ACM程序设计培训

计算机与信息学院 刘必雄

常用算法专题

内 容 提 要

COMMON ALGORITHM TOPIC

1、算法分析技术

2、枚举算法

3、递推算法

4、分治算法

5、贪心算法

COMMON ALGORITHM TOPIC

1、算法分析技术

算法分析

评价算法通常是以算法执行所需的机器时间和所占用的存储空间来判断一个算法的优劣 。因此，算法分析主要是算法**时间复杂度**和**空间复杂度**的分析。

- **算法的时间复杂度**：执行耗费CPU的时间来衡量。
- **算法的空间复杂度**：执行耗费内存空间的大小来衡量。

1、算法分析技术

程序运行时间

- 问题规模和输入数据的分布
- 利用编译程序生成的目标代码的质量
- 计算机系统的性能

算法的优劣

时间复杂度概念

② 和③表示算法的时间依赖于软件和硬件的环境。

算法的时间复杂度：指算法所需的运算量与问题规模之间的关系。

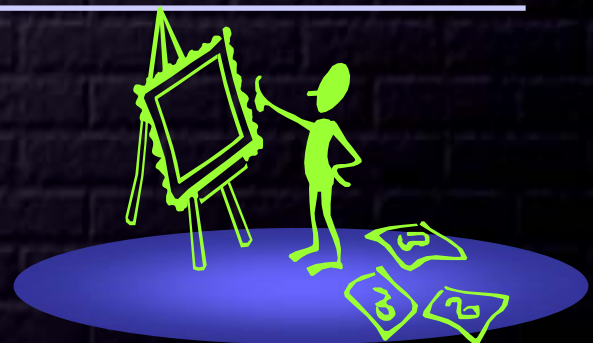
算法的时间复杂度是一种抽象的度量，它与所运行的计算机软件与硬件无关。主要体现在循环和递归上。

1、算法分析技术

循环

循环语句的时间执代价

- 对于一个循环，循环次数**乘以**每次执行简单语句的数目即为其时间代价。
- 对于多个并列循环，可先计算每个循环的时间代价，然后按**加法**规则计算总代价。
- 对于多层嵌套循环，可按**乘法**规则计算时间复杂度。



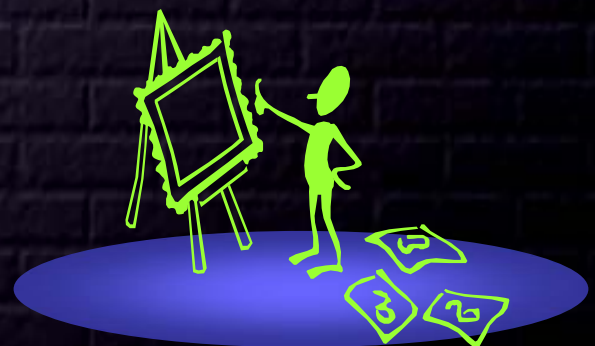
1、算法分析技术

循环

【例1】 对一个计算斐波那契（Fibonacci）数列的算法进行算法分析。

```
int Fib(int n)
{
  int a=0,b=1,c=0;
  int i;
  for (i=0;i<n;i++){
    a=b;
    b=c;
    c=a+b;
  }
  return c;
}
```

算法的时间复杂度为O(n)



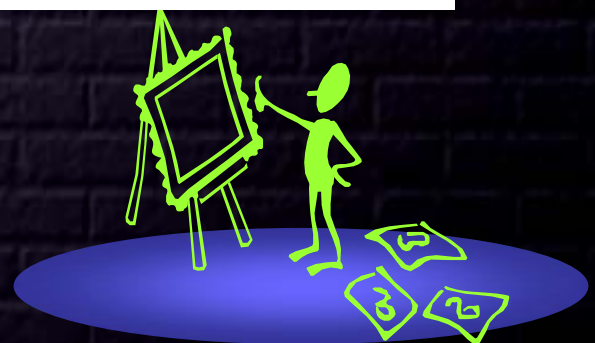
1、算法分析技术

循环

【例2】 对于两个给定的n阶方阵A和B的乘积计算的算法进行算法分析。

```
void MatrixMul(int a[][N],int b[][N],int c[][N],int n)
{
  int i,j,k;
  for (i=0;i<n;i++)
    for (j=0;j<n;j++){
      c[i][j]=0;
      for (k=0;k<n;k++)
        c[i][j]=c[i][j]+a[i][k]*b[k][j];
    }
}
```

