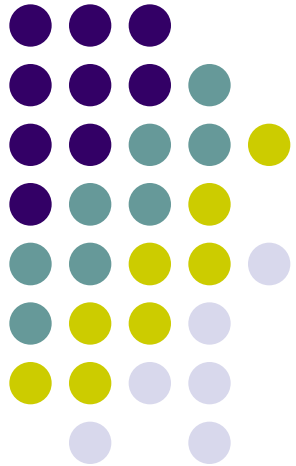
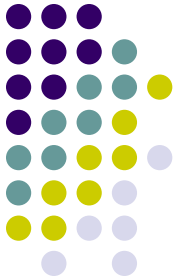
算法设计与分析

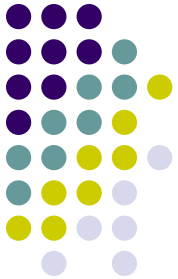
动态规划3



区间类动态规划的特点



- 区间合并：意思就是将两个或多个部分进行整合，当然也可以反过来，也就是是将一个问题进行分解成两个或多个部分。
- 特征：能将问题分解成为两两合并的形式
- 求解：对整个问题设最优值，枚举合并点，将问题分解成为左右两个部分，最后将左右两个部分的最优值进行合并得到原问题的最优值。有点类似分治算法的解题思想。
- 典型试题：整数划分，凸多边形划分、石子合并、多边形合并、能量项链等。



例16：整数划分

- 给出一个长度为n的数
- 要在其中加m-1个乘号，分成m段
- 这m段的乘积之和最大
- m<n<=20
- 有T组数据，T<=10000

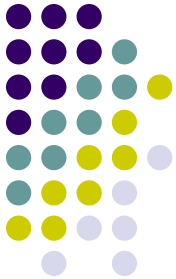
贪心法

- 尽可能平均分配各段，这样最终的数值将会尽可能大。但有反例。如**191919**分成**3**段

  19*19*19=6859

  但**191*91*9=156429**，显然乘积更大。
- 将一个数分成若干段乘积后比该数小，因为输入数不超过**20**位，因此不需高精度运算。
- 证明：
  - 假设**AB**分成**A**和**B**,且**A,B<10**,则有
  - **AB=10*A+B>A*B**(相当于**B**个**A**相加)
  - 同理可证明**A,B**为任意位也成立

动态规划



- 可以先预处理出原数第i到j段的数值A[i,j]是多少，这样转移就方便了，预处理也要尽量降低复杂度。
- F[i，j]表示把这个数前i位分成j段得到的最大乘积。
- F[i,j]=F[k,j-1]*A[k+1,i],
- 1<k<i<=n, j<=m
- 时间复杂度为$O[mn^2]$
- 由于有10000组数据，因此估计时间复杂度为$10000*20^3=8*10^7$
- 至于说输出，记录转移的父亲就可以了。

石子合并

- 在一园形操场四周摆放N堆石子(N≤100)；
- 现要将石子有次序地合并成一堆；
- 规定每次只能选相临的两堆合并成一堆,并将新的一堆的石子数,记为该次合并的得分。
  1. 选择一种合并石子的方案,使得做N-1次合并,得分的总和最少
  2. 选择一种合并石子的方案,使得做N-1次合并,得分的总和最大

示例

总得分=8+13+22=43

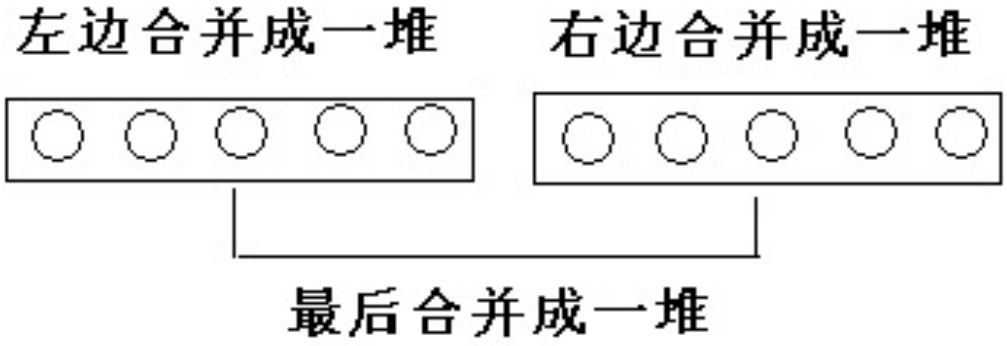
总得分=14+18+22=54

贪心法

N=5　石子数分别为3 4 6 5 4 2。	
用贪心法的合并过程如下： 第一次 3 4 6 5 4 2得分 5 第二次 5 4 6 5 4得分9 第三次 9 6 5 4得分9 第四次 9 6 9得分15 第五次 15 9得分24 第六次24 总分：62	然而有更好的方案： 第一次3 4 6 5 4 2得分 7 第二次7 6 5 4 2得分13 第三次13 5 4 2得分6 第四次13 5 6得分11 第五次 13 11得分24 第六次24 总分：61
显然，贪心法是错误的。	

分析

- 假设只有2堆石子，显然只有1种合并方案
- 如果有3堆石子，则有2种合并方案，((1,2),3)和(1,(2,3))
- 如果有k堆石子呢？
- 不管怎么合并，总之最后总会归结为2堆，如果我们把最后两堆分开，左边和右边无论怎么合并，都必须满足最优合并方案，整个问题才能得到最优解。如下图：



