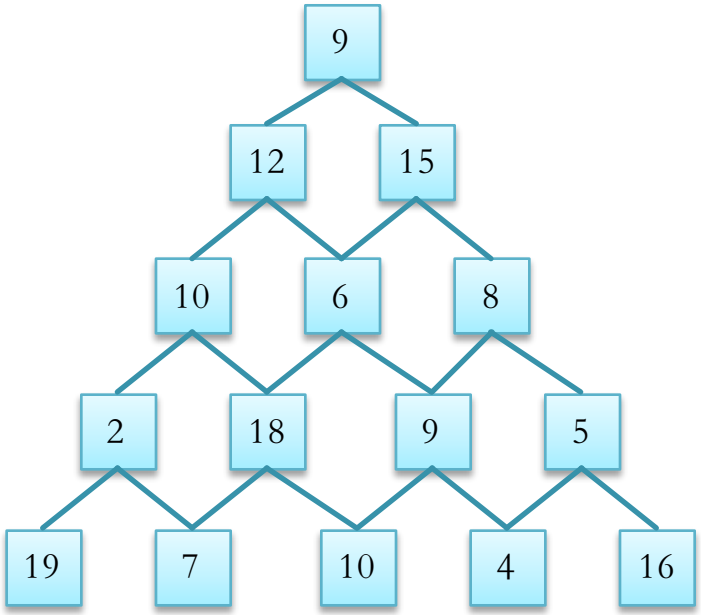
不要重复你的计算

动态规划

问题：数塔问题

- 有形如右图的一个数塔，从顶部出发，在每一结点可以选择向左走或是向右走，一直走到底层，要求找出一条路径，使路径上的数值和最大。



数据存储

9	0	0	0	0
12	15	0	0	0
10	6	8		0
2	18	9	5	0
19	7	10	4	16

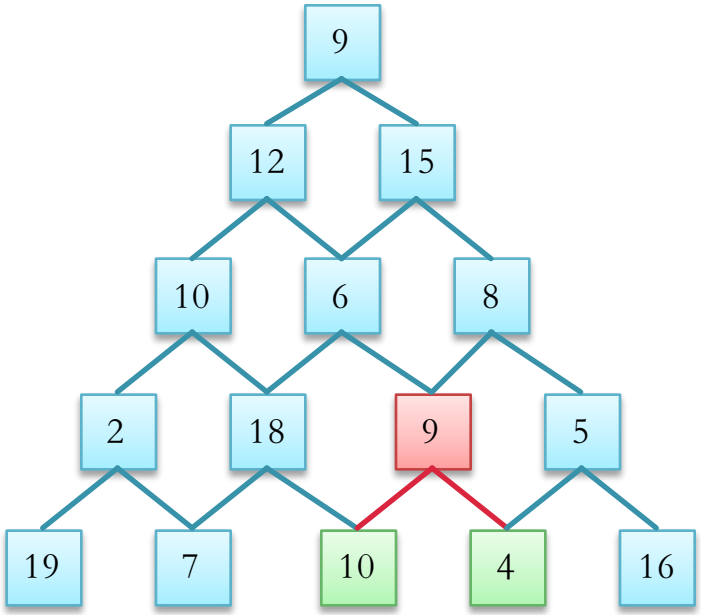
(i,j)

(i+1,j)

(i+1,j+1)

所有的节点存在一个二维数组的下三角中

第(i,j)节点的左右分支分别存放在(i+1,j)和(i+1,j+1)中



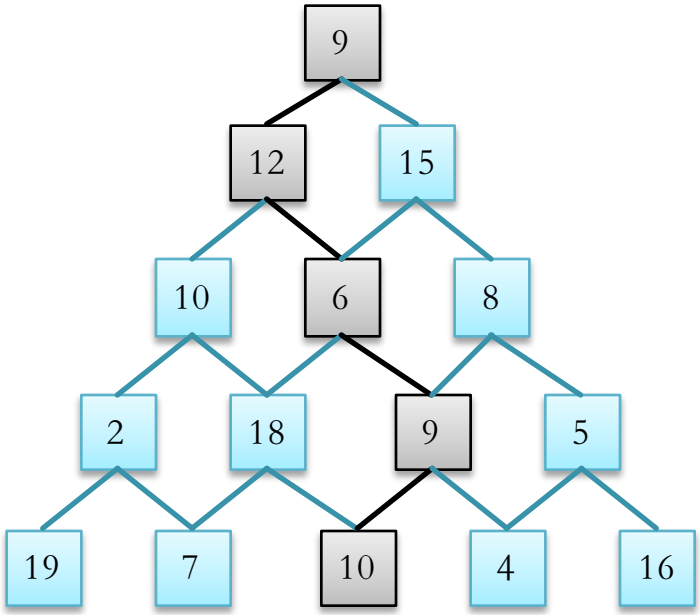
求解

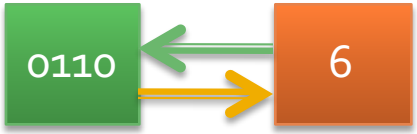
暴力求解，列举出所有可能的路径，并计算这些路径上数字的和。

如何保存这些路径呢?

每一步保存一个数据，向左是0，向右是1，然后形成一个个数和树高一样的0、1字符串，把这个字符串转化为10进制，作为数组下标

选出数组中最大的元素，再把数组元素下标从十进制转化为2进制就可以得到我们要求的路径了





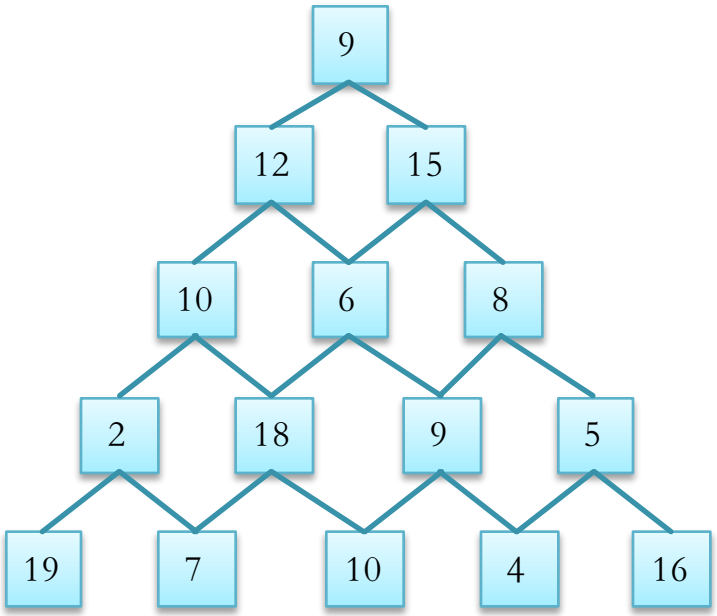
a[6] = 9+12+6+9+10 = 46

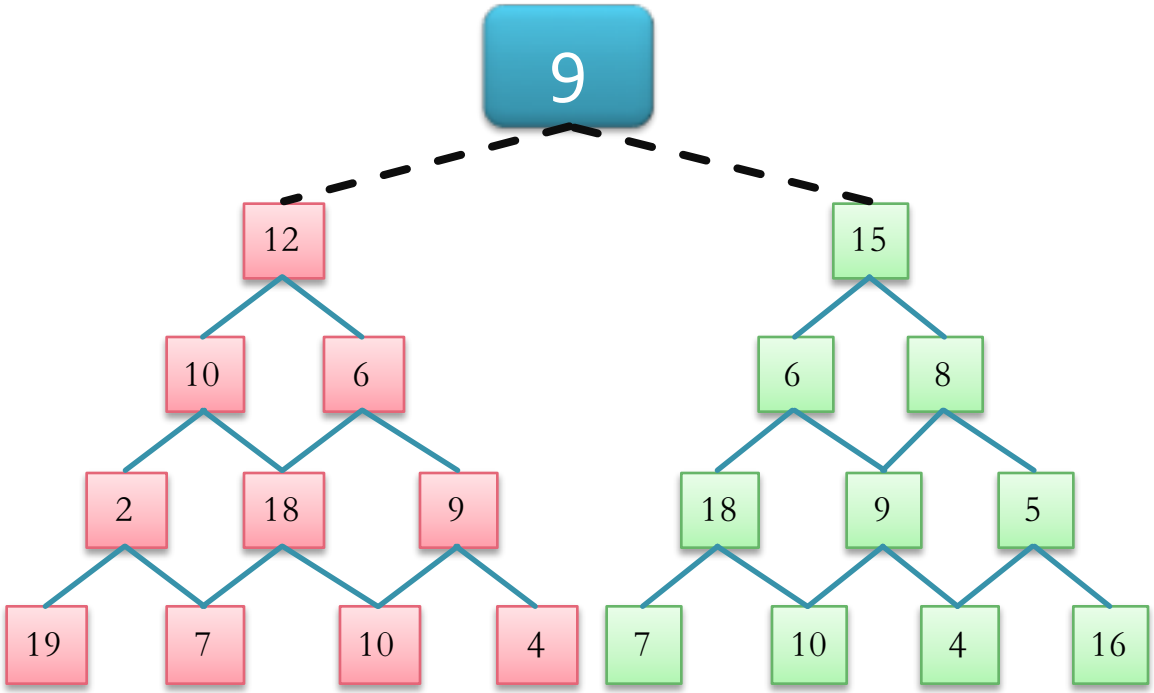
观察

考虑我们曾经讲过的分治思想

整体的最长距离相当于左右两颗子树的最长距离加上9，左右子树的最优解可以如此分解下去

最优子结构

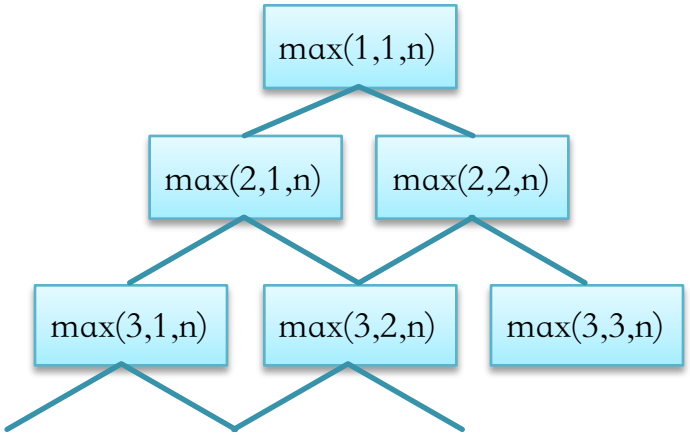




求解

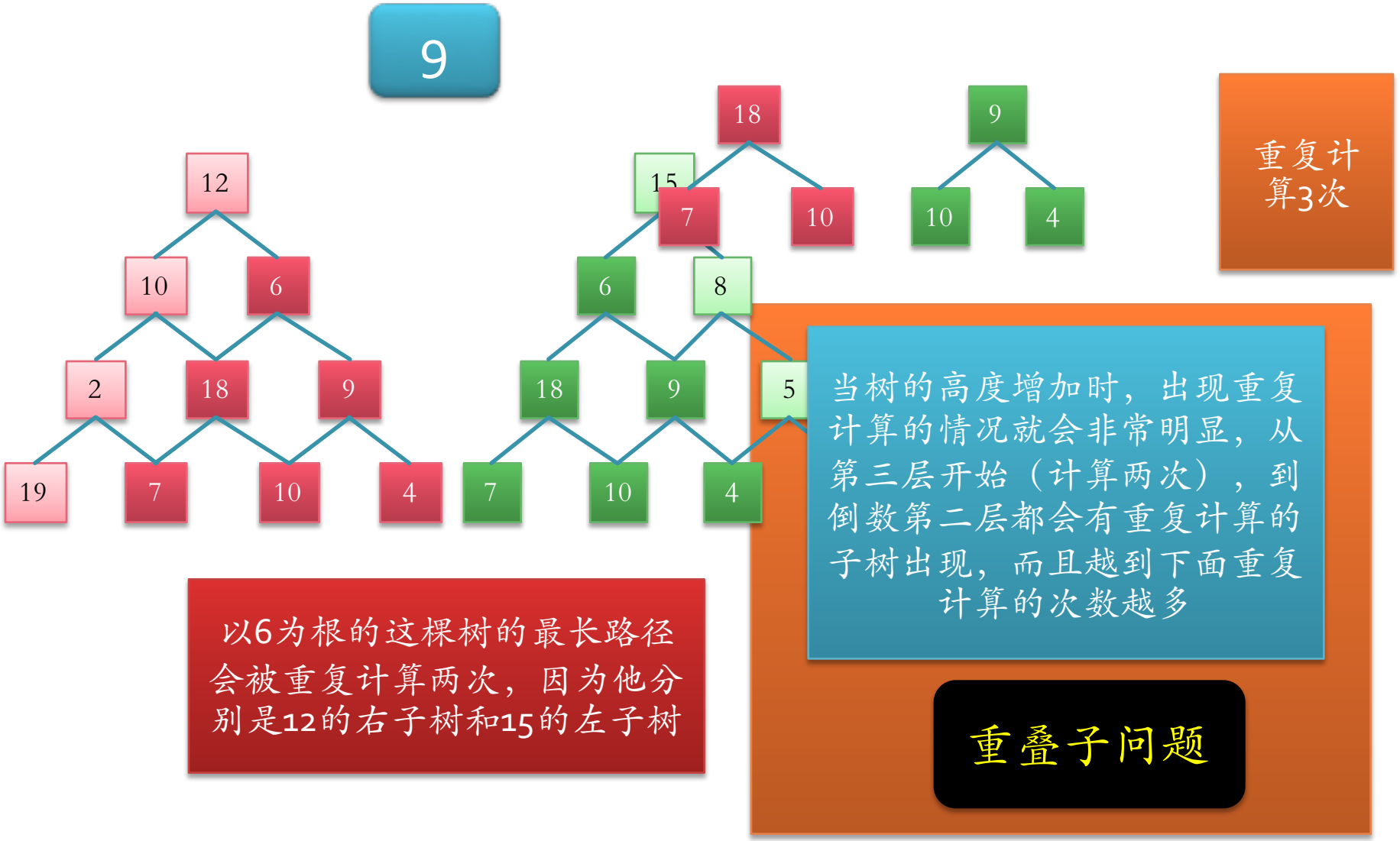
9	0	0	0	0
12	15	0	0	0
10	6	8	0	0
2	18	9	5	0
19	7	10	4	16

