
中国科学技术大学
University of Science and Technology of China
程序设计 II
Programming Design II

动态规划

主讲：吴锋

目录
CONTENTS

动态规划思想
例题1：数字三角形
例题2：最长上升子序列
例题3：最长公共子序列
例题4：Help Jimmy
例题5：最佳加法表达式

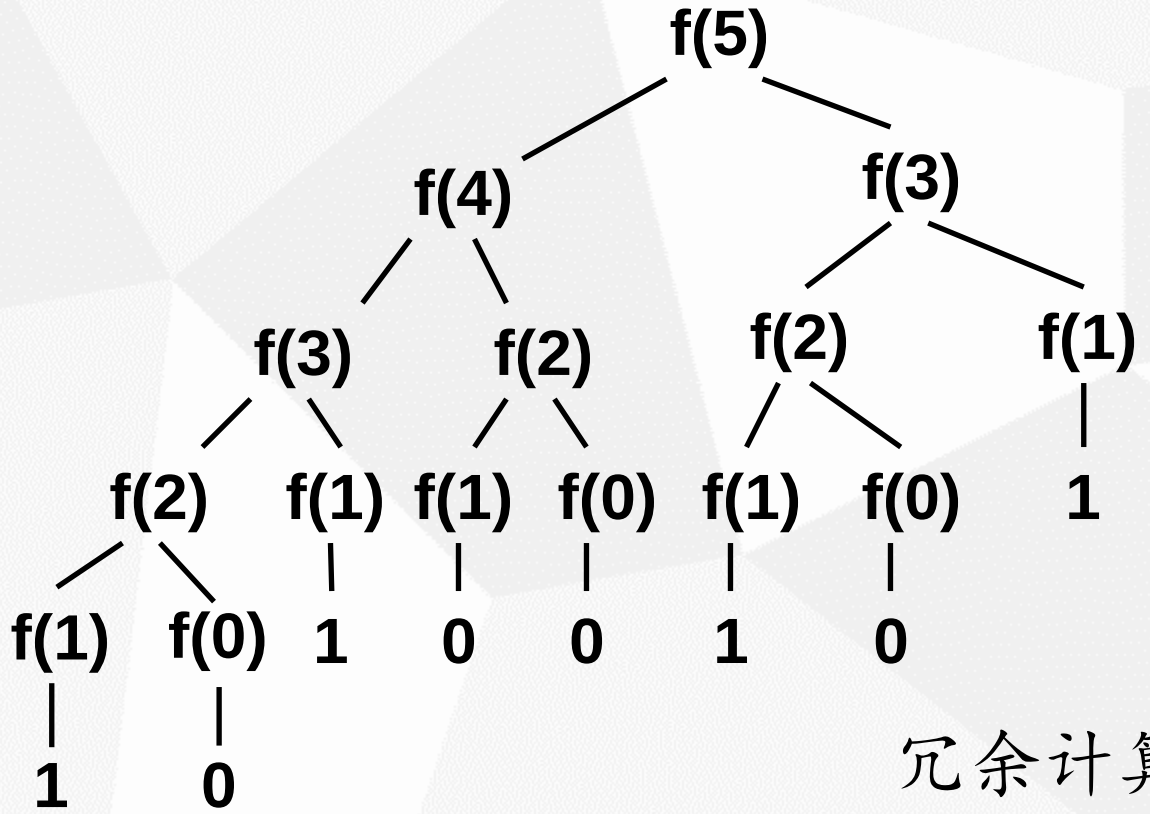
动态规划思想

- 例1：Fibonacci数列
  - 求 Fibonacci数列的第n项

```
int f(int n)
{
        if (n==0 || n==1) return n;
        return f(n-1)+f(n-2);
}
```

动态规划思想

- 树形递归存在冗余计算
  - 计算过程中存在冗余计算，为了除去冗余计算可以从已知条件开始计算，并记录计算过程中的中间结果。



冗余计算

动态规划思想

- 去除冗余：

```
int f[n+1];
f[1] = f[2] = 1;
for (int i=3; i <= n; i++)
      f[i] = f[i-1] + f[i-2];
printf("%d\n", f[n]);
```

