3.4 最大子段和

问题描述

给定由n个整数（其中可能有负数）组成的序列 $a_1,a_2,\cdots,a_n$，求该序列形如 $\sum_{k=i}^{j} a_k$ 的子段和的最大值。当所有整数均为负整数时定义其最大子段和为0。依此定义，所求的最优值为：

$$\max\{0, \max_{1\le i\le j\le n} \sum_{k=i}^{j} a_k\}$$

例如：当 $(a_1,a_2,\cdots,a_6)$ ＝（-2，$\underline{11，-4，13}$，-5，-2）时，最大子段和为：$\sum_{k=2}^{4} a_k = 20$

3.4 最大子段和

- 例：{5, -3, 4, 2}的最大子序列 {5, -3, 4, 2}，它的和是8, 达到最大；
- 例：{5, -6, 4, 2}的最大子序列是{4, 2}，它的和是6。

3.4 最大子段和

方法一、简单算法（顺序求和+比较=几个for循环）

1）用三个for循环来完成，如书 P52程序，所需时间是$O(n^3)$。

2）可以从算法的设计技巧上改进，改进后可以省去最后一个for循环，避免重复计算，从而使算法得到改进 $O(n^2)$。如书 P53 上程序所示。

3.4 最大子段和

- 方法一：简单算法 假设{a1,a2,a3,a4,a5,a6}

a1 a1,a2 a1,a2,a3 a1,a2,a3,a4 a1,a2,a3,a4,a5 a1,a2,a3,a4,a5,a6	a2 a2,a3 a2,a3,a4 a2,a3,a4,a5 a2,a3,a4,a5,a6	a3 a3,a4 a3,a4,a5 a3,a4,a5,a6
a4 a4,a5 a4,a5,a6	a5 a5,a6	a6

3.4 最大子段和

➢方法一：最大子段和的简单算法

```
int  maxsum(int n, int *a, int & besti,  int & bestj)
{  intsum=0;
  for (int i=1; i<=n; i++)
   for (int j=i; j<=n; j++)
    { int thissum=0;
      for (int k=i; k<=j; k++)  thissum+=a[k];
      if  (thissum>sum)
       { sum=thissum;  besti=i;   bestj=j;}
     }
   return  sum;
 }
```

$0(n^3)$

3.4 最大子段和

a1
a1,a2
a1,a2,a3
a1,a2,a3,a4
a1,a2,a3,a4,a5
a1,a2,a3,a4,a5,a6

$$\sum_{k=i}^{j} a^k = a_j + \sum_{k=i}^{j-1} a^k$$

3.4 最大子段和

➢方法一：最大子段和的改进算法

```
int  maxsum(int n, int *a, int & besti,  int & bestj)
{  intsum=0;
   for (int i=1; i<=n; i++)
    {  int thissum=0;
       for (int j=i; j<=n; j++)
        {   thissum+=a[k];
            if  (thissum>sum)
             { sum=thissum;  besti=i;   bestj=j;}
        } }
    return  sum;
 }
```

0 (n^2)

3.4 最大子段和

方法二、分治算法

如果将所给的序列a[1:n]分为长度相等的两段a[1:n/2]和a[n/2+1:n]，分别求出这两段的最大子段和，然后再合并。则a[1:n]的最大子段和有如下三种情形：

（1）a[1:n]的最大子段和与a[1:n/2]的最大子段和相同。

（2）a[1:n]的最大子段和与a[n/2+1:n]的最大子段和相同。

（3）a[1:n]的最大子段和为 $\sum_{k=i}^{j} a_k$，

且 $1 \le i \le n/2$，$n/2+1 \le j \le n$ 。

3.4 最大子段和

- 将a[1:n]分为长度相等的两段a[1:n/2]和a[n/2+1:n]，分别对两区段求最大子段和，则a[1:n]的最大子段和有三种情形：

① a[1:n]的最大子段和与a[1:n/2]的最大子段和相同

② a[1:n]的最大子段和与a[n/2:n]的最大子段和相同

③ a[1:n]的最大子段和为 $\sum_{k=i}^{j} a_k, 1 \le i \le n/2, n/2+1 \le j \le n$

- 对①②可递归求解
- 对③可知a[n/2]、a[n/2+1]一定在最大和子段中
  - 在a[1:n/2]中计算，$s1 = \max_{1 \le i \le n/2} \sum_{k=i}^{n/2} a[k]$
  - 在a[n/2+1:n]中计算，$s2 = \max_{n/2+1 \le i \le n} \sum_{k=n/2+1}^{n} a[k]$
  - s1+s2是该情况下的最大值。

3.4 最大子段和

方法二、分治算法

其中（1）和（2）这两种情形可以递归求得。

对于情形（3），容易看出a[n/2]和a[n/2+1]这两个元素在最优子序列中。因此，我们在a[1:n/2]中计算 $s1=\max_{1\le i\le n/2}\sum_{k=i}^{n/2}a_k$，并在a[n/2+1:n]中计算 $s2=\max_{n/2+1\le i\le n}\sum_{k=n/2+1}^{i}a_k$。

