



浙江财经大学

Zhejiang University Of Finance & Economics



区间类动态规划


信智学院 陈琰宏

区间动态规划的特点

区间DP属于线性DP的一种，以区间长度作为DP的阶段，以区间的左右端点作为状态的维度。一个状态通常由被它包含且比它更小的区间状态转移而来。阶段（长度）、状态（左右端点）、决策三者按照由外到内的顺序构成三层循环。

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：石子合并、整数划分，能量项链、凸多边形划分、多边形合并、等。





[2826] 石子合并

在一个操场上一排地摆放着N堆石子。现要将石子**有次序地合并成一堆**。规定每次只能选**相邻**的2堆石子合并成新的一堆，并将新的一堆石子数记为该次合并的得分。

计算出将N堆石子合并成一堆的最小得分。

输入格式

第一行为一个正整数N ($2 \leq N \leq 100$)；

以下N行，每行一个正整数，小于10000，分别表示第i堆石子的个数 ($1 \leq i \leq N$)。

输出格式

一个正整数，即最小得分。

7	4
13 7 8 16 21 4 18	1 3 5 2
239	22



贪心法

$N=7$ 石子数分别为 13 7 8 16 21 4 18。

用贪心法的合并过程如下：

第一次 13 7 8 16 21 4 18 得分 15

