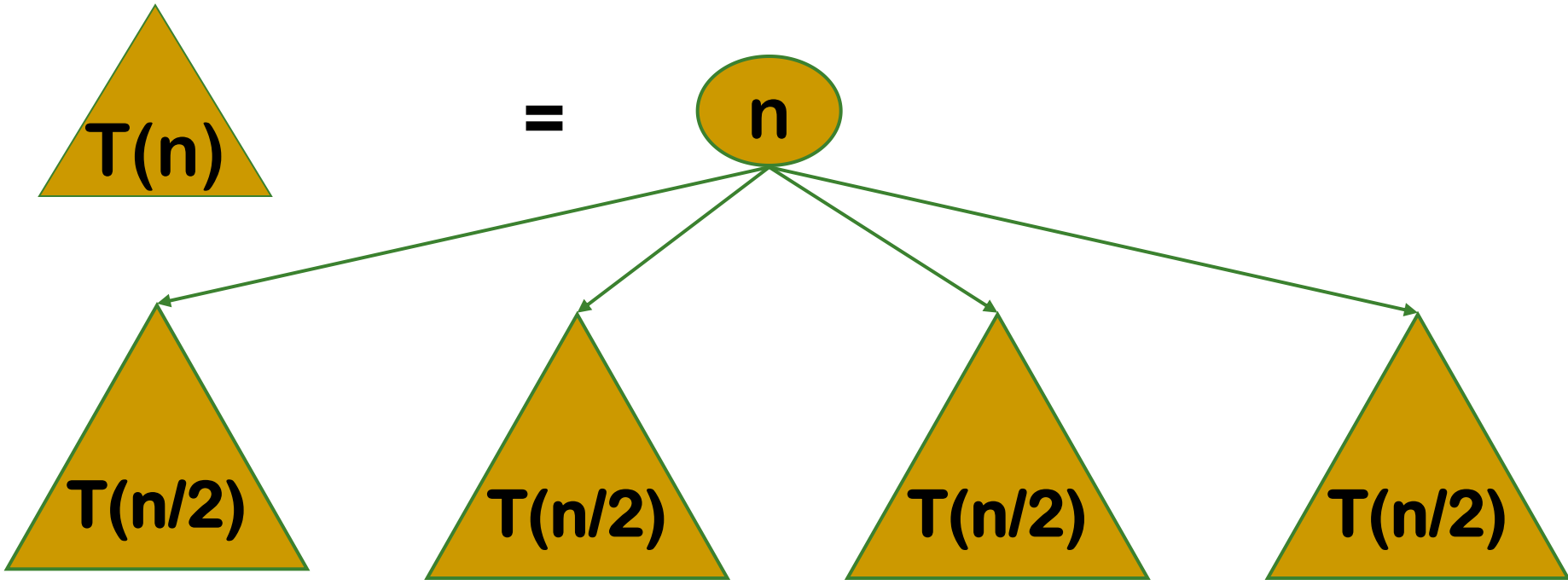
第3章　动态规划

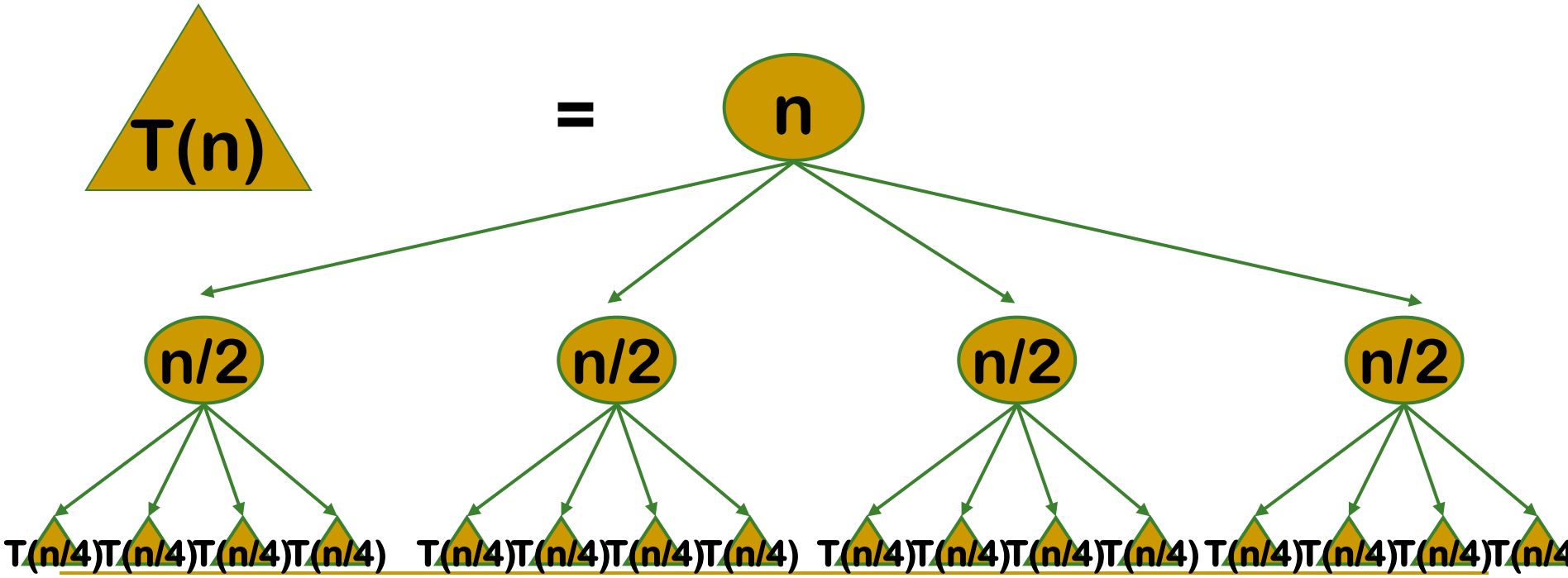
复习：算法总体思想

- 动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题



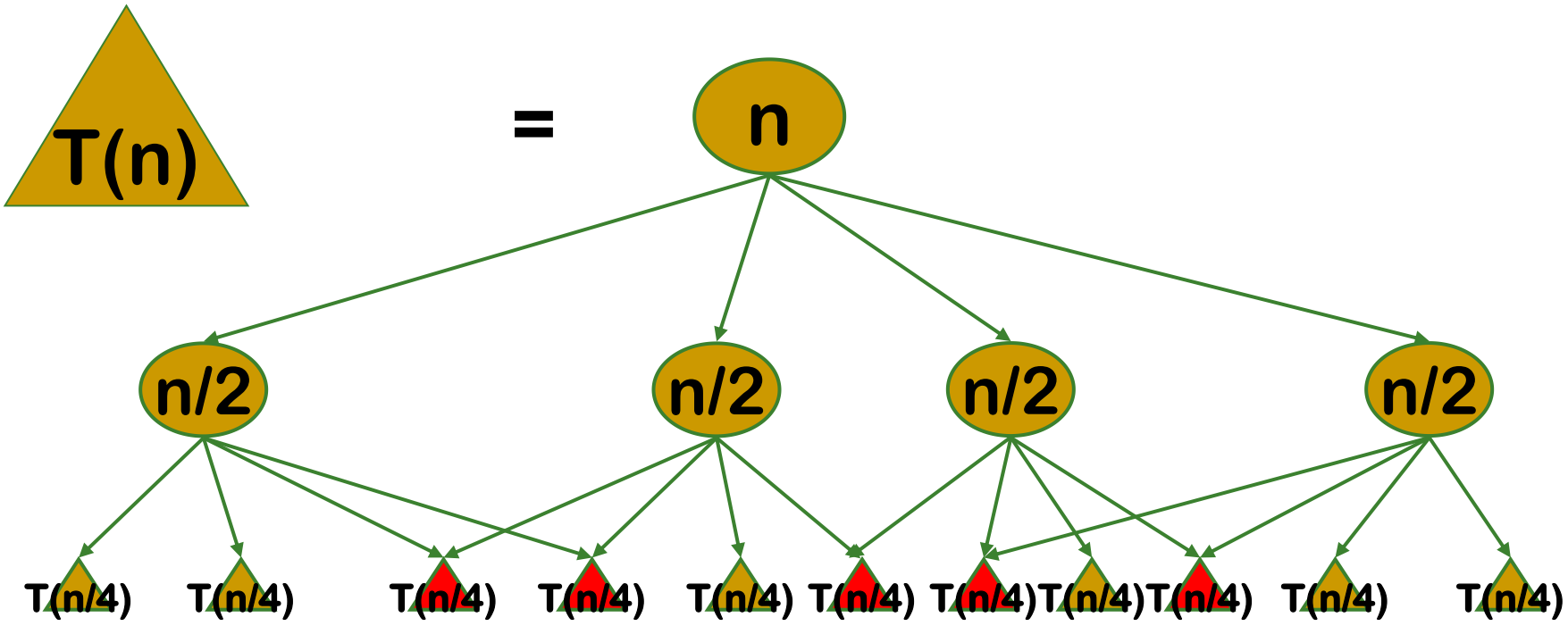
算法总体思想

- 但是在动态规划问题中，经分解得到的子问题往往不是互相独立的。不同子问题的数目常常只有多项式量级。如果使用分治法求解，有些子问题被重复计算了许多次。



算法总体思想

- 如果能够**保存已解决的子问题的答案**，而在需要时再找出已求得的答案，就可以避免大量重复计算，从而得到多项式时间算法。



复习：动态规划法 .VS. 分治法

- 动态规划法的实质也是将较大问题分解为较小的同类子问题，这一点上它与分治法类似。
- 分治法的子问题相互独立，相同的子问题被重复计算。
- 动态规划法利用问题的最优子结构特征，设计自底向上的计算过程，通过从子问题的最优解逐步构造出整个问题的最优解，避免重复计算。

复习：动态规划基本步骤

- 设计一个动态规划算法，通常可以按以下几个步骤进行：

1. **找出最优解的性质，并刻划其结构特征。**
2. **递归地定义最优值。**
3. **计算出最优值，通常采用自底向上的方式。**
4. **根据计算最优值时得到的信息，构造最优解。**

3.3 最长公共子序列

- 若给定序列$X=\{x_1, x_2, \ldots, x_m\}$，则另一序列$Z=\{z_1, z_2, \ldots, z_k\}$是X的子序列是指存在一个严格递增下标序列$\{i_1, i_2, \ldots, i_k\}$使得对于所有$j=1,2,\ldots,k$有：$z_j = x_{i_j}$。

例如，给定序列X={A，B，C，B，D，A，B}，序列Z={B，C，D，B}是X的子序列，相应的递增下标序列为{2，3，5，7}。

- 给定2个序列X和Y，当另一序列Z既是X的子序列又是Y的子序列时，称Z是序列X和Y的公共子序列。
- 问题：给定2个序列$X=\{x_1,x_2,\ldots,x_m\}$和$Y=\{y_1,y_2,\ldots,y_n\}$，找出X和Y的最长公共子序列。