动态规划

- 设**t[i,j]**表示从第**i**堆到第**j**堆石子数总和。
  Fmax(i,j)表示将从第**i**堆石子合并到第**j**堆石子的最大的得分
  Fmin(i,j)表示将从第**i**堆石子合并到第**j**堆石子的最小的得分

$$F\max(i,j)=\max_{i\le k\le j-1}\{F\max(i,k)+F\max(k+1,j)+t[i,j]\}$$

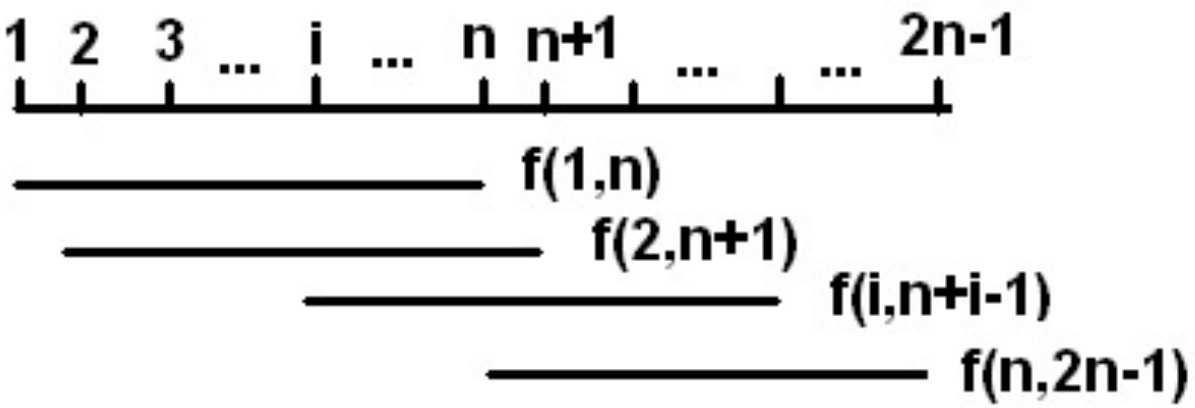
同理，

$$F\min(i,j)=\min_{i\le k\le j-1}\{F\min(i,k)+F\min(k+1,j)+t[i,j]\}$$

- **Fmax[i,i] = 0，Fmin[i,i] = 0**
- 时间复杂度为**O(n^3)**

优化

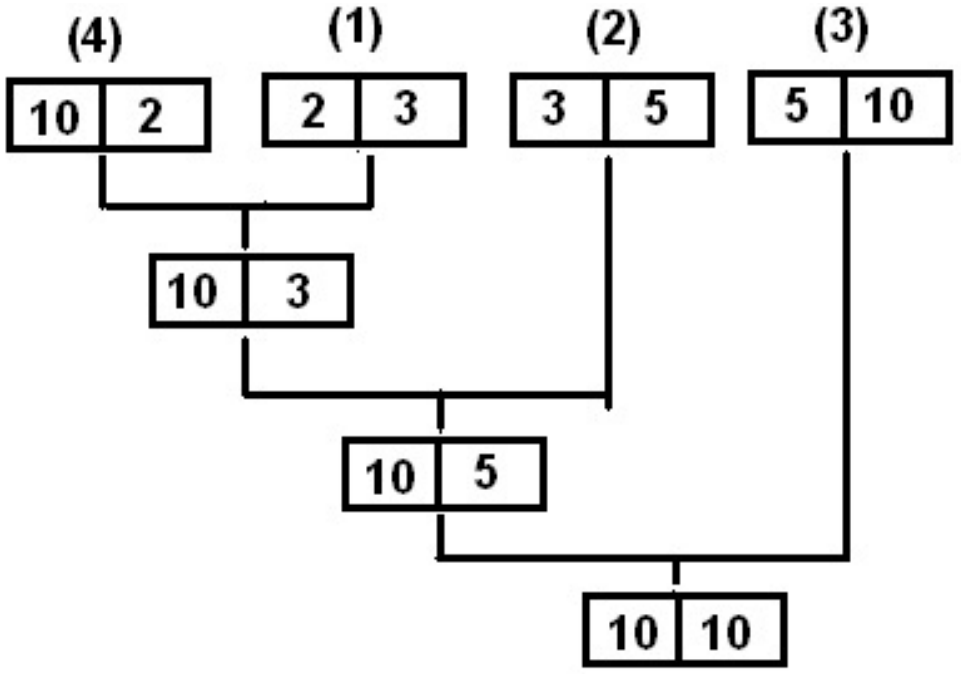
- 由于石子堆是一个圈，因此我们可以枚举分开的位置，首先将这个圈转化为链，因此总的时间复杂度为$O(n^4)$。
- 这样显然很高，其实我们可以将这条链延长2倍，扩展成2n-1堆，其中第1堆与n+1堆完全相同，第i堆与n+i堆完全相同，这样我们只要对这2n堆动态规划后，枚举f(1,n),f(2,n+1),…,f(n,2n-1)取最优值即可即可。
- 时间复杂度为$O(8n^3)$,如下图：



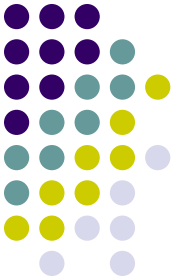
能量项链

- 在Mars星球上，每个Mars人都随身佩带着一串能量项链。
- 在项链上有N颗能量珠。
- 能量珠是一颗有头标记与尾标记的珠子，这些标记对应着某个正整数。
- 对于相邻的两颗珠子，前一颗珠子的尾标记一定等于后一颗珠子的头标记。如果前一颗能量珠的头标记为m，尾标记为r，后一颗能量珠的头标记为r，尾标记为n，则聚合后释放的能量为m×r×n（Mars单位），新产生的珠子的头标记为m，尾标记为n。
- 显然，对于一串项链不同的聚合顺序得到的总能量是不同的，请你设计一个聚合顺序，使一串项链释放出的总能量最大。

- 分析样例：
  N=4，4颗珠子的头标记与尾标记依次为
  (2，3) (3，5) (5，10) (10，2)。
- 我们用记号⊕表示两颗珠子的聚合操作，释放总能量：
  ((4⊕1)⊕2)⊕3）=10*2*3+10*3*5+10*5*10=710



