考情分析

对软件设计而言，数据结构设计解决的是软件程序中的数据组织问题，如何确定数据处理的流程，即确定解决问题的步骤是软件设计的另一个重要问题。

计算机程序需要正确、详尽地描述问题的每个对象和处理逻辑，其中问题的对象由程序的数据结构、变量来描述，而用来描述问题处理逻辑的程序结构、函数和语句构成算法（algorithm）。

算法是计算机程序的灵魂，算法和数据结构共同构成程序的两个重要方面，算法是一组确定的解决问题的步骤。它和机器以及语言并无关系，但最终还是需要使用特定的语言加以实现。

算法设计是计算机程序设计的核心内容之一，算法的优劣直接影响着程序的性能和运行效率。因此，算法设计能力在很大程度上决定了软件设计能力的高低，是软件设计师应具备的重要技能。

算法被公认为计算机科学的基石，算法的设计技术和分析技术是算法理论研究的主要内容。

从近几年的软件设计师考试题目来看，算法设计和数据流图、UML建模技术、数据库设计题目逐渐成为前四个必答题之一，算法设计的考查重点在于对常用算法的综合运用以及对算法复杂度等基本概念的理解。因此，在复习时应该注意加强对算法设计相关内容的理解与实践。

考试大纲要求分析

关于算法设计，在考试大纲中对软件设计师的要求是熟练掌握常用算法。程序设计中常用的算法有迭代、穷举搜索、递推、递归、回溯、贪心、动态规划和分治等。

在考试大纲中，对软件设计师的考试要求是能够做到对常用算法的综合运用（指对所列知识要理解其确切含义及于其他知识的联系能够进行叙述和解释，并能在实际问题的分析、综合、推理和判断等过程中运用）。

数据结构和算法的设计是软件结构设计的重要内容，也是软件设计师考试科目2：软件设计的重要考核内容。

命题特点与趋势分析

在近几年的软件设计师软件设计科目的考试中，关于算法设计的考题是前四道必选题目当中的一道。贪心法、分治法和动态规划3种常用算法，以及算法的时间复杂度多次出现在考题中。

就命题的特点而言，该类考题通过程序流程图、伪代码或程序设计语言（如C程序代码）来表示相应的算法，要求考生将空白处补充完整，或者计算相应算法的时间复杂度。

从命题的趋势上看，大多通过伪代码或程序设计语言来表示相应的算法并进行知识点考核的，说明对算法设计相关知识点的考核逐渐的更加注重灵活运用能力，尤其是结合实际要求解的问题。此外，在近几年的软件设计师下午考试中，对题目中涉及的算法的时间复杂度计算几乎是必考的问题。

鉴于对本知识点考查的方式已趋于实际应用，建议考生在学习的过程中，结合本书的典型真题和考前必练的题目，进行实际的上机编程、调试并实现（建议采用考题中出现频率较高的C程序代码实现），以加强大家对常用算法的理解并提高对常用算法的综合应用能力。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年05月06日

考点精讲

算法设计是指设计一系列解决问题的清晰指令，通过系统的方法描述解决问题的策略与机制。算法能够对一定规范的输入，在有限时间内获得所要求的输出。如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。

算法具有以下五个重要的特征：

（1）有穷性：算法将在执行又穷步后结束，且每一步的运行时间是有穷的。

（2）确定性：算法只有唯一的一条执行路径，即输入相同，输出将相同。

（3）可行性：算法中描述的操作能通过已经实现的基本运算执行有穷次来实现。

（4）输入：一个算法可能有零个或多个输入。

（5）输出：一个算法可能有一个或多个输入。

算法可以由自然语言、流程图、程序设计语言和伪代码来表示。

算法的时间复杂度和空间复杂度是衡量一个算法效率高低的重要指标。在算法设计过程中，可以通过时间来换取空间，也可以通过空间来换取时间，算法的时间复杂度分析主要是分析算法的运行时间。递归、贪心、回溯、分治和动态规划都是比较常用的算法，也是近年来有关算法设计部分的重要考试内容。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网
2014年05月06日

算法的表示

常用的表示算法的方法有自然语言、流程图、程序设计语言和伪代码等。各类表示方法的特点如下：

（1）自然语言虽然容易理解，但也容易出现二义性，并且描述起来比较冗长。

（2）程序设计语言的优点是计算机直接执行，但和具体语言相关，抽象性较差，要求算法设计者掌握程序设计语言及编程技巧。

（3）流程图优点是直观易懂，但严密性不如程序设计语言。

（4）伪代码介于自然语言和程序设计语言之间，能够比较简明扼要的表达算法设计。

流程图和伪代码都是比较常用的算法表示方式。从近几年软件设计师的考题情况来看，算法设计部分一般是通过程序流程图、伪代码和程序设计语言三种表示方式来考查考生对算法的理解和掌握情况。

另外，算法的复杂度尤其是时间复杂度是近年来频繁出现的考查点。同一问题可用不同算法解决，而一个算法的质量优劣将影响到算法乃至程序的效率。对一个算法的评价主要从时间复杂度和空间复杂度来考虑，算法评价的目的在于选择合适算法和改进算法。

随着问题规模n的不断增大，算法的时间复杂度不断增大，执行效率会降低。常见的时间复杂度有：

（1）常数阶O(1)：表示算法的运行时间为常量，与问题规模n无关。

（2）对数阶O($\log_2 n$)：表示算法的运行时间与问题规模n是对数相关，如二分查找。

（3）线性阶O(n)：表示算法的运行时间与问题规模n是线性相关。

（4）线性对数阶O($n\log_2 n$)：表示算法的运行时间与问题规模n是线性对数相关。

（5）平方阶O(n^2)：对数组机型排序的各种简单算法，如选择排序。

（6）立方阶O(n^3)：如两个n阶矩阵的乘法运算。

（7）k次方阶O(n^k)。

（8）指数阶O(2^n)等。

时间复杂度按数量级递增排列依次为：常数阶O(1)、对数阶O($\log_2 n$)、线性阶O(n)、线性对数阶O($n\log_2 n$)、平方阶O(n^2)、立方阶O(n^3)、k次方阶O(n^k)、指数阶O(2^n)，

一般情况下，如果选用指数阶的算法，问题规模不应大于10。否则，应去分析该算法是否合理，是否有复杂度较低的其他算法。

在许多情况下，算法设计遵循了人类求解问题的一般方法和思路。如“从部分到整体”和“从整体到部分”。在算法设计中，由于问题本身的复杂性与需要考虑程序实现的方便性等因素，算法设计者不得不在各种制约因素之间取得平衡，如算法的空间要求与时间要求间的平衡、算法结果的最优性与算法效率的平衡等。

下面对递归法、贪心法、回溯法、分治法和动态规划法5种常用的算法分别进行介绍。

作者：希赛教育软考学院　来源：希赛网　2014年05月06日

递归法

一个过程或函数在其定义或说明中又直接或间接调用自身的一种方法，它通常把一个大型复杂的问题转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序就可描述出解题过程所需要的多次重复计算，大大地减少了程序的代码量。递归的能力在于用有限的语句来定义对象的无限集合。一般来说，递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时，递归前进；当边界条件满足时，递归返回。

递归算法一般用于解决三类问题：

（1）数据的定义是按递归定义的。（Fibonacci函数）

（2）问题解法按递归算法实现。（回溯）

（3）数据的结构形式是按递归定义的。（树的遍历，图的搜索）

递归的执行过程可分为分解和求值两部分。首先是逐步把“大问题”分解成形式相同但规模很小的“小问题”，直至分解到递归出口。一旦遇到递归出口，分解过程结束，开始求值过程，所以分解过程是“量变”过程，即原来的“大问题”在慢慢变小，但尚未解决。遇到递归出口后，便发生了“质变”，即原递归问题转换成直接可以求值的简单问题。

递归只需要少量的步骤就可描述解题过程中所需要的多次重复计算，极大地减少了代码量。该算法设计的关键在于，找出递归方程和边界条件（递归出口）。递归关系就是使问题向边界条件转化的过程，所以递归关系必须能使问题越来越简单，规模越小。没有设定边界的递归是死循环。

递归算法设计通常需要按照以下3个步骤：

（1）分析问题，得出递归关系；

（2）设置边界条件，控制递归；

（3）设计函数，确定参数。

求解Fibonacci数列的第n项函数Fibonacci（n）就是一个典型的例子。Fibonacci数列为：1、1、2、3、5、8、13、21、…，即 Fibonacci（1）=1; Fibonacci（2）=1；Fibonacci（n）= Fibonacci（n-1）+ Fibonacci（n-2）（当n>2时）。

根据递归公式，很容易得出递归函数：

```
int Fibonacci( int n )
{
   if ( n < 1 )        //预防错误
      return 0;
if ( n = = 1 || n = =2 ) // 设置边界条件
      return 1;
   return  Fibonacci( n-1 )+ Fibonacci( n-2 );
}
```

需要说明的是上边的例子并不是递归方法的最佳例子，因为在n足够大时，则程序将变得缓慢甚至可能将计算机的内存耗尽。因此，在类似问题求解时，应限制n的最大值。

又如楼梯有n阶台阶，上楼可以一步上1阶，也可以一步上2阶，编写程序计算共有多少种不同的

走法。例如，当n=3时，共有3种走法，即1+1+1，1+2，2+1，当n=4时，共有5种走法，即1+1+1+1，2+2，2+1+1，1+2+1，1+1+2。算法分析：设n阶台阶的走法数为f(n)，显然有：

(1) f(1)= 1　　n = 1 ;

(2) f(2)= 2　　n = 2 ;

(3) f(n)=f(n-1)+ f(n-2)　n > 2。

得到相应的函数如下：

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

再如Hanoi Tower问题：设A，B，C是3个塔座。开始时，在塔座A上有一叠共n个圆盘，这些圆盘自上而下，由小到大地叠在一起。各圆盘从小到大编号为1，2，…，n，现要求将塔座A上的这一叠圆盘移到塔座C上，并仍按同样顺序叠置。

在移动圆盘时应遵守以下移动规则：

(1)每次只能移动1个圆盘；

(2)任何时刻都不允许将较大的圆盘压在较小的圆盘之上；

(3)在满足移动规则1和2的前提下，可将圆盘移至A，B，C中任一塔座上。

算法分析：这是一个典型的适合用递归算法来解决的问题。

试想要把n个盘子从柱A移到柱C上，则必须先把上面n-1个盘子从柱A全部移到柱B，然后把第n个盘子由柱A移到柱C ，再把柱B上的n-1个盘子全部移到柱C。这样就把移动n个盘子的问题变成了移动n-1个盘子的问题，如此不断减小递归的规模，直到递归出口。

递归的边界条件是，当n = =1时，直接把它由柱A移到柱C。

```
//汉诺塔，把柱A所有的盘子移到柱C
Void Hanoi( int n, char a, char b, char c )
{
   //如果只有一个盘子，直接由柱A移到柱C
   if ( n  = = 1 )
      printf( "Move disk from %c to %c\n", a, c );
   else
    {
      //先把上面n-1个盘子从柱A全部移到柱B
      Hanoi( n-1, a, c, b );
      //再把第n个盘子由柱A移到柱C
      printf( "Move disk from %c to %c\n", a, c );
      // 再把柱B上的n-1个盘子全部移到柱C
      Hanoi( n-1, b, a, c );
    }
```

}

从上述的3个例子中我们可以发现，递归算法具有以下三个基本规则：

（1）基本情形：至少有一种无需递归即可获得解决的情形，也即前面说的边界条件。

（2）进展：任意递归调用必须向基本情形迈进，即前面所说的使得问题规模变小。

（3）正确性假设：总是假设递归调用是有效的。

递归调用的有效性是可以用数学归纳法证明的，所以当我们在设计递归函数时，不必设法跟踪可能很长的递归调用途径（比如Hanoi Tower问题）。这种任务可能很麻烦，易于使设计和验证变得更加困难。所以我们一旦决定使用递归算法，则必须假设递归调用是有效的。

与递归相对应的递推算法是一种用若干步可重复的简单运算（规律）来描述复杂问题的方法。递推算法以初始（起点）值为基础，用相同的运算规律，逐次重复运算，直至运算结束。这种从“起点”重复相同的方法直至到达一定“边界”，犹如单向运动，用循环可以实现。递推的本质是按规律逐次推出（计算）先一步的结果。

