区间类动态规划

主讲老师：党东

【区间类动态规划】

区间型动态规划是线性动态规划的拓展，它在分阶段地划分问题时，与阶段中元素出现的顺序和由前一阶段的区间中哪些元素合并而来有很大的关系。

如对于状态f[i][j]，它表示划分的阶段为j，状态出现的位置为i，**它的值取决于第i个元素出现的位置和i到j这段区间的值**。这一类型的动态规划，阶段特征非常明显，求最优值时需预先设置阶段内的区间统计值，还要分动态规划的起始位置来判断。区间型动态规划的典型应用有石子合并、矩阵乘积等。

区间合并类DP

合并类动态规划的特点：

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1】合并石子 --1378

Description

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。

Input

第一行为一个正整数N (2 ≤ N ≤ 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 ≤ i ≤ N)。

Output

一个正整数，即最小得分。

Sample Input

```
7
13
7
8
16
21
4
18
```

Sample Output

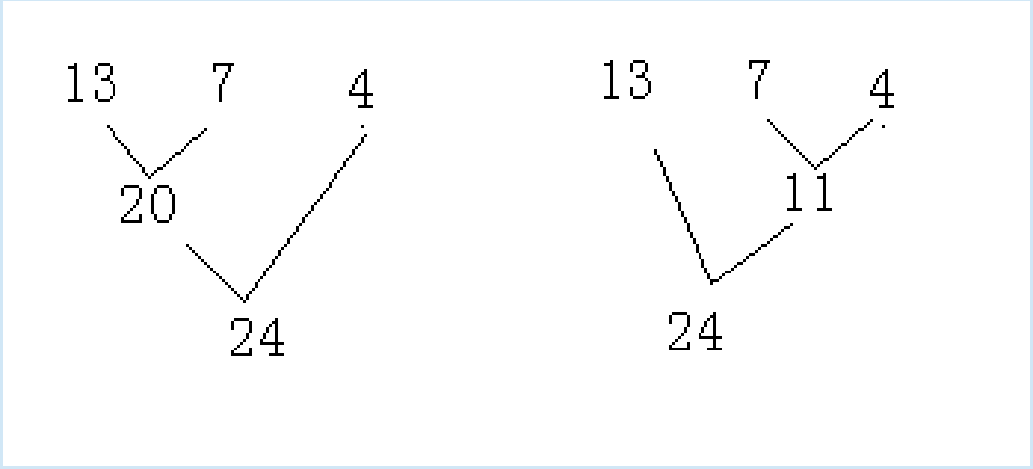
```
239
```

【例1】合并石子 --1378

【思路点拨】

（1）当N=2时，仅有一种堆法，因此总的归并代价为2堆沙子数量之和