- 用空间换时间 → 动态规划

◎ 例题1：数字三角形

- 问题描述
  - 在下面的数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。路径上的每一步都只能往左下或右下走。
  - 只需要求出这个最大和即可，不必给出具体路径。三角形的行数大于1小于等于100，数字为 0 – 99。

```
          7
        3   8
      8   1   0
    2   7   4   4
  4   5   2   6   5
```

◎ 例题1：数字三角形

- 输入格式：

```
//三角形行数。下面是三角形
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

- 要求输出最大和

例题1：数字三角形

- 解题思路：
  - 以 D(r, j) 表示第r行第j个数字，以 MaxSum(r, j) 代表从第r行的第j个数字到底边的各条路径中，数字之和最大的那条路径的数字之和，则本题是要求 MaxSum (0, 0) 。
    - 假设行编号和一行内数字编号都从0开始。
  - 动态规划的思想：从某个 D(r, j) 出发，显然下一步只能走 D(r+1, j) 或 D(r+1, j+1)，所以，对于N行的三角形：

```
if (r == N-1)
    MaxSum(r, j) = D(N-1, j)
else
    MaxSum(r, j) = Max( MaxSum(r+1, j), MaxSum(r+1, j+1) ) + D(r, j)
```

例题1：数字三角形

- 数字三角形的递归程序：

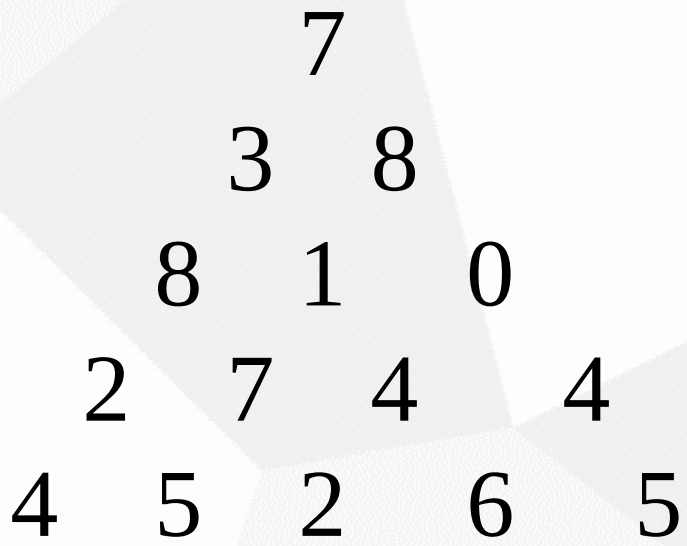
```
#include <stdio.h>
#define MAX 101
int triangle[MAX][MAX];
int n;
int longestPath(int i, int j);
int main() {
  scanf("%d", &n);
  for (int i = 0; i < n; i++)
     for (int j = 0; j <= i; j++)
       scanf("%d", &triangle[i][j]);
  printf("%d\n", longestPath(0, 0));
  return 0;
}
```

```
int longestPath(int i, int j) {
  if (i == n)
     return 0;
  int x = longestPath(i + 1, j);
  int y = longestPath(i + 1, j + 1);
  if (x < y)
     x = y;
  return x + triangle[i][j];
}
```

- 程序运行超时！

◎ 例题1：数字三角形

- 超时原因：重复计算
  - 如果采用递规的方法，深度遍历每条路径，存在大量重复计算。则时间复杂度为 2^n, 对于 n = 100，肯定超时。



◎ 例题1：数字三角形

- 改进思想：
  - 从下往上计算，对于每一点，只需要保留从下面来的路径中和最大的路径的和即可。
  - 因为在它上面的点只关心到达它的最大路径和，不关心它从那条路经上来的。

◎ 例题1：数字三角形

- 解法1：
  - 如果每算出一个 MaxSum(r, j) 就保存起来，则可以用 $O(n^2)$ 时间完成计算。因为三角形的数字总数是 n(n+1)/2 。
  - 此时需要的存储空间是：

```
// 用于存储三角形中的数字
int D[100][100];
// 用于存储每个MaxSum(r, j)
int aMaxSum[100][100];
```

例题1：数字三角形