在算法设计时，能用递推实现的算法一般都可以用递归来实现。能用递归实现的算法不一定能够通过递推实现。就算法的效率而言，递推优于递归。

上一节　本书简介　下一节

贪心法

贪心法是一种不追求最优解，只希望得到较为满意解的方法。贪心法一般可以快速得到满意的解，因为它省去了为找最优解要穷尽所有可能而必须耗费的大量时间。贪心法常以当前情况为基础作最优选择，而不考虑各种可能的整体情况。

在求最优解问题的过程中，依据某种贪心标准，从问题的初始状态出发，直接去求每一步的最优解，通过若干次的贪心选择，最终得出整个问题的最优解，这种求解方法就是贪心算法。

从贪心算法的定义可以看出，贪心算法并不是从整体上考虑问题，它所做出的选择只是在某种意义上的局部最优解，而由问题自身的特性决定了该题运用贪心算法是否可以得到最优解。

如均分纸牌问题。有 N 堆纸牌，编号分别为 1，2，…, N。每堆上有若干张，但纸牌总数必为 N 的倍数。可以在任一堆上取若干张纸牌，然后移动。移牌规则为：在编号为 1 堆上取的纸牌，只能移到编号为 2 的堆上；在编号为 N 的堆上取的纸牌，只能移到编号为 N-1 的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。例如 N=4，4 堆纸牌数分别为：9、8、17、6。设a[i]为第i堆纸牌的张数（0<=i<=n），v为均分后每堆纸牌的张数，s为最小移到次数。用贪心法按照从左到右的顺序移动纸牌。如第i堆（0<i<n）的纸牌数a[i]不等于平均值，则移动一次（即s加1），分两种情况移动：

（1）若a[i]>v，则将a[i]-v张纸牌从第i堆移动到第i+1堆；

（2）若a[i]<v，则将v -a[i]张纸牌从第i+1堆移动到第i堆；

上述两种情况可以统一看作是将a[i]-v张牌从第i堆移动到第i+1堆；移动后有：a[i]=v；a[i+1]=a[i+1]+a[i]-v；在从第i+1堆中取出纸牌补充第i堆的过程中，可能会出现第i+1堆的纸牌数小

于零（a[i+1]+a[i]-v<0 ）的情况。如n=3，三堆纸牌数为（1，2，27）这时v=10，为了使第一堆数为10，要从第二堆移9张纸牌到第一堆，而第二堆只有2张纸牌可移。

从第二堆移出9张到第一堆后，第一堆有10张纸牌，第二堆剩下-7张纸牌，再从第三堆移动17张到第二堆，刚好三堆纸牌数都是10，最后结果是对的，从第二堆移出的牌都可以从第三堆得到。

在移动过程中，只是改变了移动的顺序，而移动的次数不变，因此，此题使用贪心法可以求解。

利用贪心法求解需要注意以下两点：

（1）贪心法求解是否适合。

用贪心法解题很方便，但它的适用范围很小，判断一个问题是否适合用贪心法求解，需要在平时多加练习，依据解题经验来判断是否适合使用贪心算法。如下边的例子：

以找币问题为例，如果一个货币系统有3种币值，面值分别为一角、五分和一分，求最小找币数时，可以用贪心法求解；如果将这三种币值改为一角一分、五分和一分，就不能使用贪心法求解。

（2）选择何种贪心标准才能保证求得最优解。

在选择贪心标准时，要对所选的贪心标准进行验证才能使用，不要被表面上看似正确的贪心标准所迷惑。如下面的例子：

有N个正整数，将他们连接成一排，组成一个最大的多位整数。例如有3个正整数：13、5、20，则20513就是所求。又如13、131这2个正整数，连接组成最大的多位数是13131。分析后发现其贪心标准为：把整数化成字符串，然后再比较a+b和b+a，如果a+b>b+a，就把a排在b的前面，反之则把a排在b的后面。如果把贪心标准设定为数值大的数放在前面，有时就会得到错误的结论，比如13、131这2个正整数，连接组成的多位数是13113，但这并不是最大的多位数。

贪心法的基本原理是从问题的某一个初始解出发逐步逼近给定的目标，以尽可能快的地求得更好的解。当达到某算法中的某一步不能再继续前进时，算法停止。

因此，贪心法存在以下问题：

（1）不能保证求得的最后解是最优解；

（2）不能用来求最大或最小解问题；

（3）只能求满足某些约束条件的可行解的问题。

利用贪心法解题，通常的解题思路为：

（1）建立数学模型来描述问题；

（2）把求解的问题分成若干个子问题；

（3）对每一子问题求解，得到子问题的局部最优解；

（4）把子问题的解局部最优解合成原来解问题的一个解

利用贪心法解题，通常的步骤为以下三步：

（1）从问题的某一初始解出发；

（2）循环求解，求出可行解的一个解元素；

（3）由所有解元素组合成问题的一个可行解。

作者：希赛教育软考学院　来源：希赛网
2014年05月06日

回溯法

回溯法（探索与回溯法）是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。

回溯法首先将问题P的n元组的状态空间E表示成一棵高为n的带权有序树T，把在E中求问题P的所有解转化为在T中搜索问题P的所有解。

树T类似于检索树，它可以这样构造：

设Si中的元素可排成xi（1），xi（2），…，xi（mi-1），|Si| =mi，i=1，2，…，n。从根开始，让T的第I层的每一个结点都有mi个儿子。这mi个儿子到它们的双亲的边，按从左到右的次序，分别带权xi+1（1），xi+1（2），…，xi+1（mi），i=0，1，2，…，n-1。照这种构造方式，E中的一个n元组（x1，x2，…，xn）对应于T中的一个叶子结点，T的根到这个叶子结点的路径上依次的n条边的权分别为x1，x2，…，xn，反之亦然。

另外，对于任意的0≤i≤n-1，E中n元组（x1，x2，…，xn）的一个前缀I元组（x1，x2，…，xi）对应于T中的一个非叶子结点，T的根到这个非叶子结点的路径上依次的I条边的权分别为x1，x2，…，xi，反之亦然。特别，E中的任意一个n元组的空前缀（ ），对应于T的根。

因而，在E中寻找问题P的一个解等价于在T中搜索一个叶子结点，要求从T的根到该叶子结点的路径上依次的n条边相应带的n个权x1，x2，…，xn满足约束集D的全部约束。在T中搜索所要求的叶子结点，很自然的一种方式是从根出发，按深度优先的策略逐步深入，即依次搜索满足约束条件的前缀1元组（x1i）、前缀2元组（x1，x2）、…，前缀I元组（x1，x2，…，xi），…，直到i=n为止。

在回溯法中，上述引入的树被称为问题P的状态空间树；树T上任意一个结点被称为问题P的状态结点；树T上的任意一个叶子结点被称为问题P的一个解状态结点；树T上满足约束集D的全部约束的任意一个叶子结点被称为问题P的一个回答状态结点，它对应于问题P的一个解。

如找出从自然数1，2，…，n中任取r个数的所有组合的问题。采用回溯法求问题的解，将找到的组合以从小到大顺序存于a[0]，a[1]，…，a[r-1]中，组合的元素满足以下性质：

（1）a[i+1]>a[i]，后一个数字比前一个大；

（2）a[i]-i<=n-r+1。

按回溯法的思想，求解过程可以分析如下：

首先放弃组合数个数为r的条件，候选组合从只有一个数字1开始。因该候选解满足除问题规模之外的全部条件，扩大其规模，并使其满足上述条件（1），候选组合改为1，2。继续这一过程，得到候选组合1，2，3。该候选解满足包括问题规模在内的全部条件，因而是一个解。在该解的基础上，选下一个候选解，因a[2]上的3调整为4，以及以后调整为5都满足问题的全部要求，得到解1，2，4和1，2，5。由于对5不能再作调整，就要从a[2]回溯到a[1]，这时，a[1]=2，可以调整为3，并向前试探，得到解1，3，4。重复上述向前试探和向后回溯，直至要从a[0]再回溯时，说明已经找完问题的全部解。示意代码如下：

```
int a[100];
void comb( int m,int r )
{
```

```
int i,j;
 i = 0;
 a[i] = 1;
 do {
    if ( a[i]-i< = m-r+1) {
if ( i = =r-1 ) {
           for ( j = 0;j<r;j++ ) printf( "%4d",a[j] );
           printf( "\n" );
        }
        a[i]++;
        continue;
    } else
      { if ( i = =0 ) return;
        a[--i]++;
      }
  } while (1)
}
main( )
{
comb(5,3);
}
```

又如古老而著名的n皇后问题，是回溯算法的典型例题。该问题是十九世纪著名的数学家高斯1850年提出：在8×8格的国际象棋上摆放8个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上，问有多少种摆法。 在国际象棋中，皇后是可以横走、直走、斜走的最强大的棋子。在一个8乘8的棋盘上，如何放置最多数量的皇后，使这些皇后互相不吃，由于每行每列最多只能一个皇后，所以最多只能8个皇后出现在一个8×8格的棋盘上。

再如迷宫问题中，在寻找路径时，采用的方法通常是：从入口出发，沿某一方向向前试探，若能走通，则继续向前进；如果走不通，则要沿原路返回，换一个方向再继续试探，直到所有的可能都试探完成为止。为了保证在任何位置上都能沿原路返回（回溯），要建立一个后进先出的栈来保存从入口到当前位置的路径。而且在求解迷宫路径中，所求得的路径必须是简单路径。即在求得的路径上不能有重复的同一块通道。

回溯法其实就是试探的方法，它是一种系统地搜索问题的解的方法。回溯算法的基本思想是：从一条路往前走，能进则进，不能进则退回来，换一条路再试。利用回溯法解题，通常可分为以下三个步骤：

（1）针对所给问题，定义问题的解空间；

（2）确定易于搜索的解空间结构；

（3）以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

 作者：希赛教育软考学院 来源：希赛网 2014年05月06日

分治法

任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。

例如，对于n个元素的排序问题，当n=1时，不需任何计算。n=2时，只要作一次比较即可排好序。n=3时，只要作3次比较即可。而当n较大时，问题就不那么容易处理了。

要想直接解决一个规模较大的问题，有时是相当困难的。分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便各个击破，分而治之。

分治策略是：对于一个规模为n的问题，若该问题可以容易地解决（比如说规模n较小）则直接解决，否则将其分解为k个规模较小的子问题，这些子问题互相独立且与原问题形式相同，递归地解这些子问题，然后将各子问题的解合并得到原问题的解。

如果原问题可分割成k个子问题，$1<k\leq n$ ，且这些子问题都可解并可利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。

由分治法产生的子问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解。

这自然导致递归过程的产生。分治与递归像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法。

分治法所能解决的问题一般具有以下几个特征：

（1）该问题的规模缩小到一定的程度就可以容易地解决；

（2）该问题可以分解为若干个规模较小的相同问题，即该问题具有最优子结构性质；

（3）利用该问题分解出的子问题的解可以合并为该问题的解；

（4）该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子子问题。

绝大多数问题都具备上述的第1条特征，因为问题的计算复杂性一般是随着问题规模的增加而增加；

第2条特征是应用分治法的前提它也是大多数问题可以满足的，此特征反映了递归思想的应用；

第3条特征是关键，能否利用分治法完全取决于问题是否具有第三条特征，如果具备了第一条和第二条特征，而不具备第三条特征，则可以考虑用贪心法或动态规划法。

第4条特征涉及到分治法的效率，如果各子问题是不独立的则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然可用分治法，但时在一般情况下，用动态规划法较好。