第二次 13 15 16 21 4 18 得分 22

第三次 13 15 16 21 22 得分 28

第四次 28 16 21 22 得分 37

第五次 28 37 22 得分 59

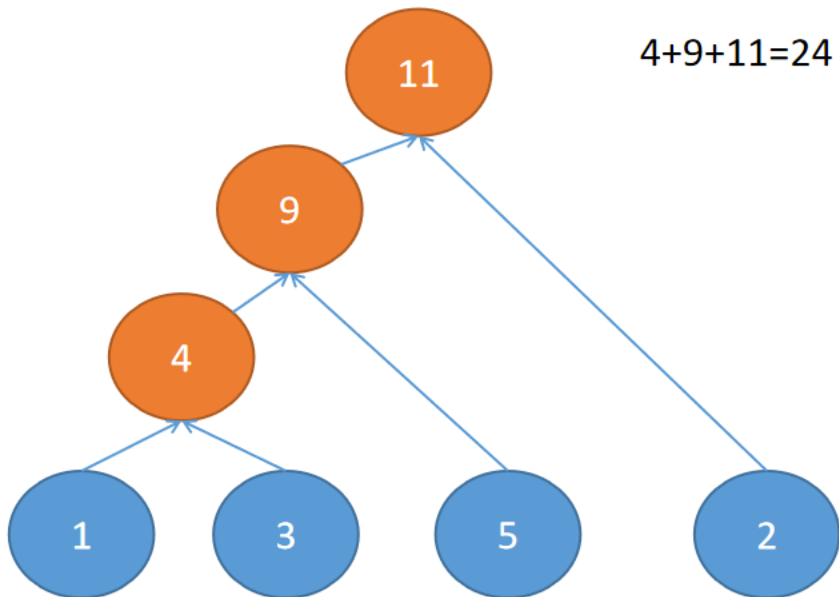
第六次 28 59 得分 87

总分：15+22+28+37+59+87=248>239

显然，贪心法是错误的。

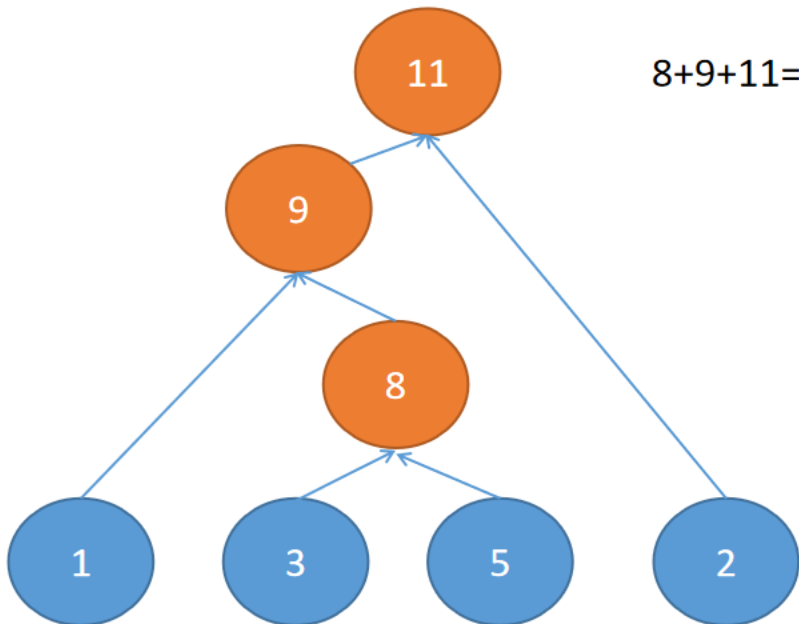
第一种合并方式

4
1 3 5 2
22



第二种合并方式

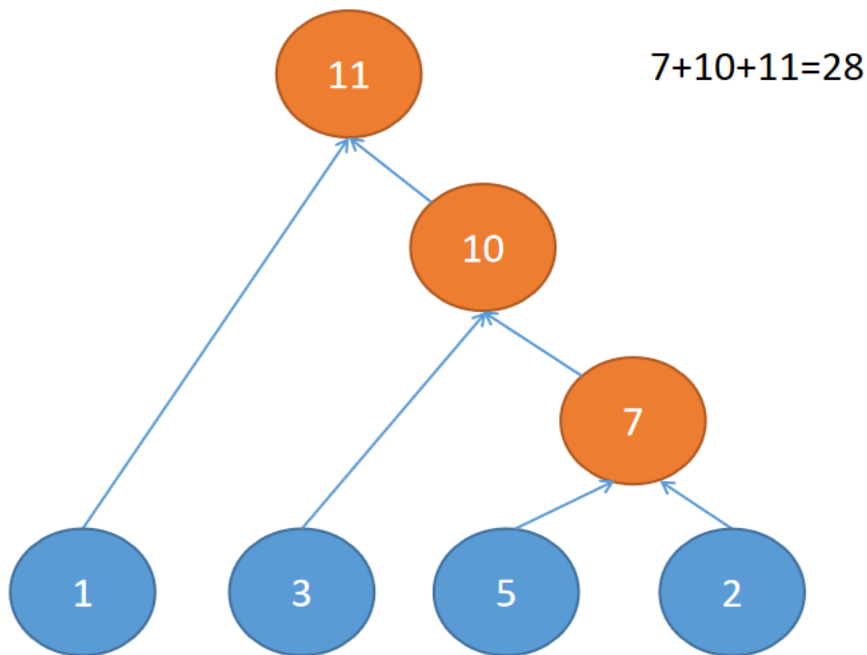
4
1 3 5 2
22



$$8+9+11=28$$

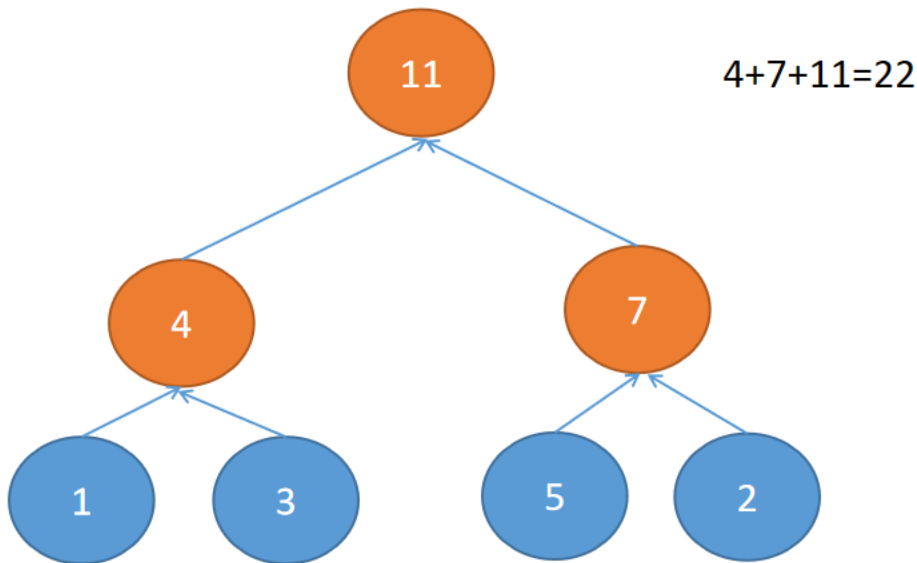
第三种合并方式

4
1 3 5 2
22



第四种合并方式

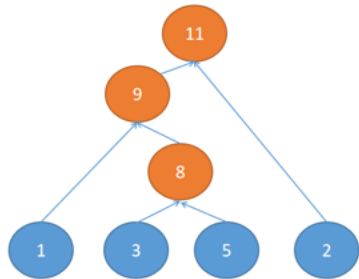
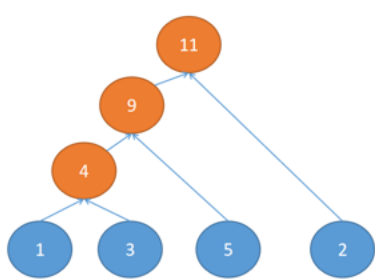
4
1 3 5 2
22





分析

将连续的两个点合并在一起，枚举所有情况，选出一个最大的

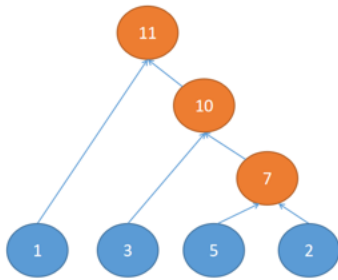
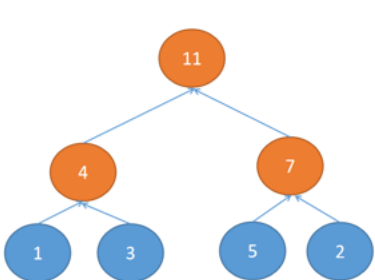