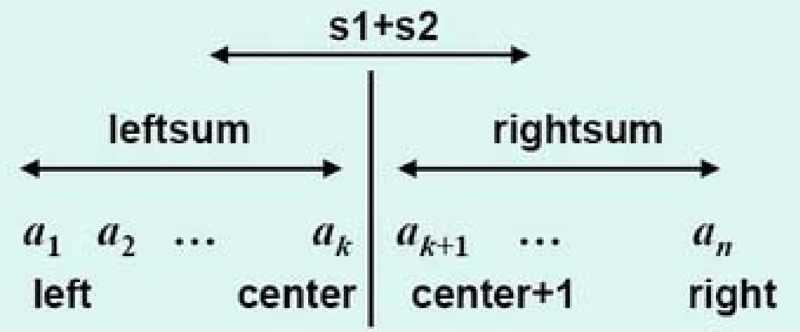
则s1＋s2即为出现情形（3）的最优值。

递归计算左段最大子段和 *leftsum*

递归计算右段最大子段和 *rightsum*

$a_{center}\to a_1$的最大和s1，$a_{center+1}\to a_n$的最大和s2

max { *leftsum*, *rightsum*, s1+s2}



方法二、分治算法

算法 MaxSubSum(*A*, *left*, *right*)

1. If $|A|=1$，则输出元素值（当值为负时输出0）
2. $center \leftarrow (left+right)/2$;
3. $leftsum \leftarrow$ MaxSubSum(A,$left$,$center$)　　//左和
4. $righsum \leftarrow$ MaxSubSum(A,$center+1$,$right$)　　//右和
5. $s1 \leftarrow A1[center]$　　//从*center*向左的最大和
6. $s2 \leftarrow A2[center+1]$　　//从*center*+1向右的最大和
7. $sum \leftarrow s1+s2$
8. if $leftsum > sum$ then $sum \leftarrow leftsum$
9. if $rightsum > sum$ then $sum \leftarrow rightsum$

复杂度分析

$$T(n)=\begin{cases} O(1) & n \le c \\ 2T(n/2)+O(n) & n > c \end{cases}$$

T(n)=O(nlogn)

3.4 最大子段和

方法二、分治算法

```
int  maxsubsum(int  *a,  int left,  int right)
{  intsum=0;
   If (left==right)  sum=a[left]>0?a[left]:0;
   else
   { int  center=(left+right)/2;
     int leftsum=maxsunsum(a, left, center);
     int rightsum= maxsunsum(a, center+1, right);

    int s1=0;   int lefts=0;
    for (int i=center; i>=left;  i--)
       { left+=a[i];    if (left>s1)  s1=left;  }
   int s2=0;   int rights=0;
    for (int i=center+1; i<=right;  i++)
       {rights+=a[i];    if (rights>s2)  s2=rights;  }
   sum=s1+s2;
   if (sum<leftsum)  sum=leftsum;
   if (sum<rightsum)  sum=rightsum;}
return  sum;}
```

复杂度分析

$$T(n)=\begin{cases} O(1) & n\le c \\ 2T(n/2)+O(n) & n>c \end{cases}$$

T(n)=O(nlogn)

方法三、动态规划

记 $b[j]=\max\limits_{1\le i\le j}\{\sum\limits_{k=i}^{j}a_k\}$，$1\le i\le j\le n$ ，b[j]表示最后一项为a[j]的序列构成的最大的子段和（从j位置往左看，aj在最大子段中），所获得的最大子段和。

例如：当（$a_1,a_2,\cdots,a_6$）=（－2，11，－4，13，－5，－2）时，

$b[1]=0$ $b[4]=20$

$b[2]=11$ $b[5]=15$

$b[3]=7$ $b[6]=13$

则$\max\limits_{1\le j\le n} b[j]$即为对于n个整数序列的最大子段和问题的所求。即：

$$\max_{1\le i\le j\le n}\sum_{k=i}^{j}a_k=\max_{1\le j\le n}\{\max_{1\le i\le j}\sum_{k=i}^{j}a_k\}=\max_{1\le j\le n}b[j]$$

3.4 最大子段和

方法三、动态规划

$$b[j]=\max_{1\le i\le j}\{\sum_{k=i}^{j}a_k\},\quad 1\le i\le j\le n$$

- 根据b[j]的定义，可以看出
  - 当b[j-1]>0时，无论a[j]为何值，b[j]=b[j-1]+a[j]
  - 当b[j-1]<0时，无论a[j]为何值，b[j]=a[j]

$$b[j]=\begin{cases}\max\{a_1,0\}, & j=1\\ \max\{b[j-1]+a_j,a_j\}, & j>1\end{cases}$$

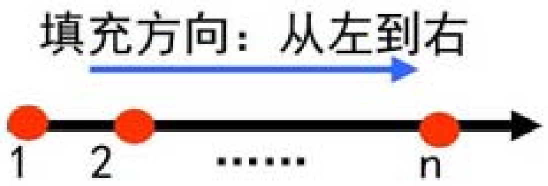
3.4 最大子段和

方法三、动态规划

b[j]的动态规划递归式如下：

$$b[j]=\begin{cases}\max\{\mathbf{0},\ a[\mathbf{1}]\} & j=\mathbf{1}\\ \max\{b[j-\mathbf{1}]+a[j],\ a[j]\} & \mathbf{1}<j\le n\end{cases}$$

此时，b[j]的解空间填充图如下：



最优解为b[1], b[2], .., b[n]中的最大值

3.4 最大子段和

方法三、动态规划

```
int maxsum(int  n, int *a)
 { intsum=0, b=0;
   for (int i=1;  i<=n;  i++)
    { if (b>0) b+=a[i];
       else b=a[i];
       if(b>sum)  sum=b;}
    return  sum;                    O(n)
}
```