算法的时间复杂度为$O(n^3)$

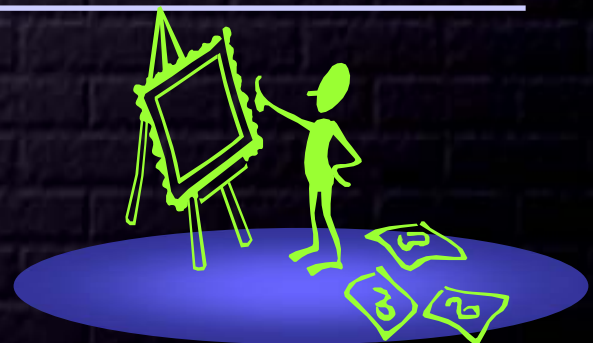


1、算法分析技术

递归

分析递归算法的时间效率

- 决定用哪些参数作为输入规模的度量。
- 找出算法的基本操作。
- 对于算法基本操作的执行次数，建立一个递推关系以及相应的初始条件。
- 求解这个递推式。



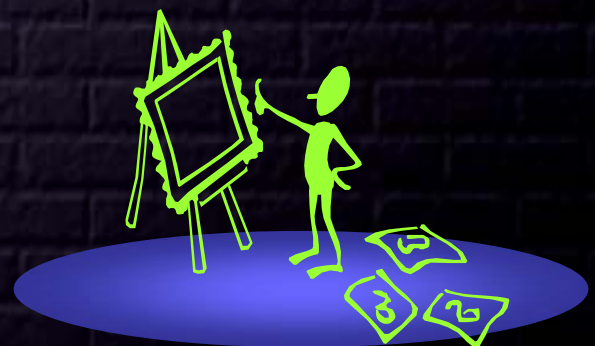
1、算法分析技术

递归

【例3】 对于任意非负整数n，使用递归算法计算阶乘函数Fact(n)=n!的值。

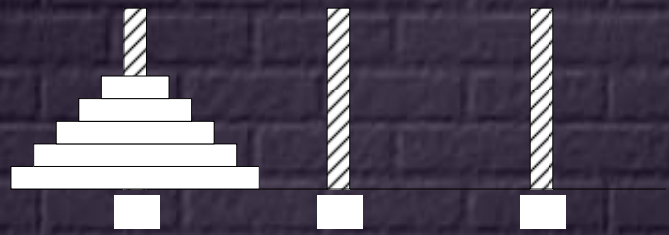
```
long Fact(int n)
{
  if (n==0)  return 1;
  else return Fact(n-1)*n;
}
```

算法的时间复杂度为O(n)



1、算法分析技术

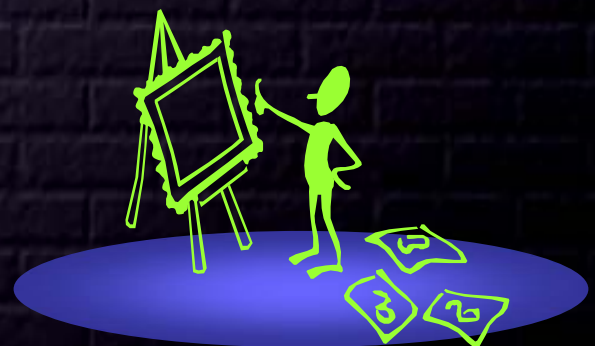
递归



【例4】 Hanoi塔问题。

```
void HanoTower(int n,char a,char b,char c)
{
  if (n==1)
    Move(a,c);
  else{
    HanoTower(n-1,a,c,b);
    Move(a,c);
    HanoTower(n-1,b,a,c);
  }
}
```