最长公共子序列的结构

设序列$X=\{x_1,x_2,\ldots,x_m\}$和$Y=\{y_1,y_2,\ldots,y_n\}$的最长公共子序列为$Z=\{z_1,z_2,\ldots,z_k\}$，从后向前推理，则

(1) 若 $x_m = y_n$，则 $z_k = x_m = y_n$，且Z_{k-1}是X_{m-1}和Y_{n-1}的最长公共子序列。

(2) 若 $x_m \neq y_n$ 且 $z_k \neq x_m$，则Z是X_{m-1}和Y的最长公共子序列。

(3) 若 $x_m \neq y_n$ 且 $z_k \neq y_n$，则Z是X和Y_{n-1}的最长公共子序列。

由此可见，2个序列的最长公共子序列包含了这2个序列的前缀的最长公共子序列。因此，最长公共子序列问题具有**最优子结构性质**。

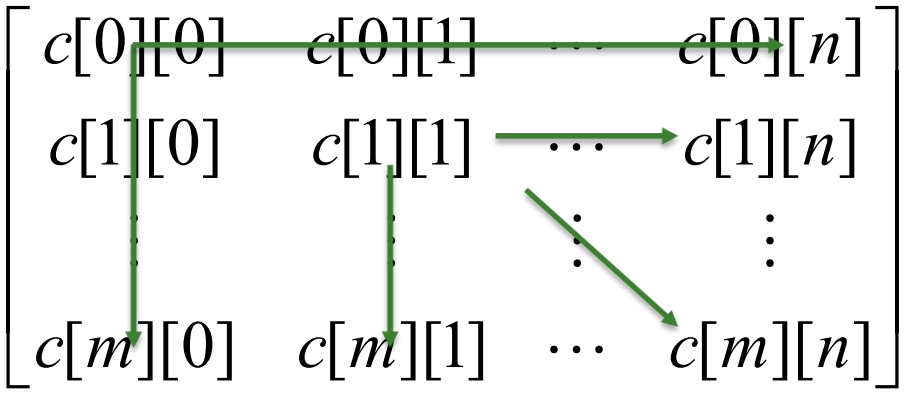
子问题的递归结构

由最长公共子序列问题的最优子结构性质建立子问题最优值的递归关系。用c[i][j]记录序列X_i和Y_j的最长公共子序列的长度。其中，$X_i=\{x_1,x_2,\ldots,x_i\}$；$Y_j=\{y_1,y_2,\ldots,y_j\}$。当i=0或j=0时，空序列是$X_i$和$Y_j$的最长公共子序列。故此时c[i][j]=0。其它情况下，由最优子结构性质可建立递归关系如下：

$$c[i][j]=\begin{cases}0 & i=0 \text{ or } j=0\\ c[i-1][j-1]+1 & i,j>0; x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0; x_i\neq y_j\end{cases}$$

计算最优值

由于在所考虑的子问题空间中，总共有 θ (mn)个不同的子问题，因此，用动态规划算法自底向上地计算最优值能提高算法的效率。



- 要得到c[i][j]，需要其左侧、上侧和左上方的值
- 因此，先对c[i][0]和c[0][j]赋初值0
- 然后计算c[1][1]...c[m][1]，再计算c[1][2]...c[m][2]
- 以此类推，最终得到c[m][n]

计算最优值

由于在所考虑的子问题空间中，总共有 θ (mn)个不同的子问题，因此，用动态规划算法自底向上地计算最优值能提高算法的效率。

```
void LCSLength(int m，int n，char *x，char *y，int **c，int **b)
{
  int i，j;
  for (i = 1; i <= m; i++) c[i][0] = 0;
  for (i = 1; i <= n; i++) c[0][i] = 0;
  for (i = 1; i <= m; i++)
    for (j = 1; j <= n; j++) {
      if (x[i]==y[j]) {
        c[i][j]=c[i-1][j-1]+1;
        b[i][j]=1;  //(1) “ 左上角”
      }
      else if (c[i-1][j]>=c[i][j-1]) {
        c[i][j]=c[i-1][j];
        b[i][j]=2;  //(2) “上侧”
      }
      else {
        c[i][j]=c[i][j-1];
        b[i][j]=3; //(3) “左侧”
      }
    }
}
```

构造最优解

```
void LCS(int i，int j，char *x，int **b)
{
    if (i ==0 || j==0) return;
    if (b[i][j]== 1){
       LCS(i-1，j-1，x，b);
       cout<<x[i];
    }
    else if (b[i][j]== 2) LCS(i-1，j，x，b);
    else LCS(i，j-1，x，b);
}
```

- b[i][j]表示c[i][j]取值的三种情况，用来构造最优解

LCS的回溯构造最优解过程

- c[i][j]存储最长公共子序列的长度
- b[i][j]记录c[i][j]是由哪个子问题的解得到的
- 没有单独考虑c[i-1][j]==c[i][j-1]的情况，对b[i][j]二维数组的取值添加一种可能，等于4，来表示c[i-1][j]==c[i][j-1]。这样通过路径回溯可以找出**所有**最长公共子序列。不做这项更改则只能找出一个最优解，不算错。

j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)	0(0)
1	A	0(0)	0(4)	0(4)	0(4)	1(1)	1(3)	1(1)
2	B	0(0)	1(1)	1(3)	1(3)	1(4)	2(1)	2(3)
3	C	0(0)	1(2)	1(4)	2(1)	2(3)	2(4)	2(4)
4	B	0(0)	1(1)	1(4)	2(2)	2(4)	3(1)	3(3)
5	D	0(0)	1(2)	2(1)	2(4)	2(4)	3(2)	3(4)
6	A	0(0)	1(2)	2(2)	2(4)	3(1)	3(4)	4(1)
7	B	0(0)	1(1)	2(2)	2(4)	3(2)	4(1)	4(4)