动态规划

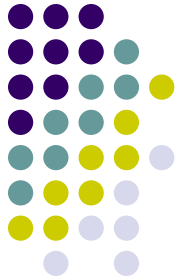


- 该题与石子合并完全类似。
- 设链中的第i颗珠子头尾标记为(S_{i-1}与S_i)。
- 令F(i,j)表示从第i颗珠子一直合并到第j颗珠子所能产生的最大能量，则有：

  $$F(i,j)=Max\{F(i,k)+F(k+1,j)+S_{i-1}*S_k*S_j,\ i<=k<j\}$$

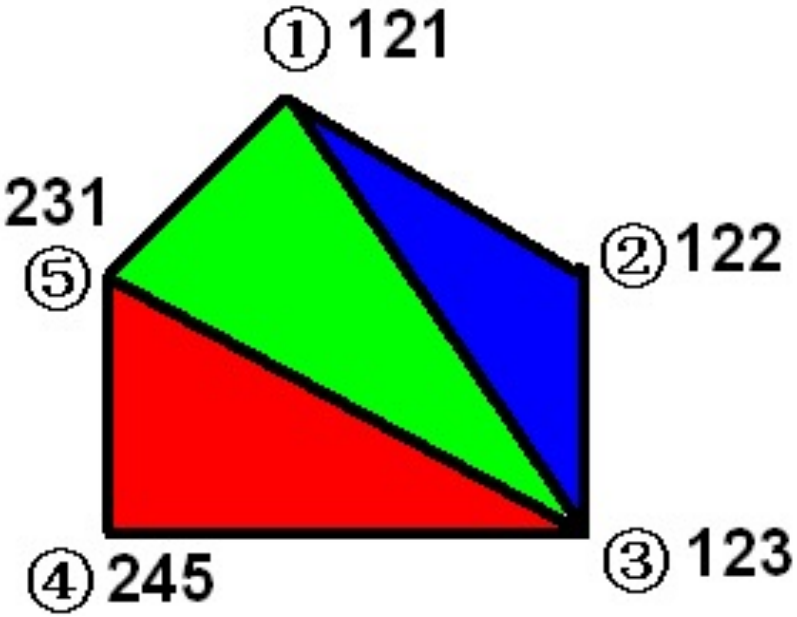
- 边界条件：F(i,i)=0
- 1<=i<k<j<=n
- 至于圈的处理，与石子合并方法完全相同，时间复杂度$O(8n^3)$ 。

凸多边形的三角剖分



- 给定由**N**顶点组成的凸多边形
- 每个顶点具有权值
- 将凸**N**边形剖分成**N-2**个三角形
- 求**N-2**个三角形顶点权值乘积之和最小？

- 样例



上述凸五边形分成△123， △135， △345

三角形顶点权值乘积之和为：

121*122*123+121*123*231+123*245*231=
12214884

分析

- 性质：一个凸多边形剖分一个三角形后，可以将凸多边形剖分成三个部分：
  - 一个三角形
  - 二个凸多边形（图**2**可以看成另一个凸多边形为**0**）

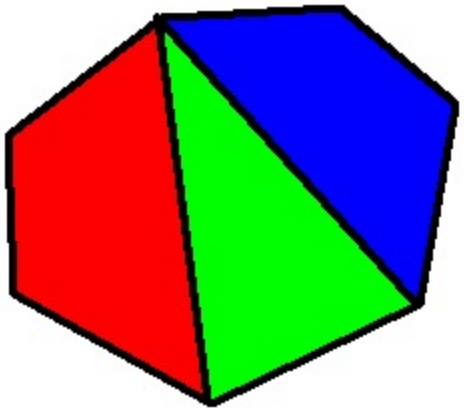


图1

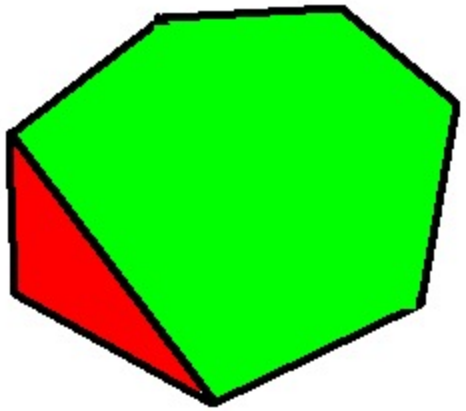
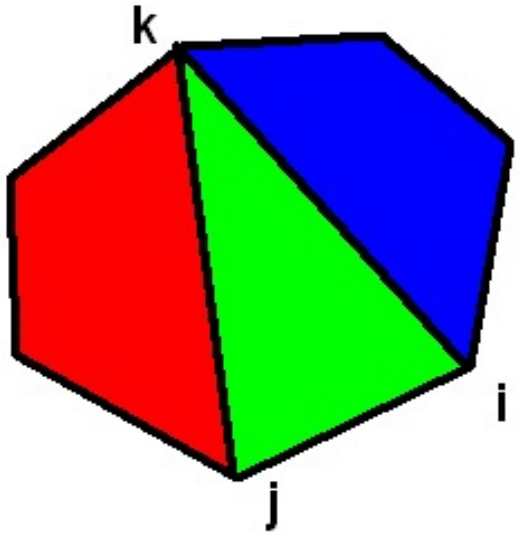


图2

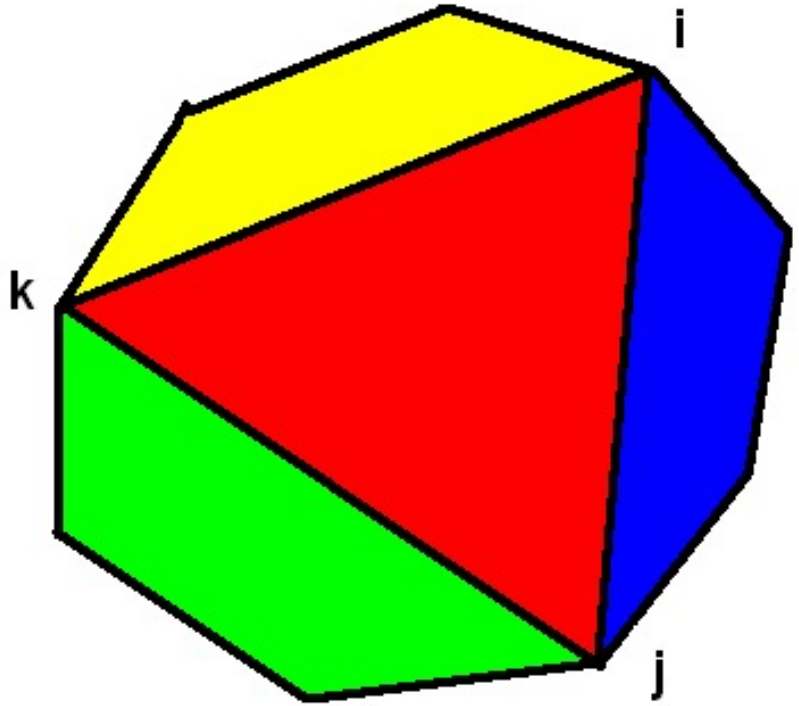
动态规划

- 如果我们按顺时针将顶点编号，则可以相邻两个顶点描述一个凸多边形。
- 设f(i,j)表示i~j这一段连续顶点的多边形划分后最小乘积
- 枚举点k，i、j和k相连成基本三角形，并把原多边形划分成两个子多边形，则有
- f(i,j)=min{f(i,k)+f(k,j)+a[i]*a[j]*a[k]}
- 1<=i<k<j<=n
- 时间复杂度$O(n^3)$