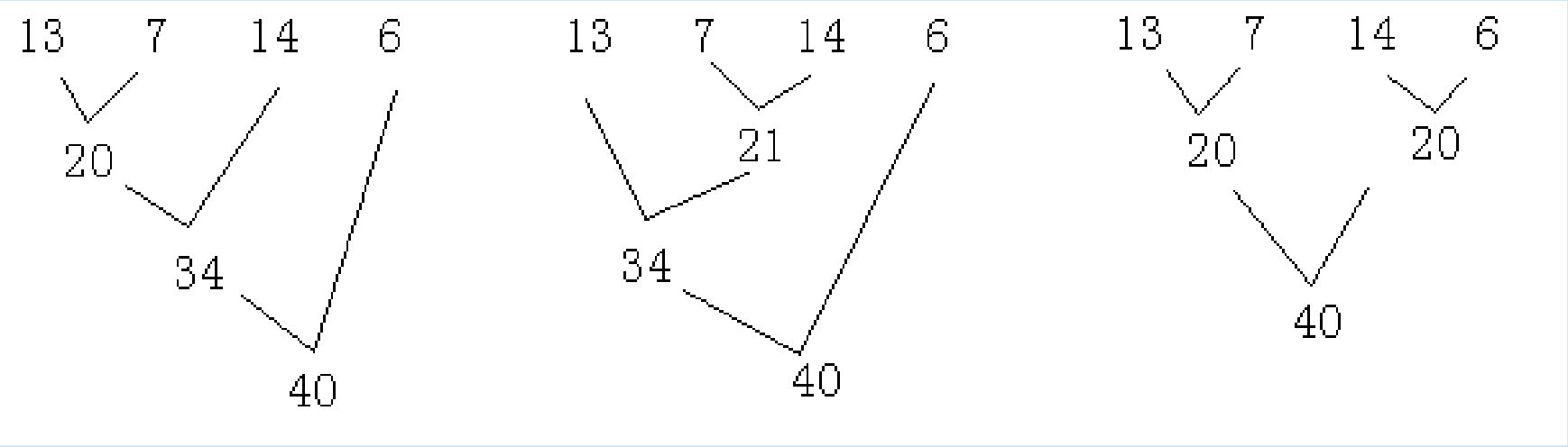
（2）当N=3时，有2种堆法。从上图可看到它们总的代价分别为44和35。



【例1】合并石子 --1378

【思路点拨】

（3）当N=4时，共有五种归并的方法，它们的总代价分别为94, 95, 80, 88, 87，最小的归并总代价为80（C）。



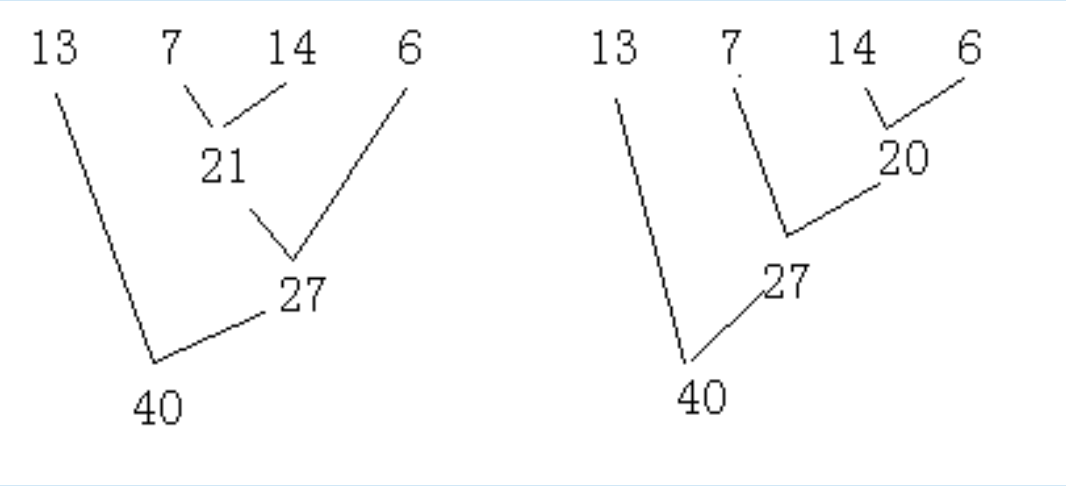
(A)　　(B)　　(C)

总代价=20+34+40=94　总代价=21+34+40=95　总代价=20+20+40=80

【例1】合并石子 --1378

【思路点拨】

（3）当N=4时，共有五种归并的方法，它们的总代价分别为94, 95, 80, 88, 87，最小的归并总代价为80（C）。



(D)　　(E)

总代价=21+27+40=88　　总代价=20+27+40=8 7

【例1】合并石子 --1378

【思路点拨】

第一类包括（A），（B），基本方法是归并前面的三堆，再归并最后一堆，由于最后一堆归并的代价是相同的，所以在归并前面三堆时不同的方法将产生不同的结果。（A）的代价小于（B）的代价，因此（A）方法优。

第二类仅有一种方法（C），分别归并2堆的代价为２０，２０，相加为４０。

第三类包括（D），（E），基本方法是先归并后面三堆，再归并第一堆，由于最后一次归并代价是相同的，所以归并后三堆的方法不同将产生不同的结果。（D）的代价大于（E）的代价，因些方法（E）优。