算法的时间复杂度为$O(2^n)$



1、算法分析技术

大O表示法

在算法分析中，不考虑**具体**的运行时间函数，只考虑运行时间函数的**数量级**。这种方法称为**渐进表示法**，最常见的是**大O表示法**。

名称	函数	名称	函数
常数阶	$O(1)$	立方阶	$O(n^3)$
对数阶	$O(\log_2 n)$	指数阶	$O(2^n)$
线性阶	$O(n)$	指数阶	$O(n!)$
线性对数阶	$O(n\log_2 n)$	指数阶	$O(n^n)$
平方阶	$O(n^2)$		

时间复杂度之间的关系：

多项式：$O(1) < O(\log_2 n) < O(n) < O(n\log_2 n) < O(n^2) < O(n^3)$

指数：$O(2^n) < O(n!) < O(n^n)$

1、算法分析技术

大O表示法

计算机不是万能的，并非所有的算法，计算机都能够计算出有用的结果。差的算法不一定有实际意义。

举一个例子加以说明。假定时间复杂性函数的时间单位为us。

函数	n=20	n=50	n=100	n=500	
1000n	.02s	.05s	.15s	.5s	易性算法
1000nlogn	.09s	.3s	.6s	4.5s	
$100n^2$	.04s	.25s	1s	25s	
$10n^3$	.02s	1s	10s	21分	
n^{logn}	.4s	1.1小时	220天	5×10^8世纪	
$2^{n/3}$	.0001s	0.1s	2.7小时	5×10^8世纪	
2^n	1s	35年	3×10^4世纪		顽性算法
3^n	58分	2×10^9世纪			

2、枚举算法

枚举法

概念

枚举法又称列举法、穷举法，是蛮力策略的具体体现，是一种简单而直接地解决问题的方法。其基本思想是**逐一列举**问题所涉及的所有情形，并根据问题提出的条件**检验**哪些是问题的解，哪些应予排除。

穷举法的框架描述：

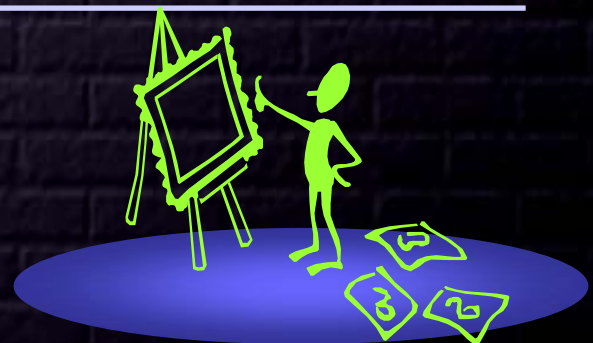
```
n=0;
for(k=<区间下限>;k<=<区间上限>;k++)      // 根据指定范围实施穷举
   if(<约束条件>){                       // 根据约束条件实施筛选
      printf(<满足要求的解>);            // 输出满足要求的解
      n++;                               // 统计解的个数
   }
```

2、枚举算法

概念

枚举的实施步骤

- 根据问题的具体情况确定穷举量（简单变量或数组）。
- 根据确定的范围设置穷举循环。
- 根据问题的具体要求确定筛选约束条件。
- 设计穷举程序并运行、调试，对运行结果进行分析与讨论。



2、枚举算法

枚举设计的意义

- 理论上，穷举可以解决可计算领域中的各种问题。
- 在实际应用中，通常要解决的问题规模不大，用穷举设计的算法其运算速度是可以接受的。
- 枚举可作为某类问题时间性能的底限，用来衡量同样问题的更高效率的算法。

2、枚举算法

整币兑零

- 问题描述

计算把一张n元（1元、2元、5元、10元）整币兑换成1分、2分、5分、1角、2角和5角共6种零币的不同兑换种数。

输入	输出
1	4562

3、递推算法

递推法

概念

递推是利用问题本身所具有的一种递推关系求解问题的一种方法。其基本思想是把一个复杂的庞大的计算过程**转化**为简单过程的多次重复，从头开始一步步地**推出**问题最终的结果。