分治策略通常使用递归



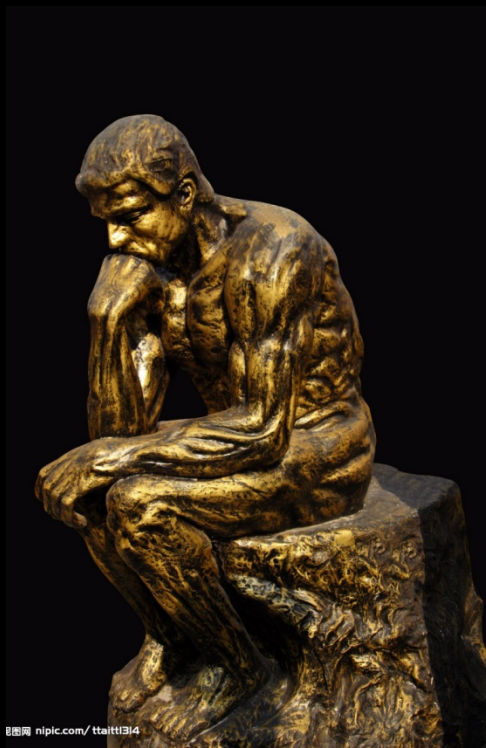

```
#define N  5
int a[N][N];
int max(int i, int j, int n)
{
    int left,right;
    if ((i==n)||(j==n))
        return a[i][j];

    left=max(i+1,j,n);
    right=max(i+1,j+1,n);
    return (left>right)? (left+a[i][j]) :
                         (right+a[i][j]);
}

int main() {
    int maxlength;
    maxlength = max(0, 0, N-1);
    return 0;
}
```

看看能不能优化


9
12
10
6
2
18
9
19
7
10
4
18
15
7
10
6
8
18
9
5
7
10
4
9
10
4
重复计算3次
当树的高度增加时，出现重复计算的情况就会非常明显，从第三层开始（计算两次），到倒数第二层都会有重复计算的子树出现，而且越到下面重复计算的次数越多
以6为根的这棵树的最长路径会被重复计算两次，因为他分别是12的右子树和15的左子树
重叠子问题

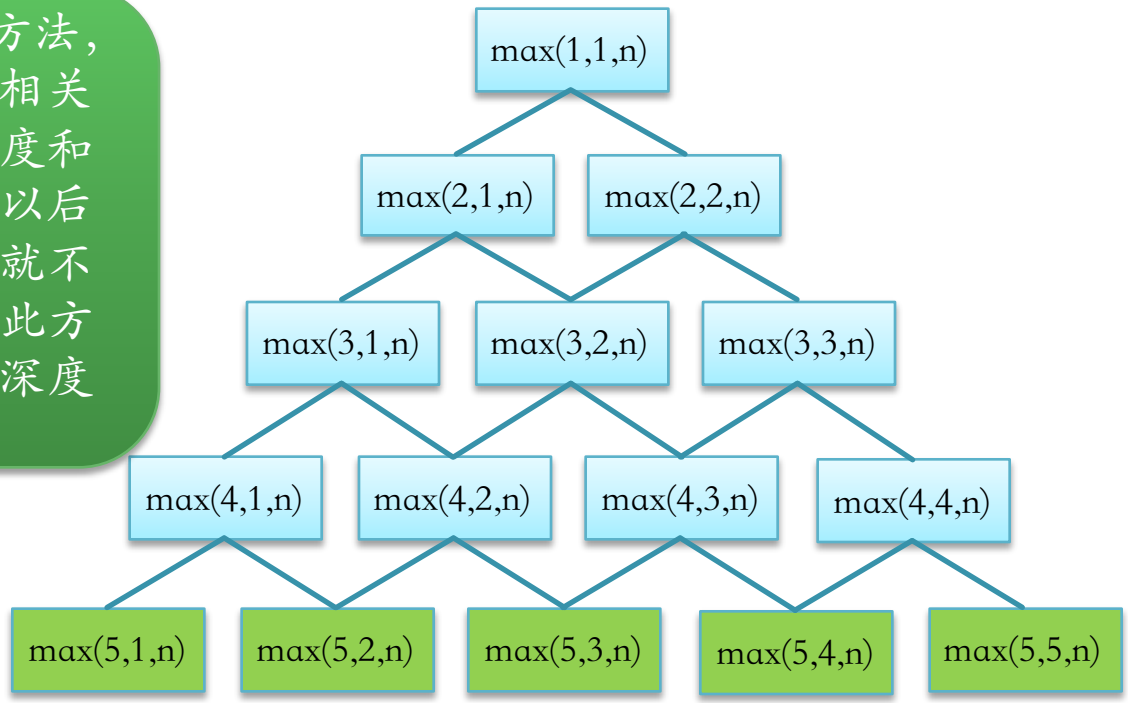
如何减少重复呢?

自顶向下记录法
自底向上方法



自顶向下记录法

依然采用自顶向下的递归方法，但是每次计算一个和节点相关的最大长度，就把这个长度和节点编号一起记录下来，以后再需要此节点的最大长度就不在计算，而是直接是用，此方法的过程相当于对树进行深度遍历的过程



m[1][1] = 59

m[2][1] = 50　　m[2][2] = 49

m[3][1] = 38　　m[3][2] = 34　　m[3][3] = 29

m[4][1] = 21　　m[4][2] = 28　　m[4][3] = 19　　m[4][4] = 21

程序实现

```
#define N  5
int a[N][N];
int max(int i, int j, int n)
{
    int left,right;
    if ((i==n)||(j==n))
        return a[i][j];

    left=max(i+1,j,n);
    right=max(i+1,j+1,n);
    return (left>right)? (left+a[i][j]) :
                         (right+a[i][j]);
}

int main() {
    int maxlength;
    maxlength = max(0, 0, N-1);
    return 0;
}
```

```
#define N  5
int a[N][N];
int data[N][N];

int max(int i, int j, int n)
{
    int left,right;
    if ((i==n)||(j==n)) {
        data[i][j] = a[i][j];
        return a[i][j];
    }else if(data[i][j] != 0 ) {
        return data[i][j];
    }

    left=max(i+1,j,n);
    right=max(i+1,j+1,n);
    data[i][j] = (left>right)? (left+a[i][j]) : (right+a[i][j]);

    return data[i][j];
}
```

如何得到选择的路径呢

a

9	0	0	0	0
12	15	0	0	0
10	6	8	0	0
2	18	9	5	0
19	7	10	4	16

data

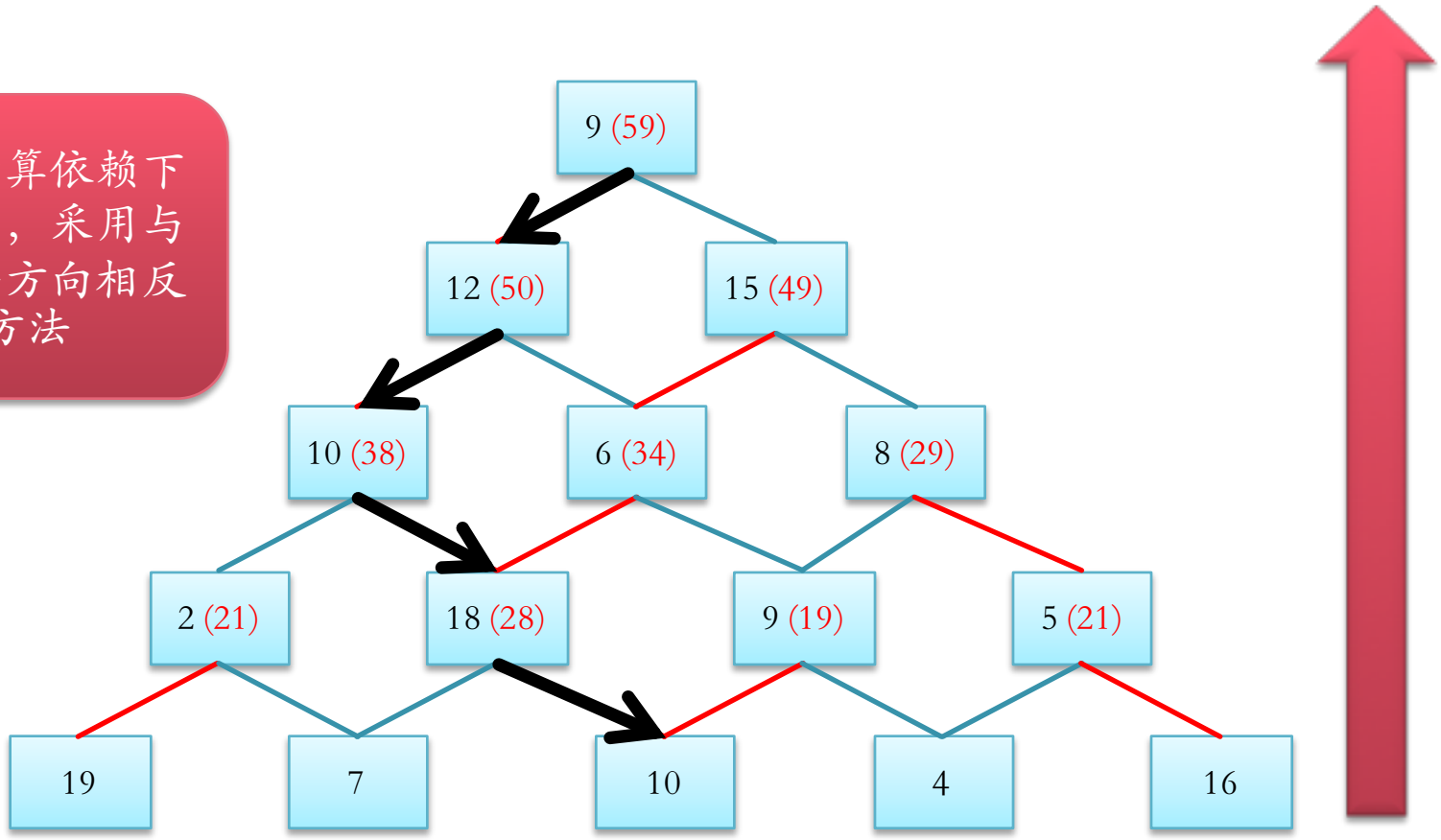
59	0	0	0	0
50	49	0	0	0
38	34	29	0	0
21	28	19	21	0
19	7	10	4	16

第(i,j)节点的左右分支分别存放在(i+1,j)和(i+1,j+1)中

左左右右

自底向上方法

上层的计算依赖下层的计算，采用与递归方法方向相反的方法



程序实现

```
#define N  5
int a[N][N];
int data[N][N];
int dir[N][N];

int main() {
    int left,right;
    for(int i=N-1; i>=0; i--) {
        for(int j=0; j<=i; j++) {
            if(i == N-1) {
                data[i][j] == a[i][j];
            }else{
                left = data[i+1][j];
                right = data[i+1][j+1];
                if (left > right) {
                    data[i][j] = left + a[i][j];
                    dir[i][j] = 0;
                }else{
                    data[i][j] = right + a[i][j];
                    dir[i][j] = 1;
                }
            }
        }
    }
}
```