【例1】合并石子 --1378

【思路点拨】

引入数据结构：

1、**F(i,j) 表示从i堆到j堆石子归并的最小代价数。**

如上讨论可知：　F(1,4)=MIN{ F(1,3), F(1,2)+F(3,4),F(2,4)}　+　40

而F(1,3)=min{F(1,2),F(2,3)}+34

F(1,2)=20　　F(3,4)=20　　F(2,3)=21

F(2,4)=MIN{F(2,3),F(3,4)}+27

2、**g[i]：表示从第1堆到第i堆的石子的重量之和；//前缀和**

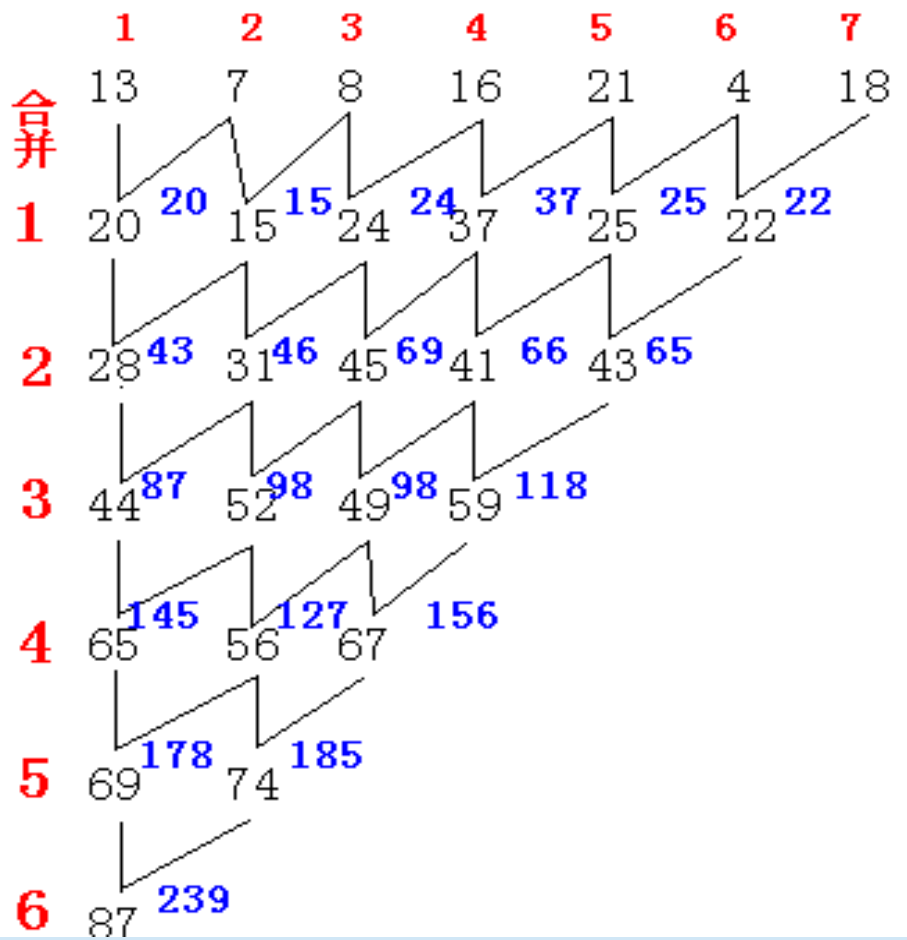
这样就有一般情况：

F(1,N)=MIN{ F(1,N-1), F(1,2)+F(3,N), F(1,3)+F(4,N), ……., F(2,N)}+G(N)