如给定一个顺序表，编写一个求出其最大值和最小值的分治算法。由于顺序表的结构没有给出，作为演示分治法这里从简，设顺序表为一整形数组，数组大小由用户定义，数据随机生成。如果数组大小为 1 则可以直接给出结果，如果大小为 2，则一次比较即可得出结果，于是我们找到求解该问题的子问题即：数组大小 <= 2。到此我们就可以进行分治运算了，只要求解的问题数组长度比 2 大就继续分治，否则求解子问题的解并更新全局解，这里以C代码为例：

```
#include<stdio.h>
```

```
#include<stdlib.h>
#include<limits.h>
#defineM40
/*分治法获取最优解*/
voidPartionGet( int s,int e,int *meter,int *max,int *min )
{
/*参数:
*s当前分治段的开始下标
*e当前分治段的结束下标
*meter表的地址
*max存储当前搜索到的最大值
*min存储当前搜索到的最小值*/
int i;
if( e-s < = 1 ) {
    /*获取局部解，并更新全局解*/
      if( meter[s] > meter[e] ){
      if( meter[s] > *max )   *max = meter[s];
      if( meter[e] < *min )   *min = meter[e];
}else{
if( meter[e] > *max )   *max = meter[s];
if( meter[s] < *min )   *min = meter[s];
       }
    return;
}
i = s+( e-s )/2;
/*不是子问题继续分治，这里使用了二分，也可以是其它*/
PartionGet( s,i,meter,max,min );
PartionGet( i+1,e,meter,max,min );
}
int main ( )
{
int i,meter[M];
/*用最小值初始化*/
int max = INT_MIN;
/*用最大值初始化*/
int min = INT_MAX;
printf( "Thearray'selementasfollowed:\n\n" );
/*初始化随机数发生器*/
randomize( );
```

```
for( i = 0;i<M;i++ ){
/*随机数据填充数组*/
   meter = rand( )%10000;
  if( !( ( i+1 )%10 ) )
    /*输出表的随机数据*/
       printf( "%-6d\n", meter );
   else    printf( "%-6d", meter );
}
PartionGet( 0,M-1,meter,&max,&min );
/*分治法获取最大值、最小值*/
printf( "\nMax:%d\nMin:%d\n", max, min );
system( "pause" );
return0;
}
```

又如在数组中查找元素，常用的算法是遍历整个数组进行查找，算法时间复杂度为O（n）；但是对于有序数组，使用二分查找法，可以使时间复杂度减少到O(log2n)。

普通查找法：

```
int IsElement( int a[], int len, int x ) {
   //判断数据x是否为数组a的元素，如果是返回该元素的下标，否则返回-1
   int i;
   for ( i = 0; i<len; i++ )
   {
      if ( a[i] = = x )
         return i;
   }
   return -1;
}
```

二分查找法：

```
int IsElement( int a[], int len, int x )
{
   int left = 0;
   int right = len - 1;
   int mid;
   while ( left <= right )
   {
      //寻找中点，以便将数组分成两半
      mid = ( left + right ) / 2;
      if ( a[mid] = = x )
         return mid;
```

```
            //比中位元素的值小，右边界左移
        else if ( a[mid] > x )
            right = mid - 1;
        else
            left = mid + 1;
    }
    return -1;
}
```

也可以写成递归的形式：

```
int IsElement( int a[], int len, int x )      //驱动程序
{
    return Binary( a, 0, len-1, x );
}
int Binary( int a[], int left, int right, int x )  //二分递归查找
{
    int mid =( left+right )/2;
    if ( left > right )              //没找到
        return -1;
    if ( a[mid] = = x )              //刚好找到
        return mid;
    else if ( a[mid] > x )           //比中点元素小，递归查找左侧数组
        return Binary( a, left, mid-1, x );
    else                        //比中点元素大，递归查找右侧数组
        return Binary( a, mid+1, right, x );
}
```

在二分查找法中，通过不断的减少查找区域的范围，把大问题分解成结构相同的小问题，直到问题得解，是分治法算法思想的典型体现。

二分法插入排序：插入排序是经典的简单排序法，它的时间复杂度最坏为O(n^2)。代码如下：

```
//插入法对数组进行排序，从第二个元素开始，依次把元素插入到其左边比其小的元素之后
void InsertSort( int a[], int len )
{
    int i, j, temp;
    for ( i = 1; i<len; i++ )      //从第二个元素开始，依次从左向右判断
    {
        temp = a[i];       //记录当前被判断的元素
        j = i - 1;
        while ( j >= 0 && a[j] > temp )    //把比temp大的元素向后移动一个位置
        {
            a[j+1] = a[j];
```

```
            j--;
        }
        a[j+1] = temp;       //把temp插入到适当位置
    }
}
```

再如插入算法，使得当前被插入的元素左侧的数组是已经排好序的。算法思想简单描述：在插入第i个元素时，对前面的0～i-1元素进行折半，先跟他们中间的那个元素比，如果小，则对前半再进行折半，否则对后半进行折半，最后把第i个元素放在目标位置上。代码如下：

```
void HalfInsertSort( int a[], int len )
{
    int i, j;
    int low, high, mid;
    int temp;
    for ( i = 1; i<len; i++ )
    {
        temp = a[i];
        low = 0;
        high = i - 1;
        while ( low <= high )      //折半查找有序插入的位置
        {
            mid = ( low + high ) / 2;
            if ( a[mid] > temp )
                high = mid - 1;
            else
                low = mid + 1;
        }
        for ( j = i-1; j>high; j-- )      //元素后移
            a[j+1] = a[j];
        a[high+1] = temp;      //插入
    }
}
```

包括希尔排序，合并排序和快速排序等排序算法都要用到了分治算法的思想。

利用分治法解题，在每一层递归上分为以下三个步骤：

（1）分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；

（2）解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题；

（3）合并：将各个子问题的解合并为原问题的解。

作者：希赛教育软考学院　来源：希赛网
2014年05月06日

动态规划法

动态规划法是20世纪50年代由贝尔曼（R. Bellman）等人提出，用来解决多阶段决策过程问题的一种最优化方法。所谓多阶段决策过程，就是把研究问题分成若干个相互联系的阶段，由每个阶段都作出决策，从而使整个过程达到最优化。许多实际问题利用动态规划法处理，常比线性规划法更为有效，特别是对于那些离散型问题。

实际上，动态规划法就是分多阶段进行决策，其基本思路是：按时空特点将复杂问题划分为相互联系的若干个阶段，在选定系统行进方向之后，逆着这个行进方向，从终点向始点计算，逐次对每个阶段寻找某种决策，使整个过程达到最优，故又称为逆序决策过程。

在这种多阶段决策问题中，各个阶段采取的决策，一般来说是与时间有关的，决策依赖于当前状态，又随即引起状态的转移，一个决策序列就是在变化的状态中产生出来的，故有“动态”的含义，这种解决多阶段决策最优化问题的方法即动态规划方法。

动态规划是一种在数学和计算机科学中使用的，用于求解包含重叠子问题的最优化问题的方法。其基本思想是将原问题分解为相似的子问题，在求解的过程中通过子问题的解求出原问题的解。

动态规划的思想是多种算法的基础，被广泛应用于计算机科学和工程领域。动态规划的实质是分治思想和解决冗余，因此，动态规划是一种将问题实例分解为更小的、相似的子问题，并存储子问题的解而避免计算重复的子问题，以解决最优化问题的算法策略。

由此可知，动态规划法与分治法和贪心法类似，它们都是将问题实例归纳为更小的、相似的子问题，并通过求解子问题产生一个全局最优解。其中贪心法的当前选择可能要依赖已经作出的所有选择，但不依赖于有待于做出的选择和子问题。因此贪心法自顶向下，一步一步地作出贪心选择；而分治法中的各个子问题是独立的（即不包含公共的子子问题），因此一旦递归地求出各子问题的解后，便可自下而上地将子问题的解合并成问题的解。但不足的是，如果当前选择可能要依赖子问题的解时，则难以通过局部的贪心策略达到全局最优解；如果各子问题是不独立的，则分治法要做许多不必要的工作，重复地解公共的子问题。用分治法递归求解问题时，每次遇到子问题都会视为新问题，如果存在大量的重叠子问题，会极大地降低算法的效率，而动态规划法总是充分利用重叠子问题，对每个重叠的子问题仅计算1次，把解保存在一个需要时就可以查询的表中，并使得每次查表的时间为常数，从而提高了算法的执行效率。动态规划法以自底向上的方式计算出最优值，和贪心法一样要求问题具有最优子结构，但不同的是贪心法要求问题具有贪心选择性质，即问题的整体最优解能够通过贪心选择来得到。

动态规划法主要应用于最优化问题，这类问题会有多种可能的解，每个解都有一个值，而动态规划找出其中最优（最大或最小）值的解。若存在若干个取最优值的解的话，它只取其中的一个。但是要保证该问题的无后效性，即无论当前取哪个解，对后面的子问题都没有影响。在求解过程中，该方法也是通过求解局部子问题的解达到全局最优解，但与分治法和贪心法不同的是，动态规划允许这些子问题不独立，（亦即各子问题可包含公共的子子问题）也允许其通过自身子问题的解作出选择，该方法对每一个子问题只解一次，并将结果保存起来，避免每次碰到时都要重复计算。

动态规划程序设计是对解最优化问题的一种途径、一种方法，而不是一种特殊算法，动态规划

方法并不具有一个标准的数学表达式和明确清晰的解题方法。

动态规划程序设计往往是针对一种最优化问题，由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划，可以解决各类最优化问题。

因此，在学习时除了要对基本概念和方法正确理解外，必须具体问题具体分析处理，以丰富的想象力去建立模型，用创造性的技巧去求解。

动态规划的基本模型分为如下5部分：

（1）确定问题的决策对象；

（2）对决策过程划分阶段；

（3）对各阶段确定状态变量；

（4）根据状态变量确定费用函数和目标函数；

（5）建立各阶段状态变量的转移过程，确定状态转移方程。

任何思想方法都有一定的局限性，超出了特定条件，它就失去了作用。同样，动态规划也并不是万能的。适用动态规划的问题必须满足最优化原理和无后效性。最优化原理（最优子结构性质）即一个最优化策略具有这样的性质，不论过去状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略。简而言之，一个最优化策略的子策略总是最优的。一个问题满足最优化原理又称其具有最优子结构性质。无后效性将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策，而只能通过当前的这个状态。换句话说，每个状态都是过去历史的一个完整总结。这就是无后向性，又称为无后效性。子问题的重叠性是指动态规划将原来具有指数级复杂度的搜索算法改进成了具有多项式时间的算法。其中的关键在于解决冗余，这是动态规划算法的根本目的。

动态规划实质上是一种以空间换时间的技术，它在实现的过程中，不得不存储产生过程中的各种状态，所以，动态规划算法的空间复杂度要大于其它的算法。

动态规划法所针对的问题有一个显著的特征，即它所对应的子问题树中的子问题呈现大量的重复。动态规划法的关键就在于，对于重复出现的子问题，只在第一次遇到时加以求解，并把答案保存起来，让以后再遇到时直接引用，不必重新求解。

作为一个非常有效的算法设计技术，在什么情况下应被采用呢？适合动态规划算法解决的问题往往具有以下两个性质：

（1）最优子结构。指一个问题的最优解中包含了其子问题的最优解。当一个问题具有最优子结构时，说明动态规划法可能适用。但是，在在这种情形下，贪心法也是有可能适用的。

（2）重叠子问题。指用来解决原问题的递归算法可反复地解同样的子问题，而不是总在产生新的子问题。即当一个递归算法不断调用同一个问题时，就可认为该问题包含重叠子问题。此时采用动态规划法优于分治法递归求解的原因是：分治法每次遇到子问题会视为是新问题，算法的效率被降低了，而动态规划法对于重叠子问题只进行一次，于是可以获得较好的计算效率。

动态规划方法的典型应用场景包括解决背包问题、图象压缩、矩阵乘法链、最短路径、无交叉子集和元件折叠等。

如背包问题：解决背包问题的方法有多种，动态规划，贪心算法，回溯法，分支定界法都能解决背包问题。其中动态规划，回溯法，分支定界法都是解决0-1背包问题的方法。背包问题与0-1背包问题的不同点在于在选择物品装入背包时，可以只选择物品的一部分，而不一定是选择物品的全

部。在这里，我们组用的有贪心法和动态规划法来对这个问题进行算法的分析设计。用动态规划的方法可以看出如果通过第n次选择得到的是一个最优解的话，那么第n-1次选择的结果一定也是一个最优解。这符合动态规划中最优子问题的性质。动态规划方法是处理分段过程最优化一类问题极其有效的方法。在该问题中，按照划分阶段、确定状态、进行决策的方法来进行动态规划：

