# 习题六

## 6.2 递归输出装配点(装配点的存储有点像是顺序存储的链表)

先往下递归到栈底再回溯即可：



(分页)

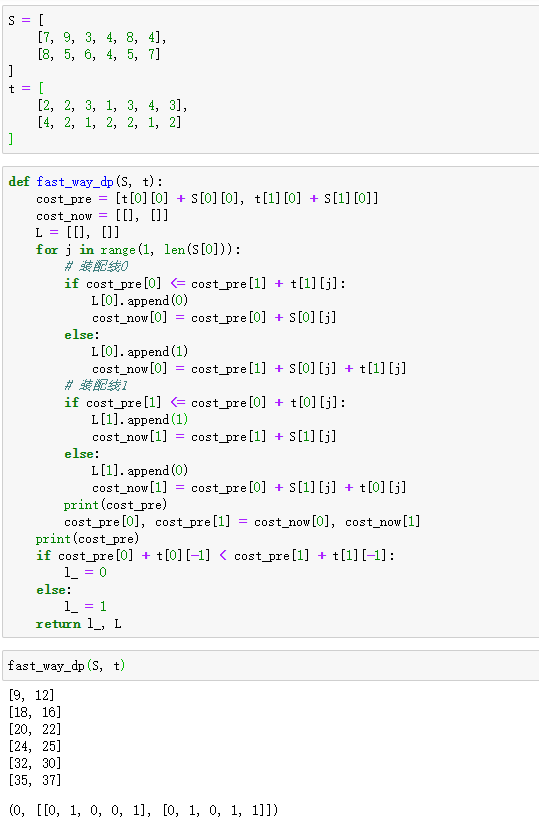
## 6.3 装配线问题中的空间使用如何从4n-2缩减到2n+2

解答中使用的空间分别是

* Cost[i][j] 用于存储最佳装配的时间使用，其中i为装配线，j为station编号
* L[i][j] 用于存储最佳装配路线，i, j同上

存储路线的存储用于构造最优解，没什么好的方式缩减存储，但是Cost[i][j]的计算中，每一个station只需要考虑上一station的最优解，因而可以缩减到只占用4个单位的存储。从而总的存储能减少到2n+2

程序：使用cost\_pre，cost\_now 代替cost，使用python实现因为课本已经给出最主要代码，python能更快的实现想法

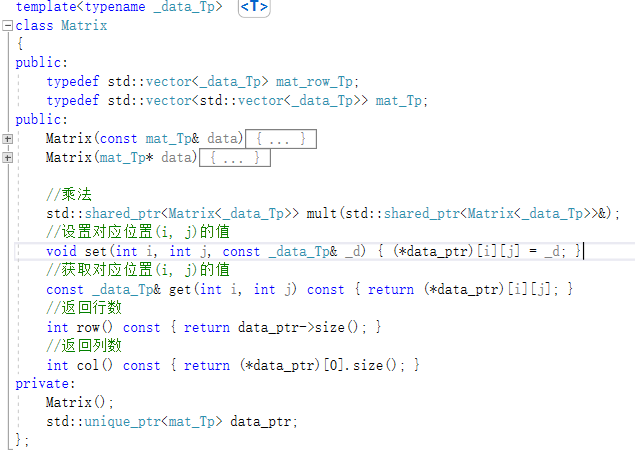


## 6.5利用矩阵的最优链乘次序算法，实现矩阵序列的乘法

课本虽然给出矩阵链乘次序的求解方法，但是并没有具体实现，考虑到试验题中有要求实现，因此本题将次序的求解和具体的乘法一起实现，工程文件为matrix\_multiply。

具体地：

* 实现简单矩阵类方便之后操作：



* 思路以及完成任务说明：

//用随机数生成矩阵，因为没有现成数据，所以随机生成

init\_matrix();

//矩阵输入

Matrixs mat\_buf; //用于存储读取的矩阵

input\_matrix(mat\_buf);

//乘法方式计算

Matrix<int>::mat\_Tp init\_mat(mat\_buf.size(),

Matrix<int>::mat\_row\_Tp(mat\_buf.size(), 0));

//用于存储计算次数和链乘方式的矩阵

Matrix<int> mult\_times(init\_mat), solute\_rec(init\_mat);

//动态规划实现的计算方法计算过程

matrix\_chain\_dp(mult\_times, solute\_rec, mat\_buf);