【例1】合并石子 --1378

【思路点拨】

f[i][j]的计算过程：



【例1】合并石子 --1378

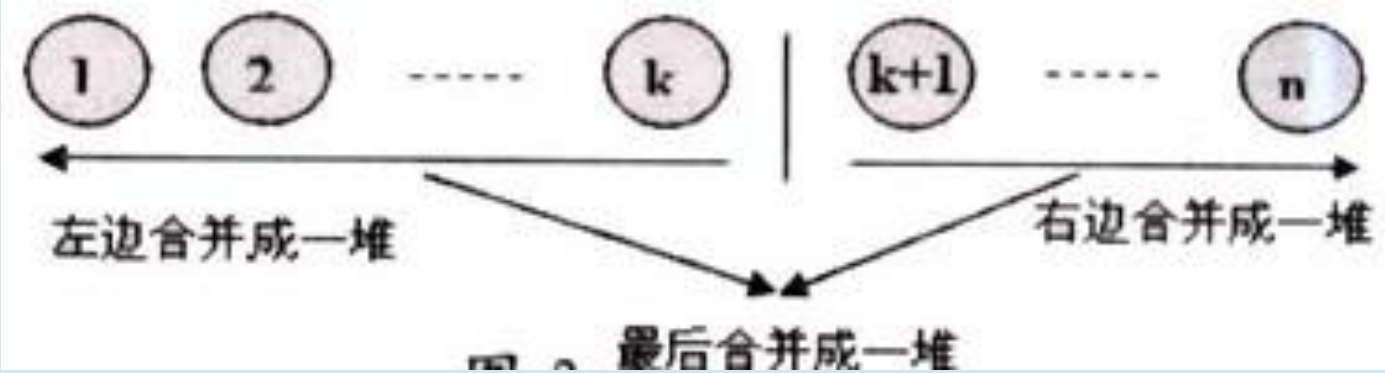
【状态转移推导】

假设有2堆石子，显然只有1种合并方案；

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。如下图设最后合并的两堆石子的位置为K和K+1：



【例1】合并石子 --1378

【题目分析】

1．阶段和状态：设n堆石子的数量依次存储在a[1]、a[2]、…、a[n]中，由于本问题中只能是相邻两堆石子才能合并（注意与合并果子的区别）：

g[i]：表示从第1堆到第i堆的石子的重量之和，即前缀和。

fmin[i][j]：表示从第i堆到第j堆石子合并为一堆石子的最小代价；

题目的阶段是：第几次合并t(1<=t<=n-1)，状态是：从第几堆开始合并石子i(1<=i<=n-t)

2．状态转移方程：

f(1,n)=min{ f(1,n-1), f(1,2)+f(3,n), f(1,3)+f(4,n), ……., f(2,n)}+g(1,n)

初始化：g[i]=g[i-1]+a[i];fmax[i][i]=0; fmin[i][i]=0;

状态转移方程：fmin[i][j]=min{f[i][k]+f[k+1][j]+g[j]-g[i-1]}(i<=k<=j-1)

Answer= fmin[1][n]

【例1】合并石子 --1378

【核心代码】

```
int main()
{
        cin>>n;
        for(int i=1;i<=n;i++)
        {   cin>>a[i];   g[i]=g[i-1]+a[i];  //求前缀和
        }
        for(int t=1;t<n;t++)   //合并阶段
                for(int i=1;i<=n-t;i++)  //起点
                {
                        int j=i+t;          //当前阶段下的终点
                        f[i][j]=0xfffffff;
                        for(int k=i;k<j;k++)  //状态转移，求f[i][j]的最优值
                                f[i][j]=min(f[i][j],f[i][k]+f[k+1][j]);
                        f[i][j]+=g[j]-g[i-1]; //累加当前合并需要的代价
                }
        cout<<f[1][n];
        return 0;
}
```

【例2】最小最大代价子母树（石子归并问题）--1510

Description

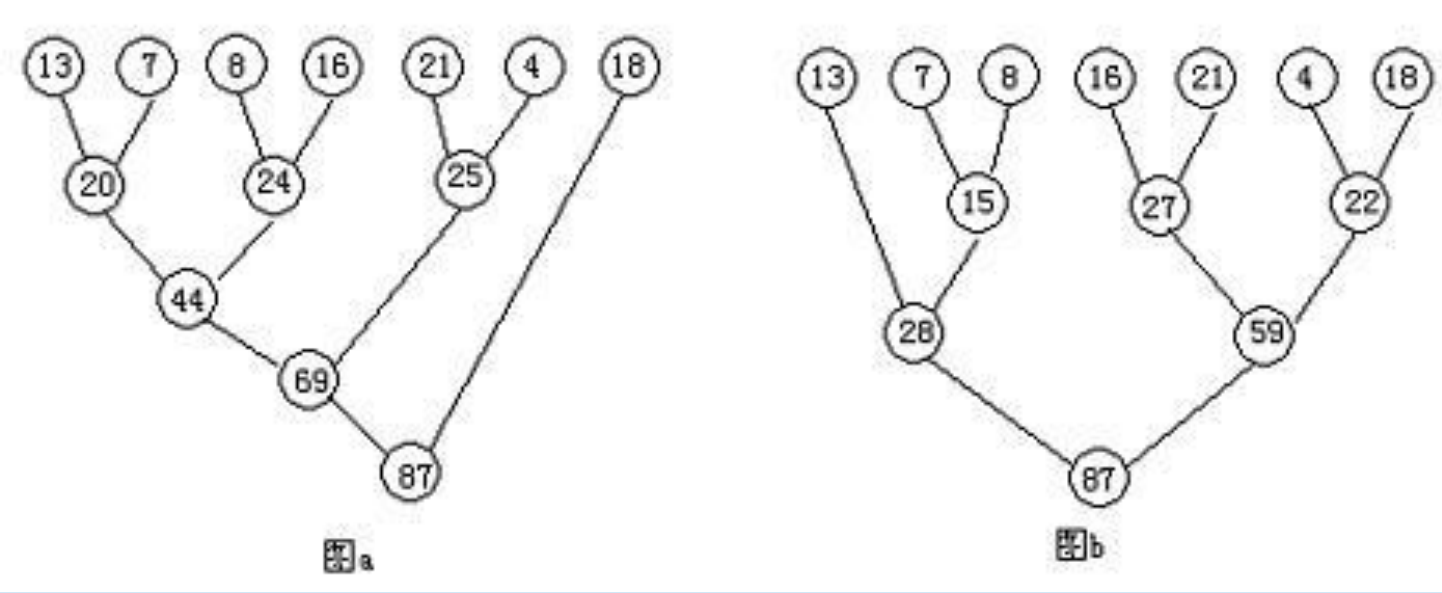
设有n堆石子排成一排，其编号为1、2、3、…、n（n<=100）。每堆石子的数量用：a[1]、a[2]、…、a[n] 表示，现将这n堆石子归并成一堆，归并的规则：