$$4+9+11=24$$

$$8+9+11=28$$

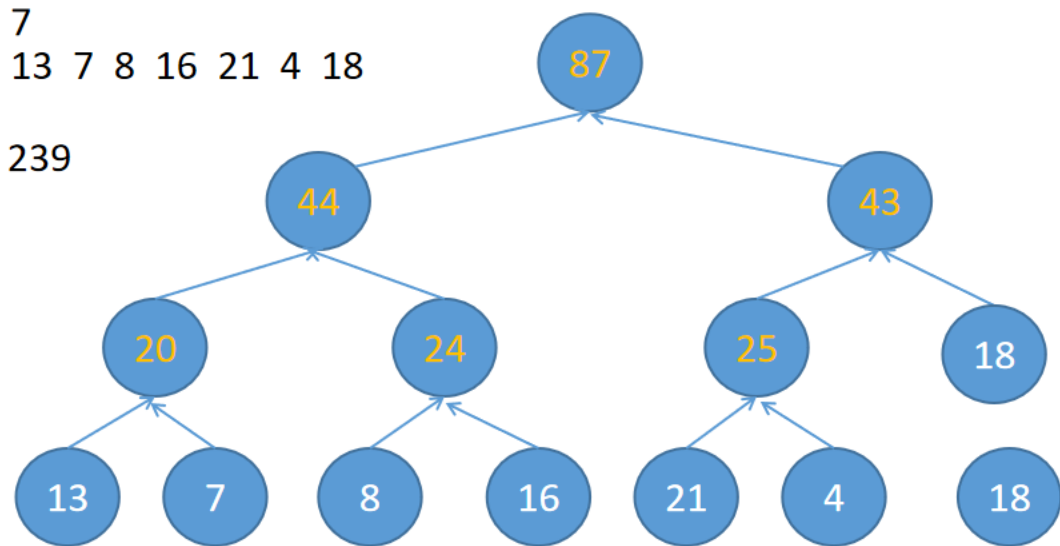
$$4+7+11=22$$

$$7+10+11=28$$





分析：如何枚举？

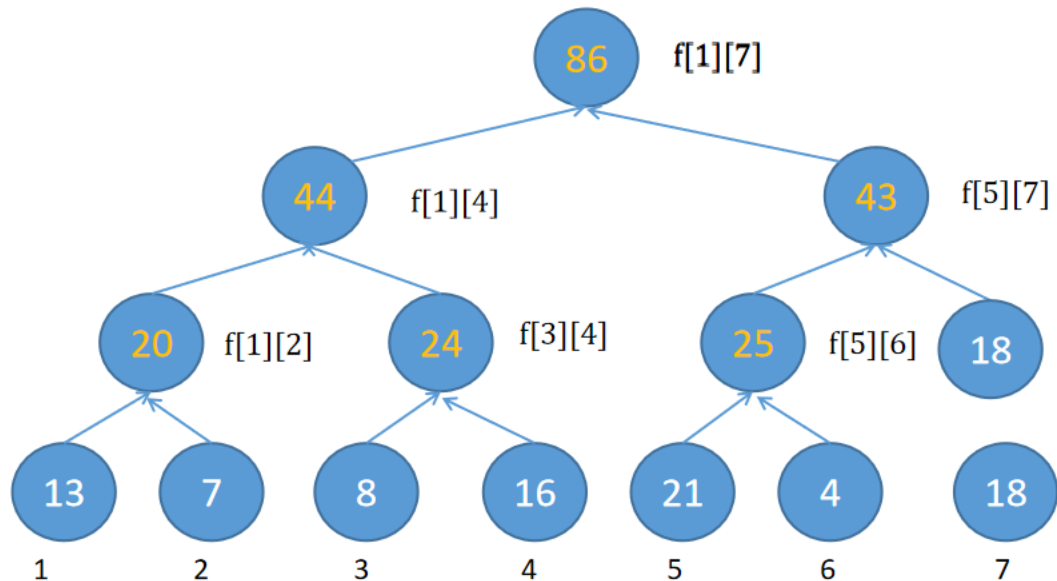


$$20+24+25+44+43+87=243$$



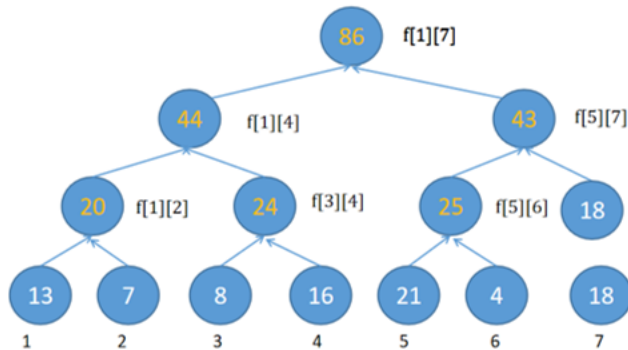
分析：状态描述

$f[i][j]$ 表示将 i 到 j 合并成一堆的方案的分





分析：状态描述

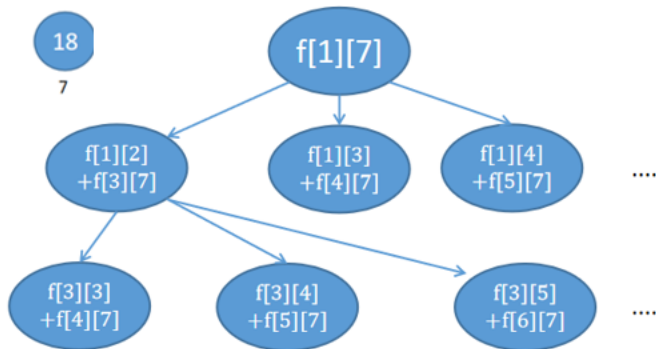


$$f[1][7] = f[1][2] + f[3][7]$$

$$f[1][7] = f[1][3] + f[4][7]$$

$$f[1][7] = f[1][4] + f[5][7]$$

... ..