```
#include <stdio.h>
#define MAX_NUM 100
int D[MAX_NUM + 10][MAX_NUM + 10];  int N;
int aMaxSum[MAX_NUM + 10][MAX_NUM + 10];
int main() {
     scanf("%d", &N);
     for (int i = 1; i <= N; i++)
          for (int j = 1; j <= i; j++)
               scanf("%d", &D[i][j]);
     for (int j = 1; j <= N; j++)
          aMaxSum[N][j] = D[N][j];
     for (int i = N ; i > 1; i--)
          for (int j = 1; j < i; j++) {
            if (aMaxSum[i][j] > aMaxSum[i][j+1])
              aMaxSum[i-1][j] = aMaxSum[i][j] + D[i-1][j];
            else
              aMaxSum[i-1][j] = aMaxSum[i][j+1] + D[i-1][j];
          }
     printf("%d", aMaxSum[1][1]);
     return 0;
}
```

例题1：数字三角形

- 解法2:
  - 没必要用二维 Sum 数组存储每一个 MaxSum(r, j), 只要从底层一行行向上递推，那么只要一维数组 Sum[100] 即可,即只要存储一行的 MaxSum 值就可以。
  - 比解法一改进之处在于：节省空间，时间复杂度不变。

例题1：数字三角形

```
#include <stdio.h>
#define MAX_NUM 100
int D[MAX_NUM + 10][MAX_NUM + 10]; int N;
int aMaxSum[MAX_NUM + 10];
int main() {
     scanf("%d", &N);
     for (int i = 1; i <= N; i++)
          for (int j = 1; j <= i; j++)
               scanf("%d", &D[i][j]);
     for (int j = 1; j <= N; j++)
          aMaxSum[j] = D[N][j];
     for (int i = N ; i > 1; i--)
          for (int j = 1; j < i; j++) {
            if (aMaxSum[j] > aMaxSum[j+1])
              aMaxSum[j] = aMaxSum[j] + D[i-1][j];
            else
              aMaxSum[j] = aMaxSum[j+1] + D[i-1][j];
          }
     printf("%d", aMaxSum[1]);
     return 0;
}
```

递归到动态规化的转化

- 递归到动态规化的转化的一般方法：
  - 原来递归函数有几个参数，就定义一个几维的数组，用于存储中间结果。
  - 数组的下标是递归函数参数的取值范围，数组元素的值是递归函数的返回值。
  - 这样就可以从边界开始，逐步填充数组，相当于计算递归函数值的逆过程。

◎ 例题2：最长上升子序列

- 问题描述（P198）
  - 一个数的序列 b_i，当 $b_1 < b_2 < \ldots < b_S$ 的时候，称这个序列是上升的。
  - 对于给定的一个序列（a_1，a_2，...，a_N），可以得到一些上升的子序列（a_{i1}，a_{i2}，...，a_{iK}），这里$1 \leqslant i_1 < i_2 < \ldots < i_K \leqslant N$。
  - 比如，对于序列（1，7，3，5，9，4，8），有它的一些上升子序列，如（1，7），（3，4，8）等等。这些子序列中最长的长度是4，比如子序列（1，3，5，8）。
  - 对于给定的序列，求出最长上升子序列的长度。

◎ 例题2：最长上升子序列

- 输入数据
  - 输入的第一行是序列的长度 N（1 ⩽ N ⩽ 1000）。第二行给出序列中的 N 个整数，这些整数的取值范围都在 0 到 10000。
- 输出要求
  - 最长上升子序列的长度。
- 输入样例

```
7
1 7 3 5 9 4 8
```

- 输出样例

```
4
```

例题2：最长上升子序列

- 解题思路

1. 找子问题

  - 经过分析，发现 “求以 a_k（k=1，2，3 … N）为终点的最长上升子序列的长度”是个好的子问题 —— 这里把一个上升子序列中最右边的那个数，称为该子序列的“终点”。
  - 虽然这个子问题和原问题形式上并不完全一样，但是只要这 N 个子问题都解决了，那么这 N 个子问题的解中，最大的那个就是整个问题的解。

2. 确定状态：

  - 上所述的子问题只和一个变量相关，就是数字的位置。因此序列中数的位置 k 就是“状态”，而状态 k 对应的“值”，就是以 a_k 做为“终点”的最长上升子序列的长度。这个问题的状态一共有N个。

例题2：最长上升子序列

- 解题思路

3. 找出状态转移方程：

- 状态定义出来后，转移方程就不难想了。假定MaxLen（k）表示以a_k做为“终点”的最长上升子序列的长度，那么：

  MaxLen（1）= 1

  MaxLen（k）= Max { MaxLen（i）: 1< i < k 且 $a_i < a_k$ 且 k≠1 } + 1

- 这个状态转移方程的意思就是，MaxLen(k) 的值，就是在 a_k 左边，“终点”数值小于 a_k ，且长度最大的那个上升子序列的长度再加1。因为 a_k 左边任何“终点”小于 a_k 的子序列，加上 a_k后就能形成一个更长的上升子序列。

- 实际实现的时候，可以不必编写递归函数，因为从 MaxLen(1)就能推算出 MaxLen(2)，有了 MaxLen(1) 和 MaxLen(2) 就能推算出 MaxLen(3) ……

例题2：最长上升子序列