讨论

- 为什么可以不考虑这种情况？



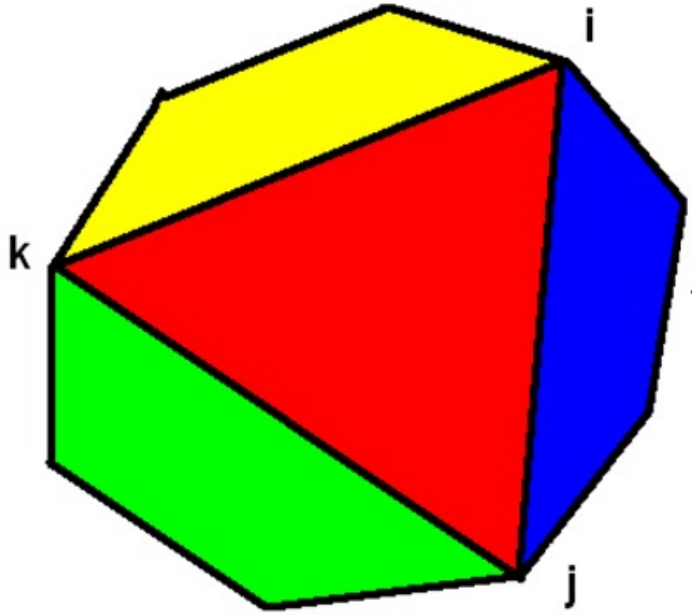


图1

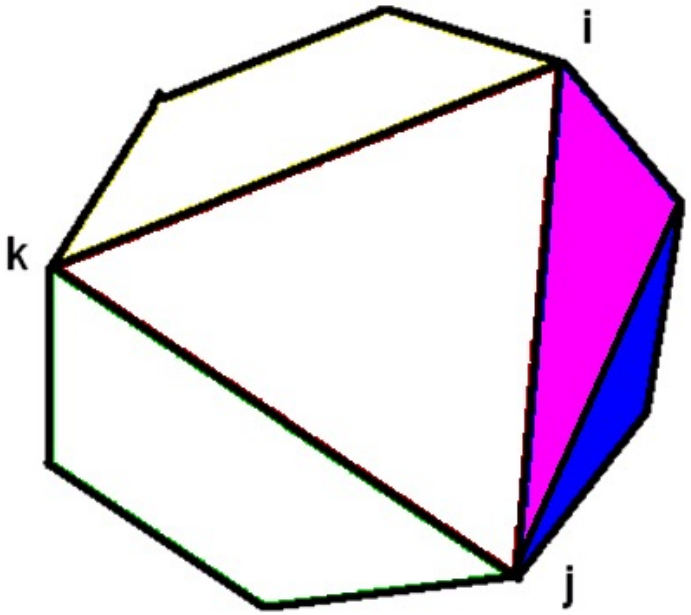


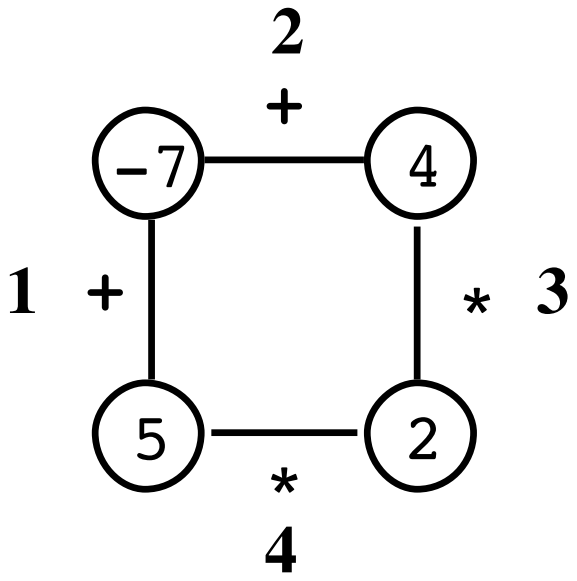
图2

- 可以看出图1和图2是等价的，也就是说如果存在图1的剖分方案，则可以转化成图2的剖分方案，因此可以不考虑图1的这种情形。

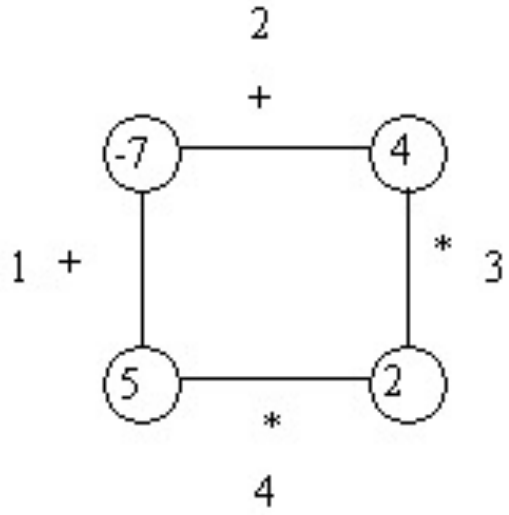
多边形（IOI98）

- 多角形是一个单人玩的游戏，开始时有一个N个顶点的多边形。如图，这里N=4。每个顶点有一个整数标记，每条边上有一个“+”号或“*”号。边从1编号到N。

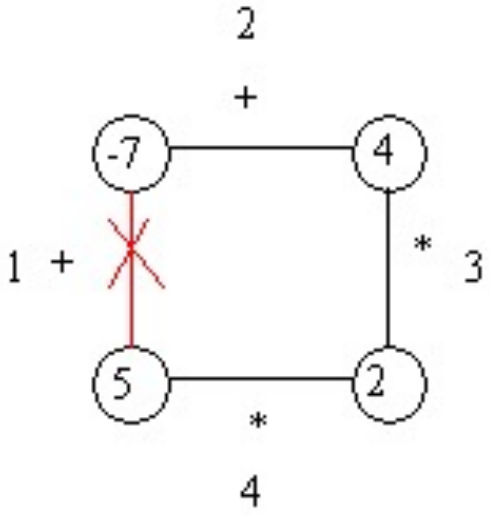
  第一步，一条边被拿走；随后各步包括如下：
- 选择一条边E和连接着E的两个顶点V1和 V2；
- 得到一个新的顶点，标记为V1与V2通过边E上的运算符运算的结果。
- 最后，游戏中没有边，游戏的得分为仅剩余的一个顶点的值。