3.4 最大子段和

方法三、动态规划

$$b[j]=\begin{cases}\max\{0,\ a[1]\} & j=1\\ \max\{b[j-1]+a[j],\ a[j]\} & 1<j\le n\end{cases}$$

例：当（$a_1,a_2,\cdots,a_6$）＝（-2，11，－4，13，－5，－2）时，如何填充b数组，（$b[j]=\max\{b[j-1]+a[j],\ a[j]\}\quad 1\le j\le n$）

解：　b[1] = max{0, a[1]}={0, -2}=0

b[2] = max{b[1]+a[2], a[2]}={0+11, 11}=11

b[3] = max{b[2]+a[3], a[3]}={11-4, -4}=7

b[4] = max{b[3]+a[4], a[4]}={7+13, 13}=20

b[5] = max{b[4]+a[5], a[5]}={20-5, -5}=15

b[6] = max{b[5]+a[6], a[6]}={15-2, -2}=13

那么，最大子段和为：$\max\limits_{1\le i\le j\le n}\sum_{k=i}^{j}a_k\ (=\sum_{k=2}^{4}a_k)=\max\limits_{1\le j\le n}b[j]=\mathbf{20}$

3.4 最大子段和

方法三、动态规划

	a1	a2	a3	a4	a5	a6
a 数组	-2	11	-4	13	-5	-2

	j=1	2	3	4	5	6
b 数组	0	11	7	20	15	13

最大子段和： $\max\limits_{1\le j\le n} b[j]$

显然，这个算法仅对原数组做了一次扫描，即计算时间 $O(n)$ ，借助的空间 $O(n)$ 。 ——“高效”的算法

小结——其他应用领域

生产与存储问题

某工厂每月需供应市场一定数量的产品。供应需求所剩余产品应存入仓库，一般地说，某月适当增加产量可降低生产成本，但超产部分存入仓库会增加库存费用，要确定一个每月的生产计划，在满足需求条件下，使一年的生产与存储费用之和最小。

小结——其他应用领域

投资决策问题

某公司现有资金Q亿元，在今后5年内考虑给A、B、C、D四个项目投资，这些项目的投资期限、回报率均不相同，问应如何确定这些项目每年的投资额，使到第五年末拥有资金的本利总额最大。

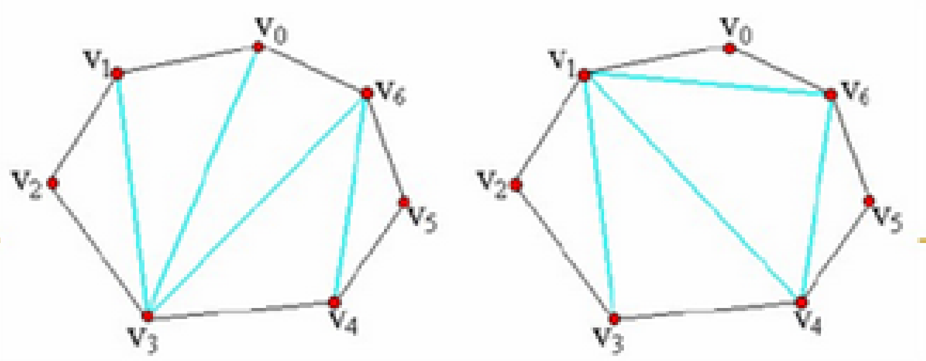
小结——其他应用领域

设备更新问题

企业使用设备都要考虑设备的更新问题，因为设备越陈旧所需的维修费用越多，但购买新设备则要一次性支出较大的费用。现在某企业要决定一台设备未来8年的更新计划，已预测到第j年购买设备的价格为K_j，G_j为设备经过j年后的残值，C_j为设备连续使用j-1年后在第j年的维修费用(j=1,2…8)，问应在哪年更新设备可使总费用最小。

3.5 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$P=\{v_0,v_1,\dots,v_{n-1}\}$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$\{v_i,v_{i+1},\dots,v_j\}$和$\{v_j,v_{j+1},\dots v_i\}$。
- **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。