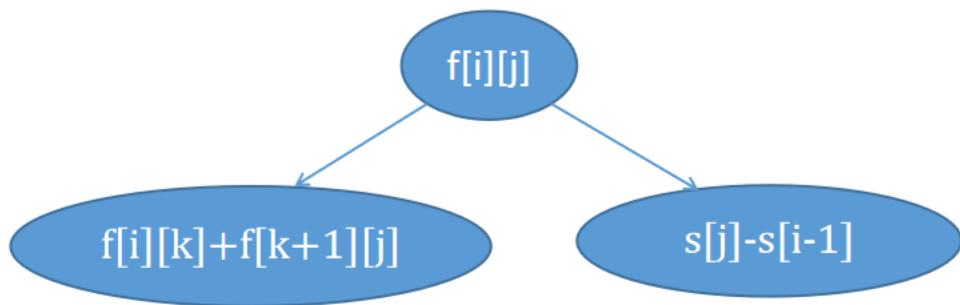


$$f[i][j] = f[i][k] + f[k+1][j]$$



分析：状态转移方程

$$f[i][j] = \min_{i \leq k \leq j-1} (f[i][k] + f[k+1][j] + s[j] - s[i-1])$$



两堆合并的代价，即从*i*到*j*的和
 $s[j]$ 表示前缀和



分析：总结

核心：最后一次合并一定是左边连续的一部分和右边连续的一部分进行合并

状态表示： $f[i][j]$ 表示将 i 到 j 合并成一堆的方案集合，属性 Min

$$(1) i < j \text{ 时, } f[i][j] = \min_{i \leq k \leq j-1} f[i][k] + f[k+1][j] + s[j] - s[i-1]$$

$$(2) i = j \text{ 时, } f[i][i] = 0 \text{ (合并一堆石子代价为 0)}$$

核心代码

算法实现：

阶段（区间长度）

状态（区间起点和终点）

决策（状态转移方程）





核心代码

所有的区间dp问题，第一维都是枚举区间长度，一般 $len = 1$ 用来初始化，枚举从 $len = 2$ 开始，第二维枚举起点 i （右端点 j 自动获得， $j = i + len - 1$ ）

```
for (int i = 1; i <= n; i++) {  
    dp[i][i] = 初始值  
}  
for (int len = 2; len <= n; len++) //区间长度  
    for (int i = 1; i + len - 1 <= n; i++) { //枚举起点  
        int j = i + len - 1; //区间终点  
        for (int k = i; k < j; k++) { //枚举分割点, 构造状态转移方程  
            dp[i][j] = max(dp[i][j], dp[i][k] + dp[k + 1][j] + w[i][j]);  
        }  
    }  
}
```


源代码



```
1 #include<bits/stdc++.h>
2 using namespace std;
3 int n,a[101],f[101][101],s[101];
4 int main(){
5     cin>>n;
6     for(int i=1;i<=n;i++){
7         cin>>a[i];
8         s[i]=s[i-1]+a[i];
9     }
10    for(int len=2;len<=n;len++){
11        for(int i=1;i+len-1<=n;i++){
12            int j=i+len-1;
13            f[i][j]=0x3f3f3f;
14            for(int k=i;k<j;k++){
15                f[i][j]=min(f[i][j],f[i][k]+f[k+1][j]+s[j]-s[i-1]);
16            }
17        }
18    }
19    cout<<f[1][n]<<endl;
20 }
```



记忆化搜索写法:

```
9 // 记忆化搜索: dp的记忆化递归实现
10 int dp(int i, int j) {
11     if (i == j) return 0; // 判断边界
12     // 一个数, 不需要合并
13     if (f[i][j] != -1) return f[i][j];
14
15     f[i][j] = 1e8;
16     for (int k = i; k <= j - 1; k++)
17         f[i][j] = min(f[i][j], dp(i,k)+dp(k+1, j)+s[j]-s[i-1]);
18
19     return f[i][j];
20 }
```



记忆化搜索写法:

```
4 using namespace std;
5 const int N = 307;
6 int a[N], s[N];
7 int f[N][N];
8
9 // 记忆化搜索: dp的记忆化递归实现
10 田 int dp(int i, int j) {
21
22 田 int main() {
23     int n;
24     cin >> n;
25 田     for (int i = 1; i <= n; i++) {
26         cin >> a[i];
27         s[i] += s[i - 1] + a[i];
28     }
29     memset(f, -1, sizeof f);
30     cout << dp(1, n) << endl;
31     return 0;
32 }
```



[2827] 乘积最大

设有一个长度为N的数字串，要求选手使用K个乘号将它分成K+1个部分，找出一种分法，使得这K+1个部分的乘积最大。

同时，为了帮助选手能够正确理解题意，主持人还举了如下的一个例子：

有一个数字串：312，当N=3，K=1时会有以下两种分法：

- 1) $3*12=36$
- 2) $31*2=62$

这时，符合题目要求的结果是： $31*2=62$ 。

现在，请你帮助你的好朋友XZ设计一个程序，求得正确的答案。



分析:

本道题是一道区间DP问题，而且对阶段数有限制 k 。可以将其按照插入乘号数来进行划分。如果插入 k 个乘号，可以把问题看做 k 个阶段的决策问题。

1、然要用一个数组用来记录组成结果的值，在合并石子中是前 i 堆石子的个数，在本道题中，是数字的大小，也就是用 $a[j][i]$ 表示从第 j 位到第 i 位所组成的自然数。

2、用 $f[j][k]$ 存储阶段 k 的每一个状态，表示前 j 位的数字用 k 个乘号能够得到的最大的结果（划分成小问题）

3、边界条件， $f[j][0]$ 当 $k=0$ 时，表示没有乘号，则 $f[j][0]$ 就是前 j 位组成的数字 $f[j][0] = a[1][j]$ 。



分析:

注意：还是阶段、状态、决策三层循环。

在考虑加一个乘号的时候要考虑这个乘号应该放在当前面有 $k-1$ 的乘号时，后面的哪个位置比较好。也就是说要确定前 j 位用 k 个乘号 $f[i][k]$ ($j \geq k+1$) 的结果，应该从有 $k-1$ 个乘号的各种方案(前几位中已经包括了 $k-1$ 个乘号)中选择哪种方案将第 k 个乘号加在后面之后结果最大。

要明白有 k 个乘号时，数字至少要有 $k+1$ 位。有 $k-1$ 个乘号时，数字至少要有 k 位。

状态转移方程：

$$f[i][k] = \max (f[j][k-1] * a[j+1][i]) (j < i)$$

核心代码:



```
2  for(int len=1;len<=k;len++)
3  {
4      for(int i=len+1;i<=n;i++)//k个乘号要有k+1位的数字
5      {
6          for(int j=len;j<i;j++)//当有k-1个乘号的时候,最少有k个数字
7              f[i][len]=max(f[i][len],f[j][len-1]*a[j+1][i]);
8      }
9  }
```





源代码：

```
4 #include<string.h>
5 #include<math.h>
6 using namespace std;
7 //仍然要用一个数组用来记录组成结果的值，在合并石子中是前i堆石子的个数
8 //在本道题中，是数字的大小，也就是用a[j][i]表示从第j位到第i位所组成的自然数
9 //k个阶段的决策问题
10
11 int main()
12 {
13     int n,k;
14     cin>>n>>k;
15     long long int a[10][10]; //表示从第j位到第i位的数值大小
16     char s[11];
17     for(int i=1;i<=n;i++)
18         cin>>s[i];
19     for(int i=1;i<=n;i++) //先将自己一个数的时候初始化一下
20         a[i][i]=s[i]-48;
21     for(int j=1;j<n;j++)
22     {
23         for(int i=j+1;i<=n;i++)
24             a[j][i]=a[j][i-1]*10+s[i]-48;
25     }
```




源代码:

```
27 long long int f[11][7]; //f[i][k]表示在前i位数中插入k个乘号所得到的的最大值
28 for(int i=0; i<=10; i++) //初始化 因为是乘积 所以先初始化为1
29     for(int j=0; j<=6; j++)
30         f[i][j]=1;
31 for(int j=1; j<=n; j++) //初始化
32     f[j][0]=a[1][j]; //边界条件 当没有乘号的时候, 结果值就是数值
33
34 //过程
35 for(int len=1; len<=k; len++)
36 {
37     for(int i=len+1; i<=n; i++) //k个乘号要有k+1位的数字
38     {
39         for(int j=len; j<i; j++) //当有k-1个乘号的时候, 最少有k个数字
40             f[i][len]=max(f[i][len], f[j][len-1]*a[j+1][i]);
41     }
42 }
43 cout<<f[n][k]<<endl;
```



[3694] 环形石子合并

在一个圆形操场的四周摆放N堆石子, 现要将石子有次序地合并成一堆. 规定每次只能选相邻的2堆合并成新的一堆, 并将新的一堆的石子数, 记为该次合并的得分.

试设计出一种算法, 计算出将N堆石子合并成一堆的最小得分和最大得分。

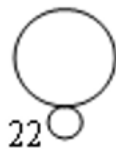
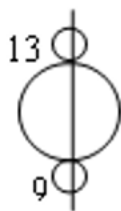
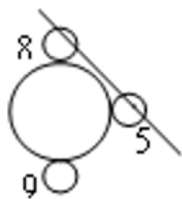
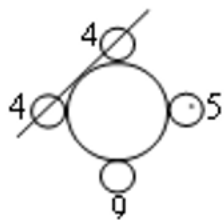
输入格式

数据的第1行试正整数N, $1 \leq N \leq 100$, 表示有N堆石子.
第2行有N个数, 分别表示每堆石子的个数

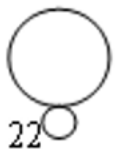
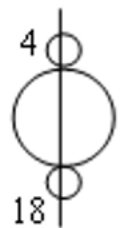
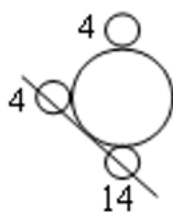
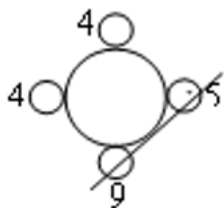
输出格式

