区间 DP

1 区间 DP

区间类 DP，是对一类序列合并类动态规划问题的分类，一般的状态方程形如：$f[i][j] = \textbf{opt}\{f[i][k], f[k][j]\} + w[i][j]$，较为经典的引例为石子合并问题。下面通过几道例题就这类问题进行展开。

2 石子合并问题

问题： *P2026. 石子合并*

描述： 给你一串无序的数（这些数字构成一个环），问将其中一段合并在一起的最小代价。

解析： 咋一看，这个问题可能与贪心一章中的合并果子有点类似，区别在于，本题的合并只产生在相邻的两个数，如果用贪心来做是错误的。容易举出一个贪心解决的错误范例：3 4 6 5 4 2，如果每次合并相邻和最小的数字，最后的答案为 62. 但实际答案为 61。

将问题进行简化，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。这样我们就得到一个类似分治的算法，通过 DP 状态来记录。设状态 $f[i][j]$ 表示区间 $[l, r]$ 的合并最优值，状态转移方程为：

$$f[i][j] = \textbf{min}\{f[i][k], f[k+1][j]\} + w[i][j]$$

理解：将区间分成 2 半，取最优的一般加上该区间的权值

本题中，所有数字构成一个环，解决思路还是一样，通过对序列延迟一倍即可。

参考代码：

```cpp
#include<bits/stdc++.h>
using namespace std;
int a[203],s[203],n;
int dpmin[203][203];
int dpmax[203][203];
int dfsmin(int l,int r){
  if(l==r)
```

```
    return 0;
  if(dpmin[l][r]!=-1)
    return dpmin[l][r];
  int ans=0x7fffffff;
  for(int k=l;k<r;k++)
    ans=min(ans,dfsmin(l,k)+dfsmin(k+1,r));
  ans=ans+s[r]-s[l-1];
  return dpmin[l][r]=ans;
}
int dfsmax(int l,int r){
  if(l==r)
    return 0;
  if(dpmax[l][r]!=-1)
    return dpmax[l][r];
  int ans=-1;
  for(int k=l;k<r;k++)
    ans=max(ans,dfsmax(l,k)+dfsmax(k+1,r));
  ans=ans+s[r]-s[l-1];
  return dpmax[l][r]=ans;
}
int main(){
  memset(dpmin,-1,sizeof(dpmin));
  memset(dpmax,-1,sizeof(dpmax));
  memset(a,0,sizeof(a));
  memset(s,0,sizeof(s));
  cin>>n;
  for(int i=1;i<=n;i++){
    cin>>a[i];
    s[i]=s[i-1]+a[i];
  }
  for(int i=1;i<n;i++){
    a[i+n]=a[i];
    s[i+n]=s[i+n-1]+a[i+n];
  }
  int minans=0x7fffffff,maxans=-1;
  for(int i=1;i<=n;i++){
    minans=min(minans,dfsmin(i,i+n-1));
```

```
    maxans=max(maxans,dfsmax(i,i+n-1));
  }
  cout<<minans<<"\n"<<maxans;
  return 0;
}
```

3 模版匹配

问题： *P2029. 括号匹配*

描述： 给一个由 ()[] 四种字符任意排列组成的字符串，求最长合法的不连续字串的长度。

解析：

方法 1: 首先能想到的是转化成 LCS（最长公共子序列），枚举中间点，求所有的 LCS 中的最大值 * 2 就是最大匹配。但是复杂度较高，光 LCS 一次就 $O(n^2)$ 的复杂度。

正解： 首先通过分解子问题，根一个大区间的括号匹配数可以由小区间的匹配数选取来。满足了最优子结构。

状态定义：区间 dp 一般应该有个区间状态，所以，定义 dp[i][j] 表示区间 [i,j] 的最大括号匹配数；考虑区间 $[i,j]$ 如果 s[i],s[j] 可以匹配, 那么

$$dp[i][j] = dp[i+1][j-1] + 2;$$

也可以将区间分成 2 半, 有：

$$dp[i][j] = max\{dp[i][j], dp[i][k] + dp[k+1][j]\}$$

状态转移方程表示为：

$$dp[i][j] = \begin{cases} dp[i+1][j-1] + 2 & s[i] =' (, [' \&\& s[j] ='],]' \\ max\{dp[i][j], dp[i][k] + dp[k+1][j]\} & other \end{cases}$$