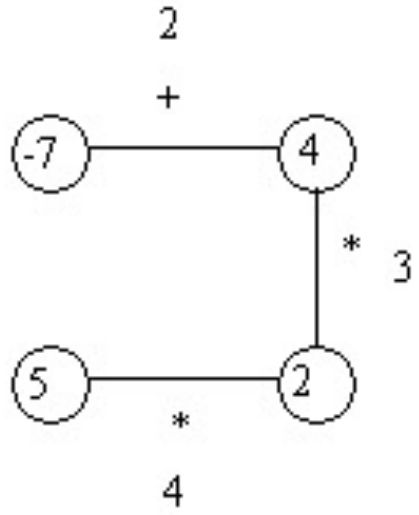


样例分析

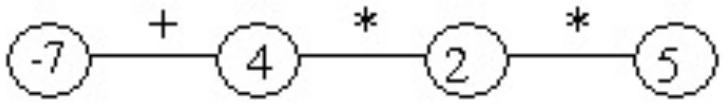


图一

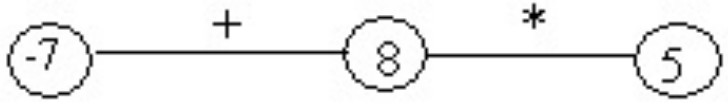




图二　　图三



图四



图五

分析

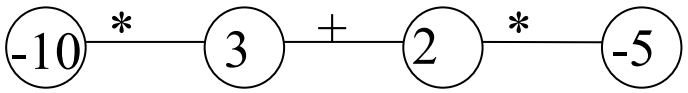
- 我们先枚举第一次删掉的边，然后再对每种状态进行动态规划求最大值 。
- 用f(i，j)表示从点i到点j进行删边操作所能得到的最大值，num(i)表示第i个顶点上的数，若为加法，那么：

$$f(i,j) = \max \sum_{k=i}^{j-1} (f(i,k) + f(k+1,j))$$

$$f(i,i) = num(i)$$

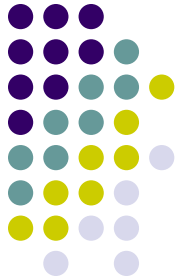
进一步分析

- 最后，我们允许顶点上出现负数。以前的方程还适不适用呢？



图六

- 这个例子的最优解应该是（3+2）*（-10）*（-5）=250，然而如果沿用以前的方程，得出的解将是（（-10）*3+2）*（-5）=125。为什么？
- 我们发现，两个负数的积为正数；这两个负数越小，它们的积越大。我们从前的方程，只是尽量使得局部解最大，而从来没有想过负数的积为正数这个问题。



分析

- 对于加法，两个最优相加肯定最优，而对于乘法
- 求最大值：
  - 正数×正数，如果两个都是最大值，则结果最大
  - 正数×负数，正数最小，负数最大，则结果最大
  - 负数×负数，如果两个都是最小值，则结果最大
- 求最小值：
  - 正数×正数，如果两个都是最小值，则结果最小
  - 正数×负数，正数最大，负数最小，则结果最小
  - 负数×负数，如果两个都是最大值，则结果最小

最终？

- 我们引入函数fmin和fmax来解决这个问题。fmax(i,j) 表示从点i开始，到但j为止进行删边操作所能得到的最大值，fmin(i,j) 表示最小值。

  当OP=‘+’

  $$Fmax(i,j)=max\{fmax(i,k)+fmax(k+1,j)\}$$

  $$Fmin(i,j)=min\{fmin(i,k)+fmin(k+1,j)\}$$

当OP=‘*’

$$f\max(i,j)=\begin{cases}\text{fmax}(i,k)*f\max(k+1,j)\\ \text{fmax}(i,k)*f\min(k+1,j)\\ \text{fmin}(i,k)*f\max(k+1,j)\\ \text{fmin}(i,k)*f\min(k+1,j)\end{cases}$$

$$f\min(i,j)=\begin{cases}\text{fmin}(i,k)*f\min(k+1,j)\\ \text{fmax}(i,k)*f\min(k+1,j)\\ \text{fmin}(i,k)*f\max(k+1,j)\\ \text{fmax}(i,k)*f\max(k+1,j)\end{cases}$$

完美解决

- 初始值

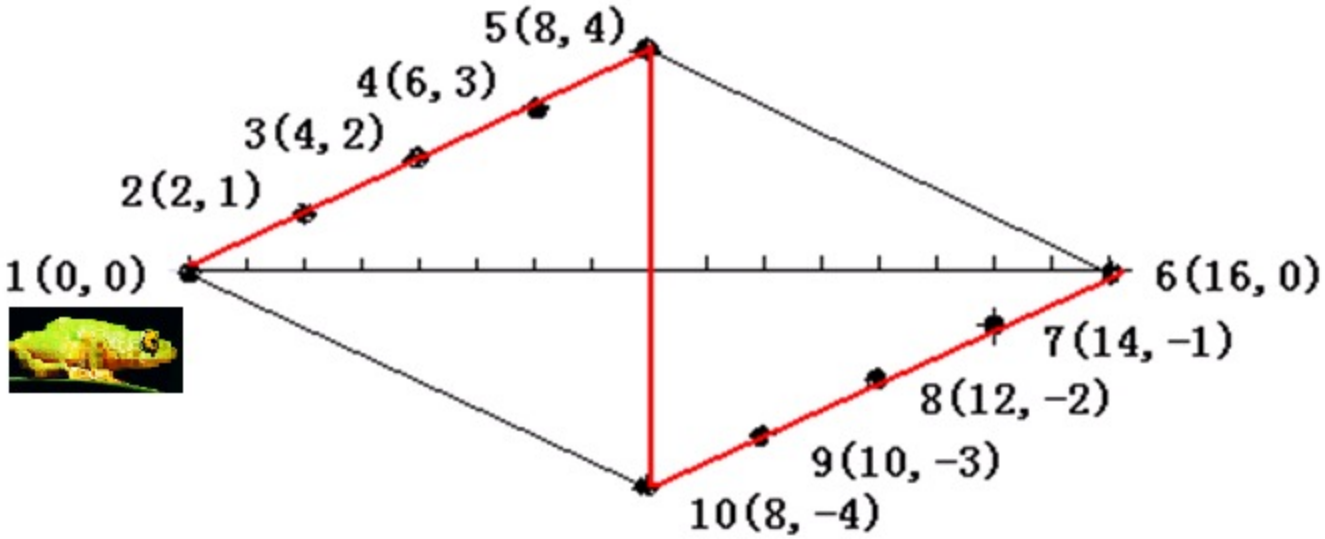
  Fmax(i,i)=num(i)