a

9	0	0	0	0
12	15	0	0	0
10	6	8	0	0
2	18	9	5	0
19	7	10	4	16

data

59	0	0	0	0
50	49	0	0	0
38	34	29	0	0
21	28	19	21	0
19	7	10	4	16

dir

0	0	0	0	0
0	0	0	0	0
1	0	1	0	0
0	1	0	1	0
0	0	0	0	0

在编程中要注意的

硬盘总比内存便宜很多

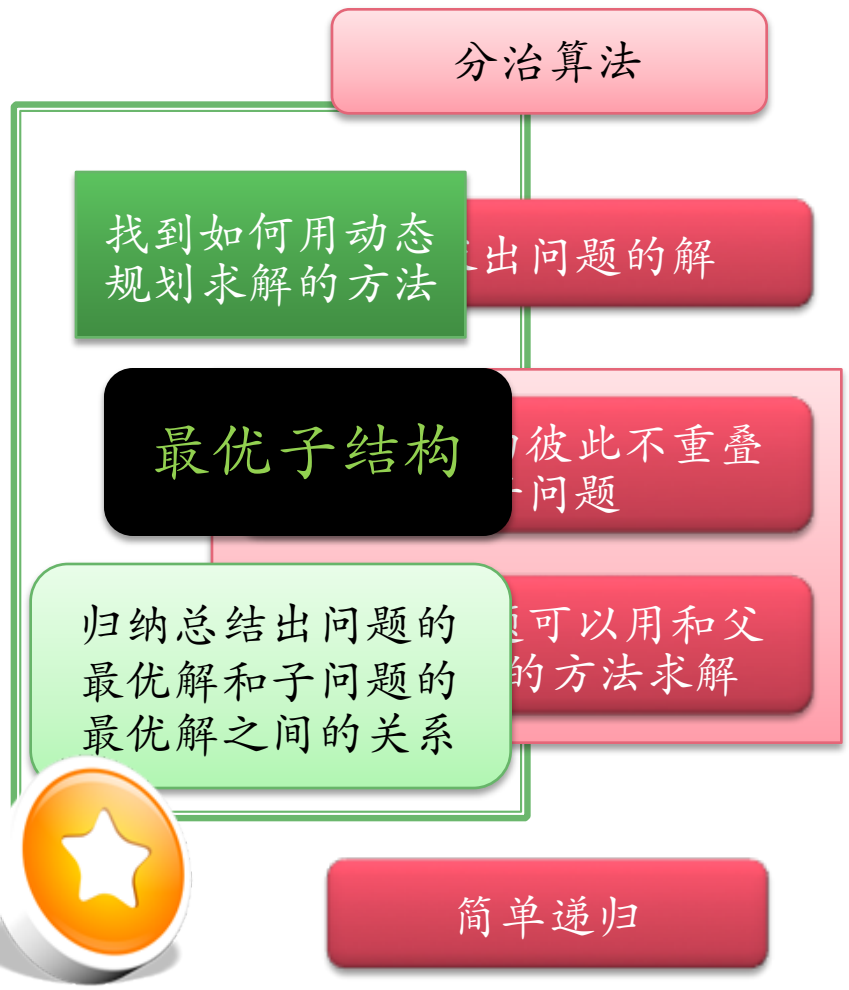
内存总比时间便宜很多

用空间换取时间

Google™

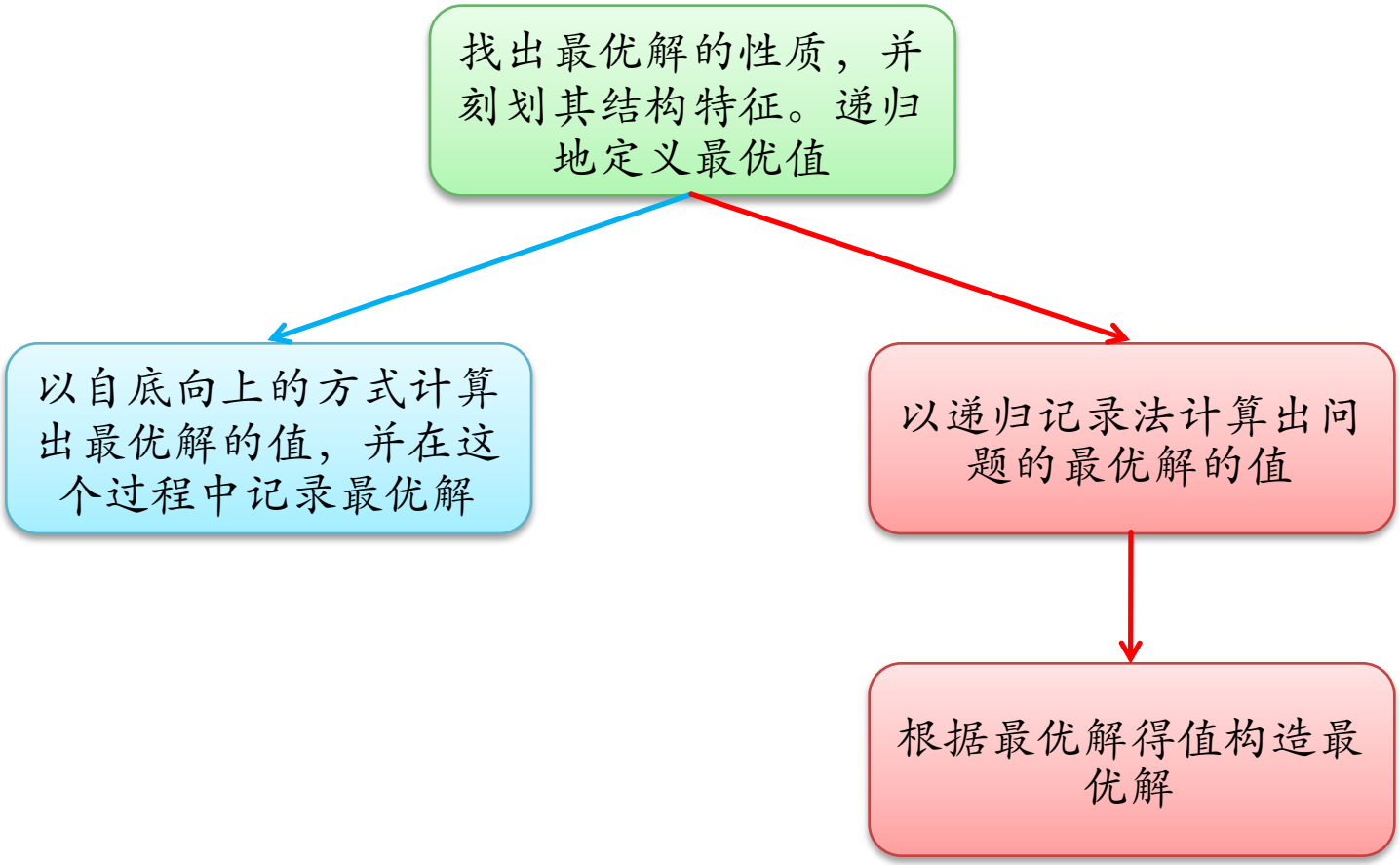
与分治算法有什么异同?

两种算法的异同


分治算法
找到如何用动态规划求解的方法
出问题的解
最优子结构
彼此不重叠
问题
归纳总结出问题的最优解和子问题的最优解之间的关系
可以用和父
的方法求解
简单递归


动态规划
如何求出问题的最优解
观察发现在使用普通循环和递归式出现的重复计算问题
父问题的最
问题的
子问题
问题，
重叠子问题
自底向
发现动态规划方法可以减少计算
递归记录法

动态规划基本步骤


找出最优解的性质，并刻划其结构特征。递归地定义最优值
以自底向上的方式计算出最优解的值，并在这个过程中记录最优解
以递归记录法计算出问题的最优解的值
根据最优解得值构造最优解

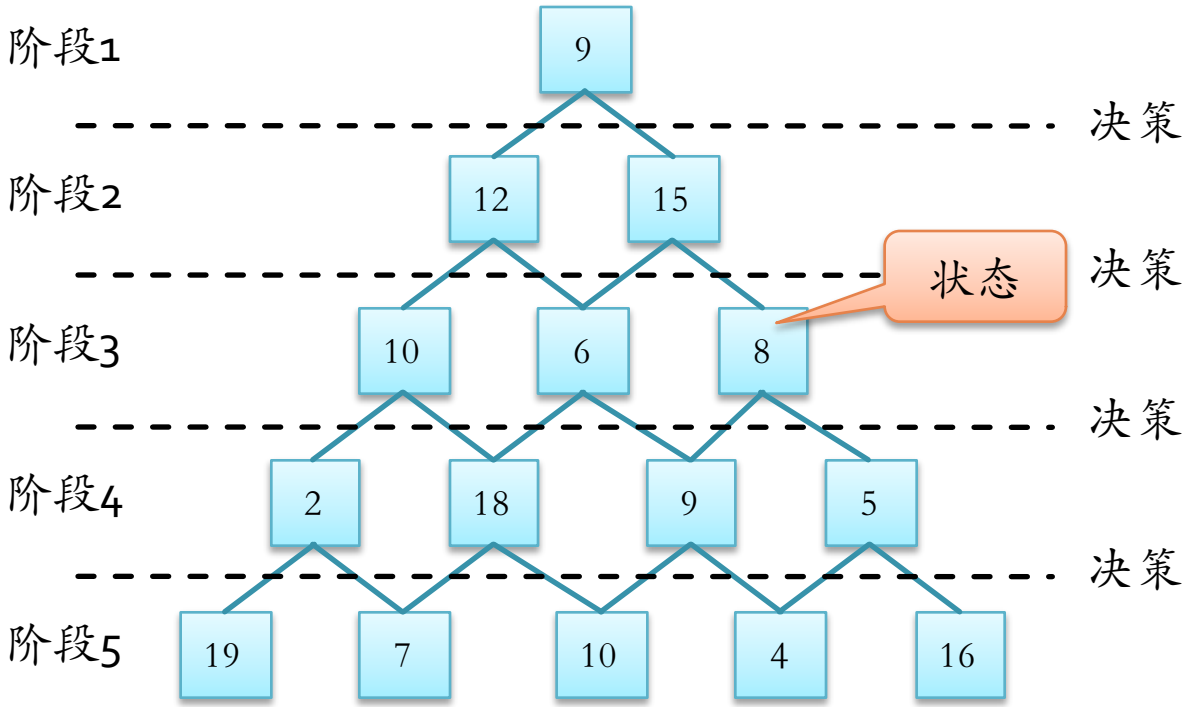
基本概念，另一个角度

阶段：把问题分成几个相互联系的有顺序的几个环节，这些环节即称为阶段

状态：某一阶段的出发位置称为状态。通俗地说状态是对问题在某一时刻的进展情况的数学描述

决策：从某阶段的一个状态演变到下一个阶段某状态的选择

状态转移方程：根据上一阶段的状态和决策导出本阶段的状态。这就像是“递推”



多阶段决策问题

从最优解得角度来说，找到最优子结构是求解的关键

从多阶段决策角度来说，找到状态转移方程式求解的关键

Problem： 滑雪问题

全国青少年信息学奥林匹克竞赛上海市队选拔赛 2002
http://poj.org/problem?id=1088

题目描述

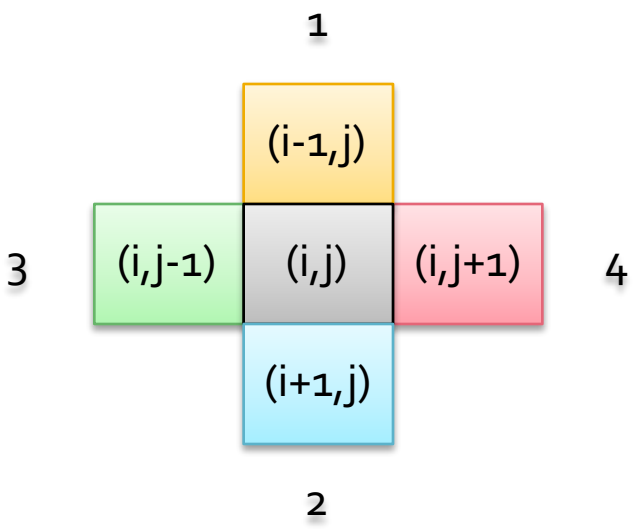
Michael喜欢滑雪百这并不奇怪，因为滑雪的确很刺激。可是为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。Michael想知道在一个区域中最长底滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。下面是一个例子。一个人可以从某个点滑向上下左右相邻四个点之一，当且仅当高度减小。在上面的例子中，一条可滑行的滑坡为24-17-16-1。当然25-24-23-...-3-2-1更长。事实上，这是最长的一条。

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Input：输入的第一行表示区域的行数R和列数C(1 <= R,C <= 100)。下面是R行，每行有C个整数，代表高度h，0<=h<=10000。

Output输出最长区域的长度。

分析与求解



数组m[N][N]保存着每个点的高度

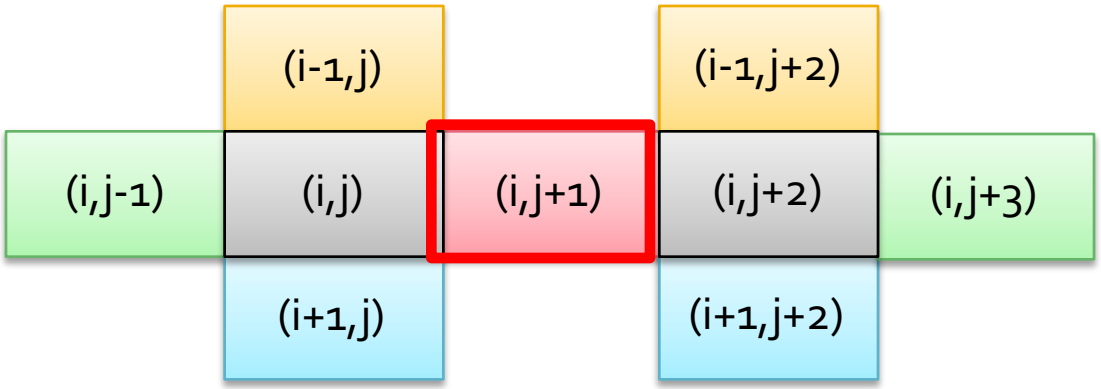
数组a[N][N]代表以每个点位起点的最长线路的长度

a[i][j]的值是，周围四个点中高度比m[i][j]的那些点中，以自身为起点的最长线路长度+1

a(i,j) = max

- if (m[i-1][j] < m[i][j]) return a[i-1][j]+1; else return 0
- if (m[i+1][j] < m[i][j]) return a[i+1][j]+1; else return 0
- if (m[i][j-1] < m[i][j]) return a[i][j-1]+1; else return 0
- if (m[i][j+1] < m[i][j]) return a[i][j+1]+1; else return 0

解结构分析



重叠子问题

用自顶向下递归记录法：每次计算了一个新的点的最大高度值就将其记录下来

求解过程—自底向上

我们发现计算任何一个点的最长距离，所需要的子问题的信息都是他周围高度比他低的点的最大长度

如果可以根据矩阵内点的从小到大的顺序计算每个点最长距离

这样当需要计算某个点的最长距离是，需要的周围的点的最长距离都已经知道了，可以避免重复计算

用三个数n*n的一维数组，分别存放矩阵中的点的高度值，点坐在的x坐标与y坐标，对高度值所在的数组进行排序，并根据排序的移动相应的移动对应的x坐标与y坐标的数组

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

v	1	2	3	4	5	16	17	18	19
x	1	1	1	1	1	2	2	2	2
y	1	2	3	4	5	1	2	3	4

Problem：矩阵连乘

经典问题

矩阵相乘基本知识

$$C_{m\times p} = A_{m\times n} B_{n\times p}$$

$$c_{ij} = \sum_{k=i}^{n} a_{ik} b_{kj}$$

两个矩阵是可相乘的充要条件第一矩阵的列数等于第二个矩阵的行数

```
#define  M 100
#define  N 200
#dfeine  P 50

float  a[M][N];
float  b[N][P];
float  c[M][P];

for(int  i=0;  i<M;  i++)   {
     for(int  j=0;  j<P;  j++)  {
          for(int  k=0;  k<N;  k++)  {
               c[i][j]  +=  a[i][k]  *  b[k][j];
          }
     }
}
```

M*N与N*P的矩阵相乘需要的计算量是M*N*P

矩阵相乘满足结合律
(AB)C = A(BC)

矩阵相乘不满足交换律，甚至不能相乘

矩阵连乘

- 给定n个要相乘的矩阵构成的序列$A_1,A_2,\ldots,A_n$，A_i与A_{i+1}是可乘的，要计算乘积

$$A_1A_2\ldots A_n$$

- 但是由于矩阵乘法具有结合律，所以可以有很多种计算顺序，我们给矩阵相乘加上括号来表示他们相乘的顺序
- 一个矩阵的乘法是加全部括号的
  - 如果他是单个矩阵
  - 两个加全部括号的矩阵的乘积外再加括号

举例

- A_1, A_2, A_3, A_4 四个矩阵相乘可能的顺序

$(A_1 (A_2 (A_3 A_4)))$

$(A_1 ((A_2 A_3) A_4))$

$((A_1 A_2)(A_3 A_4))$

$((A_1 (A_2 A_3)) A_4)$

$(((A_1 A_2) A_3) A_4)$

- 不同顺序的三个矩阵相乘所需运算量不同
  - A_1 10x100
  - A_2 100X5
  - A_3 5X50

$((A_1 A_2) A_3)$

10*100*5 +10*5*50

=7500

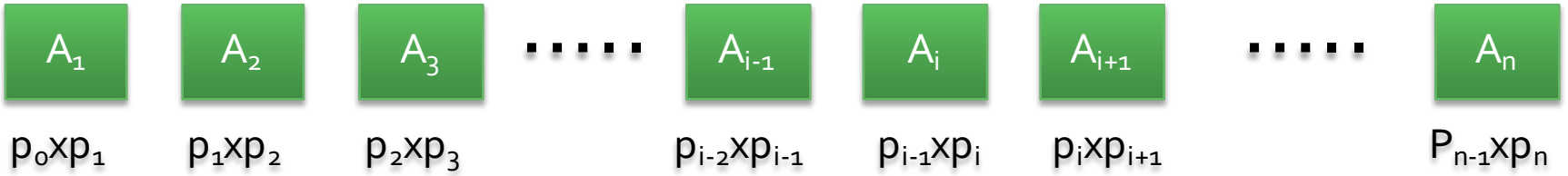
$(A_1 (A_2 A_3))$

100*5*50 +10*100*50

=75000

问题描述

- 给定n个矩阵构成一个链$<A_1,A_2,\ldots,A_n>$，矩阵A_i的维数为$p_{i-1} \times p_i$，i=1,2,3.....n；对乘积$A_1A_2\ldots A_n$以一种最小化乘法次数的方式进行加全部括号，也就是确定这个序列相乘的顺序