递推算法避开了求通项公项的麻烦，把一个复杂的问题的求解，**分解**成连续的若干步简单运算。

递推算法的首要问题是得到相邻的数据项之间的关系，即**递推关系**。

3、递推算法

概念

简单顺推方法

```
f(1…i-1)=<初始值>;          // 确定初始值
for(k=i;k<=n;k++)
   f(k)=<递推关系式>;       // 根据递推关系实施递推
printf(f(n));               // 输出 n 规模的解 f(n)
```

简单逆推方法

```
f(n…i+1)=<初始值>;          // 确定初始值
for(k=i;k>=1;k--)
   f(k)=<递推关系式>;       // 根据递推关系实施递推
printf(f(1));               // 输出解 f(1)
```

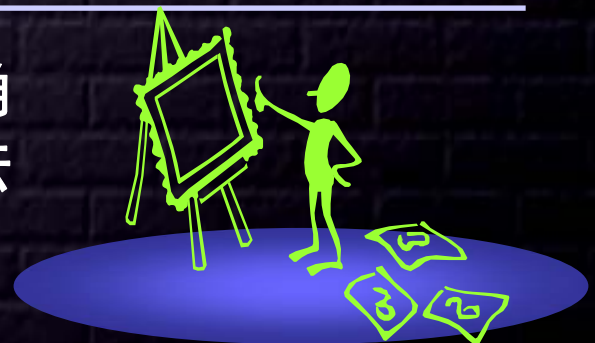
3、递推算法

概念

递推的实施步骤

- 确定递推变量。**递推变量**可以是简单变量，也可以是一维或多维数组。
- 建立递推关系。**递推关系**是指如何从变量的前一些值推出其下一个值或从变量的后一些值推出其上一个值的公式（或关系）。
- 确定初始（边界）条件。
- 对递推过程进行控制。

递推分类：一种是所需的递推次数是确定的值；另一种是所需的递推次数无法确定。



3、递推算法

摆动数列

- 问题描述

已知递推数列：
a(1)=1,a(2i)=a(i)+1,a(2i+1)=a(i)+a(i+1)，（i为正整数），试求该数列的第n项与前n项中哪些项最大?最大值为多少?

输入	输出
2011	225 321

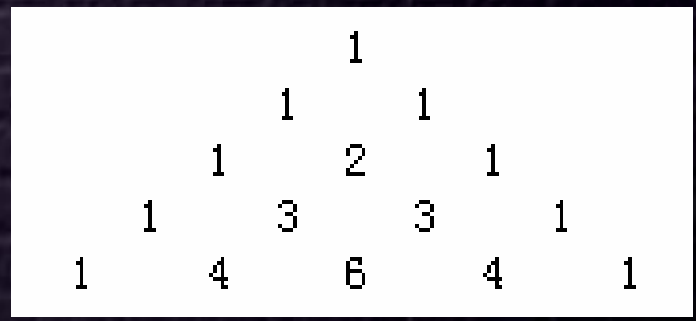
3、递推算法

杨辉三角形

- 问题描述

杨辉三角构建规律主要包括横行各数之间的大小关系以及不同横行数字之间的联系，奥妙无穷：每一行的首尾两数均为1；第k行共k个数，除首尾两数外，其余各数均为上一行的肩上两数的和。

输入	输出
5	1 4 6 4 1



4、分治算法

分治法

概念

分治法，又叫分治策略，顾名思义，分而治之。它的基本思想为：对于难以直接解决的规模较大的问题，把它分解成若干个能直接解决的相互独立的子问题，递归求出各子问题的解，再合并子问题的解，得到原问题的解。