阶段是：在前n件物品中，选取若干件物品放入背包中；

状态是：在前n件物品中，选取若干件物品放入所剩空间为w的背包中的所最大价值；

决策是：第n件物品放或者不放； 由此可以写出动态转移方程：

用f[i,j]表示在前 i 件物品中选择若干件放在所剩空间为j的背包里所能获得最大价值是：f[i,j] = max{f[i-1,j-wi]+pi(j>=wi),f[i-1,j]}。这样，可以自底向上地得出在前m件物品中取出若干件放进背包能获得的最大价值，也就是f[m,w]令f(i,j)表示用前i个物体装出重量为j的组合时的最大价值，则有：

f(i,j)=max{f(i-1,j), f(i-1, j-w[i])+v[i] } ,i>0, j>=w[i];

f(i,j)= f(i-1,j),i>0, j<w[i];

f(0,j)= v[0] , i=0, j>=w[0];

f(0,j)= 0, i=0, j<w[0]。

由上述例子可知，利用动态规划算法解题可分为以下两个主要步骤：

（1）划分阶段：按照问题的时间或空间特征，把问题分为若干个阶段。注意这若干个阶段一定要是有序的或者是可排序的（即无后向性），否则问题就无法用动态规划求解。

（2）选择状态：将问题发展到各个阶段时所处于的各种客观情况用不同的状态表示出来。当然，状态的选择要满足无后效性。

上一节 本书简介 下一节

第 5 章：算法设计 作者：希赛教育软考学院 来源：希赛网 2014年05月06日

典型真题解析

本节从历年考试真题中，精选出6道典型的试题进行分析，这6道试题所考查的知识点基本上覆盖了本章的所有内容，非常具有代表性。

上一节 本书简介 下一节

第 5 章：算法设计 作者：希赛教育软考学院 来源：希赛网 2015年04月10日

例题1

在一条农村公路的一边稀疏地分布着房子，其分布如图 5-1 所示。某电信公司需要在某些位置放置蜂窝电话基站，由于基站的覆盖范围是 6 公里，因此必须使得每栋房子到某个基站的直线距离

不超过 6 公里。为简化问题，假设所有房子在同一直线上，并且基站沿该直线放置。现采用贪心策略实现用尽可能少的基站覆盖所有的房子。

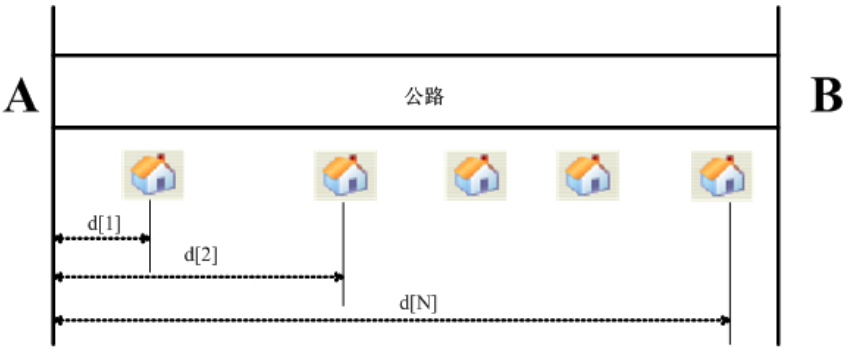

图5-1 基站覆盖图

实现贪心算法的流程如图 5-2 所示，请填充其中空白并计算该算法的时间复杂度，其中：

（1）d[i]（1≤i≤N）表示第 i 个房子到公路 A 端的距离，N 表示房子的总数，房子的编号按照房子到公路 A 端的距离从小到大进行编号。

（2）s[k]表示第 k（k ≥1）个基站到公路 A 端的距离，算法结束后 k 的值为基站的总数。

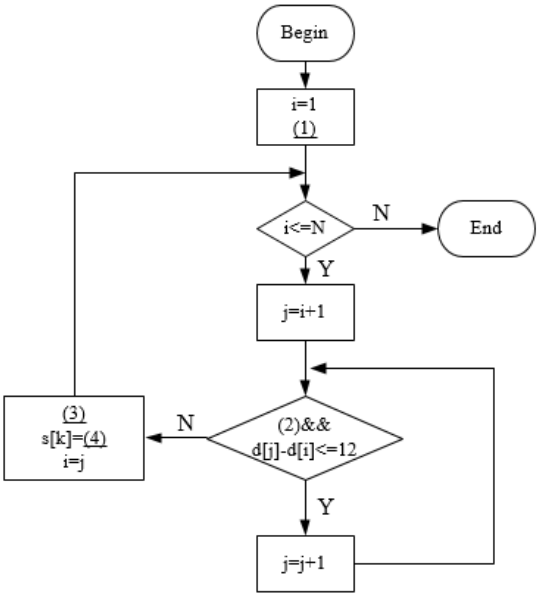

图5-2 贪心法的流程图

该算法的时间复杂度为__（5）__。

例题1分析

该题是一个应用型的算法分析题，主要考查考生对贪心算法的理解以及对程序流程图的掌握，做题的关键是对分析清楚题意，并明确流程图中的贪心条件。

该题可转化为求解能覆盖所有房子的基站部署方案的问题，即通过一系列选择求最优解的问题。不难发现，该问题具有最优子结构，并且具有贪心选择性质，可以用贪心法来求解。

贪心法是一种不求最优解，只求满意解的算法。首先初始化，k=0；若两房间的距离不超过12公立，则不建基站，否则建基站。算法思想：问题的规模为N。从第一个房子（最左端）开始部署基站，把第一个基站放置在该房子右方的6公里处，这时，该基站会覆盖从第一个房子到其右方12公里的直线的长度上的所有房子，假设覆盖了N_1个房子。此时问题规模编程了N-N_1。把第一个基站覆盖的房子去掉，再从N-N_1中选择第一个（最左端）房子开始布局基站，将第二个基站放置在该房子右方的6公里处。以此类推，直至所有的房子被覆盖。

在该算法中包含两个循环，但实际上只是遍历所有房子一次，所以算法的时间复杂度为O(N)。

例题1参考答案

（1）k=0

（2）J<=N

（3）k=k+1 或k++，或者其他等价形式

（4）d[i]+6，或者其他等价形式

（5）O(N)

上一节　本书简介　下一节

例题2

快速排序是一种典型的分治算法。采用快速排序对数组A[p..r]排序的三个步骤如下:

分解：选择一个枢轴（pivot）元素划分数组。将数组A[p..r]划分为两个子数组（可能为空）A[p..q-1]和A[q+1..r]，使得A[q]大于等于A[p..q-1]中的每个元素，小于A[q+1..r]中的每个元素。q的值在划分过程中计算。

递归求解：通过递归的调用快速排序，对子数组A[p..q-1]和A[q+1..r]分别排序。

合并：快速排序在原地排序，故不需合并操作。

【问题1】

下面是快速排序的伪代码，请填补其中的空缺。伪代码中的主要变量说明如下：

A：待排序数组

p, r：数组元素下标，从p到r

q：划分的位置

x：枢轴元素

i：整型变量，用于描述数组下标。下标小于或等于i的元素的值小于或等于枢轴元素的值

j：循环控制变量，表示数组元素下标

```
QUICKSORT( A, p, r ){
if ( p < r ){
q = PARTITION( A,p,r ) ;
QUICKSORT( A, p, q-1 );
QUICKSORT( A, q+1, r );
}
}
PARTITION( A, p, r ){
x = A[r];
i = p - 1;
for ( j = p ; j <=  r - 1; j++ ){
```

if (A[j] <= x){

i = i + 1 ;

交换A[i] 和 A[j];

}

}

交换 （1） 和 （2）

//注：空（1）和空（2）答案可互换，但两空全部答对方可得分

return （3）

}

【问题2】

（1）假设要排序包含n个元素的数组，请给出在各种不同的划分情况下，快速排序的时间复杂度，用O记号。最佳情况为 （4），平均情况为 （5），最坏情况为 （6）。

（2）假设要排序的n个元素都具有相同值时，快速排序的运行时间复杂度属于哪种情况？（7）。（最佳、平均、最坏）

【问题3】

（1）待排序数组是否能被较均匀地划分对快速排序的性能有重要影响，因此枢轴元素的选取非常重要。有人提出从待排序的数组元素中随机地取出一个元素作为枢轴元素，下面是随机化快速排序划分的伪代码—利用原有的快速排序的划分操作，请填充其中的空缺处。其中，RANDOM(i,j)表示随机取i到j之间的一个数，包括i和j。

RANDOMIZED-PARTITION(A,p,r){

i = RANDOM(p,r);

交换（8）和（9）； //注：空（8）和空（9）答案可互换，但两空全部答对方可得分

return PARTITION(A,p,r);

}

（2）随机化快速排序是否能够消除最坏情况的发生？（10）。（是或否）

例题2分析

该题主要考查考生对分治算法的快速排序的理解，对伪代码、快速排序的复杂度的掌握，做题的关键是要读懂题干，理解题干中对算法的描述。

问题1考查的是算法的伪代码表示。分治法的设计思想是将一个难以直接解决的问题，分解成一些规模较小的相同问题，各个击破。其快速排序算法的核心处理是进行划分，根据枢轴元素的值，把一个较大的数组分成两个较小的子数组。一个子数据组的所有元素的值小于等于枢轴元素的值，一个子数组的所有元素的值大于枢轴元素的值，而子数组内的元素不排序。以最后一个元素为枢轴元素进行划分，从左到右依次访问数组的每一个元素，与枢轴元素比较大小，并进行元素的交换。在问题1给出的伪代码中，当循环结束后，A[p..i]中的值小于等于枢轴元素值x，而A[i+1..r-1]中的值应大于x。此时A[i+1]是第一个比A[r]大的元素，于是A[r]与A[i+1]交换，得到划分后的两个子数组。由于划分操作（即PARTITION操作）返回枢轴元素的值，因此返回值为i+1。

问题2考查的是算法的时间复杂度分析。当每次都能做均匀划分时，是算法的最佳情况，其时间复杂度为T(N)=2T(n/2)+O(N)，即时间复杂度为O(nlgn)；算法的最坏情况是每次为极不均匀划分，即长度为n的数组划分后一个子数组为n-1，一个为0，其时间复杂度为T(N)=T(n-1)+O(N

)，即时间复杂度为O(n^2)；算法的平均情况分析起来比较复杂，假设数组每次划分为9/10:1/10，此时时间复杂度可以通过计算得到为O(nlgn)；也就是说在平均情况下快速排序仍然有较好的性能。问题2中假设要排序的n个元素都具有相同值时，快速排序的运行时间复杂度，属于最坏情况，因为每次都划分为长度为n-1和0的两个子数组。

问题3中，由于随机化的快速排序的划分调用了PARTITION操作，而传统划分每次以数组的最后一个元素作为枢轴元素。随机化的快速排序消除了输入数据的不同排列对算法性能的影响，降低了极端不均匀划分的概率，但不能保证不会导致最坏情况的发生。

例题2参考答案

【问题1】

（1）A[i+1] 或 A[r]

（2）A[r] 或 A[i+1]

（3）i+1

【问题2】

（4）O(nlgn) 或 O($nlog_2n$)

（5）O(nlgn) 或 O($nlog_2n$)

（6）O(n^2)

（7）最坏

【问题3】

（8）A[i]

（9）A[r]

（10）否

上一节　　本书简介　　下一节

例题3

现需在某城市中选择一个社区建一个大型超市，使该城市的其它社区到该超市的距离总和最小。用图模型表示该城市的地图，其中顶点表示社区，边表示社区间的路线，边上的权重表示该路线的长度。

现设计一个算法来找到该大型超市的最佳位置：即在给定图中选择一个顶点，使该顶点到其它各顶点的最短路径之和最小。算法首先需要求出每个顶点到其它任一顶点的最短路径，即需要计算任意两个顶点之间的最短路径；然后对每个顶点，计算其它各顶点到该顶点的最短路径之和；最后，选择最短路径之和最小的顶点作为建大型超市的最佳位置。

【问题 1】

本题采用Floyd-Warshall算法求解任意两个顶点之间的最短路径。 已知图G的顶点集合为V={1,2,…,n } ，W= { Wij } n*n 为权重矩阵。设 $d(k)_{ij}$=为从顶点i到顶点j的一条最短路径的权重。当k

= 0时，不存在中间顶点，因此$d(0)_{ij}=w_{ij}$

当k >0 时，该最短路径上所有的中间顶点均属于集合 {1,2, ..., k } 若中间顶点包括顶点 k ，则 $d_{ij}^{(k)}=d_{ik}^{(k-1)}+d_{kj}^{(k-1)}$ ；若中间顶点不包括顶点k ，则 $d_{ij}^{(k)}=d_{ij}^{(k-1)}$ 。