  Fmin(i,i)=num(i)
- 1<=i<=k=<j=n
- 空间复杂度:O(n^2)
- 时间复杂度:先要枚举每一条边为O(n)，然后动态规划为O(n^3)，因此总为O(n^4)。

青蛙的烦恼

- 有一个n片荷叶正好在一凸多边形顶点上
- 有一只小青蛙恰好站在1号荷叶的点
- 小青蛙可以从一片荷叶上跳到另外任意一片荷叶上
- 给出N个点的坐标
- 求小青蛙想通过最短的路程遍历所有的荷叶一次且仅一次的最短路径。

- 分析一个简单例子



- 最优遍历方法为：1,2,3,4,5,10,9,8,7,6 (图中红线)
- D=d(1,5)+d(5,10)+d(6,10)

  $=2*(8^2+4^2)^{1/2}+8$

  =25.889

分析

- 性质：青蛙遍历的路径不会相交。

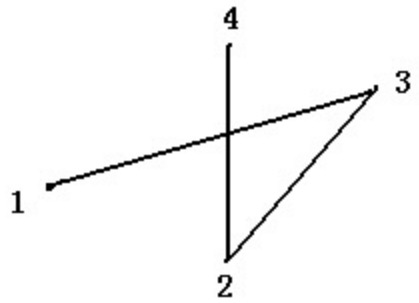


图1

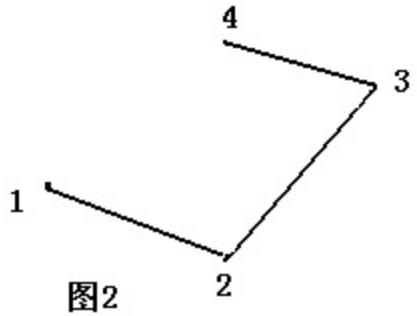


图2

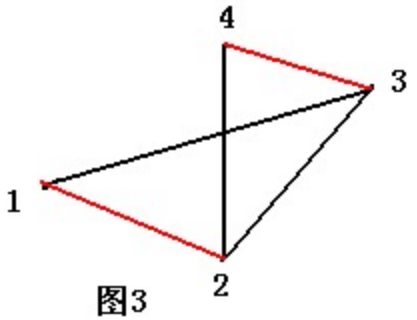


图3

- 上图中图2的路径比图1要短。
- 证明：图1：D1=d(1,3)+d(2,3)+d(2,4)

  图2：D2=d(1,2)+d(2,3)+d(3,4)

  要证明D1>D2,只要证明d(1,3) +d(2,4)>d(1,2)+d(3,4)

  连接两边，见图3，由三角形的三边关系定理即可证明。

分析

- 结论：青蛙在1号结点只能跳到2号结点或者n号结点。
  - 如果青蛙跳到了2号结点，则问题转化为：从2出发，遍历2..n一次仅一次的最短距离。
  - 如果青蛙跳到了n号结点，则问题转化为：从n出发，遍历2..n一次仅一次的最短距离。
- 这实际上是递归的思维，把问题转化为了本质相同但规模更小的子问题.如下图。

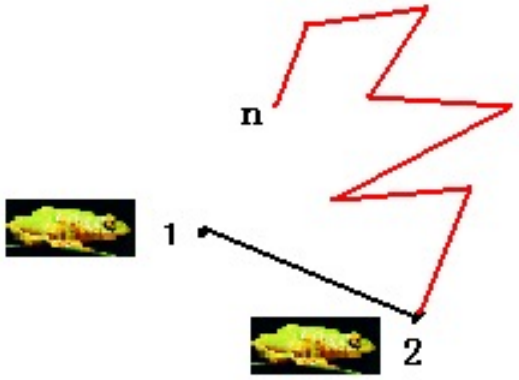


图1

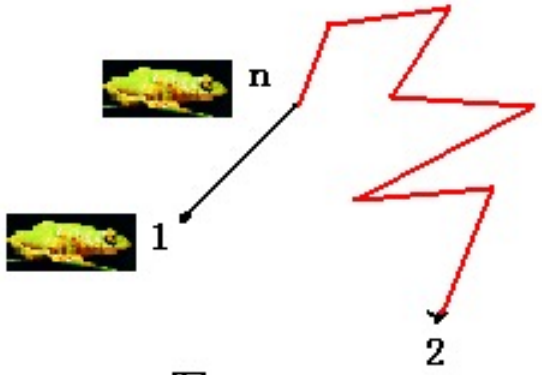


图2

动态规划(1)

- **F(i,j,0)表示还有i~j号点没访问,且青蛙停在i的最小值**
  F(i,j,1)表示还有i~j号点没访问,且青蛙停在j的最小值

$$f(i,j,0)=\min\begin{cases} f(i+1,j,0)+dist[i,i+1], \text{停在}i,\text{跳到}i+1 \\ f(i+1,j,1)+dist[i,j], \text{停在}i,\text{跳到}j \end{cases}$$

$$f(i,j,1)=\min\begin{cases} f(i,j-1,1)+dist[j,j-1], \text{停在j，跳到}j-1 \\ f(i,j\text{-}1,0)+dist[i,j], \text{停在j，跳到}i \end{cases}$$