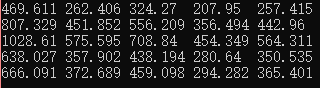
//矩阵乘法计算

auto C = matrix\_mult(solute\_rec, mat\_buf, 0, mat\_buf.size() - 1);

//输出

…

* 某次输出：



* 本题任务：其中最后一个函数就是题目要求的MatrixChainMultiply(A, s, 1, n);，参数与其要求的一致，所有函数都能在工程文件中找到，其中本函数：

std::shared\_ptr<Matrix\_t> matrix\_mult(Matrix<int>& solute\_rec, const Matrixs& mats, int i, int j)

{

if (i == j) { return mats[j]; }

auto A = matrix\_mult(solute\_rec, mats, i, solute\_rec.get(i, j));

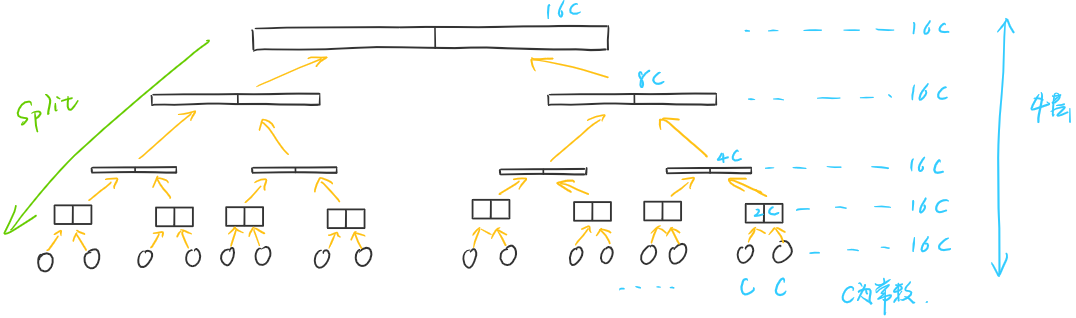
auto B = matrix\_mult(solute\_rec, mats, solute\_rec.get(i, j) + 1, j);

return A->mult(B);

}

## 6.7

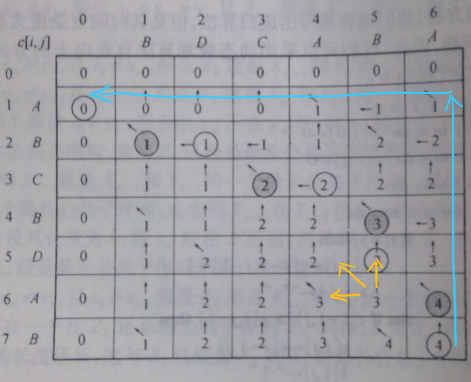
* 画16个元素的归并排序递归树：



* 如果说一个算法是好的分治算法，那它应该在需要的时候使用备忘录，因为大部分动态规划应当都能用备忘录形式的递归算法解决，而剩下的一些递归算法差不多都是子问题重叠很少或者是压根没有子问题，比如归并排序，子问题要求排序的元素个数和所有元素都相同，这样的子问题空间过于庞大，重复性太小，备忘录反而浪费资源

## 6.9 用c构造LCS的问题：

引用课本的图：



从右下角开始遍历，到达每一个位置，和如图黄色箭头所示的三个数进行对比即可判断路径的上一个元素，对于斜线的就是LCS的一部分，如图蓝色线的路径为最长可能路径，所以可以在O(m+n)时间内构造出LCS