于是得到如下递归式。

$$d_{ij}^{(k)}=\begin{cases} w_{ij} & k=0 \\ \min(\ d_{ij}^{(k-1)}, d_{ik}^{(k-1)}+d_{kj}^{(k-1)}) & k>0 \end{cases}$$

因为对于任意路径，所有的中间顶点都在集合{1,2, ..., n } 内，因此矩阵D(n) = { $d(n)_{ij}$ } n*n 给出了任意两个顶点之间的最短路径，即对所有i, j ∈V，$d(n)_{ij}$表示顶点i到顶点 j的最短路径。

下面是求解该问题的伪代码，请填充其中空缺的（1）至（6）处。 伪代码中的主要变量说明如下：

W：权重矩阵

n： 图的顶点个数

SP：最短路径权重之和数组，SP[i]表示顶点i到其它各顶点的最短路径权重之和，i从1到n

min_SP：最小的最短路径权重之和

min_v：具有最小的最短路径权重之和的顶点

i：循环控制变量

j：循环控制变量

k：循环控制变量

```
LOCATE -SHOPPINGMALL( W, n )
1   D( 0 )=W
2   for   （1）
3         for i = 1 to n
4               for j = 1 to n
5                 if d( k-1 )ij≤≤d( k-1 )ik+d( k-1 )kj
6                       （2）
7                  else
8                       （3）
9   for i = 1 to n
10        SP[i] = 0
11  for j = 1 to n
12        （4）
13  min_SP = SP[1]
14   （5）
15  for i = 2 to n
16        if min_SP > SP[i]
17              min_SP = SP[i]
18              min_v = i
19 return   （6）
```

【问题2】【问题1】中伪代码的时间复杂度为 （7） （用O符号表示）。

例题3分析

本题考查了Floyd-Warshall算法的伪代码表示以及算法的时间复杂度。

问题1的第（1）空表示主循环，k是循环控制变量，故第（1）空为k = 1 to n。第（2）和第（3）根据递归式可分别得到答案为$d_{ij}^{(k)} = d_{ij}^{(k-1)}$和$d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$。计算了任意两个顶点之间的最短路径之后，对每个顶点，开始统计其到所有其他顶点的最短路径之和，因此第（4）空填$\mathrm{SP[i]} = \mathrm{SP[i]} + d_{ij}^{(n)}$。第13和第14行初始化，假设最小的到所有其他顶点的最短路径之和为第一个顶点的最小路径之和，大型超市的最佳位置为第一个顶点，所以第（5）空填min_v = 1。最后要求返回大型超市的最佳位置，即所有其他顶点的最短路径之和最小的顶点，所以第（6）空填min_v。

问题2考查的是问题1中的伪代码的时间复杂度。第2~8行是计算任意两点之间的最短路径，有三重循环，时间复杂度为O(n^3)。第9~12行有两重循环，时间复杂度为O(n^2)。第15~18行，时间复杂度为O(n)。所以算法总的时间复杂度为O(n^3)。

例题3参考答案

【问题1】

（1）k = 1 to n

（2）$d_{ij}^{(k)} = d_{ij}^{(k-1)}$

（3）$d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$

（4）$\mathrm{SP[i]} = \mathrm{SP[i]} + d_{ij}^{(n)}$

（5）min_v = 1

（6）min_v

【问题2】

（7）O(n^3)

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年05月06日

例题4

0-1背包问题可以描述为：有n个物品，对i = 1, 2, …, n，第i个物品价值为vi，重量为wi（vi和wi为非负数），背包容量为W（W为非负数），选择其中一些物品装入背包，使装入背包物品的总价值最大，即，且总重量不超过背包容量， 即，其中，xi=0表示第i个物品不放入背包，xi=1表示第i个物品放入背包。

【问题1】

用回溯法求解此0-1背包问题，请填充下面伪代码中（1）~（4）处空缺。

回溯法是一种系统的搜索方法。在确定解空间后，回溯法从根结点开始，按照深度优先策略遍历解空间树，搜索满足约束条件的解。对每一个当前结点，若扩展该结点已经不满足约束条件，则

不再继续扩展。为了进一步提高算法的搜索效率，往往需要设计一个限界函数，判断并剪枝那些即使扩展了也不能得到最优解的结点。现在假设已经设计了BOUND(v,w,k,W)函数，其中v、w、k和W分别表示当前已经获得的价值、当前背包的重量、已经确定是否选择的物品数和背包的总容量。对应于搜索树中的某个结点，该函数值表示确定了部分物品是否选择之后，对剩下的物品在满足约束条件的前提下进行选择可能获得的最大价值，若该价值小于等于当前已经得到的最优解，则该结点无需再扩展。

下面给出0-1背包问题的回溯算法伪代码。

函数参数说明如下:

W：背包容量；n：物品个数；w：重量数组；v：价值数组；fw：获得最大价值时背包的重量；fp：背包获得的最大价值；X：问题的最优解。

变量说明如下：

cw：当前的背包重量；cp：当前获得的价值；k：当前考虑的物品编号；Y：当前已获得的部分解。

```
BKNAP( W,n,w,v,fw,fp,X )
1 cw ← cp ← 0
2 （1）
3 fp ← -1
4 while true
5     while k ≤ n and cw + w[k] ≤ W do
6         （2）
7         cp ← cp + v[k]
8         Y[k] ← 1
9         k ← k + 1
10    if k > n then
11if fp < cp then
12        fp ← cp
13            fw ← cw
14            k ← n
15            X ← Y
16    else Y( k ) ← 0
17    while BOUND( cp,cw,k,W ) ≤ fp do
18        while k ≠ 0 and Y( k ) ≠ 1 do
19            （3）
20        if k = 0 then return
21        Y[k] ← 0
22        cw ← cw - w[k]
23        cp ← cp - v[k]
24 （4）
```

【问题2】

考虑表5-2的实例，假设有3个物品，背包容量为22。图5-3中是根据上述算法构造的搜索树，其中结点的编号表示了搜索树生成的顺序，边上的数字1/0分别表示选择/不选择对应物品。除了根结点之外，每个左孩子结点旁边的上下两个数字分别表示当前背包的重量和已获得的价值，右孩子结点旁边的数字表示扩展了该结点后最多可能获得的价值。为获得最优解，应该选择物品（5），获得的价值为（6）。

表5-2 0-1背包问题实例

	物品 1	物品 2	物品 3
重量	15	10	10
价值	30	18	17
单位价值	2	1.8	1.7

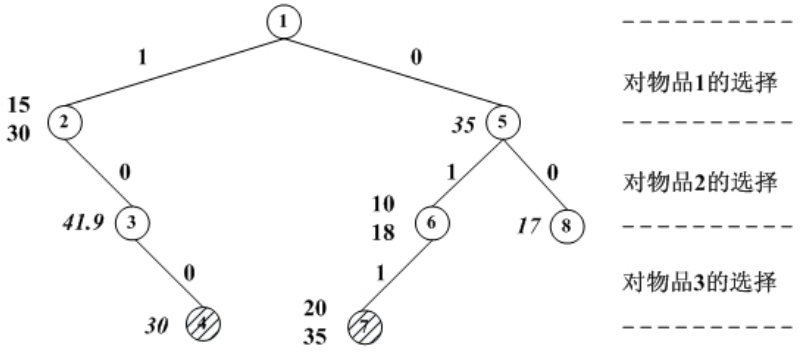

图5-3 表5-2实例的搜索树

对于表5-2的实例，若采用穷举法搜索整个解空间，则搜索树的结点数为（7），而用了上述回溯法，搜索树的结点数为（8）。

例题4分析

本题考查了用回溯法算法设计技术解决0-1背包问题。回溯法有两类算法框架：非递归形式和递归形式，本题采用的是非递归形式表示。该题目的解题要点是理解回溯法的基本思想和这两类算法框架。回溯法有“通用的解题法”之称，用它可以系统地搜索一个问题的所有解或任一解，是一种既带有系统性又带有跳跃性的搜索算法。它在包含问题的所有解得解空间树中，按照深度优先的策略，从根节点出发搜索空间树。算法搜索至空间树的任一节点时，总是先判断该节点是否可定不包含问题的解。如果肯定不包含，则跳过对以该节点为根的子树的系统搜索，逐层向其祖先接都回溯。

回溯法从第一项物品开始考虑是否应该装入包中，因此当前考试的物品编号k从1开始，即k←1。然后逐项往后检查，若能全部放入背包则将该项放入背包，此时背包的重量应该是当前的重量加上当前考虑物品的重量，即cw←cw+w[k]，当然背包中物品的价值也为当前的价值加上当前考虑物品的价值。若已经考虑完了所有的物品，则得到一个解，判断该解是否为当前最有，如果是最优，就把该解的信息放入变量fp、fw和X中。如果还没考虑完所有的物品，那意味着有些物品不能放入背包，此时先判断若不将当前的物品放入背包中，则其余物品放入背包是否可能得到比当前最优解更优的解，若得不到就回溯；否则需要继续考虑其余的物品。

根据问题1中给出的伪代码，比较容易得到此0-1背包问题的最优解，应该选择物品2和物品3，此时背包的重量为10+10=20，获得的价值为17+18=35。

如果采用穷举法搜索整个解空间，就构造成一颗完全二叉树，此时搜索树的节点数应为2^4-1=15，采用了回溯法后，搜索树的节点数仅为8个。

例题4参考答案

【问题1】

（1）k ← 1　　或其等价形式

（2）cw ← cw + w[k]　或其等价形式

（3）k ← k-1　　或其等价形式

（4）k ← k+1　　或其等价形式

【问题2】

（5）物品2和物品3

（6）35

（7）15

（8）8

　作者：希赛教育软考学院　来源：希赛网　2014年05月06日

例题5

对有向图进行拓扑排序的方法是：

（1）初始时拓扑序列为空；

（2）任意选择一个入度为0的顶点，将其放入拓扑序列中，同时从图中删除该顶点以及从该顶点出发的弧；

（3）重复（2），直到不存在入度为0的顶点为止（若所有顶点都进入拓扑序列则完成拓扑排序，否则由于有向图中存在回路无法完成拓扑排序）。

函数int* TopSort(LinkedDigraph G)的功能是对有向图G中的顶点进行拓扑排序，返回拓扑序列中的顶点编号序列，若不能完成拓扑排序，则返回空指针。其中，图G中的顶点从1开始依次编号，顶点序列为v1，v2，…，vn，图G采用邻接表示，其数据类型定义如下：

```
#define MAXVNUM 50                    /*最大顶点数*/
typedef struct ArcNode{               /*表结点类型*/
    int adjvex;                       /*邻接顶点编号*/
    struct ArcNode *nextarc;          /*指示下一个邻接顶点*/
}ArcNode;
typedef struct AdjList{               /*头结点类型*/
    char vdata;                       /*顶点的数据信息*/
    ArcNode *firstarc;                /*指向邻接表的第一个表结点*/
}AdjList;
typedef struct LinkedDigraph{         /*图的类型*/
    int n;                            /*图中顶点个数*/
    AdjList Vhead[MAXVNUM];           /*所有顶点的头结点数组*/
}LinkedDigraph;
```

例如，某有向图G如图5-4所示，其邻接表如图5-5所示。

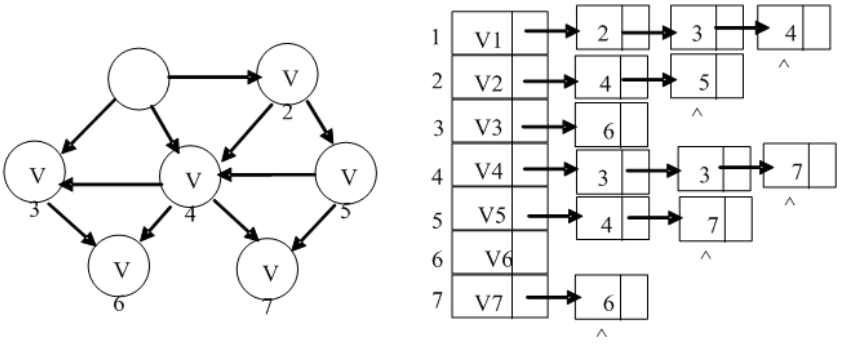

图5-4 图G及其邻接表

函数TopSort中用到了队列结构（Queue的定义省略），实现队列基本操作的函数原型如表5-3所示：

表5-3 实现队列基本操作的函数原型

函数原型	说明
Void InitQueue(Queue *Q)	初始化队列（构造一个空队列）
bool IsEmpty(Queue *Q)	判断队列是否为空，若是则返回 true，否则返回 false
Void EnQueue(Queue *Q)	元素入队列
Void DeQueue(Queue *Q)	元素出队列