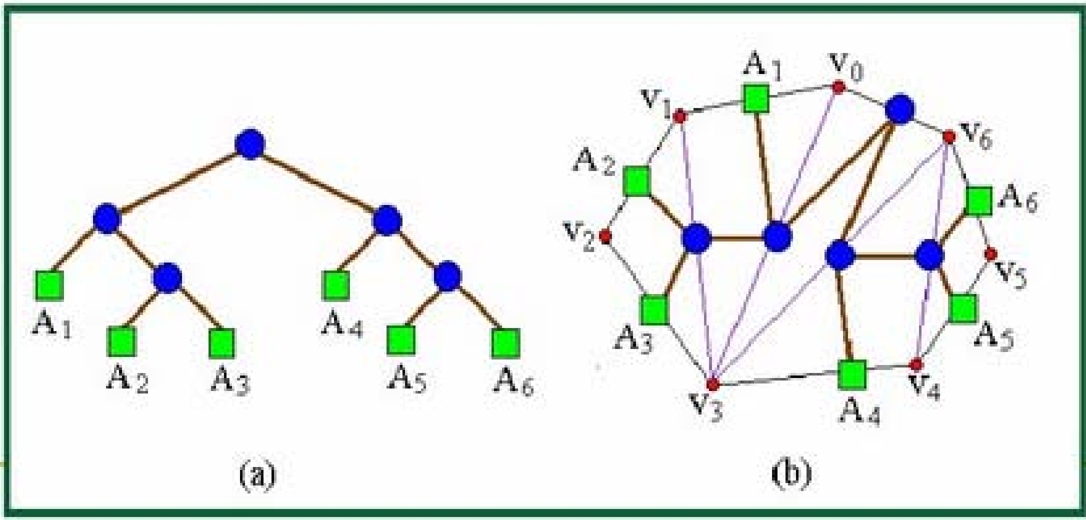
3.5 凸多边形最优三角剖分

三角剖分的结构及其相关问题

凸多边形的**三角剖分**与完全加括号的**矩阵连乘**有密切关系。

如何将它们**联系**起来?

—— 这两个问题的相关性可以从它们所对应的完全二叉树的**同构性**看出。



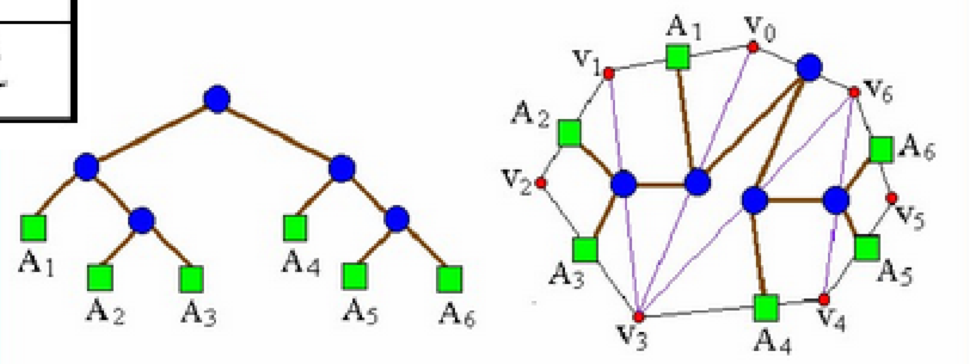
3.5 凸多边形最优三角剖分

三角剖分与矩阵连乘积同构

- 三角剖分问题和矩阵连乘积问题都对应于一个完全二叉树，也称为表达式的语法树。
- n个矩阵连乘积计算顺序同构于n个叶子的完全二叉树，凸(n+1)边形三角剖分同构于n个叶子的完全二叉树，所以n个矩阵连乘积的计算顺序问题同构于凸(n+1)边形的三角剖分问题。
- 事实上，矩阵连乘积最优计算顺序问题相当于是凸多边形最优三角剖分问题中一种特殊定义的权函数的情形。

最优三角剖分

语法树	凸多边形
根节点	边v_0v_6
内节点	三角剖分的弦
叶子节点	除v_0v_6外的各边

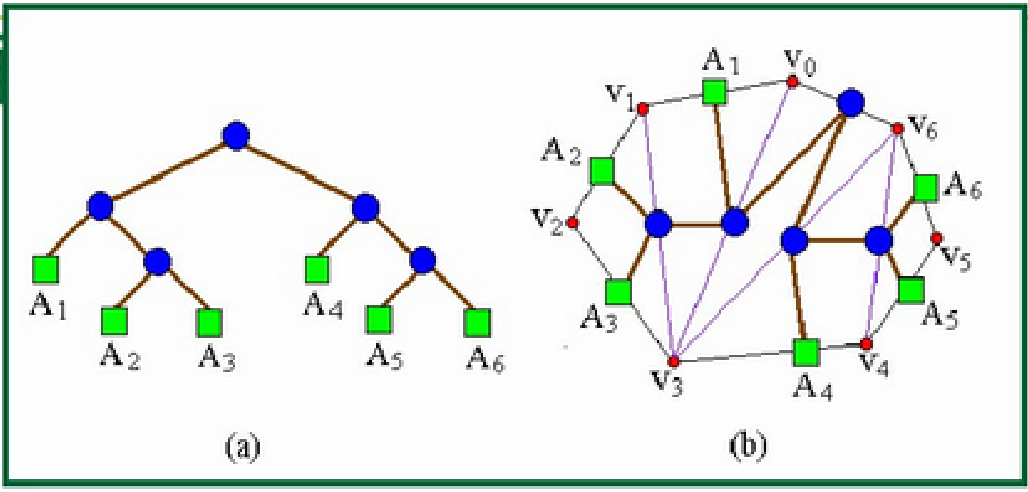


(a) (b)

- 一个表达式的完全加括号方式相应于一棵完全二叉树，称为表达式的语法树。例如，完全加括号的矩阵连乘积$((A_1(A_2A_3))(A_4(A_5A_6)))$所相应的语法树如图 (a)所示。
- 凸多边形$\{v_0,v_1,\ldots v_{n-1}\}$的三角剖分也可以用语法树表示。例如，图 (b)中凸多边形的三角剖分可用图 (a)所示的语法树表示。
- 矩阵连乘积中的每个矩阵A_i对应于凸(n+1)边形中的一条边$v_{i-1}v_i$。三角剖分中的一条弦v_iv_j，$i<j$，对应于矩阵连乘积$A[i+1:j]$。