参考代码：

```
#include<bits/stdc++.h>
using namespace std;
int f[300][300];
string s;
int dfs(int l,int r){
  if(l>=r)return 0;
  if(f[l][r]!=-1)return f[l][r];
  f[l][r]=0;
  if((s[l]=='('&&s[r]==')')||(s[l]=='['&&s[r]==']'))
```

```
    f[l][r]=dfs(l+1,r-1)+2;
  for(int k=l;k<r;k++)
    f[l][r]=max(f[l][r] ,dfs(l,k)+dfs(k+1,r));
  return f[l][r];
}
int main(){
  while(cin>>s){
    if(s[0]=='e')break;
    memset(f,-1,sizeof(f));
    cout<<dfs(0,s.size()-1)<<"\n";
  }
  return 0;
}
```

4 最优矩阵链乘

一个 n*m 的矩阵由 n 行 m 列共 n*m 排列而成。两个矩阵 A 和 B 可以相乘当且仅当 A 的列数等于 B 的行数。一个 n*m 的矩阵乘 m*p 的矩阵，运算量为 n*m*p。

矩阵乘法不满足分配律，但满足结合律。因此 A*B*C 既可以按顺序（A*B）*C 也可以按 A*（B*C）来进行。假设 A、B、C 分别是 2*3、3*4、4*5 的，则（A*B）*C 运算量是 2*3*4+2*4*5=64，A*（B*C）的运算量是 3*4*5*2*3*5=90. 显然第一种顺序节省运算量。

问题： *P2031. 矩阵链乘法*

描述： 给出 n 个矩阵组成的序列，设计一种方法把他们依次乘起来，使得总的运算量尽量小。

分析：

要计算整个表达式，一定有最后一次乘法，在最后一次前，假设序列分成 2 部分先计算了：

设 $P = A_1 * A_2 * A_3 * .. * A_k, Q = A_{k+1} * A_{k+2} * .. * A_n$

P,Q 明显是不干扰的，只需要让 P,Q 最优，这里有最优子结构

设子问题 DP[i,j] 表示从 i 到 j 的最优计算，p[i].x，p[i].y 表示第 i 个矩阵的行与列，有如下方程：

$DP[i, j] = \min\{DP[i, k] + DP[k + 1, j] + p[i].x * p[k].y * p[j].y\}$

问题： *P2028. 乘法游戏*

描述： 乘法游戏是在一行牌上进行的。每一张牌包括了一个正整数。在每一个移动中，玩家拿出一张牌，得分是用它的数字乘以它左边和右边的数，所以不允许拿第 1 张和最后 1 张牌。最后一次移动后，这里只剩下两张牌。你的目标是使得分的和最小。

解析：

对于合并一个区间，如果枚举最后一次合并的位置，可以发现左右两端的牌是不动的，这样就与上面说的矩阵链乘类似。可以设 $dp[l][r]$ 表示将区间 $[l, r]$ 移动到只剩两张卡片的最小分数, 则

有：

$dp(l,r) = \min\{dp(l,k) + dp(k,r) + a[l] * a[k] * a[r]\}$

初值：$dp[i,i+1] = 0$

5 凸多边形三角剖分

问题：*P2067. 三角剖分问题*

描述：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？

解析：**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）。

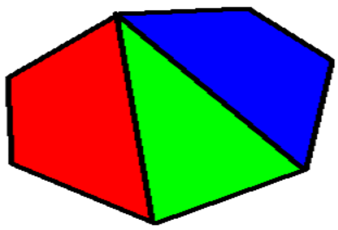


图1

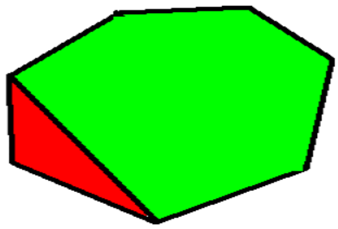
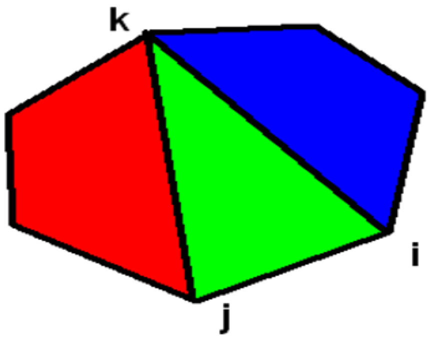