【C代码】

```
int *TopSort( LinkedDigraph G )
{
/* p临时指针，指示表结点
  Q临时队列，保存入度为0的顶点编号
  k临时变量，用作数组元素的下标
  j,w临时变量，用作顶点编号
  topOrder存储拓扑序列中的顶点编号
  inDegree存储图G中各顶点的入度
*/
 ArcNode *p;
 Queue Q;
 int k = 0;
 int j = 0, w = 0;
 int *topOrder, *inDegree;
 topOrder =( int * )malloc( ( G.n+1 )* sizeof( int ) );
 inDegree =( int * )malloc( ( G.n+1 )* sizeof( int ) );
 if( !inDegree || !topOrder ) return NULL;
    /*构造一个空队列*/
   （1）;
 for ( j = 1; j <= G.n; j++ ) {
    /*初始化*/
     topOrder[j] = 0;  inDegree[j] = 0;  }
 for ( j = 1; j <= G.n; j++ )
       /*求图G中各顶点的入度*/
      for( p = G.Vhead[j].firstarc; p; p = p->nextarc )
```

```
            inDegree[p-> adjvex] += 1;
    for ( j = 1; j <= G.n; j++ )
        /*将图G中入度为0的顶点保存在队列中*/
        if ( 0 == inDegree[j] ) EnQueue( &Q,j );
    while ( !IsEmpty( Q ) ) {
    /*队头顶点出队列并用w保存该顶点的编号*/
    （2）;
    topOrder[k++] = w;
      /*将顶点w的所有邻接顶点的入度减1
       （模拟删除顶点w及从该顶点出发的弧的操作）*/
    for( p = G.Vhead[w].firstarc; p; p = p->nextarc ) {
    （3） = 1;
         if ( 0 ==（4） ) EnQueue( &Q, p->adjvex );   }/* for */
    }/* while */
    free( inDegree );
    if （ （5） ）
    return NULL;
    return topOrder;
    } /*TopSort*/
```

【问题1】

根据以上说明和C代码，填充C代码中的空（1）～（5）。

【问题2】

对于图5-4所示的有向图G，写出函数TopSort执行后得到的拓扑序列。若将函数TopSort中的队列改为栈，写出函数TopSort执行后得到的拓扑序列。

【问题3】

设某有向无环图的顶点个数为n、弧数为e，那么用邻接表存储该图时，实现上述拓扑排序算法的函数TopSort的时间复杂度是（6）。

若有向图采用邻接矩阵表示（例如，图5-4所示有向图的邻接矩阵如图5-5所示），且将函数TopSort中有关邻接表的操作修改为针对邻接矩阵的操作，那么对于有n个顶点、e条弧的有向无环图，实现上述拓扑排序算法的时间复杂度是（7）。

	v1	v2	v3	v4	v5	v6	v7
v1	0	1	1	1	0	0	0
V2	0	0	0	1	1	0	0
V3	0	0	0	0	0	1	0
V4	0	0	1	0	0	1	1
V5	0	0	0	1	0	0	1
V6	0	0	0	0	0	0	0
V7	0	0	0	0	0	1	0

图5-5 有向图G的临接矩阵

例题5分析

该题并不是典型的针对于算法设计知识点的考题，它重点考查了C语言程序设计。该题和算法设

计有关的知识点主要在于：它通过有向图的拓补排序考查队列的定义与基本运算、C语言指针的操作，并牵出队列与栈的区别以及两种存储结构的拓补排序时间复杂度的度量。队列的特点“先进先出”，而栈的特点是“先进后出”。

算法的时间复杂度是指算法需要消耗的时间资源。一般情况下，算法的基本操作重复执行的次数是模块n的某一个函数f(n)，因此，算法的时间复杂度记做：T(n)=O(f(n))。

在计算时间复杂度的时候，先找出算法的基本操作，然后根据相应的各语句确定它的执行次数，再找出T(n)的同数量级（数量级通常有以下标准：1，Log_2n，n ，$nLog_2n$ ，n的平方，n的三次方，2的n次方，n！），找出后，f(n)=该数量级，若T(n)/f(n)求极限可得到一常数c，则时间复杂度T(n)=O(f(n))。

例题5参考答案

【问题1】

（1）InitQueue(&Q)

（2）DeQueue(&Q, &w)

（3）inDegree[p->adjvex]

（4）inDegree[p->adjvex]

（5）k!=G.n

【问题2】

函数TopSort执行后得到的拓扑队列：V1 V2 V5 V4 V3 V7 V6

改为栈后得到的拓扑队列：V1 V2 V5 V4 V7 V3 V6

【问题3】

（6）O(n)

（7）O(n^2)

上一节　本书简介　下一节

第 5 章：算法设计　作者：希赛教育软考学院　来源：希赛网　2014年05月06日

考前必练

本小节精选了历年考试中的4道真题和1道“背包问题”（涉及两种模式：递归解法和非递归解法）作为考前必练的题目，内容涵盖了回溯法、动态规划和贪心法等三种常用的算法设计和时间复杂度分析等考核点。

上一节　本书简介　下一节

第 5 章：算法设计　作者：希赛教育软考学院　来源：希赛网　2014年05月06日

试题1

考前必做的练习题“背包问题”的基本描述为：有不同重量的N件物品，其重量分别为w1,w2,...,wn，希望从中选择若干件物品，所选物品的重量之和恰能放入能盛放总重量为S的背包中，即所选重量之和等于S。

下面的C程序（程序代码1和程序代码2），均能求得“背包问题”的一组解，其中程序代码1是递归解法，而程序代码2是非递归解法。（注：仅在空白处填写相应的表达式或语句，请勿修改程序的其他内容）。

【C程序】

代码1：

```
#include <stdio.h>
#define N 7
#define S 15
int w[ N + 1 ] = {0,1,4,3,4,5,2,7};
int knap( int s , int n )
{
 if ( s = = 0 ) return 1;
 if ( ( s < 0 )||( s > 0 && n < 1 ) ) return 0;
 if (  ( 1 )  ) {
      printf ( "%4d" , w[n] );
      return 1;
 }
 return  ( 2 ) ;
}
main ( )
{
 if ( knap( S , N ) printf( "OK! \n" );
 else printf( "NO! \n" );
}
```

代码2：

```
#include <stdio.h>
#define N 7
#define S 15
typedef struct {
int s;
int n;
int job;
} KNAPTP;
int w[ N + 1 ] = {0,1,4,3,4,5,2,7};
int knap( int s,int n );
```

```
main( ){
if ( ( knap( S,N ) )
   printf( "OK! \n" );
else printf( "NO! \n" ); }
int knap( int s,int n ){
KNAPTP stack[100],x;
int top,k,rep;
x.s = s;
x.n = n;
x.job = 0;
top = 1;
stack[top] = x;
k = 0;
while( __( 3 )__ && !k ){
x = stack [top];
     rep = 1;
while ( !k&&rep ){
if ( x.s = = 0 ) k=1;      /*已求得一组解*/
         else  if( x.s <0 || x.n <= 0 )  rep=0;
 else {
   x.s =  __( 4 )__ ;
x.job = 1;
stack[++top] = x;
 }
}
if (!k )
{
 rep = 1;
 while ( top >= 1 && rep ){
  s = stack[top--];
  if( x.job = = 1 ){
x.s += w[x,n+1];
x.job = 2;
__( 5 )__  = x;
rep = 0;
}
  }
   }
}
```

```
if ( k )      /*输出一组解*/
{
 while( top >= 1 ){
 x = stack[top--];
 if( x.job = = 1 )
 printf( "%d\t",w[x,n+1] );
 }
}
return k;
}
```


试题2

假设需要将N个任务分配给N个工人同时去完成，每个人都能承担这N个任务，但费用不同。下面的程序用回溯法计算总费用最小的一种工作分配方案，在该方案中，为每个人分配1个不同的任务。

程序中，N个任务从0开始依次编号，N个工人也从0开始依次编号，主要的变量说明如下：

c[i][j]：将任务i分配给工人 j的费用；

task[i]：值为0表示任务i未分配。值为j表示任务I分配给工人j；

worker[k]：值为0表示工人k未分配任务，值为1表示工人k已分配任务；

mincost：最小总费用。

【C程序】

```
#include <stdio.h>
#define N 8 /*N表示任务数和工人数*/
int c[N][N];
unsigned int mincost = 65535; /*设置min的初始值，大于可能的总费用*/
int task[N],temp[N],worker[N];
void plan( int k,unsigned int cost )
{
int i;
if ( ____( 1 )____&& cost < mincost ){
mincost = cost;
for ( i = 0;i<N;i++ ) temp[i] = task [ i ];
}else {
for ( i = 0 ; i<N; i ++ )     /*分配任务k*/
```

```
if ( worker [i]= =0 &&  （2）  ){
worker [ i ] = 1;task [ k ]=  （3）  ;
plan(  （4）  ,cost + c [ k ] [ i ] );
 （5）  ; task [k] = 0;
} /*if*/
}
} /*Plan*/
void main ( )
{
int i,j;
for( i = 0;i < N;i ++ ){  /*设置每个任务由不同工人承担时的费用及全局数组的初值*/
worker [ i ] = 0; task [ i ] = 0; temp [ i ] = 0;
for ( j = 0 ; j < N ; j ++ )
scanf (  "%d" ,&c[ i ] [ j ] );
}
plan ( 0,0 );  /*从任务0开始分配*/
printf(  "\n最小费用 = %d\n" ,mincost );
for( i = 0; i < N; i ++ )
printf(  "Task%d is assigned to Worker%d\n" ,i,temp[ i ] );
}/*main*/
```


上一节　本书简介　下一节

第 5 章：算法设计　　作者：希赛教育软考学院　来源：希赛网　2014年05月06日

试题3

某汽车制造工厂有两条装配线。汽车装配过程如图5-6所示，即汽车底盘进入装配线，零件在多个工位装配，结束时汽车自动完成下线工作。

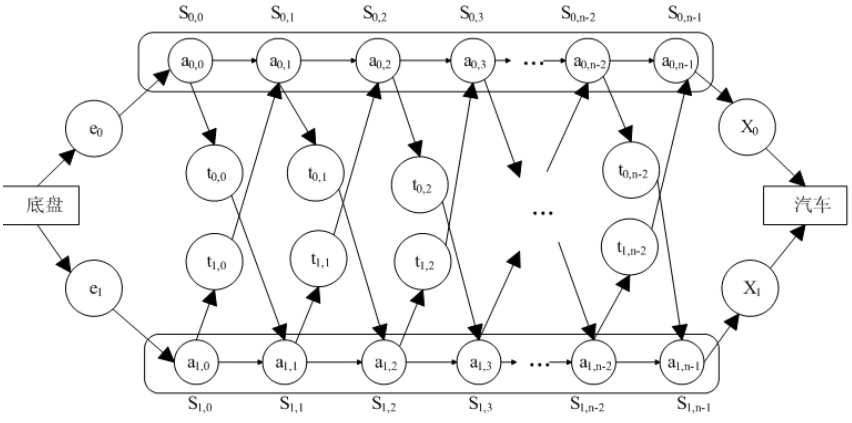

图5-6 汽车装配过程图

（1）e_0和e_1，表示底盘分别进入装配线0和装配线1所需要的时间。

（2）每条装配线有n个工位，第一条装配线的工位为$S_{0,0}$，$S_{0,1}$，…，$S_{0,n-1}$，第二条装配线的工位为$S_{1,0}$ ，$S_{1,1}$,…，$S_{1,n-1}$ 。其中S_0,k 和S_1,k （0≤k≤n-1）完成相同的任务，但所需时间可能不同。