输出共2行, 第1行为最小得分, 第2行为最大得分

示例



$$\text{总得分} = 8 + 13 + 22 = 43$$



$$\text{总得分} = 14 + 18 + 22 = 54$$



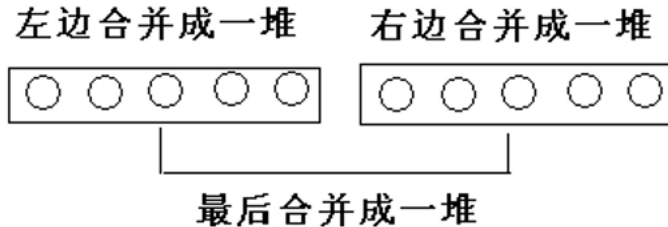
分析

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。
 $F_{\max}(i, j)$ 表示将从第 i 堆石子合并到第 j 堆石子的最大的得分
 $F_{\min}(i, j)$ 表示将从第 i 堆石子合并到第 j 堆石子的最小的得分

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

同理,

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

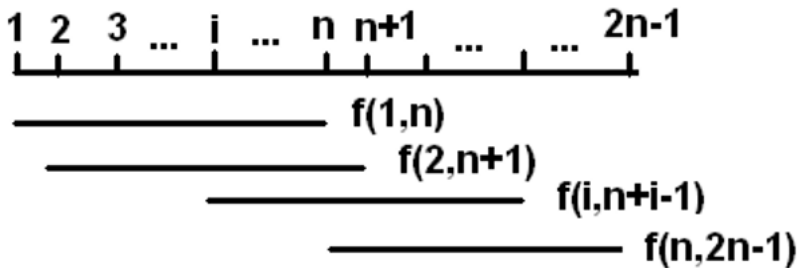
- $F_{\max}[i, i] = 0, F_{\min}[i, i] = 0$
- 时间复杂度为 $O(n^3)$





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 $f(1, n), f(2, n+1), \dots, f(n, 2n-1)$ 取最优值即可。
- 时间复杂度为 $O(8n^3)$ ，如下图：





猜想

- 合并第*i*堆到第*j*堆石子的最优断开位置*s*[*i*, *j*]要么等于*i*+1, 要么等于*j*-1, 也就是说最优合并方案只可能是:



$$\{ (i) (i+1 \dots j-1) (j) \}$$

或者



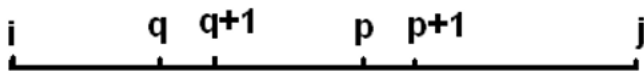
$$\{ (i \dots i+1) (i+2 \dots j) (j+1) \}$$



证明



- 设合并第*i*堆到第*j*堆石子的断开位置 *p*，且*i*<*p*<*j*-1。设在*i*,*p*之间存在一种断开方案 q 。如下图：



情况1: $t[i, p] \leq t[p+1, j]$

合并方案1: $\{ [(i \cdots q) (q+1 \cdots p)] (p+1 \cdots j) \}$ ，它的得分：

$$F1 = F_{\max}(i, q) + F_{\max}(q+1, p) + F_{\max}(p+1, j) + t[i, j] + \underline{t[i, p]}$$

合并方案2: $\{ (i \cdots q) [(q+1 \cdots p) (p+1 \cdots j)] \}$ ，它的得分：

$$F2 = F_{\max}[i, q] + F_{\max}[q+1, p] + F_{\max}[p+1, j] + t[i, j] + \underline{t[q+1, j]}$$

由于 $q < p$ ，所以 $t[i, p] \leq t[p+1, j] < t[q+1, j]$ ，所以 $F1 < F2$ 。

这说明方案2的合并方案比方案1好，因此这种情况下应尽可能靠左边决策。

情况2: $t[i, p] > t[p+1, j]$ 与情况1是对称。（证明略）





状态转移方程

- 设 $t[i, j]$ 表示从第 i 堆到第 j 堆石子数总和。
 $F_{\max}(i, j)$ 表示将从第 i 堆石子合并到第 j 堆石子的最大的得分
 $F_{\min}(i, j)$ 表示将从第 i 堆石子合并到第 j 堆石子的最小的得分

$$F_{\max}(i, j) = \max \{F_{\max}(i+1, j), F_{\max}(i, j-1)\} + t[i, j]$$

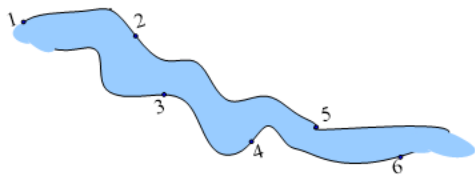
同理,

$$F_{\min}(i, j) = \min \{F_{\min}(i+1, j), F_{\min}(i, j-1)\} + t[i, j]$$

- $F_{\max}[i, i] = 0, F_{\min}[i, i] = 0$
- 时间复杂度为 $O(n^2)$

[3889] 长江一日游——游艇租赁

长江游艇俱乐部在长江上设置了 n 个游艇出租站，游客可以在这些出租站租用游艇，在下游的任何一个游艇出租站归还游艇。游艇出租站 i 到游艇出租站 j 之间的租金为 $r(i, j)$ 。现在要求出从游艇出租站1到游艇出租站 n 所需的最少的租金。