◆每次只能将相邻两堆归并成一堆，即：第 1 堆石子 a[1] 只能与第 2 堆石子 a[2] 归并，最后一堆石子 a[n] 只能与 a[n-1] 归并，中间的石子 a[i] 只能与 a[i-1] 或 a[i+1] 归并；

◆每次归并的代价是两堆石子的重量之和。

【例2】最小最大代价子母树（石子归并问题）--1510

例如：假设有这样7堆石子，各堆石子的数量分别是：13 7 8 16 21 4 18，将它们归并成一堆有多种归并方法，下图表示了其中两种归并方法：



图a所示归并方式的总代价为：20+24+25+44+69+87=267

图b所示归并方式的总代价为：15+27+22+28+59+87=248

由此可见，不同的归并方式所得到的总代价是不一样的。那么当给出了n堆石子的数量之后，求出最小的总代价和最大的总代价。

【例2】最小最大代价子母树（石子归并问题）--1510

Input

第一行一个整数 n，表示有 n 堆石子；

第二行有n个整数，表示每堆石子的个数；

Output

第一行一个整数，表示合并的最小代价；

第二行一个整数，表示合并的最大代价；

Sample Input

4

13 7 14 6

Sample Output

80

95

【例2】最小最大代价子母树（石子归并问题）--1510

【题目分析】

1．阶段和状态：设n堆石子的数量依次存储在a[1]、a[2]、…、a[n]中，由于本问题中只能是相邻两堆石子才能合并（注意与合并果子的区别）：

g[i]：表示从第1堆到第i堆的石子的重量之和，即前缀和。

fmin[i][j]：表示从第i堆到第j堆石子合并为一堆石子的最小代价；

fmax[i][j]：表示从第i堆到第j堆石子合并为一堆石子的最大代价；

题目的阶段是：第几次合并t(1<=t<=n-1)，状态是：从第几堆开始合并石子i(1<=i<=n-t)

2．状态转移方程：

f(1,n)=min{ f(1,n-1), f(1,2)+f(3,n), f(1,3)+f(4,n), ……., f(2,n)}+g(1,n)

初始化：g[i]=g[i-1]+a[i];fmax[i][i]=0; fmin[i][i]=0;

状态转移方程：fmin[i][j]=min{f[i][k]+f[k+1][j]+g[j]-g[i-1]}(i<=k<=j-1)

状态转移方程：fmax[i][j]=max{f[i][k]+f[k+1][j]+g[j]-g[i-1]}(i<=k<=j-1)

Answer= fmin[1][n]和fmax[1][n]

【例2】最小最大代价子母树（石子归并问题）--1510

【参考代码】

```
int a[101],g[101],fmax[101][101],fmin[101][101],n,i,j,k,t;
int main()
{  cin>>n;
   for(i=1;i<=n;i++)cin>>a[i];
   memset(g,0,sizeof(g));
   memset(fmax,0,sizeof(fmax));
   memset(fmin,0,sizeof(fmin));
   for(i=1;i<=n;i++){g[i]=g[i-1]+a[i];fmax[i][i]=0;fmin[i][i]=0;}
   for(t=1;t<=n-1;t++)   //阶段，第几次合并
     for(i=1;i<=n-t;i++) //状态，合并的起始堆
     {  j=i+t; //合并的结束堆
        fmin[i][j]=0xfffffff;fmax[i][j]=0; //初始化
        for(k=i;k<=j-1;k++) //决策，分界点
        {  fmin[i][j]=min(fmin[i][j],fmin[i][k]+fmin[k+1][j]);
           fmax[i][j]=max(fmax[i][j],fmax[i][k]+fmax[k+1][j]);
        }
        fmin[i][j]=fmin[i][j]+g[j]-g[i-1];
        fmax[i][j]=fmax[i][j]+g[j]-g[i-1];
     }
    cout<<fmin[1][n]<<endl;
    cout<<fmax[1][n]<<endl;
}
```