3.5 凸多边形最优三角剖分

三角



- 凸n边形的三角剖分对应于一棵有n-1个叶结点的语法树。反之亦然。即凸n边形的三角剖分与有n-1个叶结点的语法树之间存在一一对应关系。
- n个矩阵的完全加括号乘积与n个叶结点的语法树之间存在一一对应关系。
- n个矩阵的完全加括号乘积与凸(n+1)边形的三角剖分之间存在一一对应关系。

3.5 凸多边形最优三角剖分

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$P=\{v_0,v_1,\dots,v_n\}$的最优三角剖分T包含三角形$v_0v_kv_n$，$1\leqslant k\leqslant n-1$，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$\{v_0,v_1,\dots,v_k\}$和$\{v_k,v_{k+1},\dots,v_n\}$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$\{v_0,v_1,\dots,v_k\}$或$\{v_k,v_{k+1},\dots,v_n\}$的更小权的三角剖分将导致T不是最优三角剖分的矛盾。

3.5 凸多边形最优三角剖分

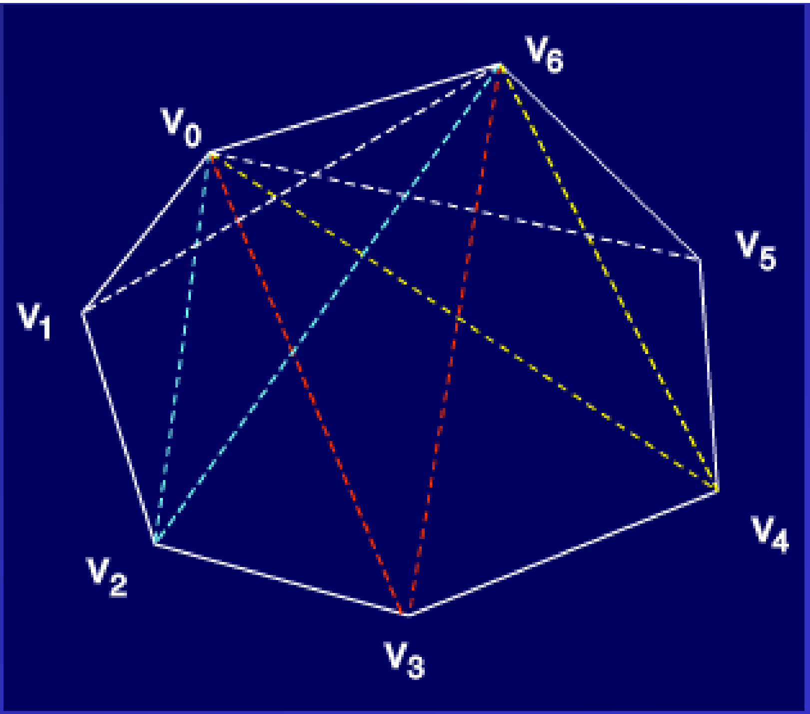
最优三角剖分的递归结构

- 定义t[i][j]，1≤i<j≤n为凸子多边形{vi-1,vi,…,vj}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{vi-1,vi}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≥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≤k≤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:

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

3.5 凸多边形最优三角剖分

- 可定义三角形上各种各样的权函数w。
- 例如$w(v_iv_jv_k) = |v_iv_j| + |v_jv_k| + |v_kv_i|$，其中$|v_iv_j|$是点$v_i$到$v_j$的欧氏距离。相应于此权函数的最优三角剖分即为最小弦长三角剖分。

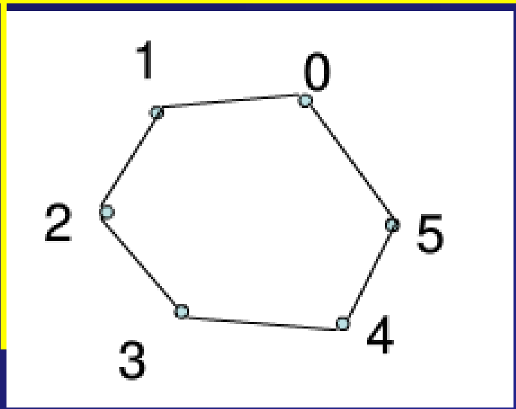


3.5 凸多边形最优三角剖分

最优三角剖分的算法

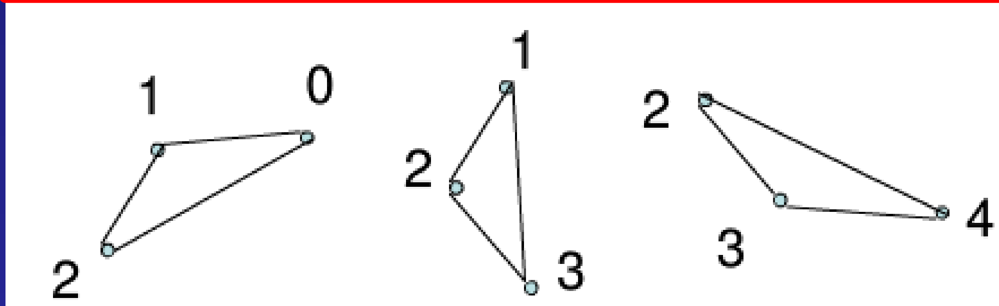
```
Void MinWeightTriangulation(int n, Type **t, int **s)
{  for (int i = 1; i <= n; i++) t[i][i] = 0;
   for (int r = 2; r <= n; r++)
     for (int i = 1; i <= n – r +1; i++) {
       int j = i + r – 1;
       t[i][j] = t[i+1][j] + w(i–1, i, j);
       s[i][j] = i;
       for (int k = i + 1; k < j; k++) {
         int u = t[i][k] + t[k+1][j] + w(i–1, k, j);
         if (u < t[i][j]) {t[i][j] = u; s[i][j] = k;}
}}} //程序中红色的部分为改动的地方
```