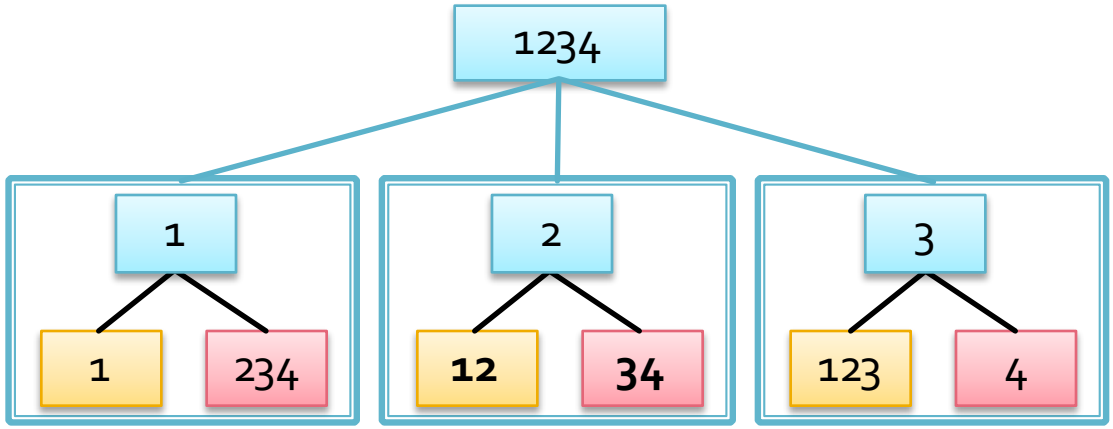


用$p_0\ p_1\ p_2\ldots.p_n$这n+1个数来表示可连乘的n个矩阵的大小

穷举法

如何例举出所有的求解次序

每次都是两个矩阵相乘，最后一步也是两个矩阵相乘，最后一步只有n-1种可能



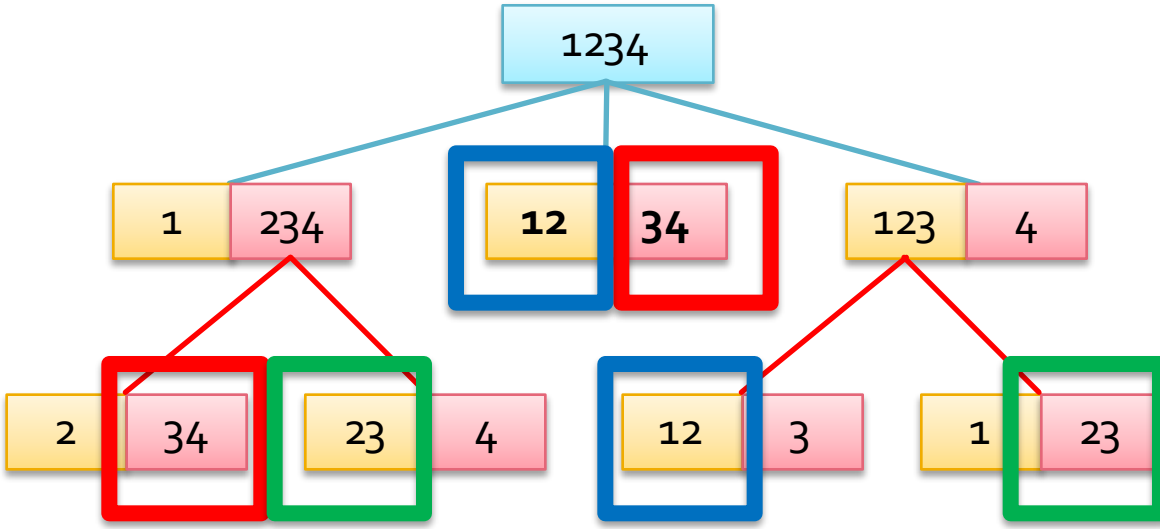
$(A_1 (A_2 (A_3 A_4)))$

$(A_1 ((A_2 A_3) A_4))$

$((A_1 A_2) (A_3 A_4))$

$(((A_1 A_2) A_3) A_4)$

$((A_1 (A_2 A_3)) A_4)$



穷举法的时间复杂度

- 用P(n)表示n个矩阵可能的所有加全括号方案数
- n=1时，P(n)=1
- n>=2时，就是两个加全部括号的子序列的乘积的乘积，分裂发生在第k处时，方案数为P(k)P(n-k)

$\Omega(4^n/n^{3/2})$

$$P(n)=\begin{cases}1 & n=1\\ \sum_{k=1}^{n-1}P(k)P(n-k) & n\geq 2\end{cases}$$

分析问题的最优子结构1

- 我们记m[i,j]代表对矩阵连乘$A_iA_{i+1}\cdots A_j$求得的最少计算量，其中i<=j
- 最少计算量乘积的最后一步肯定是两个矩阵B、C相乘，我们假设这两个矩阵分别为：
  - B：第i到k的矩阵的乘积
  - C：第k+1到j的矩阵的乘积，

$$S=A_iA_{i+1}\cdots A_kA_{k+1}\cdots A_j$$

等价于：假设最后一步在k处分可以得到最少的计算量

$$B = A_iA_{i+1}\cdots A_k$$

$$C = A_{k+1}\cdots A_j$$

分析问题的最优子结构2

$p_{i-1} \times p_j$

$S = A_iA_{i+1}\ldots A_kA_{k+1}\ldots A_j$

m[i,j]

$p_{i-1} \times p_k$

$B = A_iA_{i+1}\ldots A_k$

m[i,k]

$p_k \times p_j$

$C = A_{k+1}\ldots A_j$

m[k+1,j]

为什么整体的最优解一定要求子问题的最优解呢?

$$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} \times p_k \times p_j$$

分析问题的最优子结构3

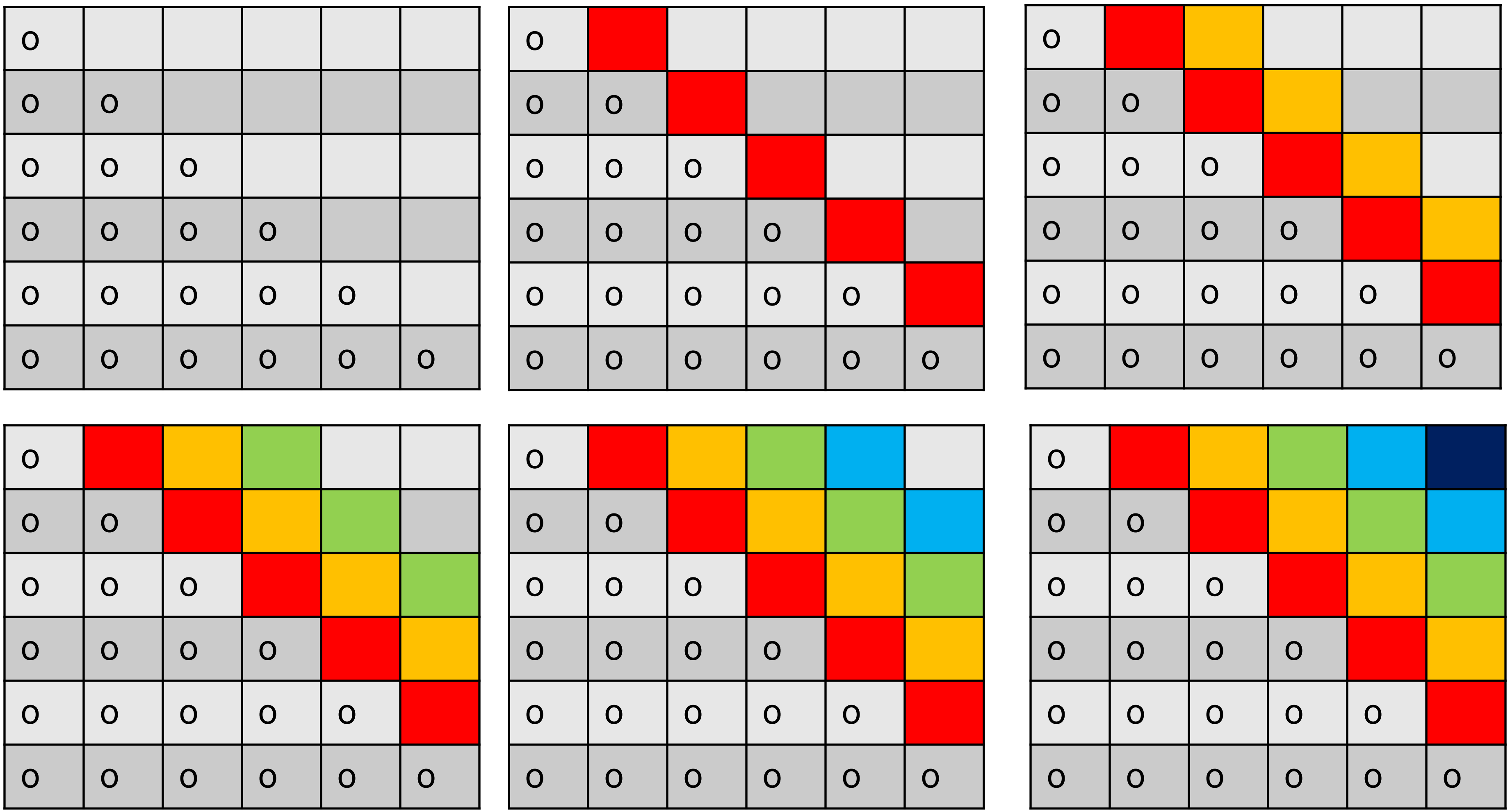
- 我们前面假设得到最优解得最后一步出现在划分k处，实际上我们不知道这个k到底是多少，但是我们知道这个k可以达到计算量最小
- 当只剩一个矩阵时，已经不需要计算了，所以有 m[i,i] = 0 , 1<=i<=j

$$m[i,j] = \min(m[i,k] + m[k+1,j] + p_{i-1} \times p_k \times p_j), \quad i<=k<j$$
$$m[i,i] = 0 , 1<=i<=j$$

求解过程

- 我们可以从m[i,i] (i=0,1,...,n-1)开始，逐步推算m[i,i+1] (i=0,1,...n-2)，m[l,i+2] (i=0,1,...n-3).....，我们最终的目标是计算m[0,n-1]
- 用一个n+1个元素的数组p存放数组维数
- 用一个n*n的二位数组m来存放m的值
- 用附加的一个n*n的数组s来存放每次取最小值的时候所决定的k，也就是可以达到最小值的分隔点，我们将通过这个矩二维数组构造一个最优解

```
#define N 6
int p[N];
int m[N][N];
int s[N][N];

for(int i=0; i<N; i++)
    m[i][i] = 0;
for(int t=1; t<N; t++) {
    for(int i=0; i<N-p; i++) {
        int j = i+t;
        m[i][j] = t * max(p) * max(p);
        s[i][j] = 0;
        for(int k=i; k<j; k++) {
            int min = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
            if (min < m[i][j]) {
                m[i][j] = min;
                s[i][j] = k;
            }
        }
    }
}
```



例子

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+p_{i-1}p_kp_j\} & i<j\end{cases}$$

矩阵	规模
A1	30×35
A2	35×15
A3	15×5
A4	5×10
A5	10×20
A6	20×25

m[2][5] $= \min\begin{cases} m[2][2]+m[3][5]+p_1p_2p_5 = 0+2500+35\times15\times20 = 13000 \\ m[2][3]+m[4][5]+p_1p_3p_5 = 2625+1000+35\times5\times20 = 7125 \\ m[2][4]+m[5][5]+p_1p_4p_5 = 4375+0+35\times10\times20 = 11375 \end{cases}$

	1	2	3	4	5	6
1	0	15750	7875			
2		0	2625	4375		
3			0	750	2500	
4				0	1000	3500
5					0	5000
6						0

构造最优解

	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						5

从3处将1-6分为1-3和4-6两段，查看s[1][3]和s[4][6]

$((A_1A_2A_3)(A_4A_5A_6))$

	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						5

将1-3从1处分为1和2-3
将4-6从5处分为4-5和6

$((A_1\,(A_2A_3))\,(A_4A_5)A_6))$

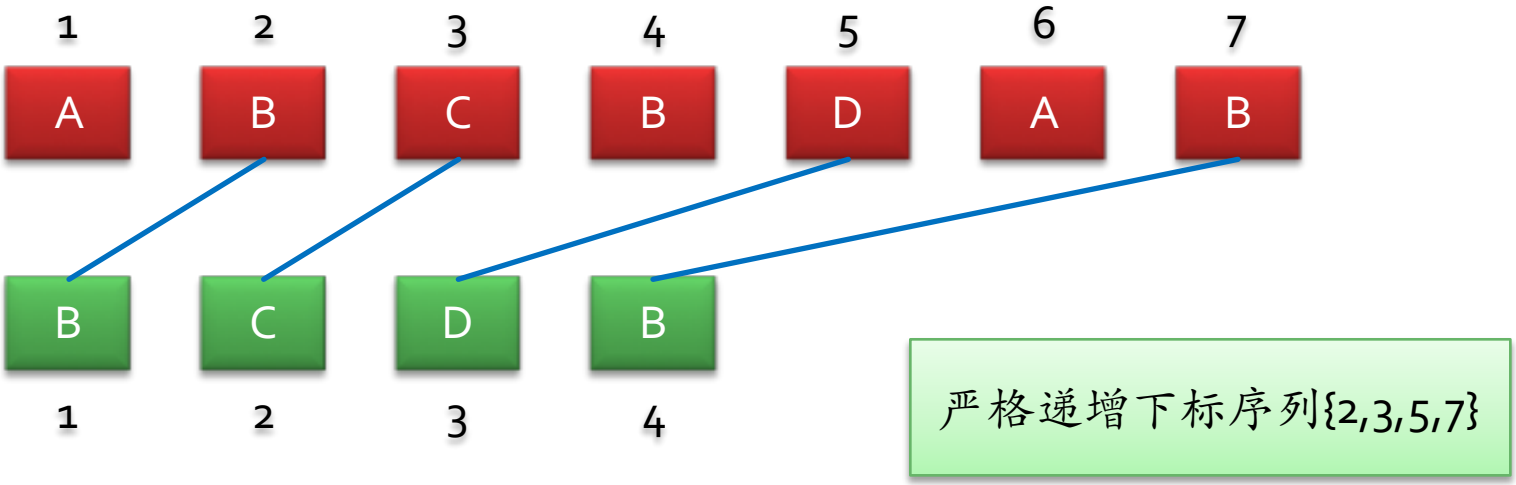
通过以上这个问题，我们已经基本了解了动态规划的整体框架以及实现步骤，在之后的问题中，我们只考虑如何找到并正确的写出最优子结构以及发现重叠子问题，而不再对整个过程加以阐述

Problem： 最长公共子序列(LCS)

经典问题

子序列

- 若给定序列$X=\{x_1,x_2,\cdots,x_m\}$，则另一序列$Z=\{z_1,z_2,\cdots,z_k\}$，是X的子序列是指存在一个严格递增下标序列$\{i_1,i_2,\cdots,i_k\}$使得对于所有$j=1,2,\cdots,k$有：$z_j=x_{i_j}$。



问题描述

- 给定2个序列X和Y，当另一序列Z既是X的子序列又是Y的子序列时，称Z是序列X和Y的公共子序列
- 给定2个序列$X=\{x_1,x_2,\cdots,x_m\}$和$Y=\{y_1,y_2,\cdots,y_n\}$，找出X和Y的最长公共子序列（LCS）

最长公共子序列的结构

- 设序列$X=\{x_1,x_2,\cdots,x_m\}$和$Y=\{y_1,y_2,\cdots,y_n\}$的最长公共子序列为$Z=\{z_1,z_2,\cdots,z_k\}$，则
  - (1)若$x_m=y_n$，则$z_k=x_m=y_n$，1且Z_{k-1}是X_{m-1}和Y_{n-1}的最长公共子序列。2
  - (2)若$x_m\neq y_n$且$z_k\neq x_m$，则Z是X_{m-1}和Y的最长公共子序列。3
  - (3)若$x_m\neq y_n$且$z_k\neq y_n$，则Z是X和Y_{n-1}的最长公共子序列。4