【例3】石子合并（提高版）--1511

Description

在一个园形操场的四周摆放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1个算法,计算出将N堆石子合并成1堆的最小得分和最大得分。

Input

输入数据的第1行试正整数N,1≤N≤1000,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。

Output

输出共2行,第1行为最小得分,第2行为最大得分.

Sample Input

4

4 4 5 9

Sample Output

43

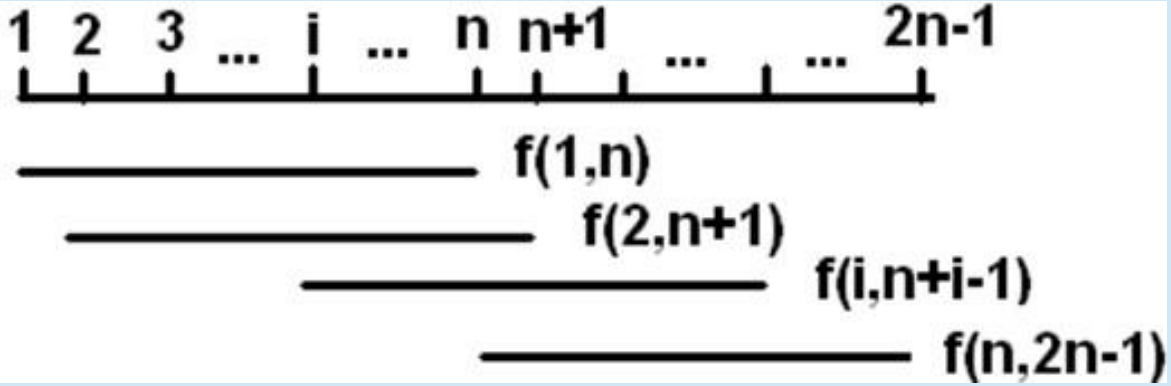
54

【例3】石子合并（提高版）--1511

【问题分析】

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(8n3),如下图：

采用循环队列的形式，见下图：



【例3】石子合并（提高版）--1511

【核心代码】

```
void init()
{  cin>>n;
   for(i=1;i<=n;i++){cin>>a[i];a[i+n]=a[i];}
   for(i=1;i<=n*2;i++){sum[i]=sum[i-1]+a[i];f2[i][i]=0;f1[i][i]=0;}
}
void dp()
{  int j,k,L,ans;
   for(L=2;L<=n;L++)   //以合并的堆数为阶段
     for(i=1;i<=2*n-L+1;i++) //合并的起始位置
     {  j=i+L-1; //合并的结束位置
        f1[i][j]=0x7fffffff/2;f2[i][j]=0;
        for(k=i;k<j;k++)
        {  f1[i][j]=min(f1[i][j],f1[i][k]+f1[k+1][j]);
           f2[i][j]=max(f2[i][j],f2[i][k]+f2[k+1][j]);
        }
        f1[i][j]+=sum[j]-sum[i-1];
        f2[i][j]+=sum[j]-sum[i-1];
     }
   ans1=0x7fffffff/2,ans2=0;
   for(i=1;i<=n;i++)ans1=min(ans1,f1[i][i+n-1]);
   for(i=1;i<=n;i++)ans2=max(ans2,f2[i][i+n-1]);
}
```

【例4】能量项链 --1512

Description

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。并且，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。因为只有这样，通过吸盘（吸盘是Mars人吸收能量的一种器官）的作用，这两颗珠子才能聚合成一颗珠子，同时释放出可以被吸盘吸收的能量。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。

需要时，Mars人就用吸盘夹住相邻的两颗珠子，通过聚合得到能量，直到项链上只剩下一颗珠子为止。显然，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。

例如：设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，(j⊕k)表示第j，k两颗珠子聚合后所释放的能量。则第4、1两颗珠子聚合后释放的能量为：