- 该问题可以分解成若干相互独立、规模较小的相同子问题。
- 子问题缩小到一定的程度能轻易得到解。
- 子问题的解合并后，能得到原问题的解。

4、分治算法

- 算法框架描述

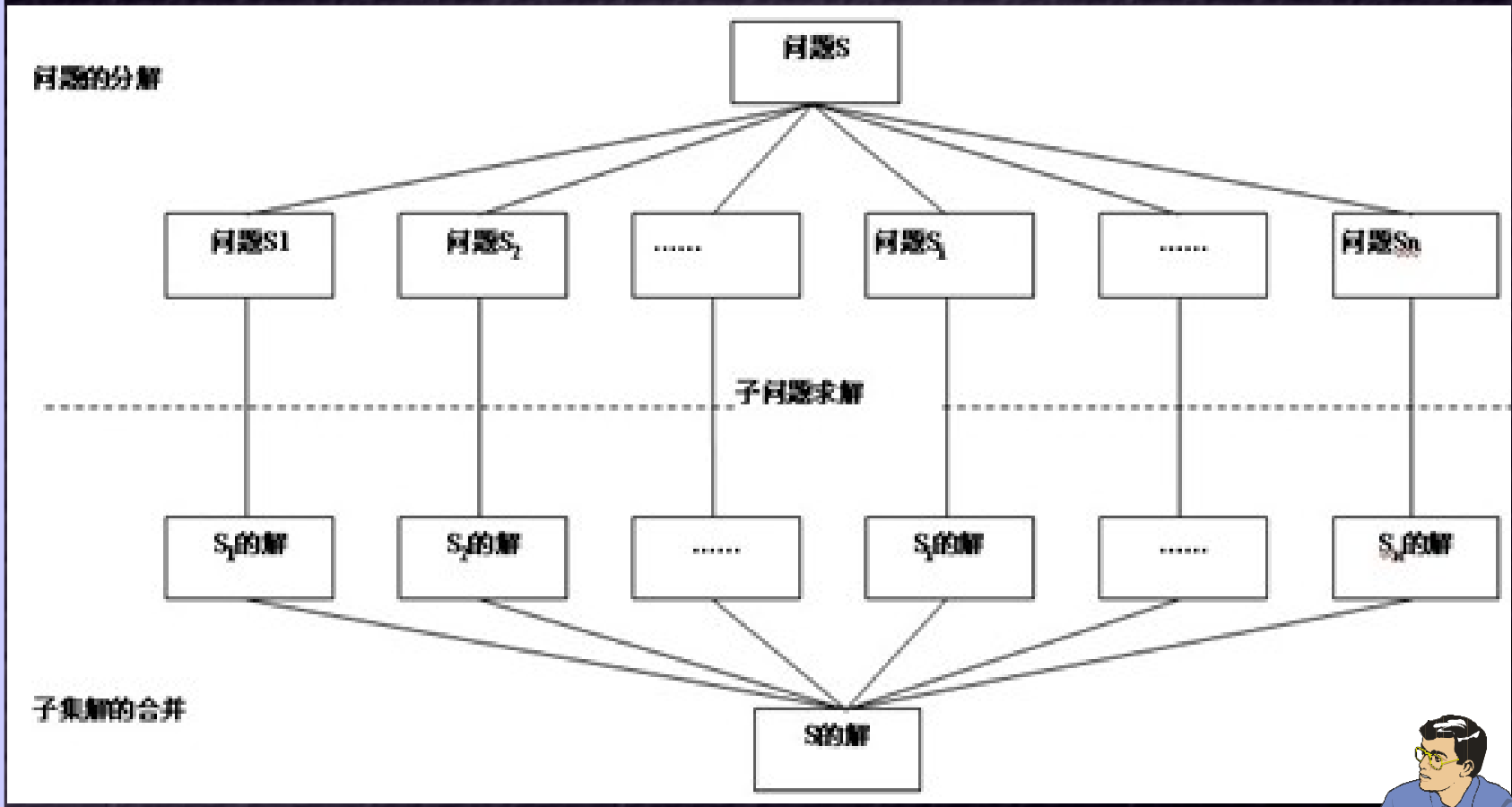
```
Divide-and-Conquer(P)
{
  if (|P|<n0) Adhoc(P);
  divide P into smaller subinstances P1,P2,...,Pk;
  for (i=1;i<=k;i++)
    Yi=Divide-and-Conquer(Pi);
  return Merge(Y1,Y2,...,Yk);
}
```