```
#include <stdio.h>
#define MAX_N 1000
int b[MAX_N + 10], aMaxLen[MAX_N + 10];
int main() {
    int N;
    scanf("%d", &N);
    for (int i = 1; i <= N; i++) scanf("%d", &b[i]);
    aMaxLen[1] = 1;
    for (int i = 2; i <= N; i++) {
        //每次求以第i个数为终点的最长上升子序列的长度
        int nTmp = 0;  //记录满足条件的，第i个数左边的上升子序列的最大长度
        for (int j = 1; j < i; j++) //察看以第j个数为终点的最长上升子序列
            if (b[i] > b[j] && nTmp < aMaxLen[j]) nTmp = aMaxLen[j];
        aMaxLen[i] = nTmp + 1;
    }
    int nMax = -1;
    for (int i = 1; i <= N; i++)
        if (nMax < aMaxLen[i]) nMax = aMaxLen[i];
    printf("%d\n", nMax);
    return 0;
}
```

例题3：最长公共子序列

- 问题描述（P203）
  - 给出两个字符串，求出这样的一个最长的公共子序列的长度：子序列中的每个字符都能在两个原串中找到，而且每个字符的先后顺序和原串中的先后顺序一致。
- 样例输入

```
abcfbc abfcab
programming    contest
abcd    mnp
```

- 样例输出

```
4
2
0
```

例题3：最长公共子序列

- 解题思路
  - 输入两个子串 s_1，s_2；设 MaxLen(i, j) 表示：s_1的左边i个字符形成的子串，与 s_2 左边的 j 个字符形成的子串的最长公共子序列的长度

    MaxLen(i, j) 就是本题的“状态”
  - 假定 len1 = strlen(s_1)，len2 = strlen(s_2)，那么题目就是要求：

    MaxLen(len1, len2)

例题3：最长公共子序列

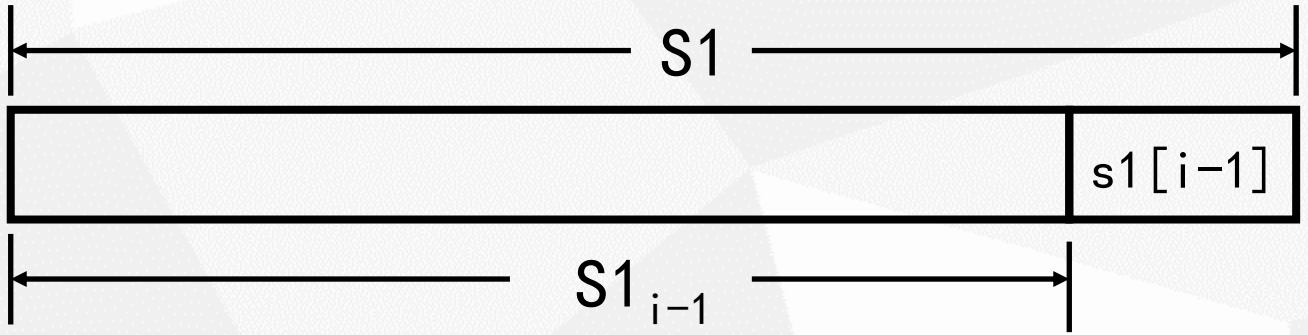
- 解题思路
  - 显然：MaxLen(n, 0) = 0 （ n = 0 … len1）

    MaxLen(0, n) = 0 （ n = 0 … len2）
  - 递推公式：

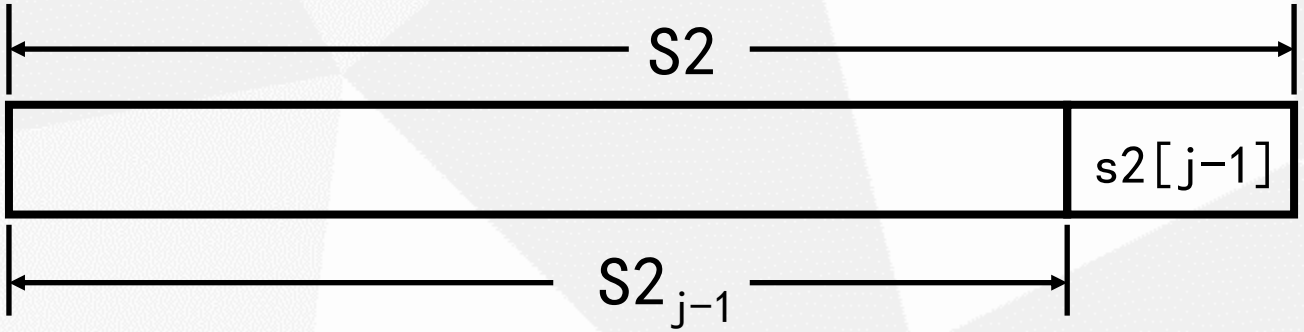
```
if (s1[i-1] == s2[j-1])
    MaxLen(i, j) = MaxLen(i-1, j-1) + 1;
else
      MaxLen(i, j) = Max( MaxLen(i, j-1), MaxLen(i-1, j) );
```

例题3：最长公共子序列

S1长度为 i



S2长度为 j



MaxLen(S1, S2) 不会比 MaxLen(S1, $S2_{j-1}$) 和

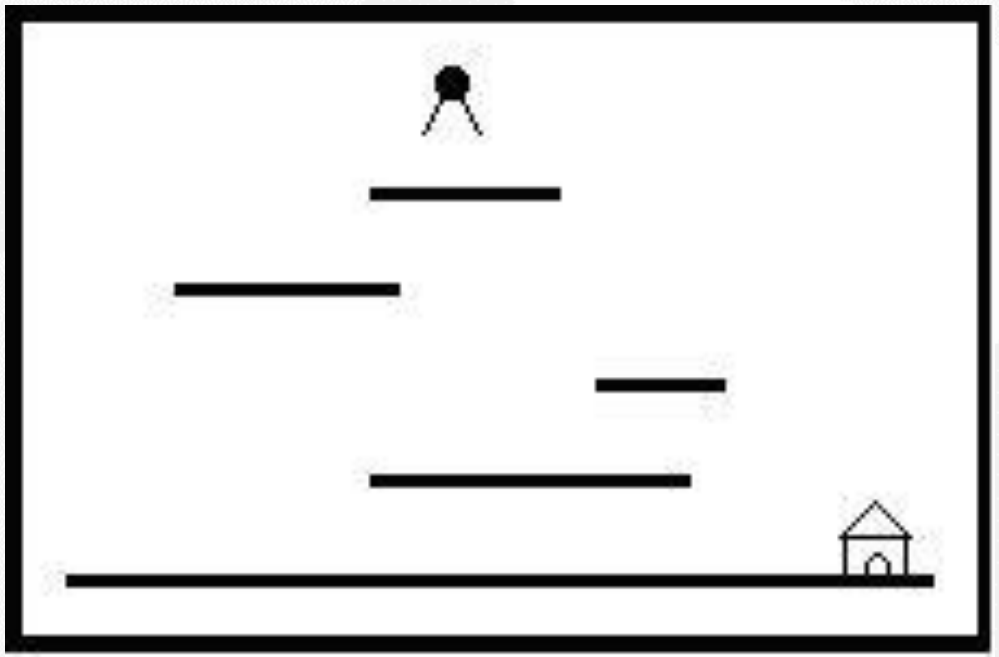
MaxLen($S1_{i-1}$, S2) 都大，也不会比两者中任何一个小

例题3：最长公共子序列