（3）$a_{i,j}$ 表示在工位$S_{i,j}$处的装配时间，其中i表示装配线（i=0或i=1），j表示工位号（0≤k≤n-1）。

（4）$t_{i,j}$ 表示从$S_{i,j}$ 处装配完成后转移到另一条装配线下一个工位的时间。

（5）x_0和x_1表示装配结束后，汽车分别从装配线0和装配线1下线所需要的时间。

（6）在同一条装配线上，底盘从一个工位转移到其下一个工位的时间可以忽略不计。图5-7所示的流程图描述了求最短装配时间的算法，该算法的输入为：

n：表示装配线上的工位数；e [i] ：表示e_1 和e_2，i取值为0或1；

a[i][j]：表示$a_{i,j}$，i的取值为0或1，j的取值范围为0～n-1；

t[i][j]：表示$t_{i,j}$ ，i的取值为0或1，j的取值范围为0～n-1；

x[i]：表示$x_{a,b}$ 和x_{1} ,i取值为0或1。

算法的输出为：

F_i：最短的装配时间；

L_i：获得最短装配时间的下线装配线号（0或者1）。

算法中使用的f[i][j]表示从开始点到$S_{i,j}$处的最短装配时间。

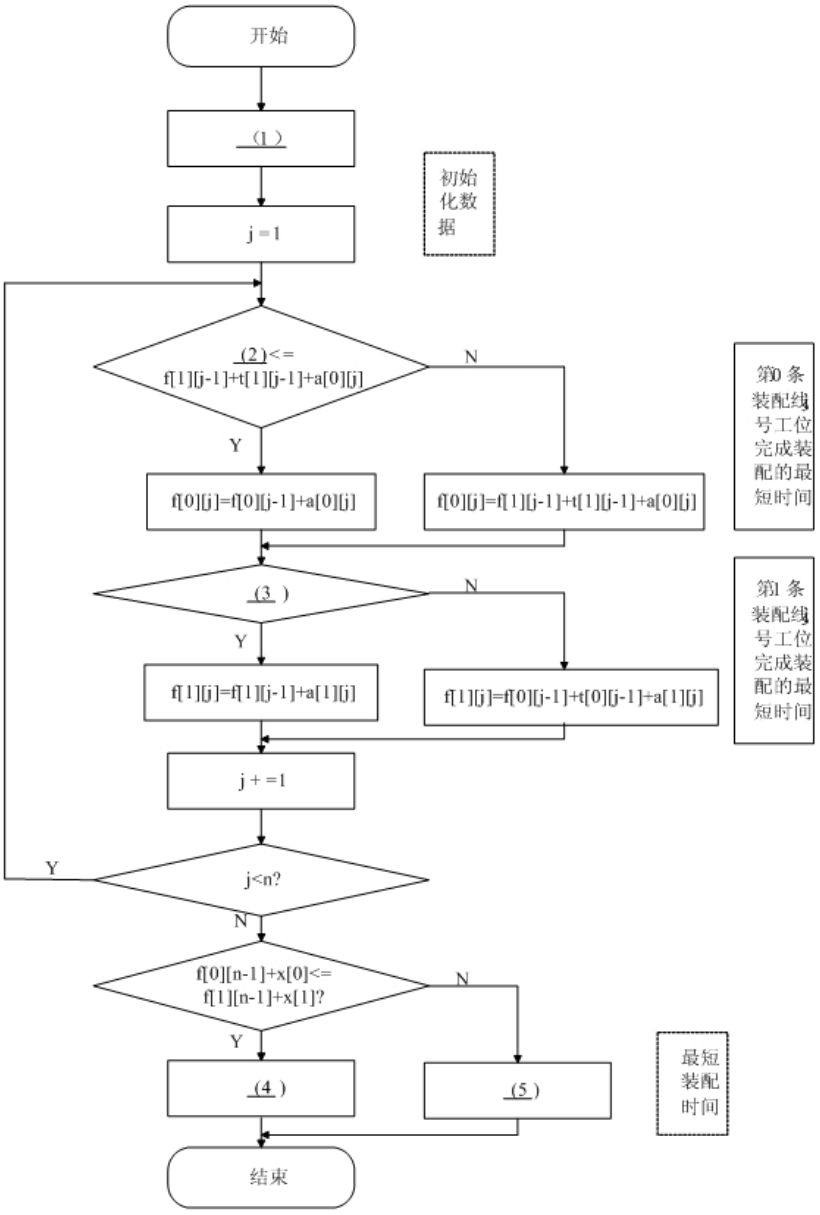

图5-7 程序流程图

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年05月06日

试题4

某机器上需要处理n个作业job1, job2, …, jobn，其中：

（1）每个作业jobi（1≤i≤n）的编号为i，jobi有一个收益值p[i]和最后期限值d[i]；

（2）机器在一个时刻只能处理一个作业，而且每个作业需要一个单位时间进行处理，一旦作业开始就不可中断，每个作业的最后期限值为单位时间的正整数倍；

（3）job1～jobn的收益值呈非递增顺序排列，即p[1]≥p[2]≥…≥p[n]；

（4）如果作业jobi在其期限之内完成，则获得收益p[i]；如果在其期限之后完成，则没有收益。

为获得较高的收益，采用贪心策略求解在期限之内完成的作业序列。图5-8是基于贪心策略求解

该问题的流程图。

（1）整型数组J[]有n个存储单元，变量k表示在期限之内完成的作业数，J[1..k]存储所有能够在期限内完成的作业编号，数组J[1..k]里的作业按其最后期限非递减排序，即d[J[1]]≤ … ≤d[J[k]]。

（2）为了方便在数组J中加入作业，增加一个虚拟作业job0，并令d[0] = 0，J[0] = 0。

（3）算法大致思想：先将作业job1的编号1放入J[1]，然后，依次对每个作业jobi（2≤i≤n）进行判定，看其能否插入到数组J中，若能，则将其编号插入到数组J的适当位置，并保证J中作业按其最后期限非递减排列，否则不插入。

jobi能插入数组J的充要条件是：jobi和数组J中已有作业均能在其期限之内完成。

（4）流程图中的主要变量说明如下：

i：循环控制变量，表示作业的编号；

k：表示在期限内完成的作业数；

r：若jobi能插入数组J，则其在数组J中的位置为r+1；

q：循环控制变量，用于移动数组J中的元素。

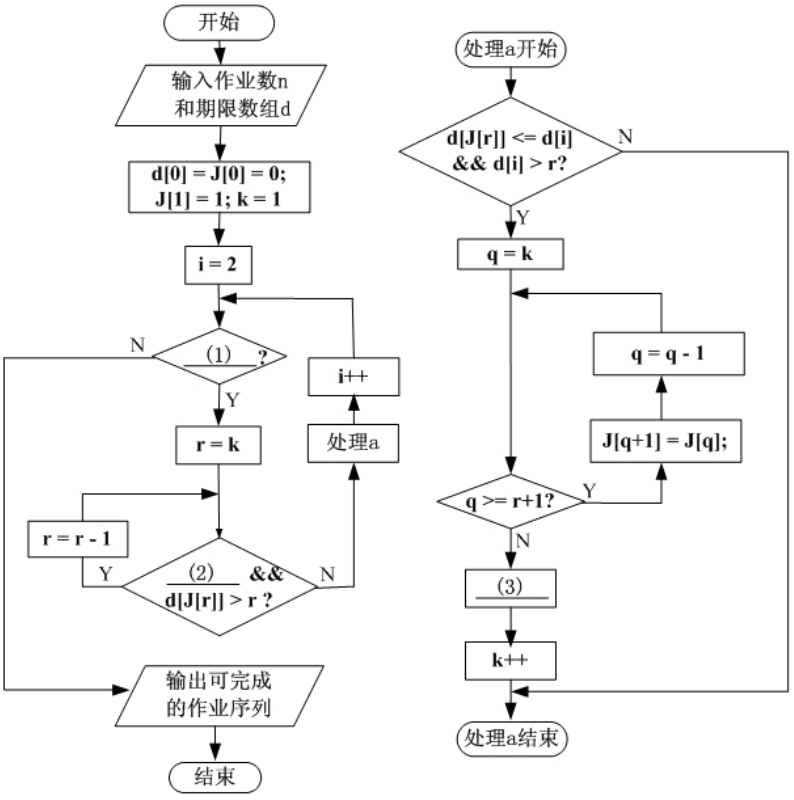

图5-9 贪心策略流程图

【问题1】

请填充图5-8中的空缺（1）、（2）和（3）处。

【问题2】

假设有6个作业job1, job2, …, job6；

完成作业的收益数组p =（p[1],p[2],p[3],p[4],p[5],p[6]）=（90,80,50,30,20,10）；

每个作业的处理期限数组d =（d[1],d[2],d[3],d[4],d[5],d[6]）=（1,2,1,3,4,3）。

请应用试题中描述的贪心策略算法，给出在期限之内处理的作业编号序列（4）（按作业处理的顺序给出），得到的总收益为（5）。

【问题3】

对于本题的作业处理问题，用图5-8的贪心算法策略，能否求得最高收益？（6）。用贪心

算法求解任意给定问题时，是否一定能得到最优解？（7）。

第 5 章：算法设计　作者：希赛教育软考学院　来源：希赛网　2014年05月06日

试题5

某餐厅供应各种标准的营养套餐。假设菜单上共有n项食物m1，m2，…，mn，每项食物mi的营养价值为vi，价格为pi，其中i=1,2,…,n，套餐中每项食物至多出现一次。客人常需要一个算法来求解总价格不超过M的营养价值最大的套餐。

【问题1】

下面是用动态规划策略求解该问题的伪代码，请填充其中的空缺（1）、（2）和（3）处。

伪代码中的主要变量说明如下：

n： 总的食物项数；

v： 营养价值数组，下标从1到n，对应第1到第n项食物的营养价值；

p： 价格数组，下标从1到n，对应第1到第n项食物的价格；

M：总价格标准，即套餐的价格不超过M；

x： 解向量（数组），下标从1到n，其元素值为0或1，其中元素值为0表示对应的食物不出现在套餐中，元素值为1表示对应的食物出现在套餐中；

nv：n+1行M+1列的二维数组，其中行和列的下标均从0开始，nv[i][j]表示由前i项食物组合且价格不超过 j 的套餐的最大营养价值。问题最终要求的套餐的最大营养价值为nv[n][M]。

伪代码如下：

```
MaxNutrientValue( n, v, p, M, x )
1    for i = 0 to n
2         nv[i][0] = 0
3    for j = 1 to M
4         nv[0][j] = 0
5    for i = 1 to n
6         for j = 1 to M
7              if j < p[i]    //若食物mi不能加入到套餐中
8                   nv[i][j] = nv[i - 1][j]
9              else if      (1)
10                  nv[i][j] = nv[i - 1][j]
11             else
12                  nv[i][j] = nv[i - 1][j – p[i]] + v[i]
13   j = M
14   for i = n downto 1
15        if    (2)
```

```
16              x[i] = 0
17          else
18              x[i] = 1
19              （3）
20    return x and nv[n][M]
```

【问题2】

现有5项食物，每项食物的营养价值和价格如表5-4所示。

表5-4 食物营养价值及价格表

编码	营养价值	价格
m1	200	50
m2	180	30
m3	225	45
m4	200	25
m5	50	5

若要求总价格不超过100的营养价值最大的套餐，则套餐应包含的食物有（4）（用食物项的编码表示），对应的最大营养价值为（5）。

【问题3】

问题1中伪代码的时间复杂度为（6）（用O符号表示）。

上一节　本书简介　下一节

第 5 章：算法设计　作者：希赛教育软考学院　来源：希赛网　2014年05月06日

练习题解析

试题1分析

试题提供了两种解决“背包问题”的方法，程序代码1是用递归的方法，程序代码2是使用非递归的方法。首先，我们分析下采用递归的方法，每次选择一个物品放入背包，则剩余的物品和背包剩余的重量，又构成一个“背包问题”。

程序从数组下表最大的物品开始，因此（1）处填“knap(s -w[n]，n-1)”，即将数组中第N个物品放入背包，如果它能够被放入，则该物品时构成解的元素之一；否则，该物品从背包中取出，不构成解的元素。因此，（2）处填“knap(s，n-1)”。在采用非递归的算法时，结构KNAPTP表示经过考查的物品，s表示考查过物品后背包所能盛放的物品的重量；n表示该物品在数组W中的下标；job表示物品当前的状态：当job等于1时，表示物品n可以放入背包；当job等于2时表示物品n不能放入背包，以后的选择将不再考虑该物品。初始时job等于0，表示背包中没有放入任何物品。K为有解的标志。