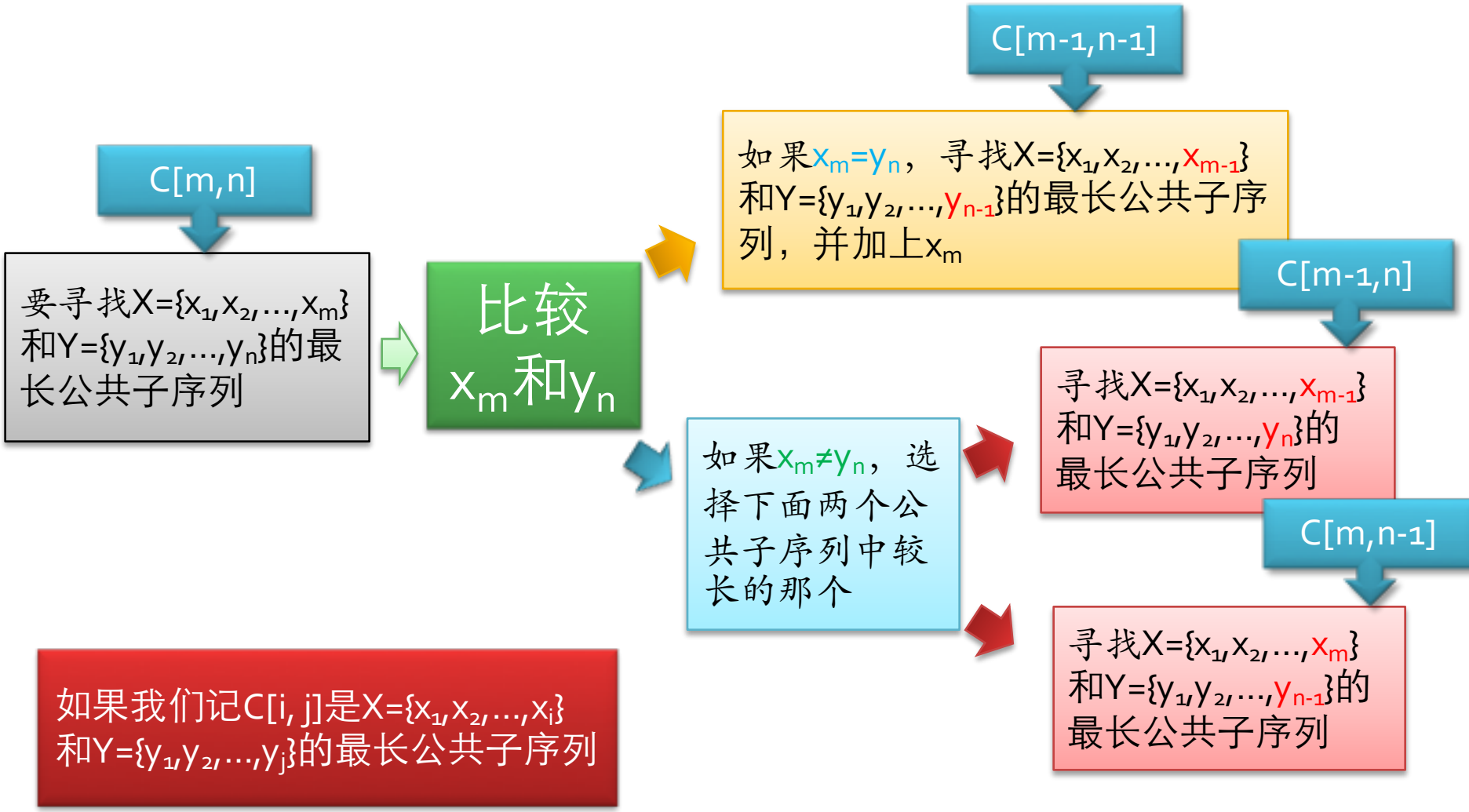
都是用反证法来证明

具体证明见《算法导论》P209 定理15.1

反证法与数学归纳法

- 有50位猎人每人养了一条狗，这些狗中至少有一条病狗，这些猎人要找出病狗，并由自己的主人把它枪杀了，第一天，没有听到枪声，第二天也没有听到，第三天听到一片枪声，问有多少只病狗?

算法运行过程


C[m,n]
要寻找X={x1,x2,…,xm}和Y={y1,y2,…,yn}的最长公共子序列
比较xm和yn
如果xm=yn，寻找X={x1,x2,…,xm-1}和Y={y1,y2,…,yn-1}的最长公共子序列，并加上xm
C[m-1,n-1]
如果xm≠yn，选择下面两个公共子序列中较长的那个
寻找X={x1,x2,…,xm-1}和Y={y1,y2,…,yn}的最长公共子序列
C[m-1,n]
寻找X={x1,x2,…,xm}和Y={y1,y2,…,yn-1}的最长公共子序列
C[m,n-1]
如果我们记C[i, j]是X={x1,x2,…,xi}和Y={y1,y2,…,yj}的最长公共子序列

最优子结构

$$c[i][j]=\begin{cases}0 & i=0,j=0\\ c[i-1][j-1]+1 & i,j>0;x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0;x_i\neq y_j\end{cases}$$

如何用递归记忆法和自底向上法求解能，如何记录到底哪些元素师两个序列的公共子序列呢?

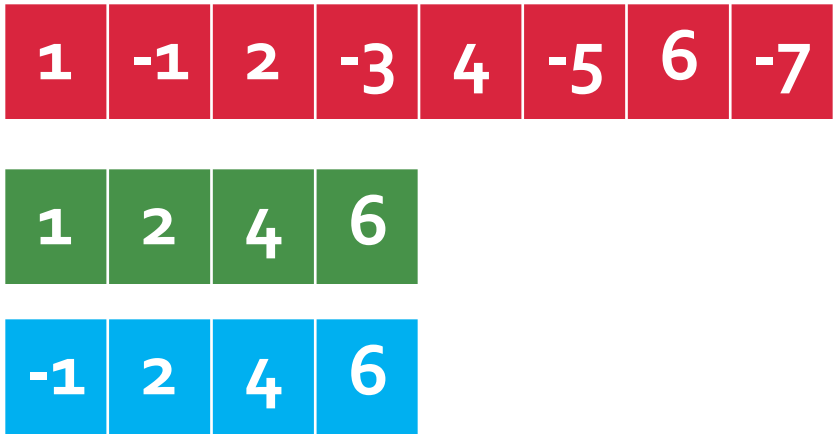
作业：参考《算法导论》15.4节用两种方法实现

Problem： 最长递增子序列(LIS)

经典问题

问题描述

- 给定一个整数序列，可以从中抽取出一个子序列，使得这个子序列中的元素是递增的，请写出一个时间复杂度尽量低的程序找到这个序列的最长的递增子序列



解法1： 利用LCS

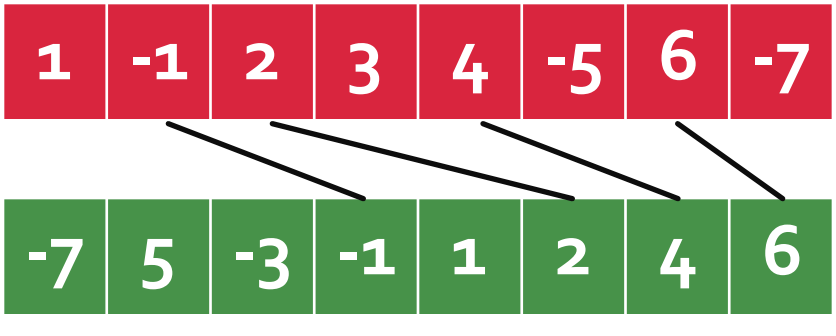
原序列

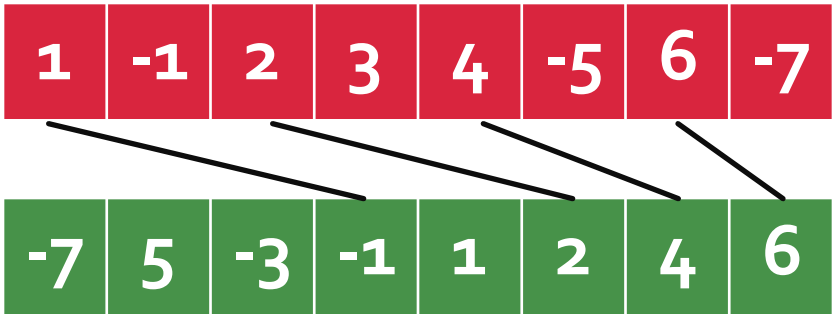
1	-1	2	3	4	-5	6	-7

我们将原序列排序

-7	-5	-3	-1	1	2	4	6

求这两个序列的LCS





-1	2	4	6

1	2	4	6

解法2：动态规划

我们令f[i]代表以array[i]结尾的最长递增子序列的长度

那么f[i+1]如何表示呢，也就是增加了一个元素的子问题f(

如果array[i+1]是array[1]....array[i+1]中最小的那个，那么f[i+1] = 1

f[i+1]

键问题是新增的这个元素

array[i]

成一个

没有记录

要写出

如果array[i+1]不是array[1]....array[i+1]中最小的那个，那么我们肯定对于那些array[k] < array[i+1] 的k，可以把array[i+1]放到那些原来以array[k]结尾的序列后面，形成新的序列，f[i+1]应该是这些序列的长度最长的

1	-1	2	3	4	-5	6	-7

1

f[i+1] = max{ 1, f[k]+1}

中k<=i, 且array[k] < array[i+1]

-1

1 2

-1 2

1 2 3

-1 2 3

我们用自底向上的方法求解f[1].....f[N]

出f[1].....f[N]的

这个最大值就

的解

计算复杂度

```
#define MAX 1000

int main() {
    int array[MAX];
    int lis[MAX];

    for(int i =0; i< MAX; i++ ) {
        lis[i] = 1;
        for(int j=0; j<i; j++ ) {
            if(array[i] > array[j] && lis[j]+1 > lis[i]) {
                lis[i] = lis[j] + 1;
            }
        }
    }
    return max(lis);
}
```

时间复杂度$O(n^2)$

我们通过添加一个数组B，B[j]表示长度为j的所有递增序列中最大值最小的那个数，可以将问题的时间复杂度降为O(nlgn)

参考：

《编程之美》2.16题的解法二和解法三

http://www.felix021.com/blog/read.php?1587

Problem： 数组最大子段和

20%

经典问题

微软亚洲研究院2006年笔试题

题目描述

- 一个有N个元素的数组A[0:N-1]，那么连续子数组之和的最大值是多少?

{1, -2, 3, 5, -3, 2}	8
{0, -2, 3, 5, -1, 2}	9
{-9, -2, -3, -5, -3}	-2

暴力求解

分别求出sum[i:j]的和，并求最大值

O(N³)

```
#define N 1000

int main() {
    int a[N];
    int maximum = a[0];
    int sum;

    for(int i=0; i<N; i++) {
        for(int j=i; j<N; j++) {
            for(int k=i; k<=j; k++) {
                sum += a[k];
            }
            if (sum > maximum) {
                maximum = sum;
            }
        }
    }
}
```

sum[i:j]的和等于sum[i:j-1]+a[j]

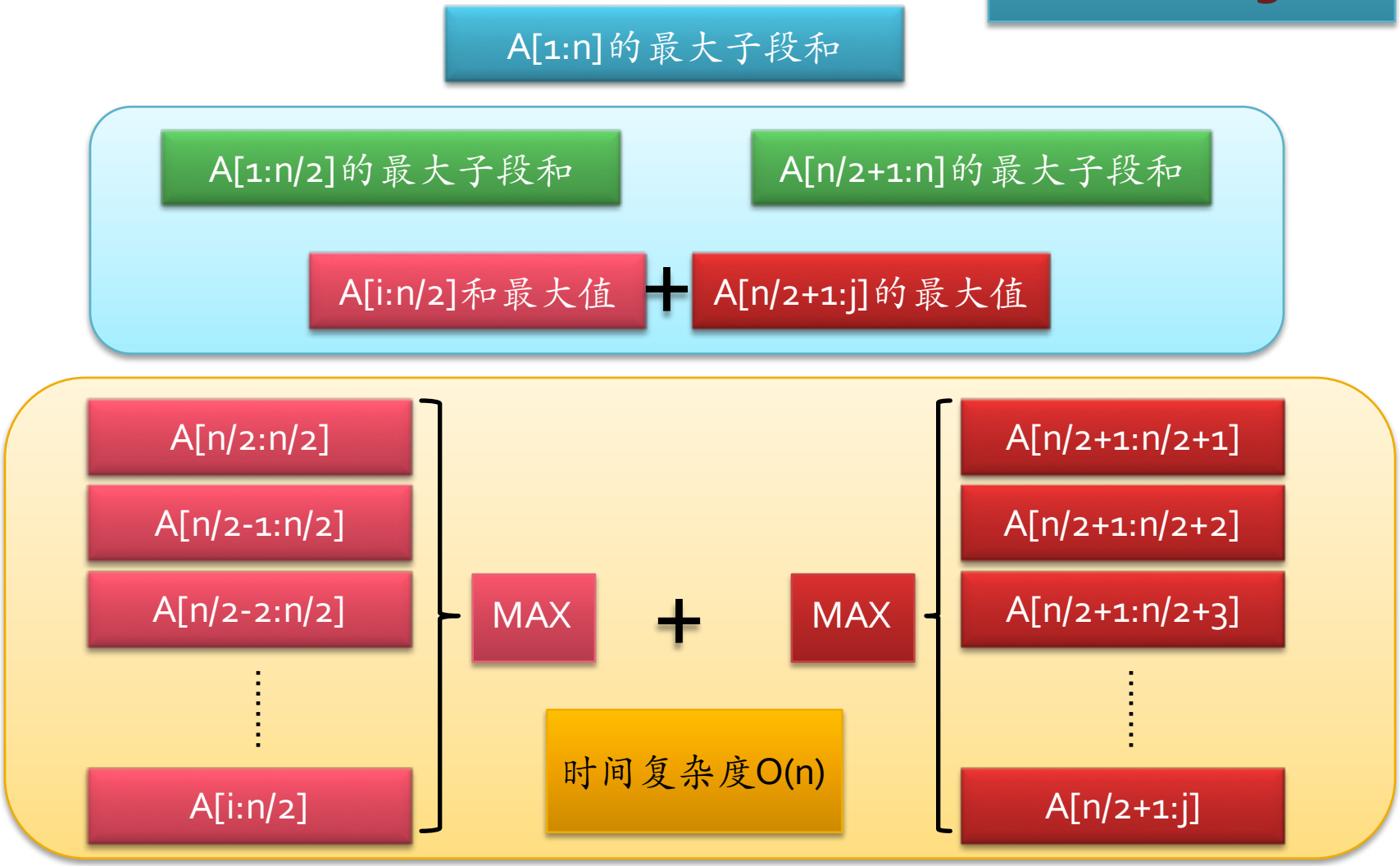
O(N²)

```
#define N 1000

int main() {
    int a[N];
    int maximum = a[0];
    int sum;

    for(int i=0; i<N; i++) {
        sum = 0;
        for(int j=i; j<N; j++) {
            sum += A[j]
            if (sum > maximum) {
                maximum = sum;
            }
        }
    }
}
```

分治策略

$T(n)= 2T(n/2)+O(n)$

$T(n) = O(nlgn)$



能否进步一优化呢

假设最大的一段子数组为A[i]……A[j]，他与A[o]之的关系

- o=i=j时,A[i]本身构成最大一段数组
- o=i<j时,最大的一段以A[o]开始
- o<i<j时,A[o]与最大的一段没关系

如果我们已知A[p]….A[N-1]的最大子段和记为All[p]，其中包含A[p]的最大子段和为Start[p]

All[p-1] = max{A[p-1], A[p-1]+Start[p], All[p]}

Start[p-1] = max{A[p-1], A[p-1]+Start[p]}

把从p-1到N-1的问题的最优解划为从p到N-1的问题的解

All[p-1] = max{A[p-1], A[p-1]+Start[p], All[p]}

Start[p-1] = max{A[p-1], A[p-1]+Start[p]}

```
#define N 1000

int main() {
    int A[N], Start[N], All[N];
    //initialize A

    Start[N-1] = A[N-1];
    All[N-1] = A[N-1];
    for(int i=n-2; i>=0; i--) {
        Start[i] = max(A[i],A[i]+Start[i+1]);
        All[i] = max(Start[i],All[i+1]);
    }
    //All[0] is the maxsum
    return 0;
}
```

时间复杂度
O(N)

扩展问题

- 如果不是一个数组，而是一个环，也就是一个首尾相连的数组，求他的最大子段和，情况会是怎么样呢?
- 如果不是一维数组而是二维数组，情况又会如何呢?

