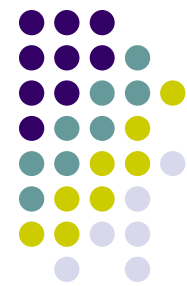


算法设计与分析（内容40+8学时）



- 一、动态规划（递推、分治、倍增等重要算法思想）
- 二、二叉堆、并查集和树状数组
- 三、线段树、**splay**及其他动态树
- 四、最小生成树、最短路径
- 五、拓扑排序
- 六、**KMP&Trie&AC**自动机
- 七、网络流
- 八、线性规划（*）
- 九、算法选讲

算法设计与分析（评分标准）



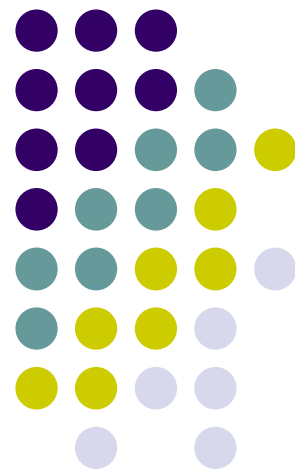
总共**10**周，每两周一次模拟考试共**5**次，课堂练习为在线作业。

考查成绩组成：

- （1）两次模拟考试(要求到教室，**10**分) + 另外**3**次模拟考试至少参加一次（**5**分）
- （2）完成课后练习的**50%**（**5**分）
- （3）**12**月份**CSP**考试折合分数

算法设计与分析

动态规划1



- 斐波纳契数列 $F(n)$



$$F(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

n	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	1	1	2	3	5	8	13	21	34	55	89

递归 vs 动态规划

递归版本:

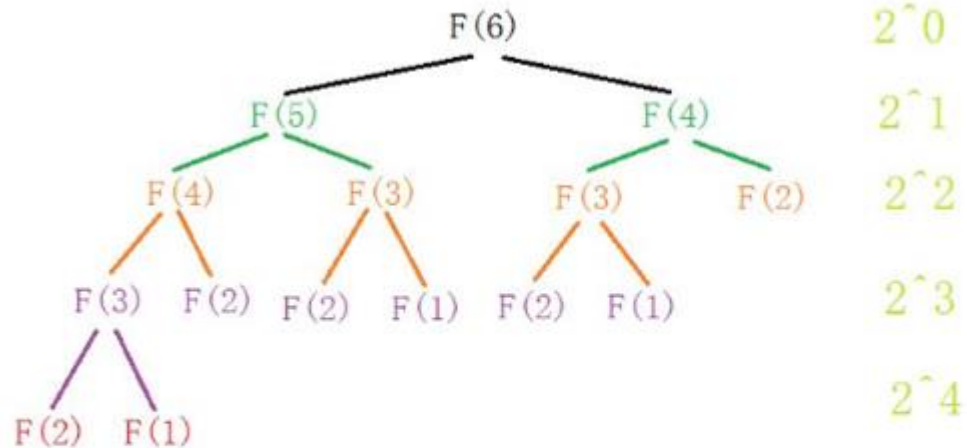
```
F(n)
1  if n==0 or n==1 then
2      return 1
3  else
4      return F(n-1) + F(n-2)
```

太慢!

初略估计:

$$T(n) = T(n-1) + T(n-2)$$

$$O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$$



时间复杂度: $(2^0 + 2^1 + 2^2 + \dots + 2^{(n-2)}) * 2$
空间复杂度: 树约有 $n-1$ 层, 所以空间复杂度约为 $O(n)$

递归 vs 动态规划



动态规划递推版本:

$F(n)$

```
1  A[0] = A[1] = 1
2  for i = 2 to n do
3      A[i] = A[i-1] + A[i-2]
4  return A[n]
```

动态规划递推强迫症版本:

$F(n)$

```
1  A=A0 = A1 = 1
2  for i = 2 to n do
3      A = A0 + A1
4      A0=A1
5      A1=A
6  return A
```

有效率!
算法复杂度是 $O(n)$
各自优缺点?

递归 vs 动态规划



动态规划递归之记忆化搜索版本:

$A[i]=0$

$F(n)$

1 $A[0] = A[1] = 1$

2 if $A[n]>0$ then

3 return $A[n]$

4 else

5 $A[n]=F(n-1)+F(n-2)$

6 return $A[n]$

有效率?
算法复杂度是 $O(?)$

方法概要



- 构造一个公式，它表示一个问题的解是与它的子问题的解相关的公式.

