**实验2-矩阵连乘**

**问题分析**

* **基础知识**

两矩阵相乘C1C2，其中元素相乘的总次数等于p\*q\*r。其中p,q为C1的行和列，q,r为C2的行列

* **最优值**

分析：应该在乘积序列的什么地方加括号从而把原序列分割成两个子序列？子序列又该如何进一步划分？

思路：将多个矩阵看作是一个区间，对于整个区间，枚举断点分为左右两个区间。形成两个相同子问题，即求子区间元素最少连乘次数。因为合并两个子区间的分别是其各自最优解，因此是具有最优子结构性质的。实现方法上在在递归调用和循环数组，选择后者，旨在尝试一般动态规划题目的解题过程。

首先定义一个二维数组。

* + 当i = j时，表示单独一个矩阵，因此元素相乘次数为0
  + 当i < j时，表示从第i个矩阵到第j个矩阵之间元素相乘的最小次数。

最终状态转移方程如下：

具体代码：

初始化条件：赋值0给区间为长度0的m数组；将每个区间先简单的定义为从区间第一个数后开始分割。

1. **for** (**int** i = 0; i < r; i++)
2. {
3. m[i][i] = 0;//i到i之间0次乘法
4. }
5. ... ...
6. m[i][j] = m[i + 1][j] + p[i-1]\*p[i]\*p[j];//初始化 直接从他后一个分割

自底向上计算最优值：与一般循环不同的是，本题不是简单的使递增循环，因为考虑到数组m的含义表示从第1个到第n个的次数。可能会导致其中有些小区间尚未计算，从而无法组成或不是最优解。因此采用区间从小到大递增。先构造小区间数组再以此向上计算大区间数组的值。

1. **for** (**int** iv = 2; iv <= r; iv++)//iv表示间隔 间隔从小到大 从小区间算到大区间
2. {
3. **for** (**int** i = 1; i <= r - iv +1; i++)
4. {
5. **int** j = i + iv - 1;
6. ... ...
7. **for** (**int** k = i + 1; k < j; k++)
8. {
9. **int** t = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];
10. **if** (t < m[i][j])
11. {
12. m[i][j] = t;
13. }
14. }
15. }
16. }

* **最优解结构**

额外增加一个pos数组用于保存每一个区间中断点的位置。初始化区间时，将其设置为区间首项；循环遍历时动态地变为上述程序中求得的最小次数位置。

1. **for** (**int** iv = 2; iv <= r; iv++)//iv表示间隔 间隔从小到大 从小区间算到大区间
2. {
3. **for** (**int** i = 1; i <= r - iv +1; i++)
4. {
5. **int** j = i + iv - 1;
6. ... ...
7. pos[i][j] = i;
8. **for** (**int** k = i + 1; k < j; k++)
9. {
10. **if** (t < m[i][j])
11. {
12. pos[i][j] = k;
13. ... ...

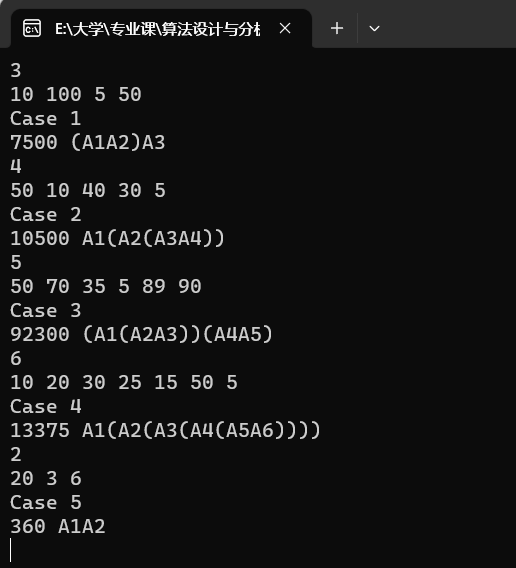
最终打印呈现用递归调用的方法。当打印的区间只包含一个值时，即直接打印当前矩阵下标并返回。除此之外先打印左括号再根据断点pos数组保存的位置，去递归处理子区间的划分情况。最后再打印右括号。当区间为时无需在外侧加括号。

1. **void** display(**int** l,**int** r)
2. {
3. **if** (l == r)
4. {
5. cout << "A" << l;
6. **return**;
7. }
8. **if** (l != 1 || r != n)
9. cout << "(";
10. display(l, pos[l][r]);
11. display(pos[l][r] + 1, r);
12. **if** (l != 1 || r != n)
13. cout << ")";
14. }