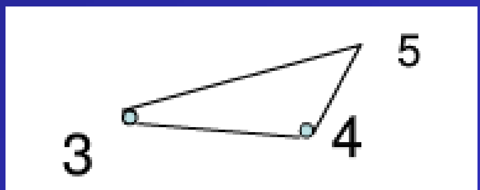
3.5 凸多边形最优三角剖分---练习题



	0	1	2	3	4	5
0	0	2	2	3	1	4
1		0	1	5	2	3
2			0	2	1	4
3				0	6	2
4					0	1
5						0

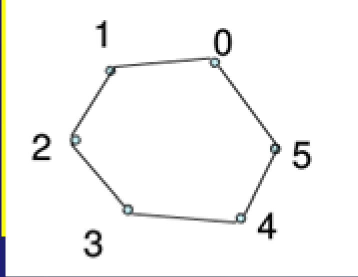
T[i][j]	1	2	3	4	5
1	0	5			
2		0	8		
3			0	9	
4				0	9
5					0





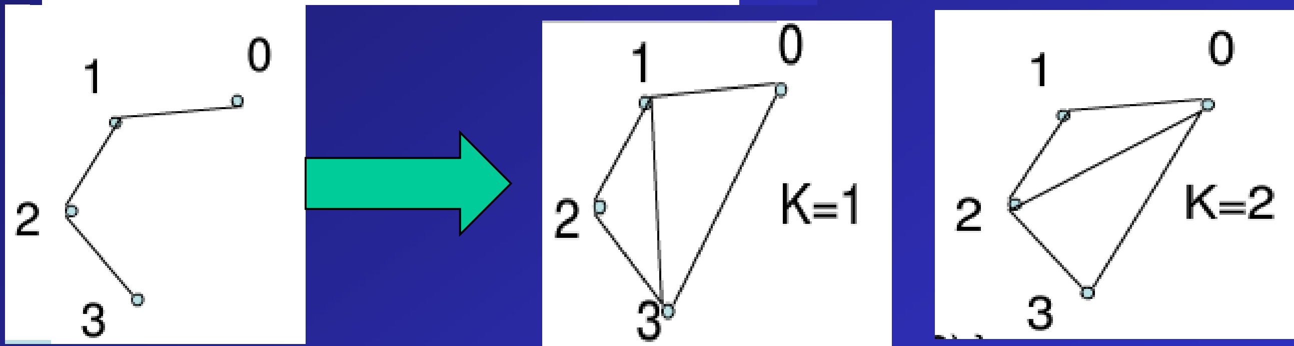
t[1][2] = 2+1+2 = 5
t[2][3] = 1+2+5 = 8
t[3][4] = 2+6+1 = 9
t[4][5] = 6+1+2 = 9

3.5 凸多边形最优三角剖分---练习题



	0	1	2	3	4	5
0	0	2	2	3	1	4
1		0	1	5	2	3
2			0	2	1	4
3				0	6	2
4					0	1
5						0

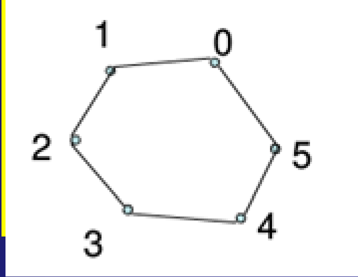
T[i][j]	1	2	3	4	5
1	0	5	12		
2		0	8		
3			0	9	
4				0	9
5					0



$$t[1][3] = \min(t[1][1] + t[2][3] + (2+5+3),\ t[1][2] + t[3][3]+(2+2+3))$$
$$= \min(0+8+10,\ 5+0+7)$$
$$= 12$$

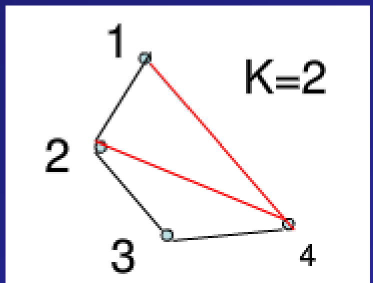
k=2

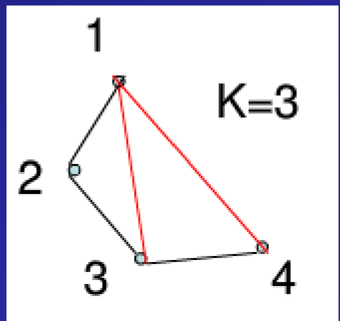
3.5 凸多边形最优三角剖分---练习题



	0	1	2	3	4	5
0	0	2	2	3	1	4
1		0	1	5	2	3
2			0	2	1	4
3				0	6	2
4					0	1
5						0

T[i][j]	1	2	3	4	5
1	0	5	12		
2		0	8	13	
3			0	9	
4				0	9
5					0





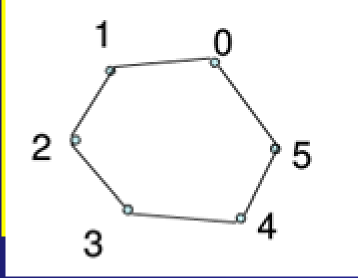
$t[2][4] = \min(\ t[2][2]+ t[3][4]+(1+1+2)\ ,\ t[2][3] + t[4][4] + (5+6+2)\ )$