(4⊕1)=10*2*3=60。

这一串项链可以得到最优值的一个聚合顺序所释放的总能量为

((4⊕1)⊕2)⊕3）=10*2*3+10*3*5+10*5*10=710。

【例4】能量项链 --1512

Input

输入文件的第一行是一个正整数N（4≤N≤100），表示项链上珠子的个数。第二行是N个用空格隔开的正整数，所有的数均不超过1000。第i个数为第i颗珠子的头标记（1≤i≤N），当1≤i < N时，第i颗珠子的尾标记应该等于第i+1颗珠子的头标记。第N颗珠子的尾标记应该等于第1颗珠子的头标记。

至于珠子的顺序，你可以这样确定：将项链放到桌面上，不要出现交叉，随意指定第一颗珠子，然后按顺时针方向确定其他珠子的顺序。

Output

输出文件只有一行，是一个正整数E（E≤2.1*10^9），为一个最优聚合顺序所释放的总能量。

Sample Input

4

2 3 5 10

Sample Output

710

【例4】能量项链 --1512

【思路点拨】

简单地说，题意为：给你一串项链，项链上有n颗珠子，相邻的两颗珠子可以合并(两个合并成一个)，合并的同时会放出一定的能量。不同珠子的合并所释放的能量是不同的。问：按照怎样的次序合并才能使释放的能量最多？

我们用head[i]表示第i颗珠子的头标记，用tail[i]表示第i颗珠子的尾标记，合并两颗相邻珠子所释放的能量为：head[i]*tail[i]*tail[i+1]

合并时不一定按照输入的顺序合并，与石子合并问题类似，**第n次合并，可以归结到第n-1次合并，具有明显的动态规划性质**。

设f[i][j]表示从第i颗珠子合并到第j颗珠子时产生的最大能量。假设最后合并的位置为k和k+1，则有：

f[i][j]=max{f[i][k]+f[k+1][j]+head[i]*tail[k]*tail[j]},1<=i<=k<j<=n

初始化：f[i][i]=0;

Answer=max{f[1][n],f[2][n+1]….+f[n][2n-1]}

时间复杂度为：O(8n3)。

【例4】能量项链 --1512

【核心代码】

```
int head[205],tail[205],f[205][205]={0},ans=0,n,i,t,j,k;
cin>>n;
for(i=1;i<=n;i++){cin>>head[i];head[i+n]=head[i];}//环变成链
for(i=1;i<=2*n-1;i++)tail[i]=head[i+1];tail[2*n]=head[1];//求尾标记
for(i=1;i<=2*n-1;i++)f[i][i]=0; //初始化
for(t=1;t<=n-1;t++) //阶段：合并次数
  for(i=1;i<=2*n-t;i++) //状态：起始位置
  {  j=i+t;  //计算结束位置
     for(k=i;k<=j-1;k++) //决策
        f[i][j]=max(f[i][j],f[i][k]+f[k+1][j]+head[i]*tail[k]*tail[j]);
  }
for(i=1;i<=n;i++)ans=max(ans,f[i][i+n-1]);//枚举断开的位置，求最值
cout<<ans<<endl;
```

【例5】添加括号 --1688

【题目背景】

给定一个正整数序列a(1)，a(2)，...，a(n),(1<=n<=20)

不改变序列中每个元素在序列中的位置，把它们相加，并用括号记每次加法所得的和，称为中间和。

例如:

给出序列是4，1，2，3。

第一种添括号方法:

((4+1)+(2+3))=((5)+(5))=(10)

有三个中间和是5，5，10，它们之和为:5+5+10=20

第二种添括号方法

(4+((1+2)+3))=(4+((3)+3))=(4+(6))=(10)

中间和是3，6，10，它们之和为19。

【问题描述】

现在要添上n-1对括号，加法运算依括号顺序进行，得到n-1个中间和，求出使中间和之和最小的添括号方法。

【例5】添加括号 --1688

Input

共两行。

第一行，为整数n。(1<=n<=20)

第二行，为a(1),a(2),...,a(n)这n个正整数，每个数字不超过100。

Output

输出3行。

第一行，为添加括号的方法。

第二行，为最终的中间和之和。

第三行，为n-1个中间和，按照从里到外，从左到右的顺序输出。

Sample Input

4

4 1 2 3

Sample Output

(4+((1+2)+3))

19

3 6 10