- **状态总数为n^2，状态转移的复杂度为O(1)，总的时间复杂度为$O(n^2)$。**

动态规划(2)

- **f(s,L,0)表示从s出发，遍历s..s+L-1一次且仅一次的最短距离；**

  f(s, L,1)表示从s+L-1出发，遍历s..s+L-1一次且仅一次的最短距离。状态转移方程为：

$$f(s,L,0)=\min\begin{cases} f(s+1,L-1,0)+dist[s,s+1], \text{跳到} s+1 \\ f(s+\mathrm{L}-1,L-1,1)+dist[s,s+L-1], \text{跳到} s+L-1 \end{cases}$$

$$f(s,L,1)=\min\begin{cases} f(s,L-1,0)+dist[s,s+L-1], \text{跳到} s \\ f(s,L-1,1)+dist[s+L-1,s+L-2], \text{跳到} s+L-2 \end{cases}$$

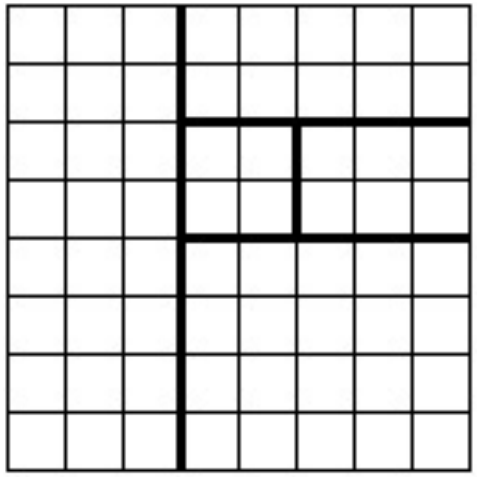
- **状态总数为n^2，状态转移的复杂度为O(1)，总的时间复杂度为$O(n^2)$。**

主程序

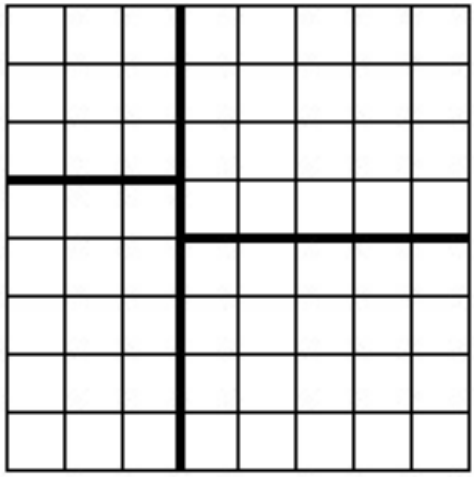
```
for i:=1 to n do
    for j:=n downto i+1 do
      begin
        update(f[i+1,j,0],f[i,j,0]+d[i,i+1]);   // 停在i,跳到
    i+1
        update(f[i+1,j,1],f[i,j,0]+d[i,j]);      // 停在i,跳到j
        update(f[i,j-1,0],f[i,j,1]+d[i,j]);      // 停在j,跳到i
        update(f[i,j-1,1],f[i,j,1]+d[j-1,j]);   // 停在j,跳到j-
    1
      end;
```

棋盘分割

- 将一个 8 × 8 的棋盘进行如下分割：将原棋盘割下一块矩形棋盘并使剩下部分也是矩形，再将剩下的部分继续如此分割，这样割了(n-1)次后，连同最后剩下的矩形棋盘共有n块矩形棋盘。(每次切割都只能沿着棋盘格子的边进行)



允许的分割方案



不允许的分割方案

任务：

棋盘上每一格有一个分值，一块矩形棋盘的总分为其所含各格分值之和。现在需要把棋盘按上述规则分割成n块矩形棋盘，并使各矩形棋盘总分的均方差最小。

- 均方差：

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

- 算术平均值：

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

均方差公式化简

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

$$\sigma^2 = \sum_{i=1}^{n}(x_i - \bar{x})^2 \Big/ n = \sum_{i=1}^{n}(x_i^2 - 2x_i\bar{x} + \bar{x}^2) \Big/ n$$

$$= (n\bar{x}^2 + \sum_{i=1}^{n} x_i^2 - 2\bar{x}\sum_{i=1}^{n} x_i) \Big/ n = \bar{x}^2 + \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \frac{2\bar{x}}{n}\sum_{i=1}^{n} x_i$$

$$= \bar{x}^2 + \frac{1}{n}\sum_{i=1}^{n} x_i^2 - 2\bar{x}^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2$$

样例

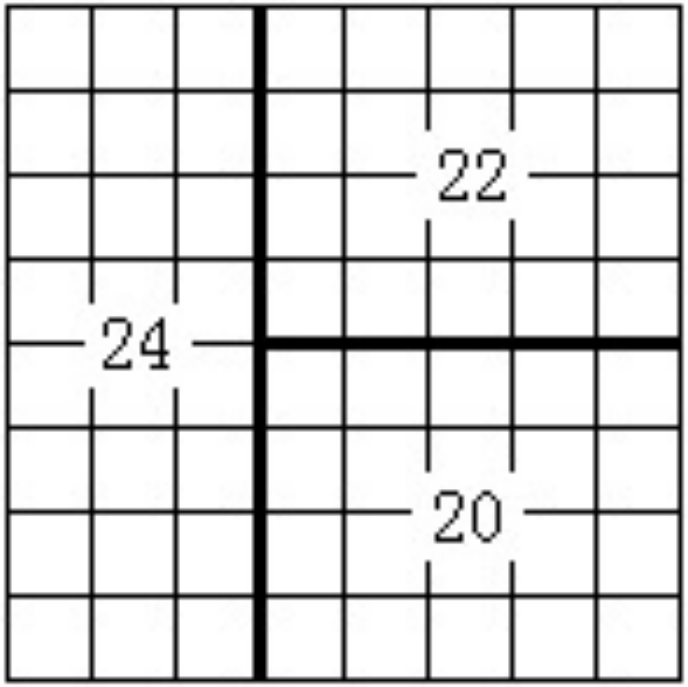
输入

3

1 1 1 1 1 1 1 3
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0
1 1 1 1 1 1 0 3