* **反证法证明最优子结构**

设(A1…Ak)(Ak+1…An) 具有最少乘法次数，则(A1…Ak)中加括号的方法使A1..Ak乘法次数最少。否则设存在另一种加括号方法(A1…Ak)'更优，则(A1…Ak)'(Ak+1…An) 比 (A1…Ak)(Ak+1…An) 更优，矛盾。同理， (Ak+1…An) 内的连乘方法也是最优的。

**运行结果截图**



**设计调试中的问题**

* 初次调式时发现结果大都为0，发现在第二层循环时遗漏了最后一位数（即m数组该位置位0），而正是底层一位数的遗漏导致后续状态转移时都将0作为最小值传递下去。

解决方法：将小于改为小于等于。

1. **for** (**int** iv = 2; iv <= r; iv++)
2. {
3. **for** (**int** i = 1; i <= r - iv +1; i++)
4. {
5. ... ...

* 此实验有多组数据输入输出，刚开始测试发现仅第一轮答案正确，后续均会出现错误。发现是每轮测试中未将m数组进行清零。解决方法：在每次循环伊始增加清空数组语句。

1. memset(m, 0, **sizeof**(m));

**实验体会**

如上文所述，此实验其实可以分为两个部分，一是求出最优值，二是构造出对应最优值的最优解结构。前者在明确了m，p数组的含义后再根据动态规划的状态转移方程可以清晰的理解整个动态规划的流程。但后者在实验过程中却带来不小的难题。第一个难点：在哪加括号，这其实是求第一部分的问题的过程。第二个难点：如何加括号，针对这个问题查询课本，上网搜索等等结合多方的方法后，最终得以理解。同时也采用了递归的方法将此问题解决，也属于复习巩固第一章的部分内容。

总的来说，此次实验让我初体验动态规划的流程，面对这类题应采取大的区间划分成小的区间进行求解。动态规划问题的精髓在于状态转移方程，通过了解问题的子问题和最后一步去确定问题的状态，从而得到转移方程。有了明确的方程后问题就变得清晰明了了，但同时初始条件和边界情况同样也需注意。在日后我也会多加对动态规划的练习，去领会其中的奥妙。

**程序源码**

1. #include<iostream>
2. #include<bits/stdc++.h>
3. **using** **namespace** std;
4. **const** **int** N = 101;
5. **int** n;
6. **int** m[N][N], pos[N][N],p[N];//m存放i,j之间最小的乘的次数，pos记录断点位置
8. **void** chain(**int** l, **int** r)
9. {
10. **for** (**int** i = 0; i < r; i++)
11. {
12. m[i][i] = 0;//i到i之间0次乘法
13. }
14. **for** (**int** iv = 2; iv <= r; iv++)//iv表示间隔 间隔从小到大 从小区间算到大区间
15. {
16. **for** (**int** i = 1; i <= r - iv +1; i++)
17. {
18. **int** j = i + iv - 1;
19. m[i][j] = m[i + 1][j] + p[i-1]\*p[i]\*p[j];//初始化 直接从他后一个分割
20. pos[i][j] = i;
21. **for** (**int** k = i + 1; k < j; k++)
22. {
23. **int** t = m[i][k] + m[k + 1][j] + p[i - 1] \* p[k] \* p[j];
24. **if** (t < m[i][j])
25. {
26. m[i][j] = t;
27. pos[i][j] = k;
28. //cout << t << endl;
29. }
30. }
31. }
32. }
33. }
35. **void** display(**int** l,**int** r)
36. {
37. **if** (l == r)
38. {
39. cout << "A" << l;
40. **return**;
41. }
42. **if** (l != 1 || r != n)
43. cout << "(";
44. display(l, pos[l][r]);
45. display(pos[l][r] + 1, r);
46. **if** (l != 1 || r != n)
47. cout << ")";
48. }
49. **int** main()
50. {
51. **int** case\_num = 0;
52. **while** (cin >> n)
53. {
54. memset(m, 0, **sizeof**(m));
55. case\_num++;
56. **for** (**int** i = 0; i <= n; i++)
57. cin >> p[i];
58. chain(1, n);
59. cout << "Case " << case\_num << endl;
60. cout << m[1][n] << ' ';
61. display(1, n);
62. cout << endl;
63. }
64. }