- 指c[i][j]=4,b[i][j]=2

比如序列 X 为 ABCBDAB，序列 Y 为 BDCABA，则其公共最长子序列为 BCBA \ BCAB \ BDAB

算法的改进

•在算法**lcsLength**和**lcs**中，可进一步将数组b省去。事实上，数组元素c[i][j]的值仅由c[i-1][j-1]，c[i-1][j]和c[i][j-1]这3个数组元素的值所确定。对于给定的数组元素c[i][j]，可以不借助于数组b而仅借助于三个值的关系在O(1)时间内来确定c[i][j]的值是由c[i-1][j-1]，c[i-1][j]和c[i][j-1]中哪一个值所确定的。

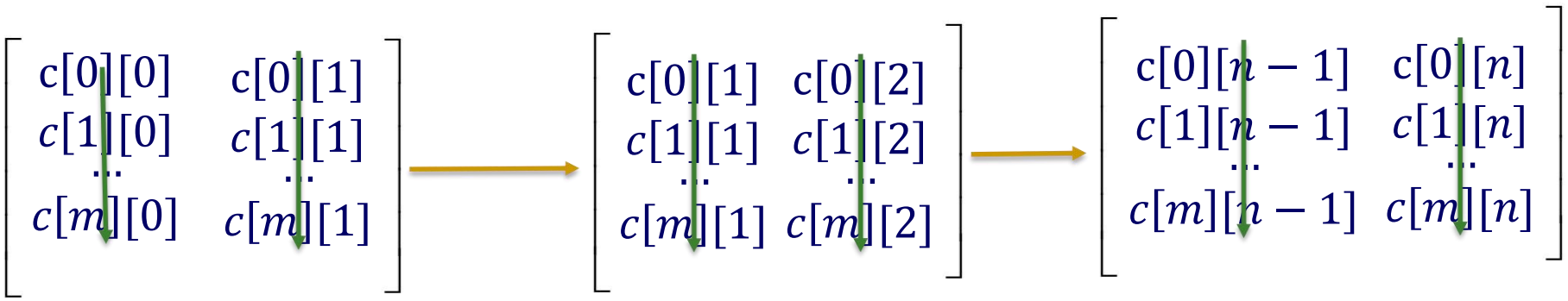
•如果只需要计算最长公共子序列的长度，则算法的空间需求可大大减少。事实上，在计算c[i][j]时，只用到数组c的第i行和第i-1行。因此，用2行的数组空间就可以计算出最长公共子序列的长度。进一步的分析还可将空间需求减至O(min(m,n))。

算法的改进

- 比如：

$$\begin{bmatrix} c[0][0] & c[0][1] & \cdots & c[0][n] \\ c[1][0] & c[1][1] & \cdots & c[1][n] \\ \vdots & \vdots & \vdots & \vdots \\ c[m][0] & c[m][1] & \cdots & c[m][n] \end{bmatrix}$$

- 只准备(m+1)*2大小的空间



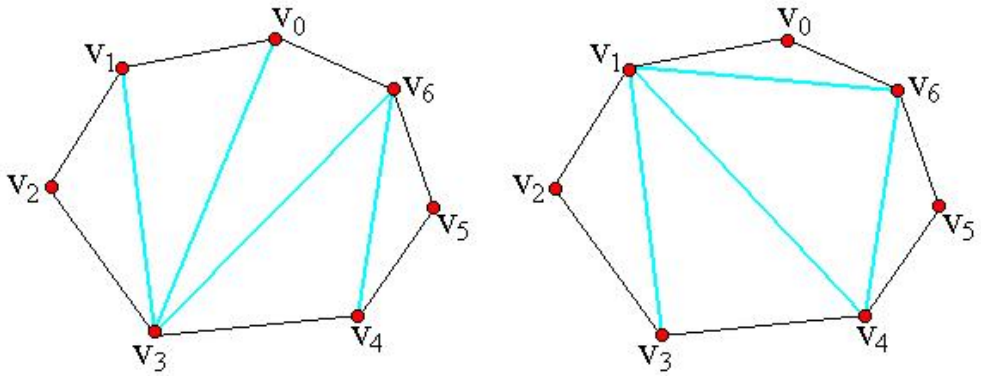
3.5 凸多边形最优三角剖分

•用多边形顶点的逆时针序列表示凸多边形，即$P=\{v_0,v_1,\ldots,v_{n-1}\}$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三个顶点组成的三角形上的权函数$w(v_iv_kv_j)$。要求确定该凸多边形的三角剖分，使得在该三角剖分中诸三角形上权之和为最小。



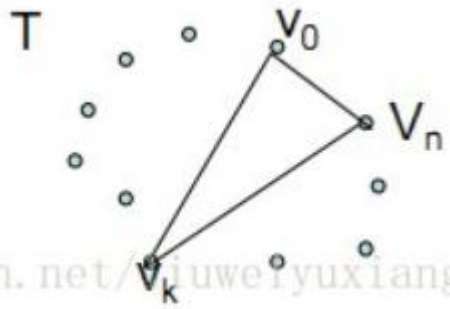
最优子结构性质

- 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$P=\{v_0,v_1,\ldots,v_n\}$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]。