E.g. $F(n) = F(n - 1) + F(n - 2)$

- 为这些子问题做索引，以便它们能够在表中更好的存储与检索 (i.e., 数组、向量)
- 以自底向上的方法来填写这表格; 首先填写最小子问题的解.
 - 这就保证了当我们解决一个特殊的子问题时, 可以利用比它更小的所有可利用的 子问题的解.



- 将待求解的问题分解成若干个子问题，并存储子问题的解而避免计算重复的子问题，并由子问题的解得到原问题的解。
- 动态规划算法通常用于求解具有某种最优性质的问题。
- 动态规划算法的基本要素：
最优子结构性性质和重叠子问题。



- **最优子结构性质：** 问题的最优解包含着它的子问题的最优解。即不管前面的策略如何，此后的决策必须是基于当前状态（由上一次决策产生）的最优决策。
- **重叠子问题：** 在用递归算法自顶向下解问题时，每次产生的子问题并不总是新问题，有些问题被反复计算多次。对每个子问题只解一次，然后将其解保存起来，以后再遇到同样的问题时就可以直接引用，不必重新求解。

解决问题的基本特征

原理



1. 动态规划一般解决最值（最优，最大，最小，最长……）问题；
2. 动态规划解决的问题一般是离散的，可以分解（划分阶段）的；
3. 动态规划解决的问题必须包含最优子结构，即可以由（ $n-1$ ）的最优推导出 n 的最优



解决问题的基本步骤

- 动态规划算法的3个步骤：
 1. 刻画最优解的结构特性。（一维，二维，三维数组）
 2. 递归的定义最优解。（状态转移方程）
 3. 以自底向上的方法来计算最优解。



实例

例题一. 斐波纳契数列 $F(n)$

步骤1: 用 $F(n)$ 表示在斐波纳契数列中第 n 个数的值;

步骤2: 状态转移方程:

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ F(n-1) + F(n-2) & \text{if } n > 1 \end{cases}$$

步骤3: 以自底向上的方法来计算最优解

n	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	1	1	2	3	5	8	13	21	34	55	89

例题二. 输入 n , 求出 $n!$

实例



步骤1: 用 $F(n)$ 表示 $n!$ 的值;

步骤2: 状态转移方程:

$$F(n) = \begin{cases} 1 & \text{if } n = 0 \text{ or } 1 \\ F(n-1) * n & \text{if } n > 1 \end{cases}$$

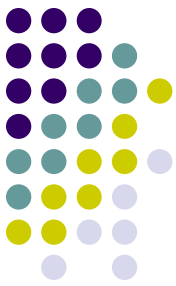
步骤3: 以自底向上的方法来计算最优解

n	0	1	2	3	4	5	6	7	8	9	10
$F(n)$	1	1	2	6	24	120	720				



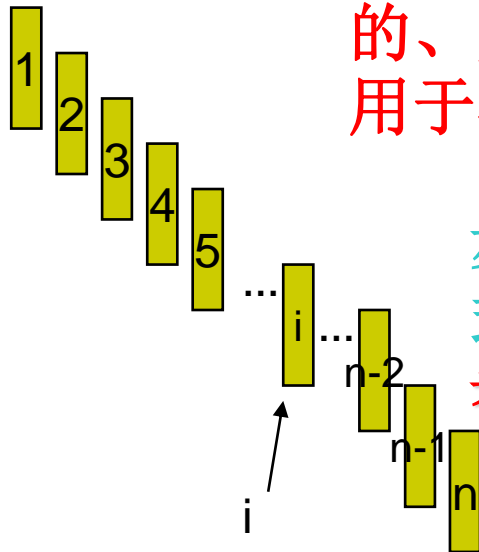
例题三：排队买票问题

- 一场演唱会即将举行。现有 n 个歌迷排队买票，一个人需一张，而售票处规定，一个人每次最多只能买两张票。假设第 i 位歌迷买一张票需要时间 T_i ($1 \leq i \leq n$)，队伍中相邻的两位歌迷（第 j 个人和第 $j+1$ 个人）也可以由其中一个人买两张票，而另一位就可以不用排队了，则这两位歌迷买两张票的时间变为 R_j ，假如 $R_j < T_j + T_{j+1}$ ，这样做就可以缩短后面歌迷等待的时间，加快整个售票的进程。现给出 n, T_j 和 R_j ，求使每个人都买到票的最短时间和方法。