```
#include <stdio.h>
#include <string.h>
char sz1[1000], sz2[1000];
int anMaxLen[1000][1000];
int main() {
     while (scanf("%s%s", &sz1, &sz2)) {
          int nLength1 = strlen(sz1), nLength2 = strlen(sz2), nTmp;
          for (int i = 0; i <= nLength1; i++) anMaxLen[i][0] = 0;
          for (int j = 0; j <= nLength2; j++) anMaxLen[0][j] = 0;
          for (int i = 1; i <= nLength1; i++)
               for (int j = 1; j <= nLength2; j++)
                    if (sz1[i-1] == sz2[j-1]) anMaxLen[i][j] = anMaxLen[i-1][j-1] + 1;
                    else {
                         int nLen1 = anMaxLen[i][j-1], nLen2 = anMaxLen[i-1][j];
                         if (nLen1 > nLen2) anMaxLen[i][j] = nLen1;
                         else anMaxLen[i][j] = nLen2;
                    }
          printf("%d\n", anMaxLen[nLength1][nLength2]);
     }
     return 0;
}
```

◎ 例题4：Help Jimmy

- 问题描述（P199）
  - 在下图所示的场景上完成的游戏：



◎ 例题4：Help Jimmy

- 问题描述
  - 场景中包括多个长度和高度各不相同的平台。地面是最低的平台，高度为零，长度无限。
  - Jimmy 老鼠在时刻0从高于所有平台的某处开始下落，它的下落速度始终为1米/秒。
  - 当 Jimmy 落到某个平台上时，游戏者选择让它向左还是向右跑，它跑动的速度也是1米/秒。
  - 当 Jimmy 跑到平台的边缘时，开始继续下落。Jimmy每次下落的高度不能超过 MAX 米，不然就会摔死，游戏也会结束。
  - 设计一个程序，计算 Jimmy 到地面时可能的最早时间。

◎ 例题4：Help Jimmy

- 输入数据
  - 第一行是测试数据的组数t（0≤t≤20）。每组测试数据的第一行是四个整数 N，X，Y，MAX，用空格分隔。N是平台的数目（不包括地面），X 和 Y 是 Jimmy 开始下落的位置的横竖坐标，MAX 是一次下落的最大高度。
  - 接下来的 N 行每行描述一个平台，包括三个整数，X1[i]，X2[i] 和 H[i]。H[i] 表示平台的高度，X1[i] 和 X2[i] 表示平台左右端点的横坐标。1≤ N ≤ 1000，-20000 ≤ X，X1[i]，X2[i] ≤ 20000，0 < H[i] < Y ≤ 20000（i = 1..N）。所有坐标的单位都是米。
  - Jimmy的大小和平台的厚度均忽略不计。如果Jimmy恰好落在某个平台的边缘，被视为落在平台上。所有的平台均不重叠或相连。测试数据保Jimmy一定能安全到达地面。

◎ 例题4：Help Jimmy

- 输出要求
  - 对输入的每组测试数据，输出一个整数，Jimmy到地面时可能的最早时间。
- 输入样例

```
1
3 8 17 20
0 10 8
0 10 13
4 14 3
```

- 输出样例

```
23
```

◎ 例题4：Help Jimmy

- 解题思路
  - Jimmy 跳到一块板上后，可以有两种选择，向左走，或向右走。走到左端和走到右端所需的时间，是很容易算的。
  - 如果我们能知道，以左端为起点到达地面的最短时间，和以右端为起点到达地面的最短时间，那么向左走还是向右走，就很容选择了。
  - 因此，整个问题就被分解成两个子问题，即 Jimmy 所在位置下方第一块板左端为起点到地面的最短时间，和右端为起点到地面的最短时间。
  - 这两个子问题和原问题在形式上是完全一致的。

◎ 例题4：Help Jimmy

- 解题思路
  - 将板子从上到下从1开始进行无重复的编号（越高的板子编号越小，高度相同的几块板子，哪块编号在前无所谓），那么，和上面两个子问题相关的变量就只有板子的编号。
  - 不妨认为 Jimmy 开始的位置是一个编号为0，长度为0的板子，假设 LeftMinTime(k) 表示从 k 号板子左端到地面的最短时间，RightMinTime(k) 表示从 k 号板子右端到地面的最短时间，那么，求板子 k 左端点到地面的最短时间的方法如下：

◎ 例题4：Help Jimmy

- 解题思路

出口条件

递推公式