当要租用游艇从一个站到另外一个站时，中间可能经过很多站点，不同的停靠站策略就有不同的租金。

如果穷举所有的停靠策略，例如一共有10个站点，当求子问题4个站点的停靠策略时，子问题有

(1, 2, 3, 4) , (2, 3, 4, 5) , (3, 4, 5, 6)
(4, 5, 6, 7) , (5, 6, 7, 8) , (6, 7, 8, 9)
(7, 8, 9, 10) 。



如果再继续求解子问题，会发现大量的子问题重叠，其算法时间复杂度为 2^n ，暴力穷举的办法是不可取的。



分析

- (1) 确定状态。 $dp[i][j]$ 表示第*i*个站点到第*j*个站点的最少租金。
- (2) 划分阶段。 区间长度。
- (3) 决策选择。 原问题与子问题之间的关系。

$i, i+1, \dots, k$

$k, k+1, \dots, j$

状态转移方程: $dp[i][j] = \min_{i < k < j} \{dp[i][k] + dp[k][j], dp[i][j]\}$

- (4) 边界条件。 $i < j, dp[i][j] = r[i][j]$ 。
 - (5) 求解目标。 $dp[1][n]$ 。
-



边界条件: $i < j$, $dp[i][j] = r[i][j]$ 。

状态转移方程:

$$dp[i][j] = \min_{i < k < j} \{dp[i][k] + dp[k][j], dp[i][j]\}$$

$r[i][j]$	1	2	3	4	5	6
1		2	6	9	15	20
2			3	5	11	18
3				3	6	12
4					5	8
5						6
6						

$dp[i][j]$	1	2	3	4	5	6
1		2	6	9	15	20
2			3	5	11	18
3				3	6	12
4					5	8
5						6
6						



求解目标: $dp[1][6]=15$

dp[]	1	2	3	4	5	6
1		2	5	7	11	15
2			3	5	9	13
3				3	6	11
4					5	8
5						6
6						





算法实现：

阶段（区间长度）

状态（区间起点和终点）

决策（状态转移方程）



算法实现:

```
void rent(){
    for(int d=3;d<=n;d++){//区间长度d
        for(int i=1;i<=n-d+1;i++){//状态起点i, 终点j
            int j=i+d-1;
            for(int k=i+1;k<j;k++){//枚举决策点k
                if(dp[i][j]>dp[i][k]+dp[k][j])
                    dp[i][j]=dp[i][k]+dp[k][j];
            }
        }
    }
}
```





算法分析

时间复杂度：3层for循环中嵌套，执行次数为 $O(n^3)$ ，
时间复杂度为 $O(n^3)$ 。

空间复杂度：数组 $dp[n][n]$ ，空间复杂度为 $O(n^2)$ 。





延伸思考

上面得到的最优值只是第1个站点到第 n 个站点之间的最少租金，并不知道停靠了哪些站点，如果还想知道停靠的站点，怎么办？

状态转移方程：

$$dp[i][j] = \min_{i < k < j} \{dp[i][k] + dp[k][j], dp[i][j]\}$$

用辅助数组 $s[i][j]$ 记录各个子问题 $(i..j)$ 的最优决策 k （停靠站点）。

构造最优解:

dp[]	1	2	3	4	5	6
1		2	5	7	11	15
2			3	5	9	13
3				3	6	11
4					5	8
5						6
6						

s[][]	1	2	3	4	5	6
1		0	2	2	2	2
2			0	0	3	4
3				0	0	4
4					0	0
5						0
6						

答案: 1—2—4—6。



```

8  for(int p=1;p<=t;p++){
9      cin >> n;
10     memset(m,0,sizeof(m));
11     for (int i = 1; i <= n - 1; i++)
12         {//初始化, 将所有r(i,j)都先存在数组m中
13         for (int j = i + 1; j <= n; j++)
14             cin >> m[i][j];
15     }
16     for (int r = 2; r <= n; r++)
17         {//接着从长度为 2 的开始找较优解 (比如说从 1 到 2 就是长度为 2 的)
18         //直到找到长度为 n 的
19         for (int i = 1; i <= n - r + 1; i++)
20             {
21                 int j = i + r - 1; // r(i, j) 的长度为 r
22                 for (int k = i; k <= j; k++)
23                     {//在 i 到 j 中找某一站 k, 使得r(i,k)+r(k,j)最小
24                     int t = m[i][k] + m[k][j];
25                     if (t < m[i][j])
26                         m[i][j] = t; //用较优解替换原来的r(i,j)
27                     }
28             }
29     }
30     cout << m[1][n] << endl;

```

```

1  #include<bits/stdc++.h>
2  using namespace std;
3  int main()
4  {
5      int n,t;
6      int m[200][200] = { 0 };
7      cin>>t;
8      for(int p=1;p<=t;p++){
33
34
35 }

```



[3695] 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 $m \times r \times n$ （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。