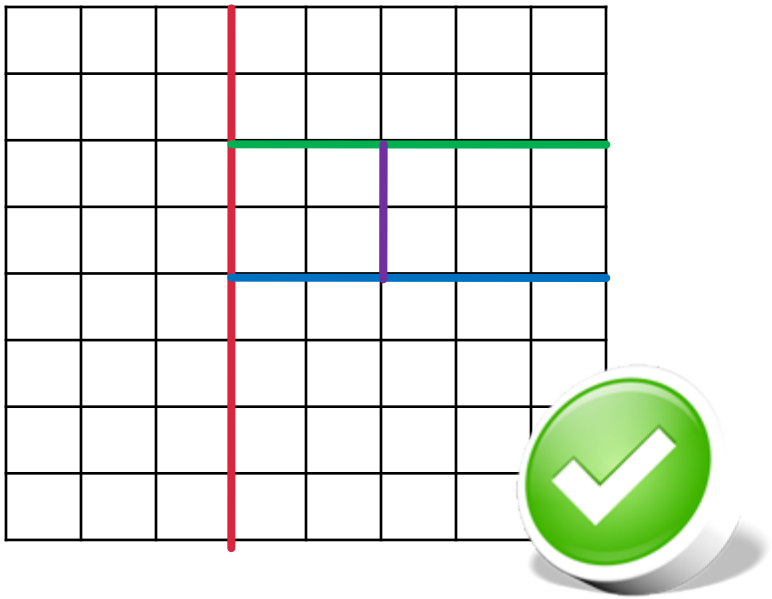
参考：
《编程之美》2.14题与2.15题

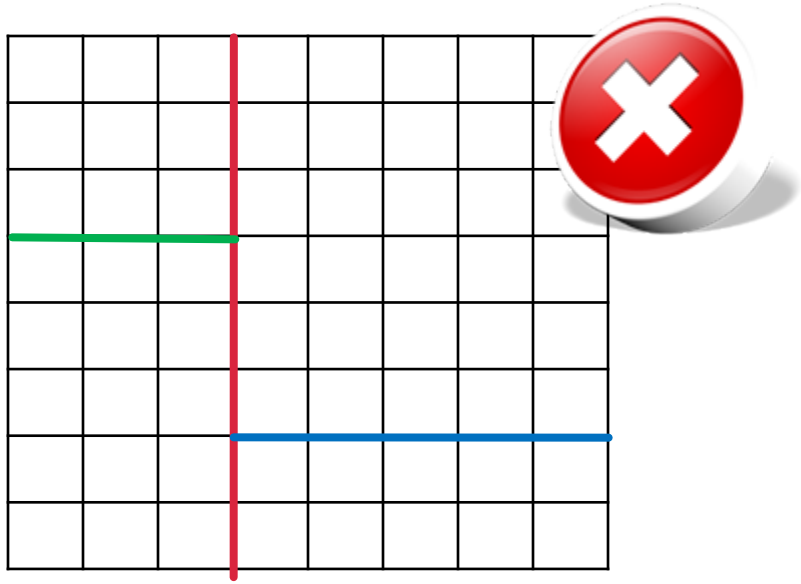
Problem：棋盘分割问题

全国青少年信息学奥林匹克竞赛
1999年 第二天 问题1 Chess

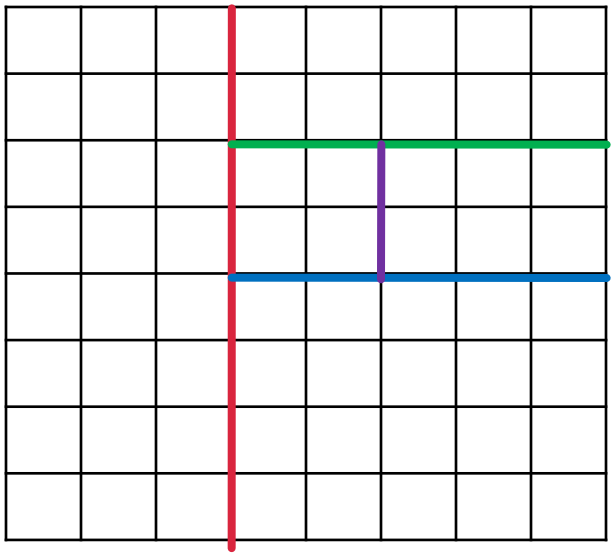
题目描述

- 将一个8X8的棋盘进行分割，每次割下一块矩形的棋盘并使剩下的棋盘也是矩形，再将剩下的继续分割，这样分割了n-1次之后，连同最后剩下的矩阵棋盘共有n块矩形棋盘。





- 原棋盘上每一格有一个分值，一块矩形棋盘的总分值为其所含各格的分值之和，现在需要将棋盘按上述规则分割成n块，并使各矩形棋盘的总分的均方差最小



分析问题

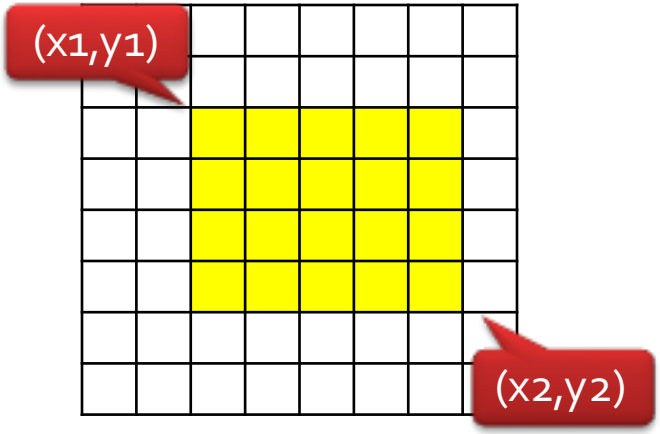
$$\text{均方差}\sigma=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n}},\text{其中}\bar{x}=\frac{\sum_{i=1}^{n}x_i}{n}$$

x_i是第i块矩形棋盘的总分

$$\sigma^2=\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n}=\frac{1}{n}\left(n(\bar{x})^2+\sum_{i=1}^{n}x_i^{\,2}-2\bar{x}\sum_{i=1}^{n}x_i\right)$$

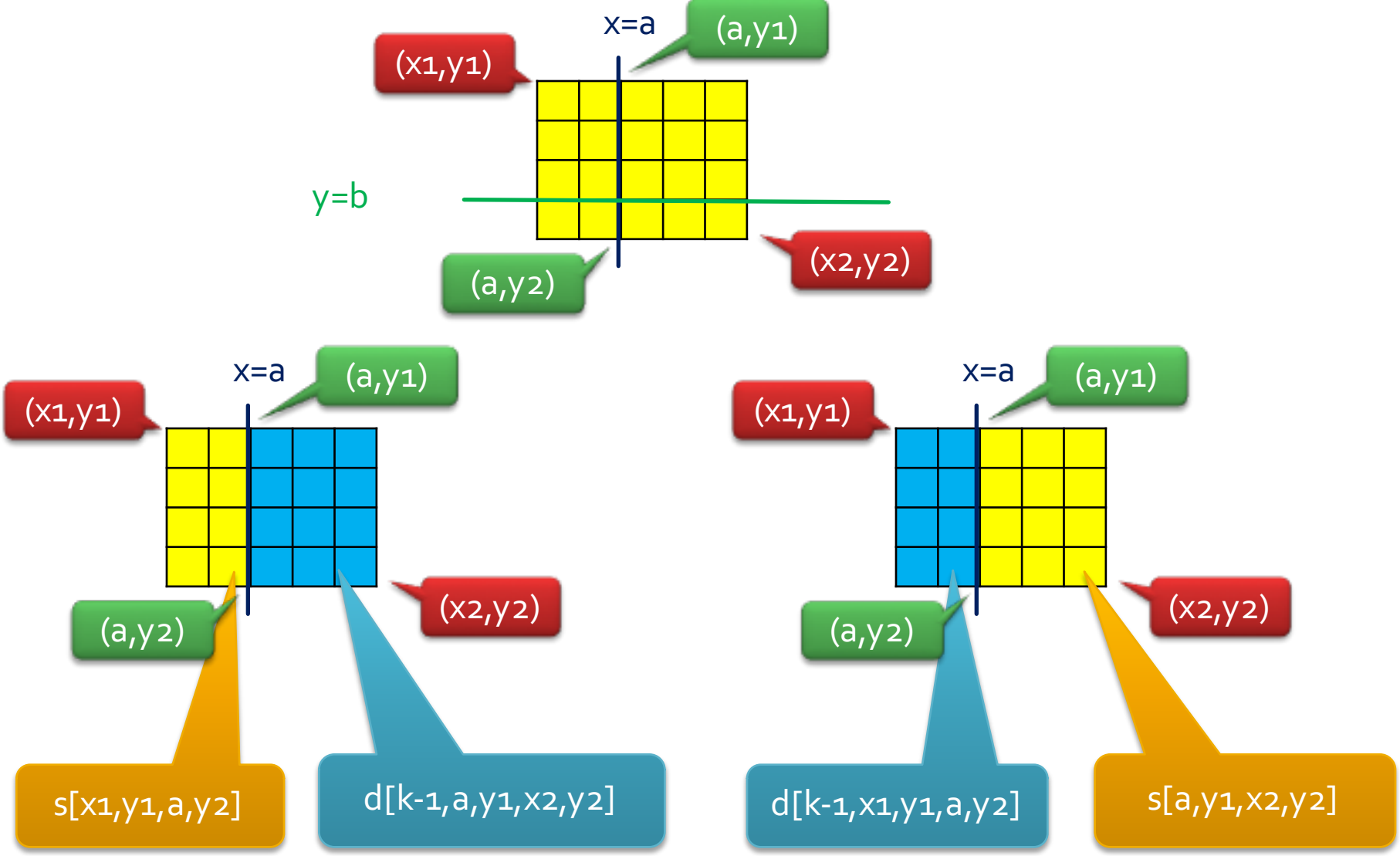
$$=\frac{1}{n}\sum_{i=1}^{n}x_i^{\,2}-(\bar{x})^2$$

设左上角坐标为(x1,y1)右下角坐标为(x2,y2)的棋盘的总分值的平方为s[x1,y1,x2,y2]，切割k次之后得到的k+1块棋盘的总平方和最小为d[k,x1,y1,x2,y2]



相当于求每个棋盘的总分的平方和最小

我们最后要求的是d[n,o,o,8,8]



$$d[k,x1,y1,x2,y2] = \min \{$$
$$\min\{s[x1,y1,a,y2]+d[k-1,a,y1,x2,y2],\ s[a,y1,x2,y2]+d[k-1,x1,y1,a,y2]\} \quad x1<a<x2,$$
$$\min\{s[x1,y1,x2,b]+d[k-1,x1,b,x2,y2],\ s[x1,b,x2,y2]+d[k-1,x1,y1,x2,b]\} \quad y1<b<y2$$
$$\}$$

Problem： 编辑距离

腾讯2007年校园招聘
面试题

问题的实际意义

系统是如何知道的？

问题描述：给出一个词典，词典中没有的单词被认为是错误的单词，根据词典中的单词对用户输入的错误单词进行纠错

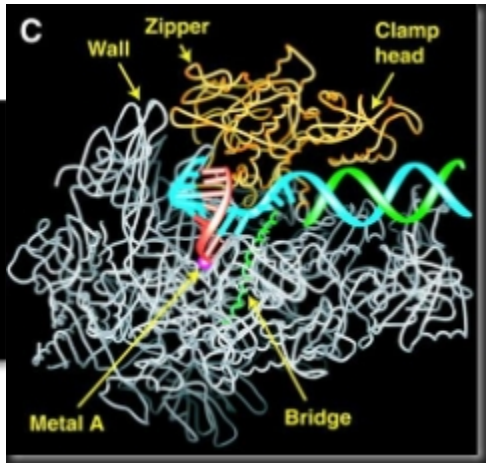
问题转化

o	c		u	r	r	a	n	c	e
o	c	c	u	r	r	e	n	c	e

在字典中找到与输入单词最“相近”的单词

如何定义两个字符串的相似度呢?

生物学中，有机体的基因组被划分为巨大的成为染色体的线状DNA分子，基因可以表示成包含了A、C、G、T的一个串。那如何通过判断两个病毒的基因是否相似来判断是否能使用同一种药物呢?