## 6.12 设计出寻找最长单调递增子序列的算法

* 全部代码在侯mework\_12.cpp中
* T(n)=O()版本：

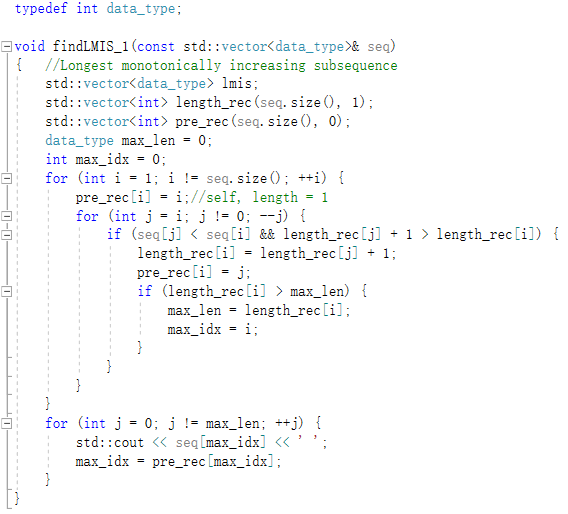
每个数都从后向前遍历，寻找第一个比它小的数，T(n)== O()，测试数据：

3, 2, 1, 3, 5, 6, 7, 8, 1, 3, 3, 2

输出(倒序)：



代码：



* T(n)=O(nlog(n))版本

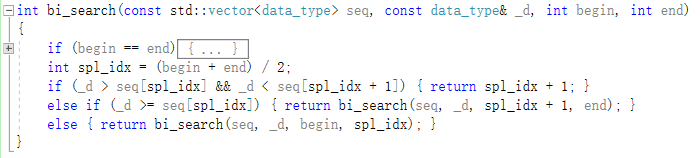
除了维护一个用于记录向前索引的列表外，维护一个记录长度为i的单调递增子序列最小首元素的数组B[i]。遍历元素，利用二分搜索插入B[i]。

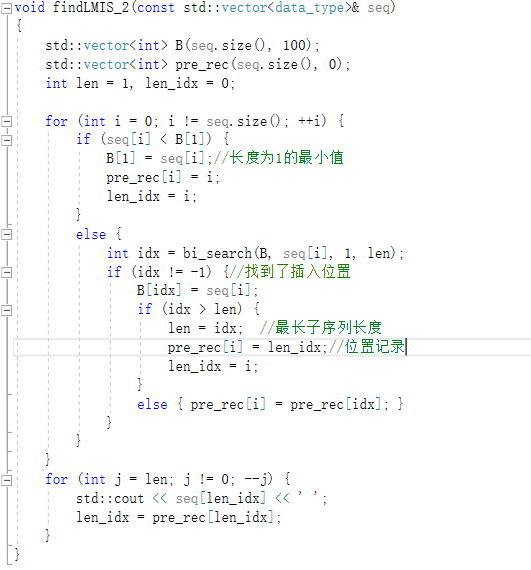
二分搜索T(n)=O(nlog(n))，所以整体复杂度

输出：(输入：3, 2, 1, 3, 5, 6, 9, 8, 1, 3, 10, 2)



代码：



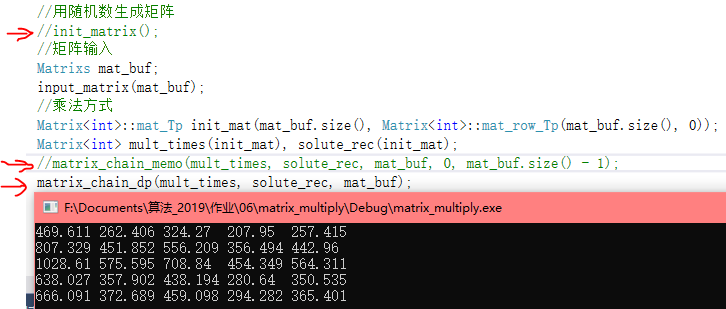


（分页）

## 6.25 实现矩阵链乘的三种算法，并用实验方法对比

* 所有代码在vs2017的工程matrix\_multiply中
* 动态规划版本：(在6.5(上面)中已经说明过)