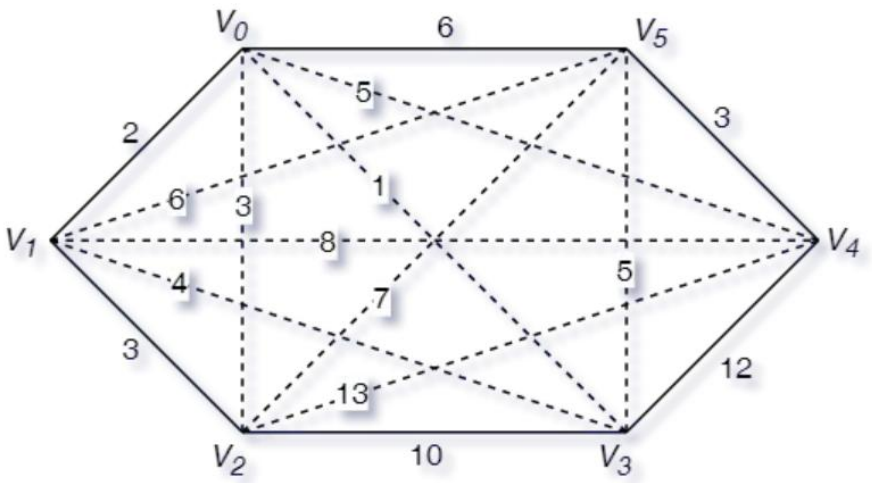
•t[i][j]的值可以利用最优子结构性质递归地计算。当$j-i \ge 1$时，凸子多边形至少有3个顶点。由最优子结构性质，t[i][j]的值应为t[i][k]的值加上t[k+1][j]的值，再加上三角形$v_{i-1}v_kv_j$的权值，其中$i \le k \le j-1$。由于在计算时还不知道k的确切位置，而k的所有可能位置只有j-i个，因此可以在这j-i个位置中选出使t[i][j]值达到最小的位置。

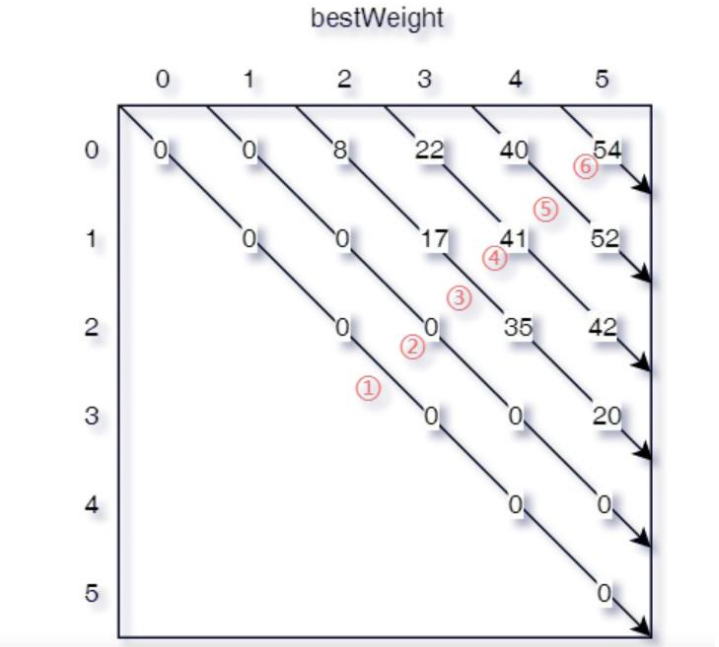


最优三角剖分的递归结构

•由此，t[i][j]可递归地定义为：

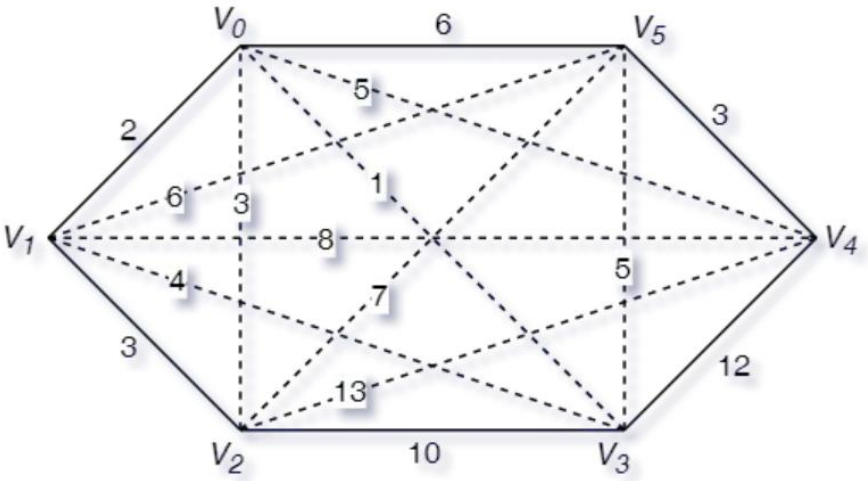
$$t[i][j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t[i][k]+t[k+1][j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