4、分治算法

- 实施分治的步骤
  - **分解**：将要解决的问题分解成若干个规模较小的同类子问题。
  - **解决**：当子问题划分得足够小时，求解出子问题的解。
  - **合并**：将子问题的解逐层合并成原问题的解。

4、分治算法

- 算法设计过程图



4、分治算法

最值求解

- 问题描述

给n个整数，求它们之中最大值和最小值，要求比较次数尽量小。

输入	输出
10 12 34 67 10 18 98 86 76 23 23	10 98

4、分治算法

一元三次方程求解

- 问题描述

$ax^3+bx^2+cx+d=0$一元三次方程中给出该方程中各项的系数（a、b、c、d均为实数），并约定该方程存在三个不同实根（根的范围在-100至100之间），且根与根之差的绝对值>=1。要求由小到大依次在同一行输出这三个实根（根与根之间留有空格），并精确到小数点后4位。

输入	输出
1 -5 -4 20	-2.0000 2.0000 5.0000

4、分治算法

一元三次方程求解

- 问题分析

  - 如果精确到小数点后两位，可用简单枚举法：将x从-100.00 到100.00（步长0.01）逐一枚举，得到20000个 f(x)，取其值与0最接近的三个f(x)，对应的x即为答案。而题目已改成精度为小数点后4位，枚举算法时间复杂度将达不到要求。
  - 直接使用求根公式，极为复杂。
  - 采用二分法逐渐缩小根的范围，从而得到根的某精度的数值。

4、分治算法

一元三次方程求解

- 问题分析

用二分法求根，若区间(a,b)内有根，则必有f(a)*f(b)<0。重复执行如下的过程：

- 若a+0.0001>b或f((a+b)/2)=0，则可确定根为(a+b)/2并退出过程；
- 若f(a)*f((a+b)/2)<0，则由题目给出的定理可知根在区间(a,(a+b)/2)中，故对区间重复该过程；
- 若f(a)*f((a+b)/2)>0，则必然有f((a+b)/2)*f(b)<0，根在((a+b)/2,b)中，对此区间重复该过程。
- 执行完毕，就可以得到精确到0.0001的根。

4、分治算法

一元三次方程求解

- 问题分析

求方程的所有三个实根：

所有的根的范围都在-100至100之间，且根与根之差的绝对值>=1。因此可知：在[-100,-99]、[-99,-98]、……、[99,100]、[100,100]这201个区间内，每个区间内至多只能有一个根。即：除区间[100,100]外，其余区间[a,a+1]，只有当f(a)=0或f(a)·f(a+1)<0时，方程在此区间内才有解。若f(a)=0，解即为a；若f(a)·f(a+1)<0，则可以利用区间[a,b]中所述的二分法迅速出找出解。

5、贪心算法

贪心法

概念

贪心法是指在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。

- 贪心算法没有固定的算法框架，算法设计的关键是贪心策略的选择。
- 贪心算法不是对所有问题都能得到整体最优解，选择的贪心策略必须具备无后效性。

5、贪心算法

概念

- 特点

  - **贪心选择性质**：算法中每一步选择都是当前看似**最佳**的选择，这种选择依赖于已做出的选择，但不依赖于未作出的选择。
  - **最优子结构性质**：算法中每一次都取得了最优解（即局部最优解），要保证最后的结果最优，则必须满足全局最优解包含局部最优解。

5、贪心算法

概念

贪心法的一般步骤

- 产生问题的一个初始解。
- 循环操作，当可以向给定的目标前进时，就根据局部最优策略，向目标前进一步。
- 得到问题的最优解（或较优解）。

- **优点**：一个正确的贪心算法拥有很多优点，比如思维复杂度低、代码量小、运行效率高、空间复杂度低等。
- **缺点**：贪心法的缺点集中表现在他的“**非完美性**”。很难找到一个简单可行并且保证正确的贪心思路。