分析:

问题的最优解是根据子问题的最优解计算出来的。每个子问题的最优解：**唯一的、无后效性的、能表示当前状态且可用于状态转换的。**



步骤1: 用 $F(i)$ 表示前 i 个人买票的最优方式, 即所需最短时间; **现在要决定 $F(i)$ 需要考虑两种情况:**

$$\min \begin{cases} (1) \text{ 第 } i \text{ 个人的票自己买} \\ (2) \text{ 第 } i \text{ 个人的票由第 } i-1 \text{ 个人买} \end{cases}$$

步骤2: 状态转移方程:

0	T_1						
---	-------	--	--	--	--	--	--

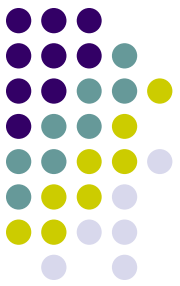
步骤3: 以自底向上的方法来计算最优解

程序的实现



BuyTicks(T, R)

```
1   $n = \text{length}[T]$ 
2   $f[0] = 0$ 
3   $f[1] = T[1]$ 
4  for  $i = 2$  to  $n$  do
5       $f[i] = \min\{f[i - 2] + R[i - 1],$ 
6           $f[i - 1] + T[i]\}$ 
7  return  $f[n]$ 
```



例题四：硬币问题

- 如果我们有面值为1元、3元和5元的硬币足够多，如何用最少的硬币凑够104元？(22枚)
- 如果我们有面值为1元、5元和11元的硬币足够多，如何用最少的硬币凑够103元？(11枚)

$$f(i) = \begin{cases} 1, i = 1, 5, 11 \\ i, i = 2, 3, 4 \\ i - 4, i = 6, 7, 8, 9 \\ 2, i = 10 \\ \min\{f(i - 1), f(i - 5), f(i - 11)\} + 1, i > 11 \end{cases}$$

- 如果我们有面值为 $a_1 < a_2 < \dots < a_n$ 元的硬币足够多，如何用最少的硬币凑够 M 元？(如何初始化？)



例题四：硬币问题

$$f(i) = \begin{cases} 0, i = 0 \\ \min\{f(i - a_j) | a_j \leq i\} + 1, i > 0 \end{cases}$$

例题五：求最长不降子序列 LIS

实例



1. 问题描述

设有一个正整数的序列： b_1, b_2, \dots, b_n ，对于下标 $i_1 < i_2 < \dots < i_m$ ，若有 $b_{i_1} \leq b_{i_2} \leq \dots \leq b_{i_m}$ ，则称存在一个长度为 m 的不下降序列。

例如，下列数列

13 7 9 16 38 24 37 18 44 19 21 22 63 15

对于下标 $i_1 = 1, i_2 = 4, i_3 = 5, i_4 = 9, i_5 = 13$ ，满足 $13 < 16 < 38 < 44 < 63$ ，则存在长度为5的不下降序列。

但是，我们看到还存在其他的不下降序列： $i_1 = 2, i_2 = 3, i_3 = 4, i_4 = 8, i_5 = 10, i_6 = 11, i_7 = 12, i_8 = 13$ ，满足： **$7 < 9 < 16 < 18 < 19 < 21 < 22 < 63$** ，则存在长度为8的不下降序列。

问题为：当 b_1, b_2, \dots, b_n 给出之后，求出最长的不下降序列。

$O(n^2)$

步骤1：用 $F(i)$ 表示经过第 i 项，且到 i 为止最长不下降序列的长度值；

步骤2：状态转移方程；

$d[i]$ 表示数列中第 i 项的值；

$$F(i) = \begin{cases} 1 & i = 1 \\ \max\{F(j) | d[j] \leq d[i], j < i\} + 1 & i > 1 \end{cases}$$