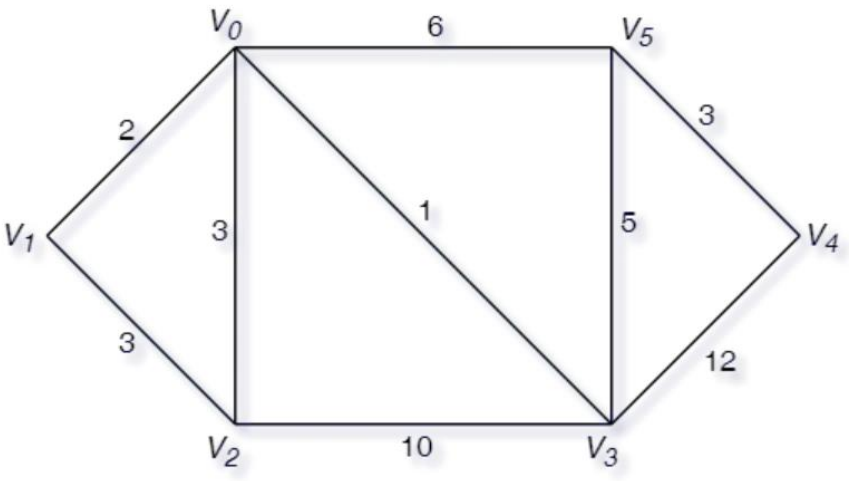




- 得到该递归式就可以写出动态规划算法

最优三角剖分





•此凸 6 边形的最优三角剖分的权值之和为 (2 + 3 + 3) + (3 + 10 + 1) + (1 + 5 + 6) + (5 + 12 + 3) = 54 。

```
public static void minWeightTriangulation(int n) {
  for (int i = 1; i <= n; i++) {
      t[i][i] = 0;
  }
  for (int r = 2; r <= n; r++) {//i与j的差值
      for (int i = 1; i <= n - r + 1; i++) {
          int j = i + r - 1;
          m[i][j] = m[i + 1][j] + w(i-1,i,j);//k==i的情况
          s[i][j] = i;
          for (int k = i + 1; k < i+r-1; k++) {
              int u = m[i][k] + m[k + 1][j] + w(i-1,k,j);
              if (u < m[i][j]) {
                  m[i][j] = u;
                  s[i][j] = k; }}}}}
```