输出

1.633



分析

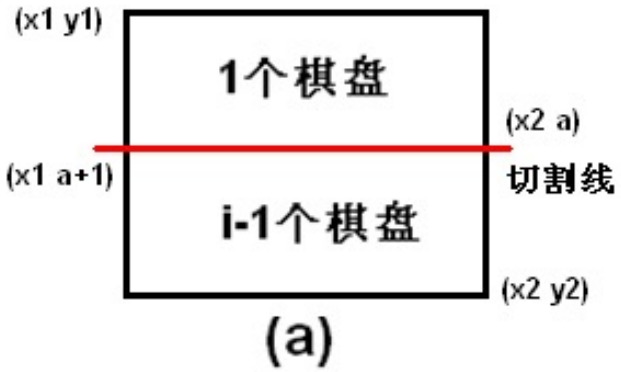
- 由化简后的均方差公式：

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n} x_i^{\,2} - \bar{x}^{2}$$

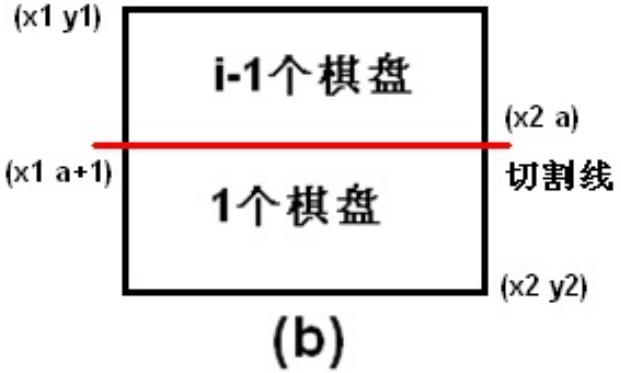
- 可知，均方差的平方为每格数的平方和除以**n**，然后减去平均值的平方，而后者是一个已知数。
- 因此，在棋盘切割的各种方案中，只需使得每个棋盘内各数值的平方和最小即可。
- 因此，我们需要求出各棋盘分割后的每个棋盘各数平方和的最小值，设为**w**，那么
- 答案为：

$$ans = \sqrt{w/n - \bar{x}^{2}}$$

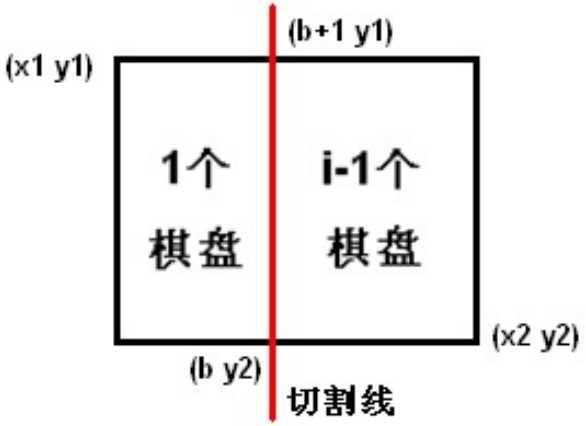
棋盘切割后的四种情况



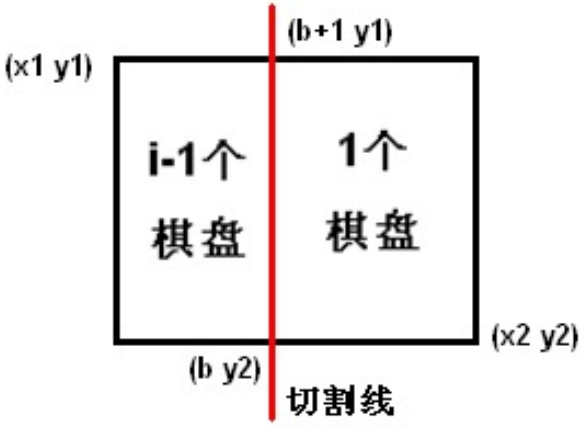
(a)



(b)



(c)



(d)

动态规划

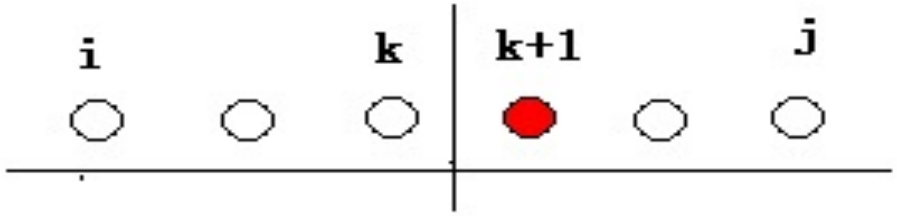
- 设$F(i,x_1,y_1,x_2,y_2)$表示以$[x_1,y_1][x_2,y_2]$为四边形对角线的棋盘切割成i块的各块数值总平方和的最小值，则有：

$$f(i,x_1,y_1,x_2,y_2)=\min\begin{cases} f(i-1,x_1,y_1,x_1,a+1)+D[x_1,y_1][x_2,a], \text{横切,上面不动} \\ f(i-1,x_1,y_1,x_2,a)+D[x_1,y_1],[x_1,a+1], \text{横切,下面不动} \\ f(i-1,b+1,y_1,x_2,y_2)+D[x_1,y_1],[b,y_2], \text{竖切,左面不动} \\ f(i-1,x_1,y_1,b,y_2)+D[b+1,y_1],[x_2,y_2], \text{竖切,右面不动} \end{cases}$$

- $1<=X1,x2,x3,x4<=8,1<=i<=n$。
- 设棋盘边长为m,则状态数为nm^4 ,决策数最多m。
- 先预处理从左上角(1,1)到右下角(i,j)的棋盘和时间复杂度为$O(m^2)$，因此转移为O(1),总时间复杂度为$O(nm^5)$。

总结

- 该类问题的基本特征是能将问题分解成为两两合并的形式。解决方法是对整个问题设最优值，枚举合并点，将问题分解成为左右两个部分，最后将左右两个部分的最优值进行合并得到原问题的最优值。有点类似分治的解题思想。
- 设前i到j的最优值，枚举剖分（合并）点，将(i,j)分成左右两区间，分别求左右两边最优值，如下图。



- 状态转移方程的一般形式如下：

  F(i,j)=Max{F(i,k)+F(k+1,j)+决策，k为划分点

继续题目类型整理