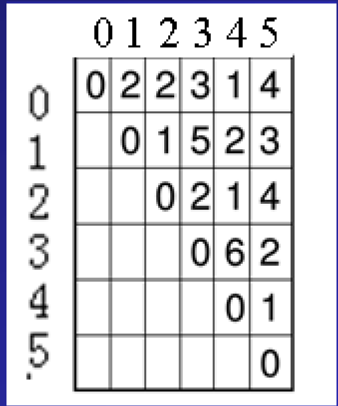
$= \min(0+9+4,\ 8+0+13)$

$= 13$

$k=2$

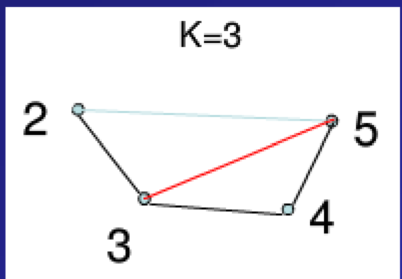
3.5 凸多边形最优三角剖分---练习题

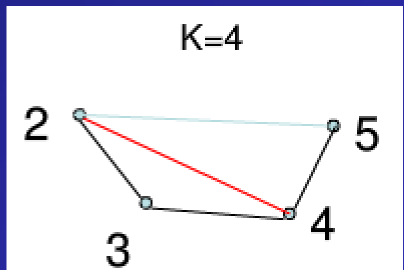




	0	1	2	3	4	5
0	0	2	2	3	1	4
1		0	1	5	2	3
2			0	2	1	4
3				0	6	2
4					0	1
5						0

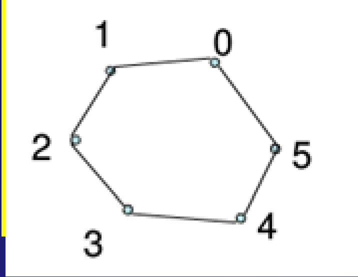
T[i][j]	1	2	3	4	5
1	0	5	12		
2		0	8	13	
3			0	9	15
4				0	9
5					0



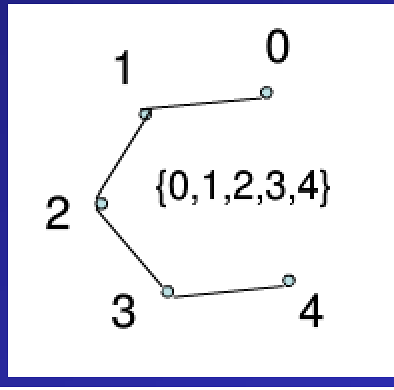


$$t[3][5] = \min(\ t[3][3] + t[4][5]+(2+4+2)\ ,\ t[3][4] + t[5][5] + (4+1+1)\ )$$
$$= \min(0+9+8,\ 9+0+6)$$
$$= 15$$
$$k=4$$

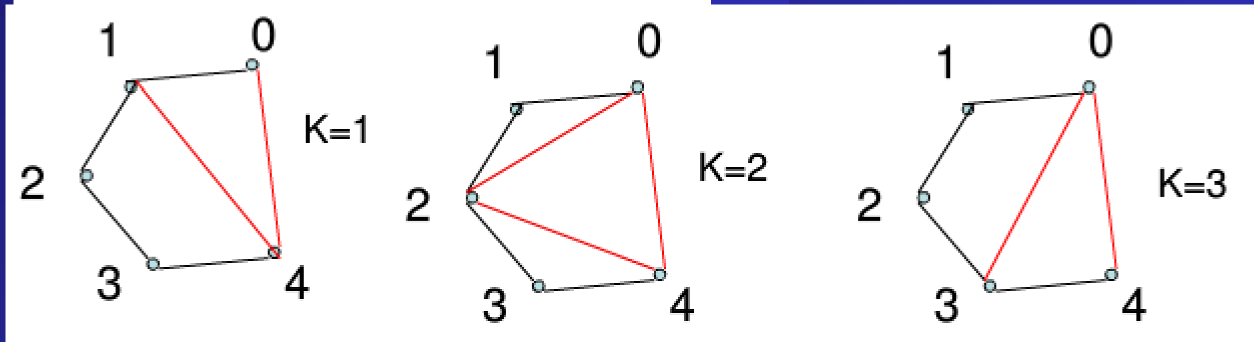
3.5 凸多边形最优三角剖分---练习题



	0	1	2	3	4	5
0	0	2	2	3	1	4
1		0	1	5	2	3
2			0	2	1	4
3				0	6	2
4					0	1
5						0



T[i][j]	1	2	3	4	5
1	0	5	12		
2		0	8	13	
3			0	9	15
4				0	9
5					0



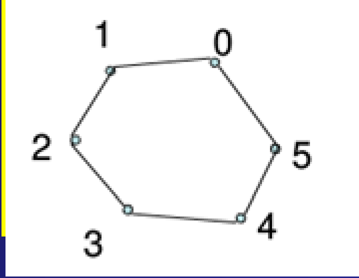
t[1][4] =

min(t[1][1] + [2][4]+(2+2+1) , t[1][2] + t[3][4] + (2+1+1), t[1][3] + t[4][4] + (3+6+1))