3.6 多边形游戏

多边形游戏是一个单人玩的游戏，开始时有一个由n个顶点构成的多边形。每个顶点被赋予一个整数值，每条边被赋予一个运算符“+”或“*”。所有边依次用整数从1到n编号。

游戏第1步，将一条边删除。

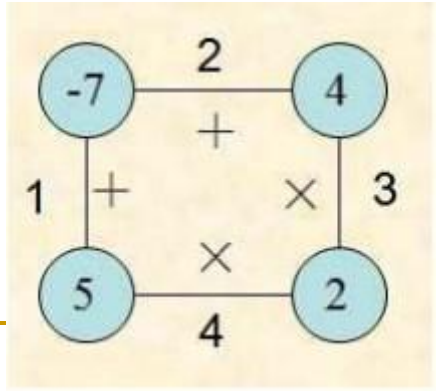
随后n-1步按以下方式操作：

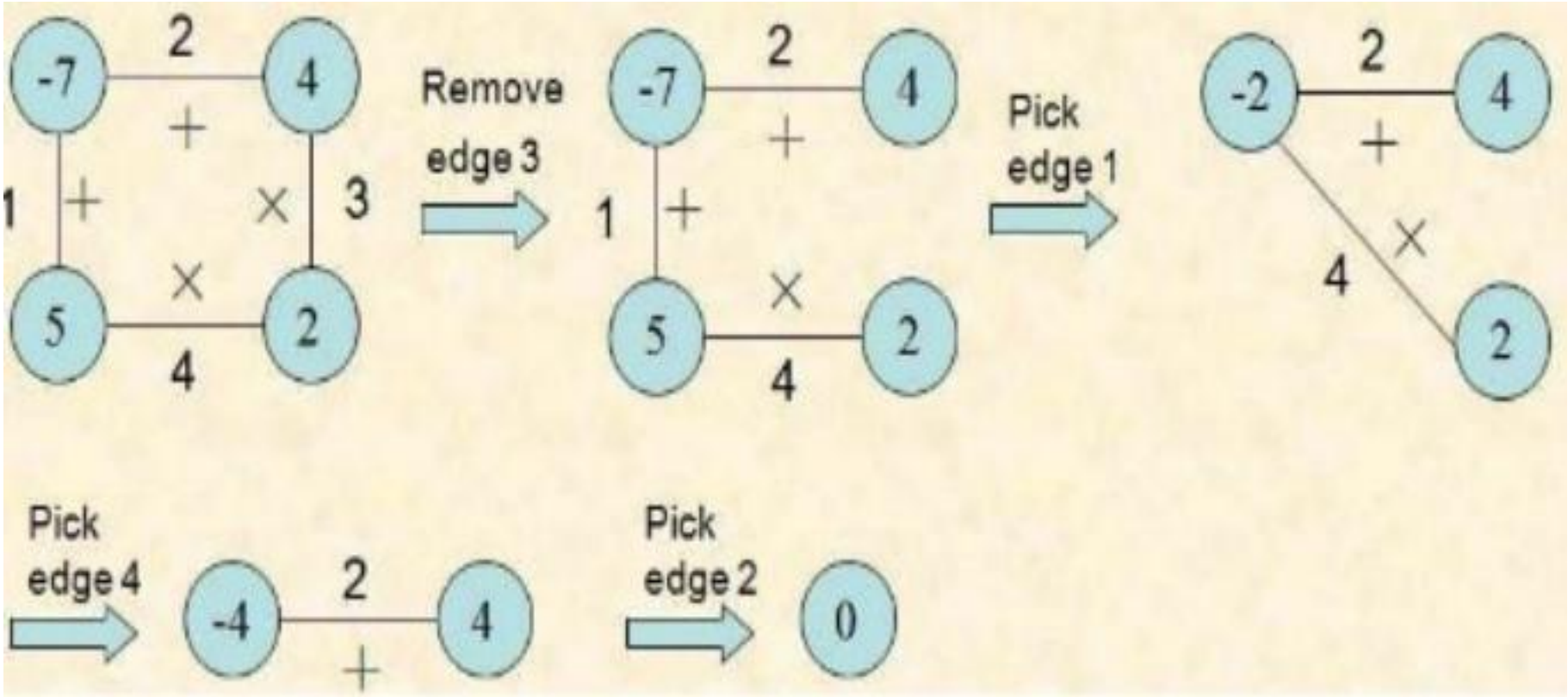
(1)选择一条边E以及由E连接着的2个顶点V_1和V_2；

(2)用一个新的顶点取代边E以及由E连接着的2个顶点V_1和V_2。将由顶点V_1和V_2的整数值通过边E上的运算得到的结果赋予新顶点。

最后，所有边都被删除，游戏结束。游戏的得分就是所剩顶点上的整数值。

问题:对于给定的多边形，计算最高得分。




-7
2
4
1
3
5
2
4
Remove
edge 3
-7
2
4
1
5
2
4
Pick
edge 1
-2
2
4
4
2
Pick
edge 4
-4
2
4
Pick
edge 2
0

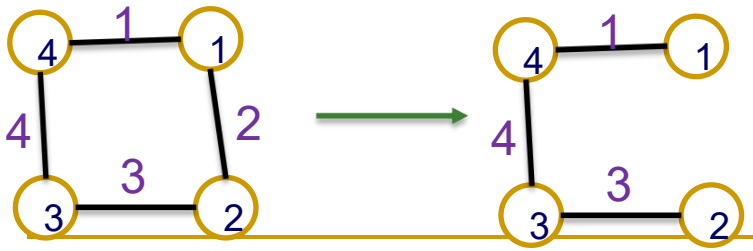
多边形游戏规则

最优子结构性质

- 设所给的多边形的边和顶点的顺时针序列为op[1], v[1], op[2], v[2], op[3],…,op[n],v[n] 其中，op[i]表示第i条边所对应的运算符，v[i]表示第i个顶点上的数值，i=1~n。

- 在所给多边形中，从顶点i(1≤i≤n)开始，长度为j (链中有j个顶点)的顺时针链p(i，j) 可表示为v[i]，op[i+1]，…，v[i+j-1]。则将一条边op[i]删去后，得到的链为p(i,j)。

- 如果这条链的最后一次合并运算在op[i+s]处发生(1≤s≤j-1)，则可在op[i+s]处将链分割为2个子链p(i， s)和p(i+s，j-s)。
- 这里的加法需要进行取模运算，即mod n



p(2,4)=v(2),op(3),,,,v(1)

最优子结构性质

- 设m_1是对子链p(i，s)的任意一种合并方式得到的值，而a和b分别是在所有可能的合并中得到的最小值和最大值。m_2是p(i+s，j-s)的任意一种合并方式得到的值，而c和d分别是在所有可能的合并中得到的最小值和最大值。依此定义有$a \le m_1 \le b$，$c \le m_2 \le d$

(1)当op[i+s]='+'时，显然有$a+c \le m \le b+d$
(2)当op[i+s]='*'时，有$\min\{ac，ad，bc，bd\} \le m \le \max\{ac，ad，bc，bd\}$

- 换句话说，主链的最大值和最小值可由子链的最大值和最小值得到。

多边形游戏的递归求解

- 设m[i,j,0]是链p(i,j)合并的最小值，而m[i,j,1]是最大值。若最优合并在op[i+s]处将p(i,j)分为两个长度小于j的子链 p(i, s)和p(i+s，j-s) 的最大值和最小值均已计算出。即：

$$a = m[i, s, 0], \qquad b = m[i, s, 1]$$
$$c = m[i + s, j - s, 0], \qquad d = m[i + s, j - s, 1]$$

(1) 当op[i+s]=’+’时

$$m[i, j, 0] = a + c$$
$$m[i, j, 1] = b + d$$

(2) 当op[i+s]=’*’时

$$m[i, j, 0] = \min\{ac, ad, bc, bd\}$$
$$m[i, j, 1] = \max\{ac, ad, bc, bd\}$$

多边形游戏的递归求解

- 综合 (1)和(2),将p(i, j)在op[i+s]处断开的最大值记为maxf(i, j, s)，最小值记为minf(i, j, s)

$$\text{minf}(i,j,s)=\begin{cases}a+c & op[i+s]='+'\\ \min\{ac,ad,bc,bd\} & op[i+s]='*'\end{cases}$$

$$\text{maxf}(i,j,s)=\begin{cases}b+d & op[i+s]='+'\\ \max\{ac,ad,bc,bd\} & op[i+s]='*'\end{cases}$$

