算法第六次作业2022-10-31

10205101530-赵晗瑜

1.

1. Find an optimal parenthesization of a matrix-chain product whose sequence of dimensions is <10,5,20,40,25,8>.

(A1(((A2A3)A4)A5))

总次数：

$$5*20*40+5*40*25+5*25*8+10*5*8=10400$$

编译运行如下C++代码，求出矩阵链最优括号化方案

```
#include<iostream>
using namespace std;
//p为矩阵链，p[0],p[1]代表第一个矩阵，p[1],p[2]代表第二个矩阵，length为p的长度
//所以如果有六个矩阵，length=7，m为存储最优结果的二维矩阵，t为存储选择最优结果路线的
//二维矩阵
void MatrixChainOrder(int* p, int(*m)[10], int(*t)[10], int length)
{
    int n = length - 1;
    int i, j, k, q, num = 0;
    //A[i][i]只有一个矩阵，所以相乘次数为0，即m[i][i]=0;
    for (i = 1; i < length; i++)
    {
        m[i][i] = 0;
    }
    //i代表矩阵链的长度，i=2表示有两个矩阵相乘时如何划分
    for (i = 2; i <= n; i++)
    {
        //j表示从第j个矩阵开始的i个矩阵如何划分是最优
```

```
        for (j = 1; j <= n - i + 1; j++)
        {
            //k为从第j个数i个矩阵就是k，从j到k表示他们之间的i个矩阵如何划分
            k = j + i - 1;
            //m[j][k]存储了从j到k使用最佳划分所得到的最优结果
            m[j][k] = 0x7fffffff;
            //q为介于j到k-1之间的数，目的是利用q对j到k之间的矩阵进行试探性的划
分，
            //从而找到最优划分，这是一种遍历性的试探。
            for (q = j; q <= k - 1; q++)
            {
                num = m[j][q] + m[q + 1][k] + p[j - 1] * p[q] *
p[k];
                if (num < m[j][k])
                {
                    m[j][k] = num;
                    t[j][k] = q;
                }
            }
        }
    }
}
void PrintAnswer(int(*t)[10], int i, int j)
{
    if (i == j)
    {
        cout << "A" << i;
    }
    else
    {
        cout << "(";
        PrintAnswer(t, i, t[i][j]);
        PrintAnswer(t, t[i][j] + 1, j);
        cout << ")";
    }
}
int main()
{
    int p[7] = { 10,5,20,40,25,8 };
    int m[10][10], t[10][10];
    MatrixChainOrder(p, m, t, 6);
    PrintAnswer(t, 1, 5);
```

```
    return 0;
}
```

15.2-2

设计递归算法 MATRIX-CHAIN-MULTIPLY(A，s，i，j)，实现矩阵链最优代价乘法计算的真正计算过程，其输入参数为矩阵序列$\langle A_1, A_2, \cdots, A_n\rangle$，MATRIX-CHAIN-ORDER 得到的表 s，以及下标 i 和 j。(初始调用应为 MATRIX-CHAIN-MULTIPLY(A，s，1，n)。)

```
MATRIX-CHAIN-MULTIPLY(A,S,i,j)
    if i == j then
        return A[i]
    end if
    return MATRIX-CHAIN-MULTIPLE(A,s,i,s[i,j]) * MATRIX-CHAIN-
MULTIPLY(A,s,s[i,j]+1,j)
```

15.2-4

对输入链长度为 n 的矩阵链乘法问题，描述其子问题图：它包含多少个顶点？包含多少条边？这些边分别连接哪些顶点？

对于任意的 $1\le i\le j\le n$，v_{ij} 表示求解 $A_i..A_j$ 的一次调用，对于顶点 v_{ij}:

① 若 $i=j$ 则 v_{ij} 没有出边

② 若 $i<j$ 则对所有的 $i\le k<j$ 子问题图中 都有有向边 (v_{ij}, v_{ik}) 和 $(v_{ij}, v_{k+1,j})$ 存在，这些边表明为了解决 $A_i\ldots A_j$ 的子问题的最优括号乘积，我们需要解决 $A_i\ldots A_k$ 和 $A_{k+1}\ldots A_j$ 的最优括号乘积子问题

总节点数: $\sum_{j=1}^{n}\sum_{i=1}^{j} 1 = \frac{n(n+1)}{2}$

对区间长度 $l=j-i$ 和区间起点 i 作规纳，总边数为:

$$\sum_{l=1}^{n}\sum_{i=1}^{n-l+1}(l-1)\times 2 = \frac{(n-1)n(n+1)}{3} = \frac{n^3-n}{3}$$

长为 l 的区间有 n-l+1 个，这个区间有 l-1 个可以分隔的点，每次分割都衍生出两条边

15.4-5

设计一个 $O(n^2)$ 时间的算法，求一个 n 个数的序列的最长单调递增子序列。

可以对输入序列 A 生成一个新的已排序的序列 A'，耗时 $O(n\log n)$，然后利用已有的 LCS-LENGTH 以 A 和 A'

作为输入，然后用 LCS-PRINT 输出，总耗时

$$O(nlogn) + O(n^2) + O(2n) = O(n^2)$$

```
LONGEST-INCREASING-SUBSEQUENCE(A)
  Let A′ be a copy of A
  sort A
  (c, b) = LCS-LENGTH(A, A′)
  PRINT-LCS(b, A, 1, A.length)
```

15.4-6

设计一个 $O(n\lg n)$ 时间的算法，求一个 n 个数的序列的最长单调递增子序列。（提示：注意到，一个长度为 i 的候选子序列的尾元素至少不比一个长度为 $i-1$ 候选子序列的尾元素小。因此，可以在输入序列中将候选子序列链接起来。）

算法 LONG-MONOTONIC(S) 返回 S 的最长单调递增子序列,其中S的长度为n。

该算法的工作原理如下：

- 创建一个新数组B，使得B[i]包含长度为i的最长单调递增子序列的最后一个值
- 创建一个新数组C，使得C[i]包含长度为i的最长单调递增子序列，其中包含迄今为止看到的最小的最后一个元素。
- 为了分析运行时间，观察 到B 的条目是有序的，所以我们可以在 O(log(n)) 时间内执行第 9 行。 由于 for 循环中的每一行都需要恒定时间，因此总运行时间为 O(n

log n)

遍历复杂度为 O(n), 二分查找复杂度 O(lgn)。总的算法复杂度为 O(nlgn)。

```
Algorithm 6 LONG-MONOTONIC(S)
Initialize an array B of integers length of n, where every value is set equal to ∞.
Initialize an array C of empty lists length n.
L = 1
for i = 1 to n do
    if A[i] < B[1] then
        B[1] = A[i]
        C[1].head.key = A[i]
    else
        Let j be the largest index of B such that B[j] < A[i]
        B[j + 1] = A[i]
        C[j + 1] = C[j]
        C[j + 1].insert(A[i])
        if j + 1 > L then
            L = L + 1
        end if
    end if
end for
Print C[L]
```