……………………………

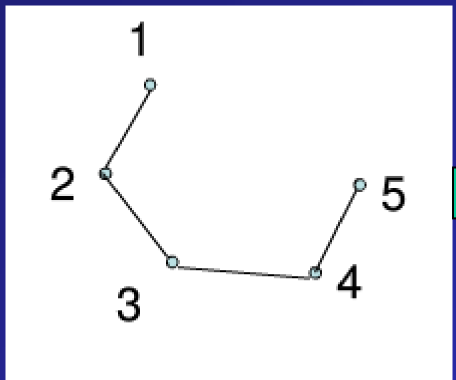
(以下计算略)

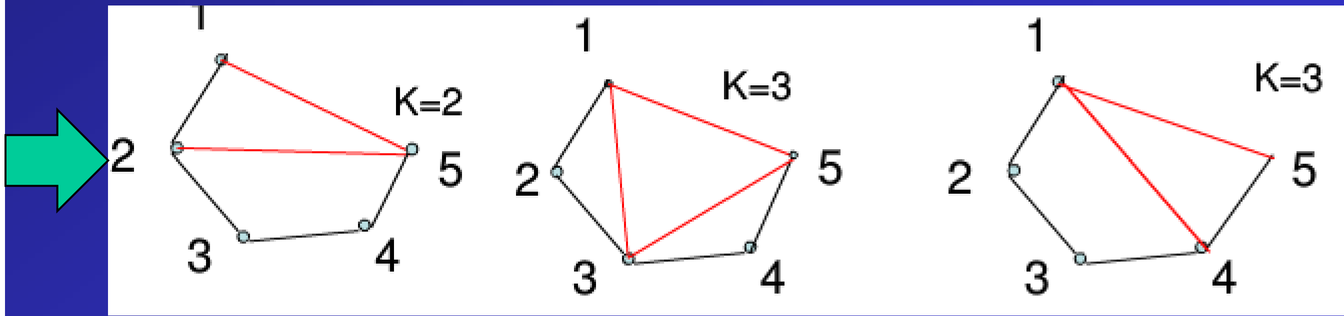
3.5 凸多边形最优三角剖分---练习题



	0	1	2	3	4	5
0	0	2	2	3	1	4
1		0	1	5	2	3
2			0	2	1	4
3				0	6	2
4					0	1
5						0

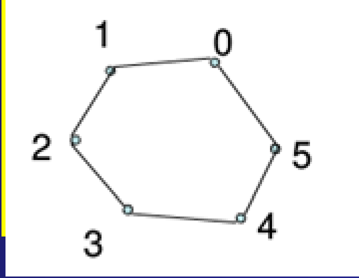
T[i][j]	1	2	3	4	5
1	0	5			
2		0	8		
3			0	9	
4				0	9
5					0



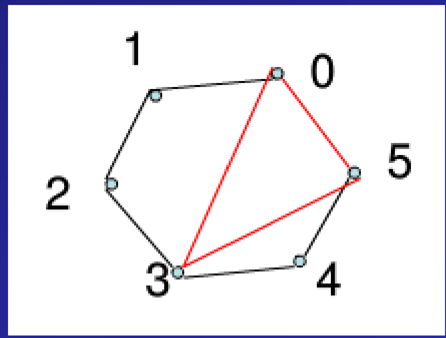


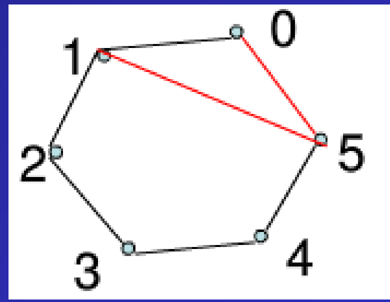
t[2][5] =。。。。。。
(计算略)

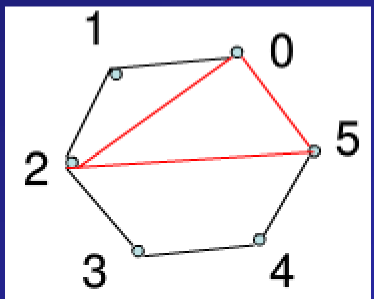
3.5 凸多边形最优三角剖分---练习题

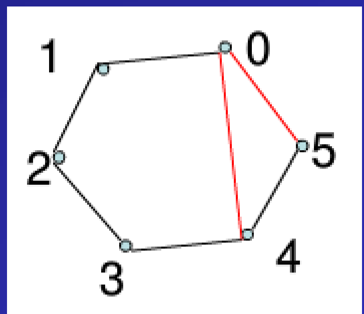


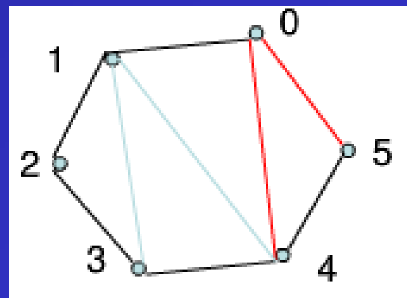
	0	1	2	3	4	5
0	0	2	2	3	1	4
1		0	1	5	2	3
2			0	2	1	4
3				0	6	2
4					0	1
5						0











t[1][5] =。。。。。。
(计算略)