- 由于最优断开位置s有$1\le s\le j-1$的j-1中情况。

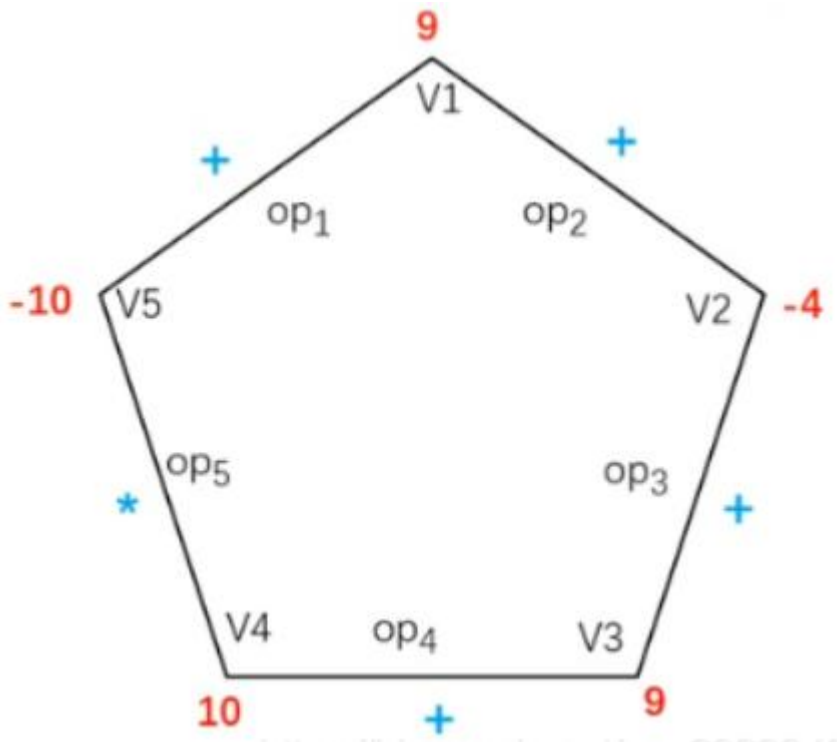
$$m[i,j,0]=\min_{1\le s\le j}\{\text{minf}(i,j,s)\},1\le i,j\le n$$

$$m[i,j,1]=\max_{1\le s\le j}\{\text{maxf}(i,j,s)\},1\le i,j\le n$$

- 初始边界值为 m[i,1,0]=v[i]　m[i,1,1]=v[i]　$1\le i\le n$
- 多边形是封闭的，当i+s>n时，顶点i+s实际编号为(i+s)%n。

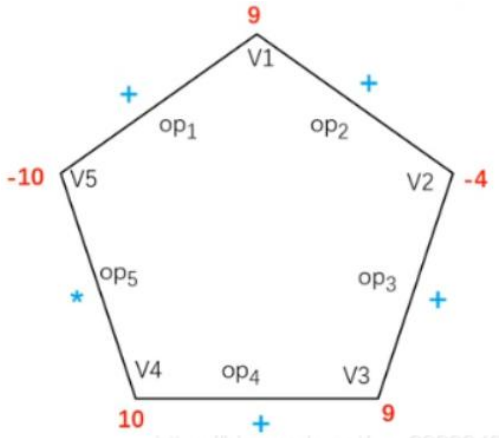
m[i,n,1]记为游戏首次删除第i条边后得到的最大得分。

多边形游戏样例



m(1,1,1) = 9
m(1,1,0) = 9
m(2,1,1) = -4
m(2,1,0) = -4
m(3,1,1) = 9
m(3,1,0) = 9
m(4,1,1) = 10
m(4,1,0) = 10
m(5,1,1) = -10
m(5,1,0) = -10

多边形游戏样例

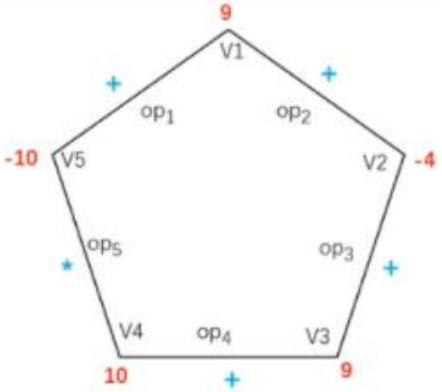


m(1,2) = m(1,1)op(2)m(2,1); op(2) = “+”
m(1,2,1) = m(1,1,1) + m(2,1,1) = 5
m(1,2,0) = m(1,1,0) + m(2,1,0) = 5
m(2,2) = m(2,1)op(3)m(3,1); op(3) = “+”
m(2,2,1) = m(2,1,1) + m(3,1,1) = 5
m(2,2,0) = m(2,1,0) + m(3,1,0) = 5
m(3,2) = m(3,1)op(4)m(4,1); op(4) = “+”
m(3,2,1) = m(3,1,1) + m(4,1,1) = 19
m(3,2,0) = m(3,1,0) + m(4,1,0) = 19
m(4,2) = m(4,1)op(5)m(5,1); op(5) = “*”
m(4,2,1) = max{m(4,1,1)*m(5,1,1), m(4,1,1)*m(5,1,0), m(4,1,0)*m(5,1,1), m(4,1,0)*m(5,1,0)} = -100
m(4,2,0) = min{m(4,1,1)*m(5,1,1), m(4,1,1)*m(5,1,0), m(4,1,0)*m(5,1,1), m(4,1,0)*m(5,1,0)} = -100
m(5,2) = m(5,1)op(1)m(1,1)
m(5,2,1) = m(5,1,1) + m(1,1,1) = -1
m(5,2,0) = m(5,1,0) + m(1,1,0) = -1

max = max{m(1,5,1), m(2,5,1), m(3,5,1), m(4,5,1), m(5,5,1)} = 40

min = min{m(1,5,0), m(2,5,0), m(3,5,0), m(4,5,0), m(5,5,0)} = -240

	1	2	3	4	5
1	9 9	5 5	14 14	24 24	-86 -240
2	-4 -4	5 5	15 15	-95 -194	-5 -185
3	9 9	19 19	-91 -190	-1 -181	-5 -185
4	10 10	-100 -100	-10 -91	-14 -95	40 -86
5	-10 -10	-1 -1	-5 -5	4 4	14 14



所以

max = 40

min=-240