编辑距离

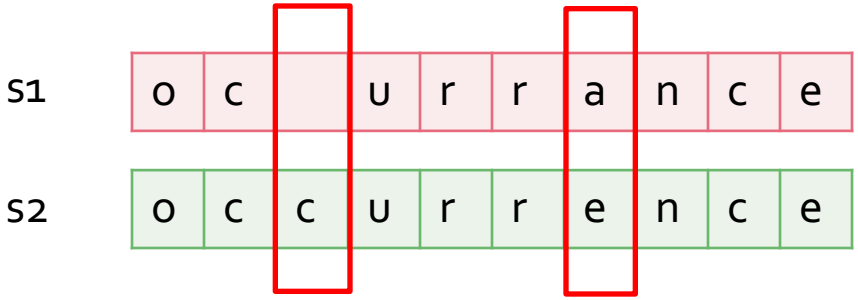
俄国科学家Vladimir Levenshtein在1965年发明，用他的名字命名，称为**Levenshtein distance，也叫做Edit distance**

两个字符串S1,S2的编辑距离是指，把S1和S2变成相同字符串（或者将一个字符串变为另一个字符串）需要下面操作的最小次数

1.把某个字符ch1变成ch2

2. 删除某个字符

3.插入某个字符



dis(S1,S2) = 2

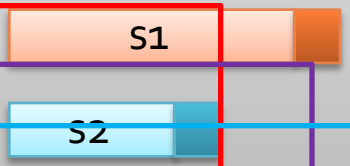
考虑最优子结构

计算两个字符串s1+ch1, s2+ch2的编辑距离有这样的性质:

d(s1,"") = d("",s1) = |s1|

d(ch1,ch2) = ch1 == ch2 ? 0 : 1;

d(s1+ch1, s2+ch2) = min(d(s1,s2)+ ch1==ch2 ? 0 : 1 ,
d(s1+ch1,s2)+1,
d(s1,s2+ch2)+1);



假设求两个字符串x[1:M], y[1:N]的编辑距离
我们记dis(i,j)为x[1:i]和y[1:j]的编辑距离

最终要计算dis[M,N]

dis(i,0) = i, dis(0,i) = i

dis(1,1) = x[i]==y[1]?0:1

dis(i+1,j+1) = min { dis(i,j)+x[i+1]==y[j+1]?0:1,
dis(i+1,j)+1,
dis(i,j+1)+1 }

假设求两个字符串x[1:M], y[1:N]的编辑距离
我们记dis(i,j)为x[1:i]和y[1:j]的编辑距离

P	O	L	Y	N	O	M	I	A	L

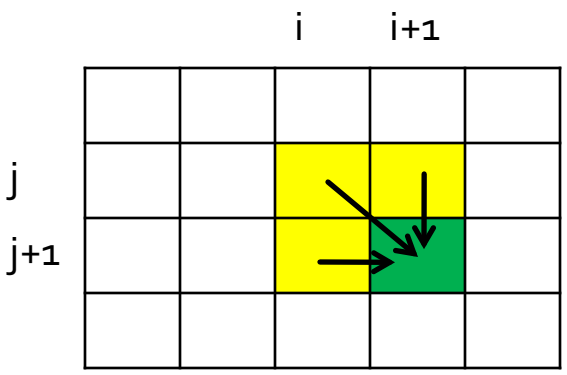
E	X	P	O	N	E	N	T	I	A	L

最终要计算dis[M,N]

dis(i,0) = I, dis(0,i) =

dis(1,1) = x[i]==y[1]?0:1

dis(i+1,j+1) = min{
dis(i,j)+x[i+1]==y[j+1]?0:1,
dis(i+1,j)+1,
dis(i,j+1)+1 }



		P	O	L	Y	N	O	M	I	A	L
	0	1	2	3	4	5	6	7	8	9	10
E	1	1									
X	2										
P	3										
O	4										
B	5										
E	6										
N	7										
T	8										
I	9										
A	10										
L	11										6

自底向上

更近一步考虑

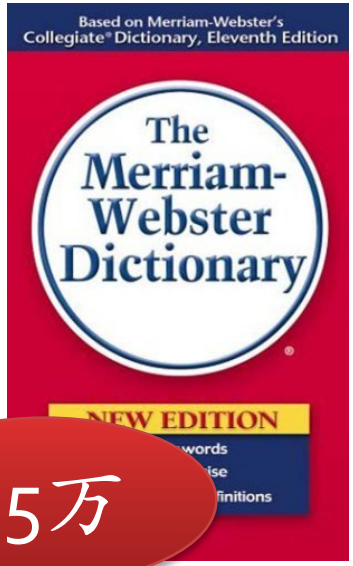
- 编辑距离的缺陷
  - 用户输入bsg是和bug更相似还是与bag更相似呢
  - 是否替换、插入、删除有着不同的概率呢?

回到最初的问题

- 既然已经知道如何定义两个字符串的相似度了，那么如何进行纠错呢。难道需要遍历词典中的所有词条么。

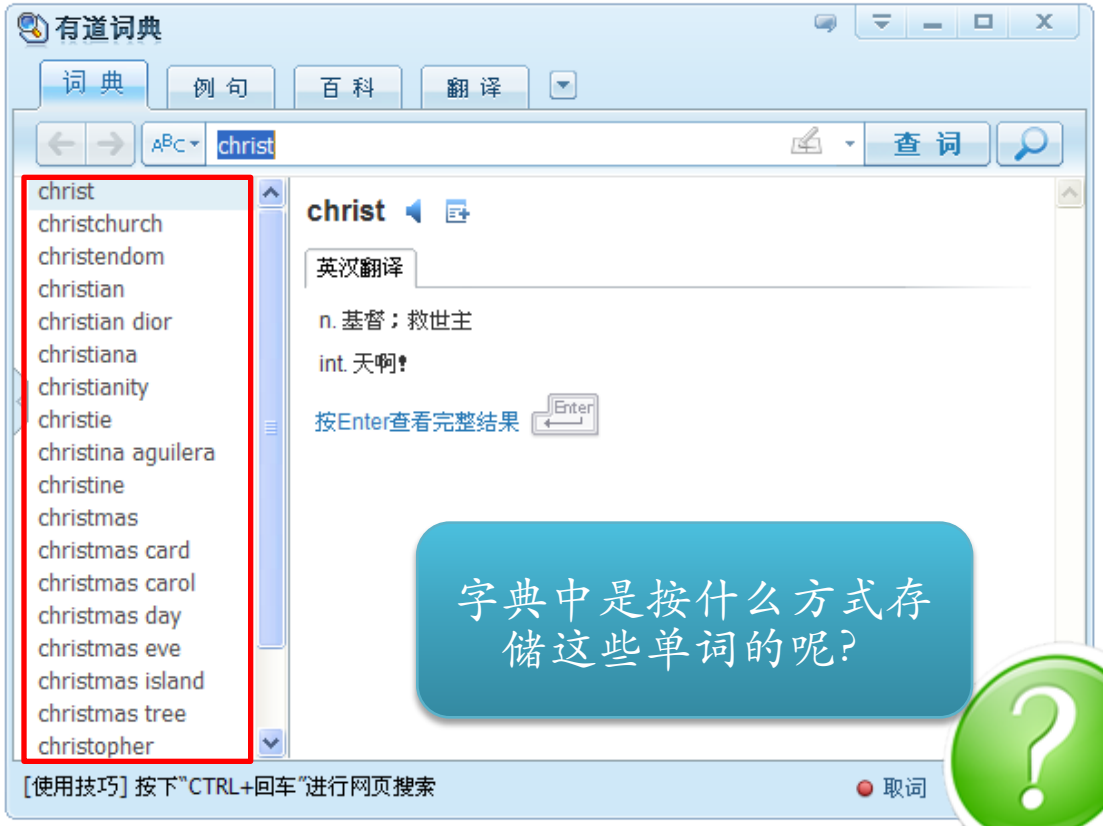
又让我想起了百度的一道面试题

给定一个词典，词典中有很多的单词，现给出一篇英文文章，问词典中的单词一共在文章中出现了多少次?



45万

还是把问题转化一下



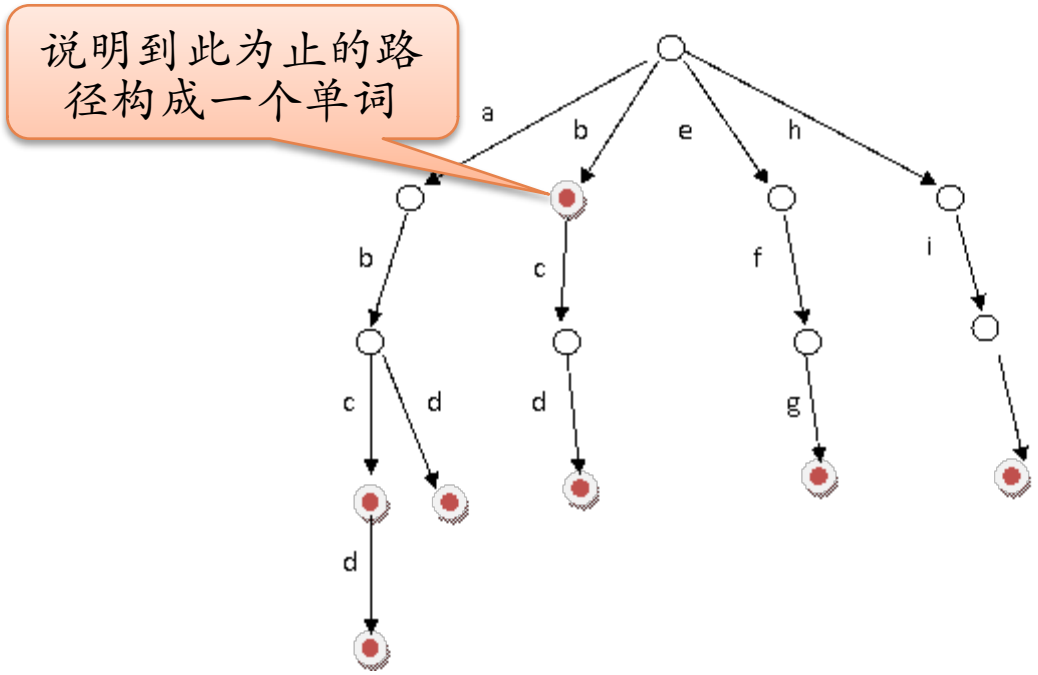
字典中是按什么方式存储这些单词的呢?

有道词典

金山词霸

TRIE树（Retrieval）

假设有b，abc，abd，bcd，abcd，efg，hii这六个单词



一棵26叉树

还是回到纠错的问题

要找和用户输入相近的，也就是两者编辑距离较小的
例如dis(error,correct) <= MD

dis(error,correct) >= abs(length(error) – length(correct))

MD>=abs(length(error) – length(correct))

length(error)-MD <= length(correct) <= length(error)+MD

所以我们将字典中具有相同长度的字符串分为一组，每组的代号就是这组中单词的长度，得到K个组

通常MD为1或者2

更好的选择是错误单词长度的四分之一

编辑距离性质

- 编辑距离的性质：令d(x,y)表示字符串x到y的编辑距离，那么显然
  - d(x,y) = o 当且仅当 x=y （编辑距离为o <==> 字符串相等）
  - d(x,y) = d(y,x) （从x变到y的最少步数就是从y变到x的最少步数）
  - d(x,y) + d(y,z) >= d(x,z) （从x变到z所需的步数不会超过x先变成y再变成z的步数）

那是不是在这2MD个组中的每个都要检查呢?

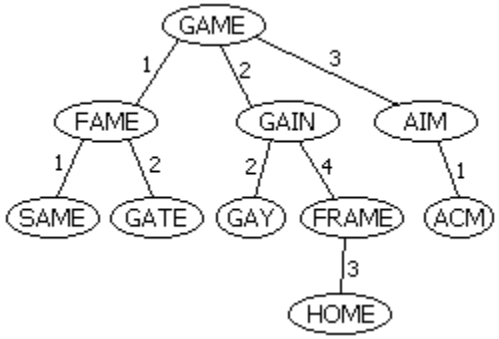
进一步考虑

如果dis(error, y) = 1 <MD，dis(y,z) >MD+1，即y和z有至少MD+2个字符不一样

dis(y, z) <= dis(z,error) + dis(error,y) = dis(error,z) + dis(error,y)

怎么知道哪些该检查哪些不该呢

1973年，Burkhard和Keller提出的BK Tree方法



推荐阅读

http://www.matrix67.com/blog/archives/333

http://hi.baidu.com/daizongh/blog/item/669d631636e6bc4e21a4e954.html

Problem：01背包问题

经典问题

问题描述

- 你去一个岛上寻宝，发现了一个宝藏，其中有n块重量不同的金砖，我们给这n块金砖进行编号1,2,….,n，他们的重量分别为w1,w2,……,wn。我们希望拿走的金装的总重量尽量重。但是背包最多只能承重W重量。请问如何选择。

简单起见：假设所有数值都是正整数

选择一个子集S，使得在满足$\sum_{i\in S} w_i \le W$的前提下，$\sum_{i\in S} w_i$ 达到最大

问题转化

选择一个子集S，使得在满足$\sum_{i\in S} w_i \le W$的前提下，$\sum_{i\in S} w_i$ 达到最大

我们可以将问题改写成

$$\max \sum_{i=1}^{n} w_i x_i$$

$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

如果让你手动去选该怎样呢?

大问题的最优解和子问题最优解

如果我们用m(k,j)代表如下子问题

$$\max \sum_{i=1}^{k} w_i x_i \qquad \begin{cases} \sum_{i=1}^{k} w_i x_i \le j \\ x_i \in \{0,1\}, 1 \le i \le k \end{cases}$$

即m(k,j)是背包容量为j，可选择物品为1,2,...,k时问题的最优值

$$m(i+1,j) = \begin{cases} \max\{m(i,j), m(i,j-w_{i+1}) + w_{i+1}\} & j \ge w_{i+1} \\ m(i,j) & 0 \le j < w_{i+1} \end{cases}$$

$$m(1,j) = \begin{cases} w_1 & j \ge w_1 \\ 0 & 0 \le j < w_1 \end{cases}$$

问题描述

- 你去一个岛上寻宝，发现了一个宝藏，其中有n块重量不同的宝贝，我们给这n块金砖进行编号1,2,….,n，他们的重量分别为w1,w2,……,wn；他们的价值分别为v1,v2,….,vn。我们希望可以拿走的宝贝的总价值尽力那高。但是背包最多只能承重W重量。请问如何选择。

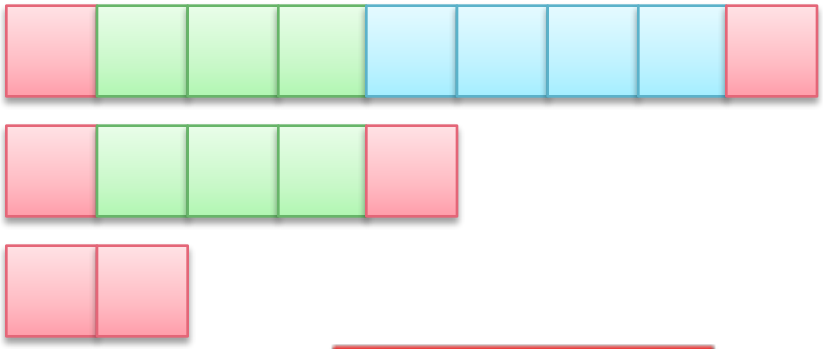
选择一个子集S，使得在满足$\sum_{i \in S} w_i \le W$的前提下，$\sum_{i \in S} v_i$达到最大