禁用备忘录方法，禁用随机数据生成，运行一次6.5中的程序：

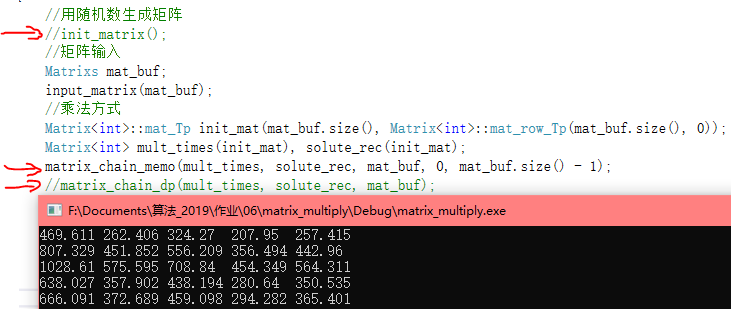


监视中矩阵相乘次数的记录：

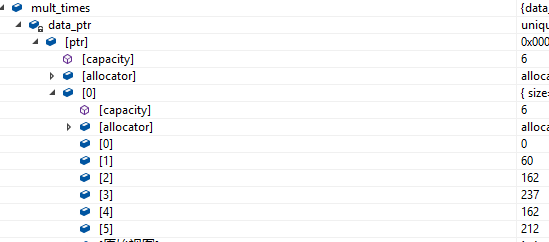


* 备忘录(memo)法

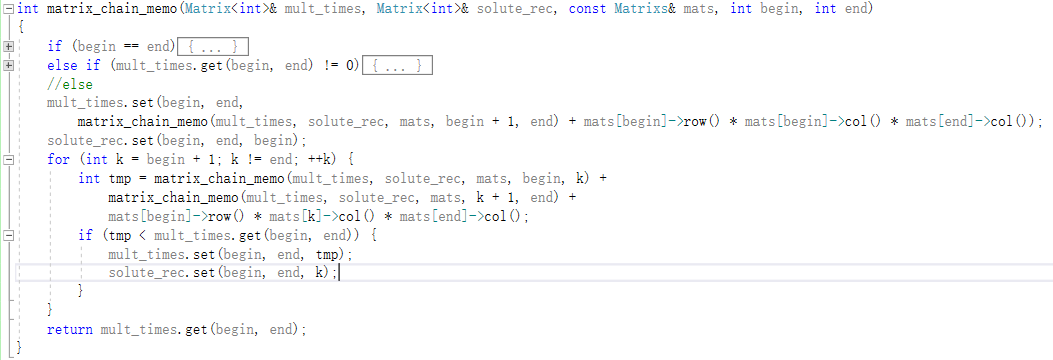
禁用动态规划方法，禁用随机数据生成，运行一次6.5中的程序，可以看到结果是一致的。



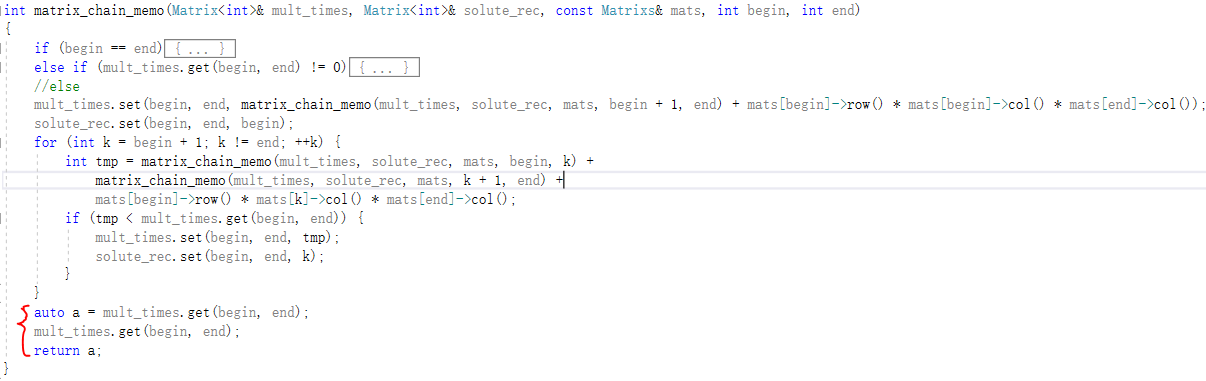
监视中矩阵相乘次数的记录：



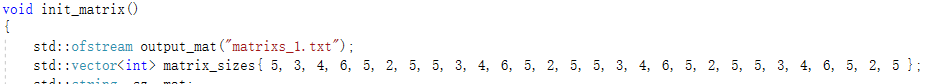
实现的代码：



* 完全递归的方法：只需要稍微修改一下备忘录方法(最终的代码还是改回备忘录，因此递归部分参考以下代码)：



* 实验：为了对比，将矩阵个数增加：



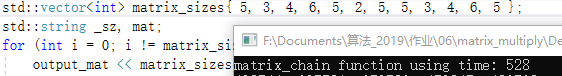
* + 动态规划方法使用时间(ms):



* + 备忘录方法使用时间(ms):



* + 完全递归方法使用时间(ms)：由于之前的长度一分钟还没有结果，只能修改矩阵序列的长度：



可以看到动态规划和备忘录方法几乎相差无几，而正常的递归会耗费非常多的时间

## oj上的题目，至少两题：

选做了xoj上的三题：

