
In game 2, the greedy strategy might lose by as many as 4 points.
In game 3, the greedy strategy might lose by as many as 5 points.

2、解题思路

由于第二名玩家的选择策略固定为贪心策略，故只需确定第一名玩家的策略即可。

采用策略：

✓ 贪心策略

✓ 动态规划

✓ 采用贪心策略不可行

例如：卡片序列为3　2 10 4

如果第一名玩家采用贪心策略：

第一名玩家先选4，第二名玩家会选择10，第一名玩家再选择3，第二名玩家只能选择2，点数差为4+3-10-2=-5

如果第一名玩家不采用贪心策略：

第一名玩家先选3，第二名玩家会选择4，第一名玩家再选择10，第二名玩家只能选择2，点数差为3+10-4-2=7

✓ 采用动态规划

由于贪心策略只着眼于当前，没有考虑以后的情况，故此题不适用。

题目满足最优化原理和无后效性，故可采用动态规划。

划分阶段：

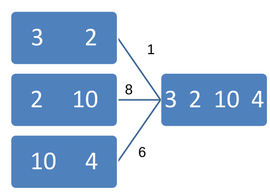
可按照每一轮来划分阶段。由于每轮双方各取一张卡片，卡片序列总数每次减少2，卡片的总数又为偶数，因此最后而且最简单的情况是只剩两张卡片，这时选择的策略确定，即肯定取数字大的。由这最简单的情况出发，每次加上两张卡片，即可递推出结果。

状态转移：

第一名玩家每次只能选择最左或最右边的卡片，由于第二名玩家采用贪心策略，只需比较第一名玩家取走卡片后的最左与最右边卡片的大小，即可确定第二名玩家要取的卡片，从而确定该阶段的点数差。

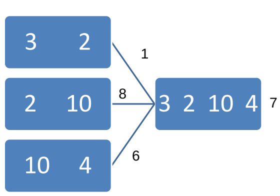
因此比较第一名玩家取最左边和最右边卡片时的点数差，选择点数差大的。

求解过程：对于序列3　2 10 4，可分为两个阶段



阶段2：序列可能为3　2，2　10或10　4
由于只有两张卡，故第一名玩家肯定要选择数字较大的，因此对应的点数差为：3-2=1，10-2=8，10-4=6。

求解过程：对于序列3　2 10 4，可分为两个阶段



阶段1：序列为3　2　10　4
第一位玩家如果选择3，由于4>2，因此第二名玩家选择4，剩下序列2　10可由前面得出，故点数差为3-4+8=7。
第一名玩家如果选择4，由于10>3，因此第二名玩家选择10，剩下序列3　2可由前面得出，故点数差为4-10+1=-5。
由于7>-5，故得到的最大点数差为7。

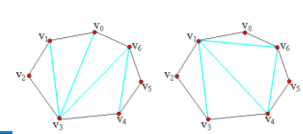
4.14凸多边形最优三角剖分

•用多边形顶点的逆时针序列表示凸多边形，即$P=\{v_0,v_1,\ldots,v_{n-1}\}$表示具有$n$条边的凸多边形。

•若v_i与v_j是多边形上不相邻的2个顶点，则线段v_iv_j称为多边形的一条弦。弦将多边形分割成2个多边形$\{v_i,v_{i+1},\ldots,v_j\}$和$\{v_j,v_{j+1},\ldots v_i\}$。

•**多边形的三角剖分**是将多边形分割成互不相交的三角形的弦的集合T。

•给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得即该三角剖分中诸三角形上权之和为最小。



最优子结构性质

- 凸多边形的最优三角剖分问题有最优子结构性质。
- 事实上，若凸$(n+1)$边形$P=\{v_0,v_1,\ldots,v_{n-1}\}$的最优三角剖分$T$包含三角形$v_0v_kv_n$，$1\le k\le n-1$，则$T$的权为3个部分权的和：三角形$v_0v_kv_n$的权，子多边形$\{v_0,v_1,\ldots,v_k\}$和$\{v_k,v_{k+1},\ldots,v_n\}$的权之和。可以断言，由$T$所确定的这2个子多边形的三角剖分也是最优的。因为若有$\{v_0,v_1,\ldots,v_k\}$或$\{v_k,v_{k+1},\ldots,v_n\}$的更小权的三角剖分将导致$T$不是最优三角剖分的矛盾。

最优三角剖分的递归结构

- 定义 $t[i][j]$，$1\le i<j\le n$为凸子多边形$\{v_{i-1},v_i,\ldots,v_j\}$的最优三角剖分所对应的权函数值，即其最优值。为方便起见，设退化的多边形$\{v_{i-1},v_i\}$具有权值0。据此定义，要计算的凸$(n+1)$边形P的最优权值为$t[1][n]$。
- $t[i][j]$的值可以利用最优子结构性质递归地计算。当$j-i\ge 1$时，凸子多边形至少有3个顶点。由最优子结构性质，$t[i][j]$的值应为$t[i][k]$的值加上$t[k+1][j]$的值，再加上三角形$v_{i-1}v_kv_j$的权值，其中$i\le k\le j-1$。由于在计算时还不知道k的确切位置，而k的所有可能位置只有$j-i$个，因此可以在这$j-i$个位置中选出使$t[i][j]$值达到最小的位置。由此，$t[i][j]$可递归地定义为：

$$t[i][j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t[i][k]+t[k+1][j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$