```
if ( 板子 k 左端正下方没有别的板子 ) {
    if ( 板子k的高度 h(k) 大于Max )
        LeftMinTime(k) = ∞;
    else
        LeftMinTime(k) = h(k);
}
else if ( 板子 k 左端正下方的板子编号是 m )
    LeftMinTime(k) = h(k) - h(m) +
        Min( LeftMinTime(m) + Lx(k) - Lx(m),
                RightMinTime(m) + Rx(m) - Lx(k) );
}
```

◎ 例题4：Help Jimmy

- 解题思路
  - 上面，h(i) 就代表i号板子的高度，Lx(i) 就代表i号板子左端点的横坐标，Rx(i) 就代表 i 号板子右端点的横坐标。那么 h(k) – h(m) 当然就是从 k 号板子跳到m号板子所需要的时间，Lx(k) – Lx(m) 就是从 m 号板子的落脚点走到 m 号板子左端点的时间，Rx(m) – Lx(k) 就是从 m 号板子的落脚点走到右端点所需的时间。
  - 求 RightMinTime(k) 的过程类似。
  - 不妨认为 Jimmy 开始的位置是一个编号为0，长度为0的板子，那么整个问题就是要求 LeftMinTime(0)。
  - 输入数据中，板子并没有按高度排序，所以程序中一定要首先将板子排序。

例题5：最佳加法表达式

- 问题描述
  - 有一个由 1..9 组成的数字串。问如果将 m 个加号插入到这个数字串中，在各种可能形成的表达式中，值最小的那个表达式的值是多少。
- 解题思路
  - 假定数字串长度是n，添完加号后，表达式的最后一个加号添加在第 i 个数字后面。
  - 那么整个表达式的最小值，就等于在前 i 个数字中插入 m - 1个加号所能形成的最小值，加上第 i + 1到第 n 个数字所组成的数的值。
  - 子问题（前 i 个数字的最小值）与原问题（整个表达式的最小值）在形式上是完全一致的。

例题5：最佳加法表达式

- 解题思路
  - 设 V(m, n) 表示在 n 个数字中插入 m 个加号所能形成的表达式最小值，那么：

```
if (m = 0)
    V(m,n) = n 个数字构成的整数
else if (n < m + 1)
   V(m,n) = ∞
else
    V(m,n) = Min{ V(m-1,i) + Num(i+1,n) } ( i=m…n-1 )
```

  - Num(k, j) 表示从第 k 个字符到第 j 个字符所组成的数
  - 数字编号从1开始算

作业

- 1.矩形覆盖（P208）
  - 在平面上给出了n 个点，现在需要用一些平行于坐标轴的矩形把这些点覆盖住。每个点都需要被覆盖，而且可以被覆盖多次。每个矩形都至少要覆盖两个点，而且处于矩形边界上的点也算作被矩形覆盖。注意：矩形的长宽都必须是正整数，也就是说矩形不能退化为线段或者点。 问：怎样选择矩形，才能够使矩形的总面积最小。
- 6. 集合加法（P210）
  - 给出2 个正整数集合A = {pi | 1 <= i <= a}，B = {qj | 1 <= j <= b}和一个正整数s (1 <= s <= 10000)。问题是：使得pi + qj = s 的不同的(i, j)对有多少个。

作业

- 7. 木材加工（P210）
  - 木材厂有N 根原木，现在想把这些木头切割成一些长度相同的小段木头，需要得到的小段的数目K 是给定了。我们希望得到的小段越长越好，你的任务是计算能够得到的小段木头的最大长度。木头长度的单位是cm。原木的长度都是正整数，我们要求切割得到的小段木头的长度也要求是正整数 (1 <= N <= 10000，1 <= K <= 10000) 。

作业

- 8. 最大子矩阵（P210）
  - 将“矩阵的大小”定义为矩阵中所有元素的和。给定一个N * N的矩阵(0 < N <= 100)，你的任务是找到最大的非空(大小至少是1 * 1)子矩阵。比如，如下4 * 4 的矩阵

```
0 -2 -7 0
9 2 -6 2
-4 1 -4 1
-1 8 0 -2
```

它的最大子矩阵是：

```
9 2
-4 1
-1 8
```

这个子矩阵的大小是15。