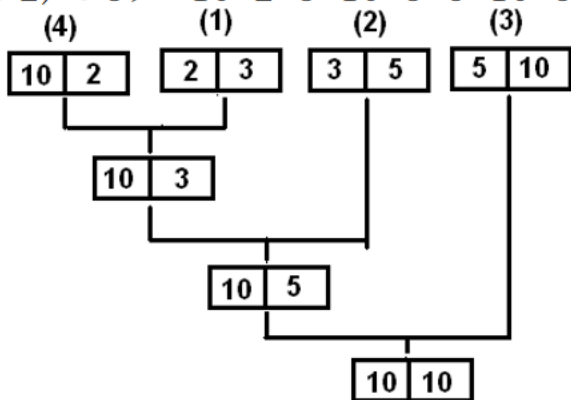


- 分析样例：

$N=4$ ，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 \oplus 表示两颗珠子的聚合操作，释放总能量：

$$((4 \oplus 1) \oplus 2) \oplus 3 = 10 * 2 * 3 + 10 * 3 * 5 + 10 * 5 * 10 = 710$$



分析



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：

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

边界条件： $F(i, i) = 0$

$$1 \leq i < k < j \leq n$$

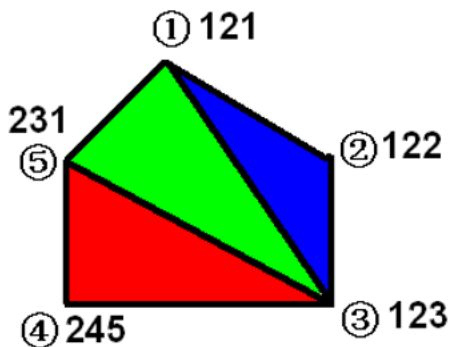
至于圈的处理，与石子合并方法完全相同，时间复杂度 $O(8n^3)$ 。



[3696] 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?





上述凸五边形分成 $\triangle 123$ ， $\triangle 135$ ， $\triangle 345$

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）

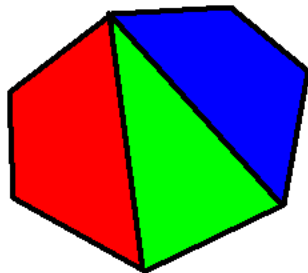


图1

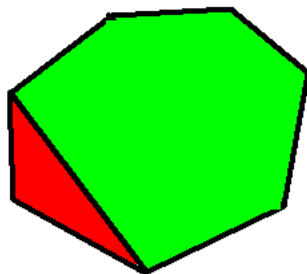


图2



动态规划

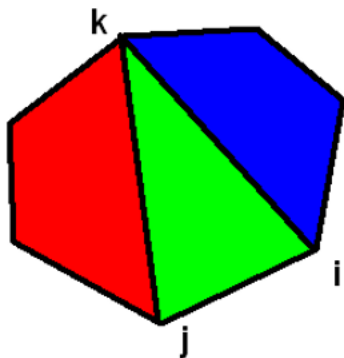
如果我们按顺时针将顶点编号，则可以相邻两个顶点描述一个凸多边形。

设 $f(i, j)$ 表示 $i \sim j$ 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

$$f(i, j) = \min \{ f(i, k) + f(k, j) + a[i] * a[j] \}$$

$$1 \leq i < k < j \leq n$$

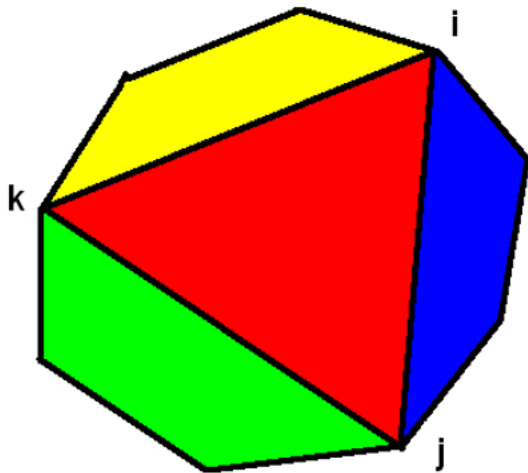
时间复杂度 $O(n^3)$

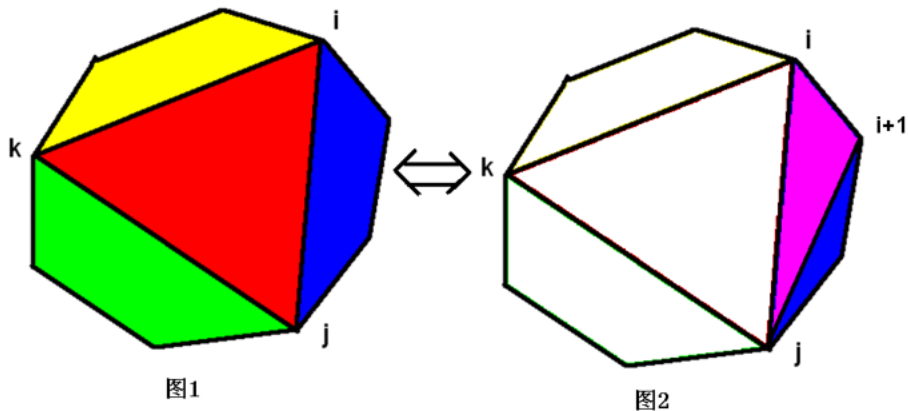


讨论



➤为什么可以不考虑这种情况？





- 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。