步骤3：以自底向上的方法来计算最优解。最终结果： $\max\{F(i)\}$

例题五：求最长不降子序列 LIS

实例



13 7 9 16 38 24 37 18 44 19 21 22 63 15 $d[i]$

构造单调不减数组 A ， $A[i]$ 表示长度为 i 的不降子序列的最小尾元素的值， $Alen$ 表示数组 A 中存储的元素个数，等于当前的最长子序列长度，初始 $Alen=0$ 过程如下：

13	$Alen++$,
7	
7 9	$Alen++$
7 9 16	$Alen++$
7 9 16 38	$Alen++$
7 9 16 24	
7 9 16 24 37	$Alen++$

从数组 $d[i]$ 的第1项开始依次处理（保证 $A[i]$ 表示长度为 i 的不降子序列的最小尾元素的值）：

比如，处理到24了，在最长不降序列中，24肯定不能接在38后面，所以前面算法中和38比较就是浪费时间，只需和不大于24的数据进行比较($\log n$ 复杂度)。

现在发现 $16 < 24$ ，到16的最长不降子序列长度为3，那么可以把24接在16的后面，形成长度为4的子序列，于是用24替换掉38（24后面的数据，如果要接在长度为4的序列后面，24比38更优质）。



拓展1：拦截导弹

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度（雷达给出的高度数据是不大于30000的正整数），计算这套系统最多能拦截多少导弹，如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

样例：

INPUT

389 207 155 300 299 170 158 65

OUTPUT

6（最多能拦截的导弹数）

2（要拦截所有导弹最少要配备的系统数）

最长不升

拓展2：低价购买



“低价购买”这条建议是在奶牛股票市场取得成功的一半规则。要想被认为是伟大的投资者，你必须遵循以下的问题建议：“低价购买；再低价购买”。每次你购买一支股票，你必须用低于你上次购买它的价格购买它。买的次数越多越好！你的目标是在遵循以上建议的前提下，求你最多能购买股票的次数。你将被给出一段时间内一支股票每天的出售价(2^{16} 范围内的正整数)，你可以选择在哪些天购买这支股票。每次购买都必须遵循“低价购买；再低价购买”的原则。写一个程序计算最大购买次数。

这里是某支股票的价格清单：

日期 1 2 3 4 5 6 7 8 9 10 11 12

价格 68 69 54 64 68 64 70 67 78 62 98 87

最优秀的投资者可以购买最多4次股票，可行方案中的一种是：

日期 2 5 6 10

价格 69 68 64 62


输入

第1行: N ($1 \leq N \leq 5000$), 股票发行天数

第2行: N 个数，是每天的股票价格。

输出

输出文件仅一行包含两个数:最大购买次数和拥有最大购买次数的方案数($\leq 2^{31}$)
当二种方案“看起来一样”时（就是说它们构成的价格队列一样的时候），这2种方案被认为是相同的。



同时记录两个状态
最长子序列、方案数

拓展3：合唱队形



N 位同学站成一排，音乐老师要请其中的 $(N - K)$ 位同学出列，使得剩下的 K 位同学排成合唱队形。

合唱队形是指这样的一种队形：设 K 位同学从左到右依次编号为 $1, 2, \dots, K$ ，他们的身高分别为 T_1, T_2, \dots, T_K ，则他们的身高满足 $T_1 < \dots < T_i > T_{i+1} > \dots > T_K (1 \leq i \leq K)$ 。

你的任务是，已知所有 N 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。

输入：第一行是一个整数 $N (2 \leq N \leq 100)$ ，表示同学的总数。第二行有 n 个整数，用空格分隔，第 i 个整数 $T_i (130 \leq T_i \leq 230)$ 是第 i 位同学的身高(厘米)。

输出：包括一行，这一行只包含一个整数，就是最少需要几位同学出列。

样例输入

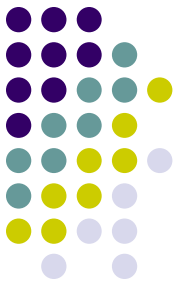
8

186 186 150 200 160 130 197 220

样例输出：

4

两个方向分别做



列表总结

题目	关键点	类型
最长不降子序列 LIS	$max, O(n^2), O(n \log n)$	一维 DP
导弹拦截	最长不升	
斐波纳契数列	$O(n)$, 简单递推, ...	一维 DP
上楼梯	$O(n)$, 简单递推, ...	