Problem： 方块消除

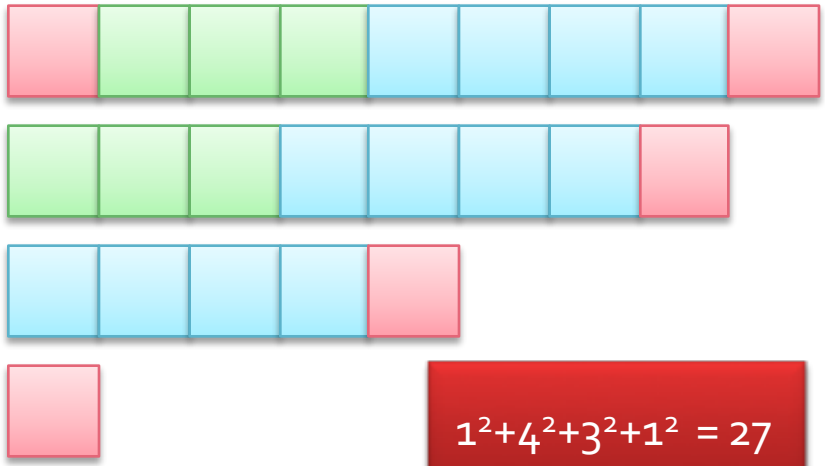
IOI（International Olympiad in Informatics）
2003 中国国家集训队讨论题
http://poj.org/problem?id=1390

问题描述

N个带颜色的方块排成一列，相同颜色的方块连成一个区域，游戏时你可以选择任意一个区域消除，设这个区域包含的方块数是x，则将得x^2到个分值，方块消去之后其右边的所有方块就会向左移动，与被消去的方块区域的左边相连。不同的消去方法所得到的分值不同，请设计一种方法得到最大的分值。



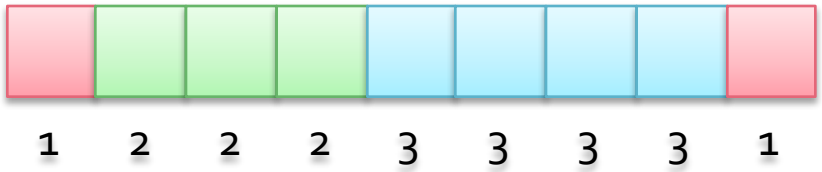
$4^2+3^2+2^2 = 29$



$1^2+4^2+3^2+1^2 = 27$

祖玛游戏

问题表示



(1, 1) (2, 3) (3, 4) (1, 1)
(a, b)表示b个a颜色的块组成的区域

Color = {1, 2, 3, 1}
Len = {1, 3, 4, 1}

对于一个具有l段的方块行，我们可以用两个长度为l一维数组来表示
color[i]表示第i个区域的颜色
len[i]比奥斯第i区域的长度

分析

寻找最优子结构，如何将问题的最优解划为子问题的最优解的组合的形式

我们设f[i][j]为消去第i个区域到第j个区域得到的最高得分

长度为leng[j]的第j块要不要马上消掉

] * leng[j]

第q+1段到第j-1段可以得到的最高积分 f[q+1][j-1]

一起消除，

假设若干段中最后一段是q则此时

组成

第i段到第q-1段，以及第q段和第j段组成的新的段的最高得分，但是由于新段大小发生了变化，我们不能再用原来的f[i][q]的形式表示了

最优子结构

(color[i],leng[i]),(color[i+1],leng[i+1]),……,(color[j-1,leng[j-1]]), (color[j],leng[j]+k)

我们设f[i][j][k]为消去第i个区域到第j-1个区域以及第j个区域和之后的颜色相同的k块合成的区域得到的最高得分

得分 = f[i][j-1][o] + leng[j] * leng[j]

如果要和前面的若干段一起消除，假设若干段中最后一段是q则此时的得分由两部分组成

第q+1段到第j-1段可以得到的最高积分 f[q+1][j-1][o]

第i段到第q-1段，以及第q段和第j段组成的新的段的最高得分，f[i][q][k+leng[j]]

f[i][j][k] = max(f[i][j-1][o] + leng[j] * leng[j] ,
f[q+1][j-1][o] + f[i][q][k+leng[j]])
其中color[q] = color[j]

Problem: 括号序列

ACM/ICPC Regional Contest
Northeast Europe 2001. Problem B

问题描述

- 定义一下序列是规则序列
  - 空序列是规则序列
  - 如果S是规则序列，那么[S]和(S)也是规则序列
  - 如果A、B是规则序列，那么AB也是规则序列
- 例如下面这些都是规则序列
  - ()、[]、(())、([])、()[]、()[()]
- 下面几个都不是规则序列
  - (、]、)(、([()
- 现给出由‘(’，‘)’，‘[’，‘]’构成的序列请添加尽量少的括号构造一个规则序列

观察

① 如果S是形如(S′)或[S′]，那么只需将S′划为规则的，S就是规则的

② 如果S是形如(S′，那么只需将S′变成规则的，然后在最后加上一个)

③ S形如[S、S]、S)，和上一种情况类似

④ 我们把序列分成两部分S_i ….. S_k,S_{k+1}…..S_j ，只要将S_i ….. S_k和S_{k+1}…..S_j都变成规则的，那么S就是规则的

找到了问题最优解和子问题最优解之间的关系

但是那种方法是添加括号最好的方法呢?

① 如果S是形如(S')或[S']，那么只需将S'划为规则的，S就是规则的

② 如果S是形如(S'，那么只需将S'变成规则的，然后在最后加上一个)

③ S形如[S、S]、S)，和上一种情况类似

④ 我们把序列分成两部分S_i ..... S_k,S_{k+1}.....S_j ，只要将S_i ..... S_k和S_{k+1}.....S_j都变成规则的，那么S就是规则

```
#define N 100;
char a[N];
int bracket(int i, int j) {
    if(i > j)
        return 0;
    else if (i ==j)
        return 1;

    int answer = N;
①   if( ( a[i] == '(' && a[j] == ')' ) || ( a[i] == '[' && a[j] == ']' ) ) {
        answer = min(answer, bracket(i+1, j-1));
    else if ( a[i] == '(' || a[i] == '[')
        answer = min(answer, bracket(i+1, j)+1);
    else if ( a[j] == ')' || a[j] == ']')
        answer = min(answer, bracket(i, j-1)+1);

    for(int k = i; k < j; k++ ) {
④       answer = min(answer, bracket(i,k) + bracket(k+1,j));
    }
    return answer;
}

int main() {
    int minbracket;
    minbracket = bracket(0, N-1);
}
```

②+③

简单递归法

观察有什么问题

还是存在非常严重的重复计算重叠子问题

采用递归记录法

更容易用来改进原始简单递归算法的性能，需要对源程序改动较小

开辟一个data[N][N]的数组，用data[i][j]，记录S_i .....S_j 这个序列需要加入括号的最少个数，这样data[o][N-1]就是我们最后的结果

```
#define N 100;
char a[N];
int bracket(int i, int j) {
    if(i > j)
        return 0;
    else if (i ==j)
        return 1;

    int answer = N;
    if( ( a[i] == '(' && a[j] == ')' ) ||
        ( a[i] == '[' && a[j] == ']' ) ) {
        answer = min(answer, bracket(i+1, j-1));
    else if ( a[i] == '(' || a[i] == '[')
        answer = min(answer, bracket(i+1, j)+1);
    else if ( a[j] == ')' || a[j] == ']')
        answer = min(answer, bracket(i, j-1)+1);

    for(int k = i; k < j; k++ ) {
        answer = min(answer, bracket(i,k) + bracket(k+1,j));
    }
    return answer;
}
```

```
#define N 100;
char a[N];
int data[N][N];      //initialize al

int bracket(int i, int j) {
    if(i > j)
        return 0;
    else if (i ==j) {
        data[i][i] = 1;
        return 1;
    }else if(data[i][j] >= 0 ) {
        return data[i][j];
    }

    int answer = N;
    if( ( a[i] == '(' && a[j] == ')' ) || ( a[i] == '[' && a[j] == ']' ) ) {
        answer = min(answer, bracket(i+1, j-1));
    else if ( a[i] == '(' || a[i] == '[')
        answer = min(answer, bracket(i+1, j)+1);
    else if ( a[j] == ')' || a[j] == ']')
        answer = min(answer, bracket(i, j-1)+1);

    for(int k = i; k < j; k++ ) {
        answer = min(answer, bracket(i,k) + bracket(k+1,j));
    }
    data[i][j] = answer;
    return answer;
}
```

采用递归记录法

递归记录法的优缺点

还是存在非常严重的重复计算重叠子问题

采用递归记录法

更容易用来给进原始简单递归算法的性能，需要对源程序改动较小

开辟一个data[N][N]的数组，用data[i][j]，记录S_i .....S_j 这个序列需要加入括号的最少个数，这样data[o][N-1]就是我们最后的结果

无法通过递归过程得到除最优解的数值之外的其他信息，例如本题只能得到最少需要加多少个，无法直接得到如何加括号的过程

自底向上法

还是存在非常严重的重复计算重叠子问题

自底向上方法

```
if( ( a[i] == '(' && a[j] == ')' ) ||
    ( a[i] == '[' && a[j] == ']' ) ) {
    answer = min(answer, bracket(i+1, j-1));
else if ( a[i] == '(' || a[i] == '[')
    answer = min(answer, bracket(i+1, j)+1);
else if ( a[j] == ')' || a[j] == ']')
    answer = min(answer, bracket(i, j-1)+1);


for(int k = i; k < j; k++ ) {
    answer = min(answer, bracket(i,k) + bracket(k+1,j));
```

为了计算data[i][j]，需要知道data[i+1][j-1]，data[i+1][j]，data[i][j-1]，d[i][k]，d[k+1][j] 其中i<=k<j；可见令p=j-i，为计算 data[i][j]，需首先计算所有data[m][n] 其中n-m<p

依次计算
data[i][i]（0<=i<N-1）
data[i][i+1]（0<=i<N-2）
.....
data[0][N-1]

1					
0	1				
0	0	1			
0	0	0	1		
0	0	0	0	1	
0	0	0	0	0	1

1					
0	1				
0	0	1			
0	0	0	1		
0	0	0	0	1	
0	0	0	0	0	1

1					
0	1				
0	0	1			
0	0	0	1		
0	0	0	0	1	
0	0	0	0	0	1

1					
0	1				
0	0	1			
0	0	0	1		
0	0	0	0	1	
0	0	0	0	0	1

1					
0	1				
0	0	1			
0	0	0	1		
0	0	0	0	1	
0	0	0	0	0	1

1					
0	1				
0	0	1			
0	0	0	1		
0	0	0	0	1	
0	0	0	0	0	1

```
char a[N];
int data[N][N];
int minadd;
for(int p=1; p<n; p++) { //迭代每一个对角线
    for(int i=1; i<N-p; i++) {//迭代对角线中的每一个元素
        j = i + p;
        data[i][j] = N;

        if( ( a[i] == '(' && a[j] == ')' ) || ( a[i] == '[' && a[j] == ']' ) ) {
            data[i][j] = min(data[i][j], data[i+1, j-1]);
        else if ( a[i] == '(' || a[i] == '[')
            data[i][j] = min(data[i][j], data[i+1, j]+1);
        else if ( a[j] == ')' || a[j] == ']')
            data[i][j] = min(data[i][j], data[i, j-1]+1);

        for(int k = i; k < j; k++ ) {//迭代每一个元素可以从中间分的位置
            data[i][j] = min(data[i][j], data[i,k] + data[k+1,j]);
        } //loop k
    } //loop i
} //loop p

minadd = data[0][N-1]
```

自底向上法

自底向上法

① 如果S是形如(S')或[S']，那么只需将S'划为规则的，S就是规则的
② 如果S是形如(S'，那么只需将S'变成规则的，然后在最后加上一个)
③ S形如[S、S]、S)，和上一种情况类似
④ 我们把序列分成两部分S_i ..... S_k,S_{k+1}.....S_j ，只要将S_i ..... S_k和S_{k+1}.....S_j都变成规则的，那么S就是规则

在计算data[i][j]，如果最小值取自data[i+1][j]则在第j个之后加上和a[i]对应的结束括号，如果最小值取自data[i][j-1]则在第i个之前加上和a[j]对应的开始括号

```
if( ( a[i] == '(' && a[j] == ')' ) ||
    ( a[i] == '[' && a[j] == ']' ) ) {
    data[i][j] = min(data[i][j], data[i+1, j-1]);
else if ( a[i] == '(' || a[i] == '[')
    data[i][j] = min(data[i][j], data[i+1, j]+1);
else if ( a[j] == ')' || a[j] == ']')
    data[i][j] = min(data[i][j], data[i, j-1]+1);

for(int k = i; k < j; k++ ) {
    data[i][j] = min(data[i][j], data[i,k] + data[k+1,j]);
}
```

```
char a[N];
int data[N][N];
int position[N][N];
int minadd;
for(int p=1; p<n; p++) {
    for(int i=1; i<N-p; i++) {
        j = i + p;
        data[i][j] = N;

        if( ( a[i] == '(' && a[j] == ')' ) || ( a[i] == '[' && a[j] == ']' ) ) {
            data[i][j] = min(data[i][j], data[i+1, j-1]);
            position[i][j] = -1;
         //if position[i][j] = -1, we consider the sting from i+1 to j-1
        }else if ( a[i] == '(' || a[i] == '[') {
            data[i][j] = min(data[i][j], data[i+1, j]+1);
            position[i][j] = i-1;
         //if position[i][j] = i-1, add the begin a[j] before i
        }else if ( a[j] == ')' || a[j] == ']')  {
            data[i][j] = min(data[i][j], data[i, j-1]+1);
            position[i][j] = j;
         //if position[i][j] = j, add the end a[i] after j
        }
        for(int k = i; k < j; k++ ) {
            data[i][j] = min(data[i][j], data[i,k] + data[k+1,j]);
            position[i][j] = k;
         //if i < = position[i][j] < j, depart from the position after k
        } //loop k
    } //loop i
} //loop p

minadd = data[0][N-1];
```

自底向上法

还是存在非常严重的重复计算重叠子问题

自底向上方法

依次计算
data[i][i]（0<=i<N-1）
Data[i][i+1]（0<=i<N-2）
.....
Data[0][N-1]

在计算data[i][j]，如果最小值取自data[i+1][j]则在第j个之后加上和a[i]对应的结束括号，如果最小值取自data[i][j-1]则在第i个之前加上和a[j]对应的开始括号

通过整个过程生成的position就可以知道如何加括号

需要对整理过程有更细致的分析，包括哪些是最底层的计算，另外还需要开辟计算位置的数组进行保存

Problem：格式化文本

ACM/ICPC Mid-Central European Regional Contest 1999
http://poj.org/problem?id=1093

问题描述

Writings e-mails is fun, but, unfortunately, they do not look very nice, mainly because not all lines have the same lengths. In this problem, your task is to write an e-mail formatting program which reformats a paragraph of an e-mail (e.g. by inserting spaces) so that, afterwards, all lines have the same length (even the last one of each paragraph).
The easiest way to perform this task would be to insert more spaces between the words in lines which are too short. But this is not the best way. Consider the following example:

```
**************************
This is the example you are
actually considering.
```

$1^2+7^2=50$

```
**************************
This_is_the_example_you__are
actually_______considering.
```

$1+1+1+4+1+4=12$

```
**************************
This__is__the__example___you
are__actually___considering.
```

Of course, this has to be formalized. To do this, we assign a badness to each gap between words. The badness assigned to a gap of n spaces is (n - 1)^2. The goal of the program is to minimize the sum of all badnesses

Problem： 数组分割问题

经典问题