5、贪心算法

背包问题

- 问题描述

假设有一个需要使用某一资源的n个活动所组成的集合S，S={1,…,n}。该资源一次只能被一个活动所占用，每一个活动有一个开始时间b_i和结束时间e_i(b_i<=e_i)。若b_i>=e_j或b_j>=e_i，则称活动i和活动j兼容。选择由互相兼容的活动所组成的最大集合，输出该集合的活动数量。

输入	输出
11 3 5 1 4 12 14 8 12 0 6 8 11 6 10 5 7 3 8 5 9 2 13	4

5、贪心算法

背包问题

- 问题分析

  - 贪心策略：每一步总是选择这样一个活动来占用资源：它能够使得余下的未调度的时间最大化，使得兼容的活动尽可能多。
  - 为了达到这一目的，将n个待选活动按结束时间递增的顺序排列。
  - 首先让这一序列中的活动1进入集合S，再依次分析递增序列中的活动2、活动3、…，每次将与S中的活动兼容的活动加入到集合中。

5、贪心算法

背包问题

- 问题深入

如何来证明上述的贪心选择得出的解一定就是最优解呢？

- 构造一个最优解，然后对该解进行修正，使其第一步为一个贪心选择，证明总是存在一个以贪心选择开始的求解方案。
- 证明经过若干次贪心选择后，可以得出一个最优解。
- 贪心选择的设计及其产生最优解的证明是对实际问题分析归纳的结果。所以，归纳分析对贪心求解问题很关键，有的问题看起来很复杂，但用贪心法求解的话，则会惊人地简单。

5、贪心算法

部分背包问题

- 问题描述

给定一个最大载重量为M的卡车和N种食品，有食盐、白糖、大米等。已知第i种食品的最多拥有w[i]公斤，其商品价值为p[i]元/公斤，编程确定一个装货方案，使得装入卡车中的所有物品总价值最大。

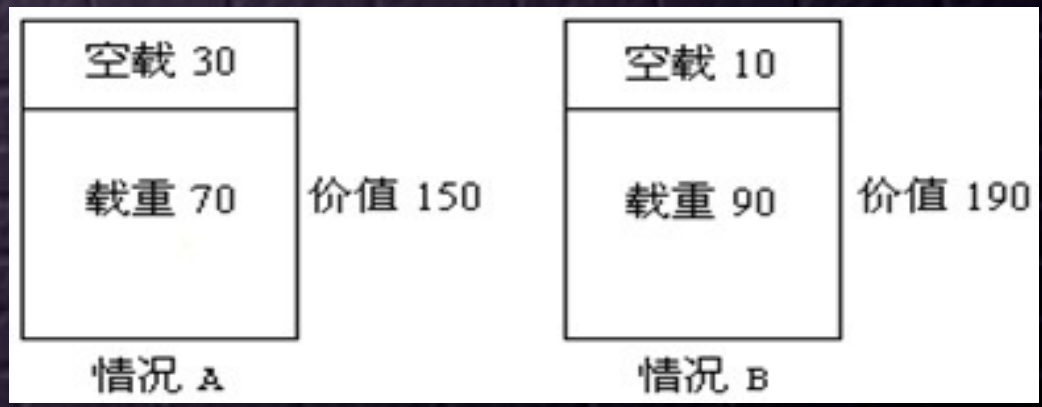
输入	输出
11 4 2 4 6 7 6 10 12 13	139

5、贪心算法

0/1背包问题

- 问题描述

有一个容量为M的背包，现在要从N件物品中选取若干件装入背包中，第i种物品的重量为w[i]、价值为p[i]。背包中物品的总重不能超过背包的容量，并且一件物品要么全部选取，要么不选取。编程确定一个装包方案，使得装入包中的所有物品总价值最大。

输入	输出
100 3 40 50 70 80 100 150	

谢 谢

欢迎批评指正