图2

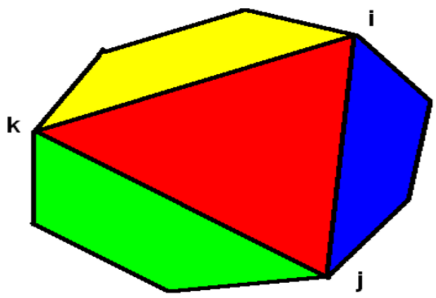
如果我们按顺时针将顶点编号，则可以相邻两个顶点描述一个凸多边形。设 f(i,j) 表示 i j 这一段连续顶点的多边形划分后最小乘积；枚举点 k，i、j 和 k 相连成基本三角形，并把原多边形划分成两个子多边形，则有

$f(i,j) = \min\{f(i,k) + f(k,j) + a[i] * a[j] * a[k]\}\ 1 <= i < k < j <= n$

时间复杂度 $O(n^3)$



思考：为什么可以不考虑下面的情况？



6 棋盘分割

问题： *P2067.* 三角剖分问题

描述： 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)。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。

解析： 首先，对均方差公式进行变形观察：

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

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

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

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

由化简后的均方差公式：

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

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。

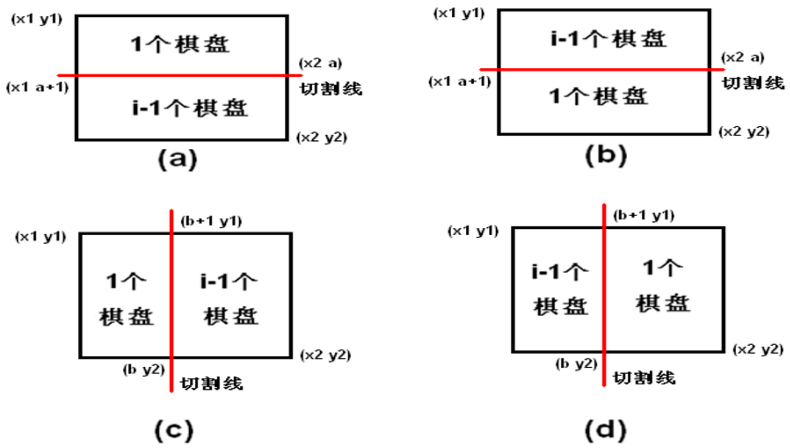
因此，我们需要求出各棋盘分割后的每个棋盘各数平方和的最小值，设为 w，那么答案为：$ans = \sqrt{w/n - \bar{x}^2}$

设 F(i,x1,y1,x2,y2) 表示以 [x1,y1][x2,y2] 为四边形对角线的棋盘切割成 k 块的各块数值总平方和的最小值，则有：

$$f(i,x_1,y_1,x_2,y_2) = \begin{cases} f(i-1,x_1,a+1,x_2,y_2) + D[x_1,y_1][x_2,a] & case\ A \\ f(i-1,x_1,y_1,x_2,a) + D[x_1,a+1][x_2,y_2] & case\ B \\ f(i-1,b+1,x_1,x_2,y_2) + D[x_1,y_1][b,y_2] & case\ C \\ f(i-1,x_1,y_1,b,y_2) + D[b+1,y_1][x_2,y_2] & case\ D \end{cases}$$

$$1 \le x1, x2, x3, x4 \le 8, 1 \le i \le n$$

其中 case A 表示横切，上面不动，见下图四种情况。



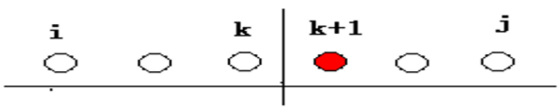
设棋盘边长为 m, 则状态数为 nm^4 , 决策数最多 m。

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 $O(nm^5)$。

7 小结

区间类 DP 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