该物品满足放入背包的条件，第（4）和（5）空将完成将物品放入背包的操作，因此（4）处填“x.s - w[x.n--]”，修改背包的可容纳物品的重量；（5）处填“stack[++top]”，将下一个要考查的物品放入栈中。若物品不满足放入背包的条件，则将物品从背包中取出，即将rep置为0，结束循环while(!k&&rep)。第（3）空要求给出可以继续选择物品的条件，在此处填“top >= 1”。本题的解题要点是熟悉背包问题的本质，掌握递归法的综合运用，并分析清楚每个循环的目的以及结

束条件。

试题1参考答案

（1）knap(s -w[n] , n-1)

（2）knap(s , n-1)

（3）top >= 1

（4）x.s - w[x.n--]

（5）stack[++top]

试题2分析

根据题目中的说明，程序用回溯法计算总费用最小的一种工作分配方案，因此在得到每一个分配方案时需要和先前得到的分配方案中的最小费用进行比较。通过程序注释可知，在plan(int k,unsigned int cost)中，k表示任务编号、cost表示费用。

由于需要将N个任务分配给N个工人，以任务为序时，最后一个任务即第N-1个任务分配之后便得到一种方案。因此空（1）处应填入“k>=N ”，或“k == N”。

同时，在分配任务k时，需要考查所有工人的情况，此时如果工人i尚未接收任务(worker[i] = = 0)，并且将任务 k 分配给工人i不超出前面某个方案的费用，则可将任务分配给他(task[k] = i)，再分配第k+1个任务。回溯时需要撤销工人i的任务，以便与其他的分配方案做比较。本题的解题要点是掌握回溯法算法设计的基本原理。

试题2参考答案

（1）k>=N 或k = = N或其他等价形式

（2）cost + c[k][i] < mincost或其他等价形式

（3）i

（4）k+1

（5）worker[i] = 0或其他等价形式

试题3分析

该题用流程图表示了动态规划算法。在问题有两个特性，即最优子结构和重叠子结构时，可以考虑用动态规划求解问题。动态规划算法解题有以下四个步骤：

（1）刻画问题的最优子结构，描述问题的最优解包含子问题的最优解。以本题为例，最短装配时间等于经过装配线0的第n个恭维的最短装配时间加上x[0]，或者等于经过装配线1的第n个恭维的最短装配时间加上x[1]，取哪条装配线取决于哪个值更小。而经过某个装配线0/1的第i个工位的最短装配时间又等于经过装配线0/1的第i-1个工位的最短装配时间，或者等于经过装配线1/0的第i-1个工位的最短装配时间加上从这个工位到装配线0/1的迁移时间，取决于哪个值更小。

（2）建立最优子结构的递归关系，这一步是关键。对本题而言，递归关系为：

$$f(0,j)=\begin{cases} e0+a(0,0) & j=0 \\ \min(f(0,j-1)+a(0,j),f(1,j-1)+t(1,j-1)+a(0,j)) & j>0 \end{cases}$$

$$f(1,j)=\begin{cases} e1+a(1,0) & j=0 \\ \min(f(1,j-1)+a(1,j),f(0,j-1)+t(0,j-1)+a(1,j)) & j>0 \end{cases}$$

（3）根据递归关系求最优解的值。对于本题来说，最优解记录在fi中，fi=min(f(0,n-1)+x_0,f(1,n-1)+x_1);

（4）构造最优解，对本题而言，只是求出最优解是从哪条装配线装配出来，并没有记录最优解。

在仔细阅读说明后，根据流程图，不难得出求最短装配时间的算法。f[i][j]里实际存储的是第0/1条装配线上第i（i > 1且i <= n-1）个工位完成装配所需的最短时间。则可知第（1）空的内容：f[0][0] = e[0] +a[0][0] ,f[1][0] = e[1] +a[1][0]；

第（2）和（3）空是一个最优路径的选择问题，需要做出判断，则可知：

（2）f[0][j-1] + a[0][j]

（3）f[1][j-1] + a[1][j] <= f[0][j-1]+t[0][j-1]+a[1][j]

对于第（4）和第（5）空提示比较明显，就是要对输出结果赋值。

（4）fi = f[0][n-1] + x [0]

li =0

（5）fi = f[1][n-1] + x [1]

li =1

试题3参考答案

（1）f[0][0] = e[0] +a[0][0]

f[1][0] = e[1] +a[1][0]

（2）f[0][j-1] + a[0][j]

（3）f[1][j-1] + a[1][j] <= f[0][j-1]+t[0][j-1]+a[1][j] 或其他等价形式

（4）fi = f[0][n-1] + x [0]

li =0

（5）fi = f[1][n-1] + x [1]

li =1

试题4分析

本题考查的是贪心算法的流程图表示以及贪心算法策略。

问题1考查的是贪心算法的流程图。第（1）空表示第2个作业到第n个作业的主循环，i是循环控制变量，因此第（1）空填入 i <= n。

数组J中的作业J[i]（1<=i<=k）是期限之前完成的作业，且d[J[i]]<=d[J[i+1]]（1<=i<k）。主循环内嵌套了两个循环，第一个循环判断当前的作业i插入到J中的位置，循环控制变量r表示当前考虑的J中的作业。为保证J中的作业期限按升序排序，作业J[r]如果 比作业i的期限大，那么循环控制变量r需要自减，因此第（2）空填入D[j[r]] > d[i]。本算法的第二个循环的作用为使作业依次向后移动，采用了插入排序的思想，最后把作业i插入到J[r+1]处，所以第（3）空填入J[r+1] = i。

问题2是本题算法的一个实例。6个作业的收益已经按降序排好序。根据流程图，将作业1，2，4，5依次放入数组J中，总收益为220，具体过程参见表5-5所示。

表5-5 作业执行的具体过程

J数组	收益	作业	期限	操作
——	0	Job1	1	放入J中
1	90	Job2	2	放入J中
1,2	170	Job3	1	不放入J中
1,2	170	Job4	3	放入J中
1,2,4	200	Job5	4	放入J中
1,2,4,5	220	Job6	3	不放入J中
1,2,4,5	220			

问题3考查算法策略。贪心法在该题能够求得最优解。但是，不是所有的问题都能够通过贪心策略来获得最优解。

试题4参考答案

【问题1】

（1）i <= n

（2）D[j[r]] > d[i]

（3）J[r+1] = i 或者 J[q+1] = i

【问题2】

（4）1，2，4，5 或其他等价描述形式

（5）220

【问题3】

（6）能

（7）不能

试题5分析

该题考核了动态规划策略算法，其本质上是一个0-1背包问题，该最优化问题的目标函数是：

$$\max \sum_{i=1}^{n} v_i x_i (x_i = 0,1)$$

约束函数是：

$$\max \sum_{i=1}^{n} p_i x_i \le M (x_i = 0,1)$$

0-1背包问题可用动态规划策略求得最优解，求解的递归式为：

$$mv[i][j] = \begin{cases} o & i = 0 \text{或} j = 0 \\ m[i-1][j] & j < p_i \\ \max(m[i-1][j], mv[i-1][j-p_i] + v_i) & j >= p_i \end{cases}$$

其中，nv[i][j]表示由前i项食物组合且价格不超过j的套餐的最大营养价值。问题最终要求的套餐的最大营养价值为nv[n][M]。根据上边的递归式，自底向上的方式来编写伪代码。问题1中的伪代码第1行到第12行计算数组nv的元素值，第1行到第4行计算i为0或者j为0时的nv[i][j]的值，对应递归式的第一种情况；第7行和第8行对应递归式的第二种情况；第9行和第12行对应于第三种情况。因此，得空（1）的答案为：nv[i-1][j] >= nv[i-1][j-p[i]] + v[i]。伪代码的第13行到第19行求解放入到套餐中的食物，知空（2）的答案是：nv[i][j] = nv[i-1][j]，空（3）的答案为：j = j - p[i]。

问题2要求总价格不超过100，根据上述递归式，计算出要选择的食物为m_2，m_3，m_4，对应的总价值为605，总价格为100。

依据题目给出的伪代码，第1行到第2行、第3行到第4行以及第14行到第19行的时间复杂度均为O(n)，第5行到第12行的时间复杂度为O(nM)，因而得出算法总的时间复杂度为O(nM)。

试题5参考答案

【问题1】

（1）nv[i-1][j] >= nv[i-1][j-p[i]] + v[i]

（2）nv[i][j] = nv[i-1][j]

（3）j = j - p[i]

【问题2】

（4）m_2，m_3，m_4（答案中的食物边贸无前后顺序关系）

（5）605

【问题3】

（6）O(nM)，或其他等价的方式

　作者：希赛教育软考学院　来源：希赛网　2014年05月06日

试题解答方法

算法设计类题目要求考生掌握基本的算法设计技术，包括递归法、贪心法、回溯法、分治法、动态规划法等，需要结合具体的问题，用对应的算法设计技术来解决问题。

总体上，此类题目的解题步骤可以分为两步：

第1步：弄清题意，分析出题目所采用的算法；

同一个问题往往可以采用多种不同算法来求解，但在考题中的问题通过认真阅读题目要求，比较容易弄清问题采用的是哪种算法。在表5-6中，对常用的算法定义、特征和典型的应用场合做了对比分析。

表5-6 常用的算法定义、特征和典型的应用场合

算法名称	算法定义	特征和典型的应用场合
递归法	递归法是指一个过程或函数在其定义或说明中又直接或间接调用自身的一种方法。在使用递归策略时，必须有一个明确的递归结束条件，称为递归出口。在递归调用的过程当中系统为每一层的返回点、局部量等开辟了栈来存储。	递归次数过多容易造成栈溢出。 场合 1：数据的定义是按递归定义的。（Fibonacci函数） 场合 2：问题解法按递归算法实现。（回溯） 场合 3：数据的结构形式是按递归定义的。（树的遍历，图的搜索）
贪心法	贪心法是一种不追求最优解，只希望得到较为满意解的方法。贪心法常以当前情况为基础作最优选择，而不考虑各种可能的整体情况，所以贪心法不要回溯。	这类问题一般具有2个重要的性质：1、贪心选择性质； 2、最优子结构性质。 贪心法经典应用有背包问题、活动安排问题等。
回溯法	回溯法是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。	回溯是递归的一个特例，但它又有别于一般的递归法，用回溯法解题的一个显著特征是在搜索过程中动态产生问题的解空间。在任何时刻，算法只保存从根结点到当前扩展结点的路径。经典应用有：迷宫搜索、N皇后问题、骑士巡游、正则表达式匹配等。
分治法	分治法是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题，直到最后子问题可以简单的直接求解。而原问题的解就是子问题的解的合并。	问题规模缩小到一定的程度就可以容易地解决，可以分解为若干个规模较小的相同问题，利用该问题分解出的子问题的解可以合并为该问题的解；该问题所分解出的各个子问题是相互独立的。
动态规划法	动态规划法用于求解包含重叠子问题的最优化问题的方法。其基本思想是，将原问题分解为相似的子问题，在求解的过程中通过子问题的解求出原问题的解。动态规划实质上是一种以空间换时间的技术，它在实现的过程中，不得不存储产生过程中的各种状态，所以它的空间复杂度要大于其它的算法。	不论过去状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略；将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策，而只能通过当前的这个状态；最后，考虑采用动态规划的关键在于解决冗余。

下面以0-1背包问题为例，来说明算法的选择以及不同算法适合的应用场合的区别。

【问题】

给定n种物品和一个背包。物品i的重量是Wi，其价值为Vi，背包的容量为C。应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

【分析】

在选择装入背包的物品时，对每种物品m只有2种选择，即装入背包或不装入背包。不能将物品m装入背包多次，也不能只装入部分的物品m。

根据动态规划法和贪心法的定义与特征不难得出：动态规划法的确可以有效地求解0-1背包问题，而贪心法不能得到最优解。因为对于0-1背包问题，贪心法无法保证最终能将背包装满，部分闲置的背包空间使每公斤背包空间的价值降低了。

贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心法可行的第一个基本要素，也是贪心法与动态规划法的主要区别。

事实上，在考虑0-1背包问题时，应比较选择该物品和不选择该物品所导致的最终方案，然后再作出最好选择。由此就导出许多互相重叠的子问题，这正是用动态规划算法求解的另一重要特征。

第2步：结合实际问题要求，运用算法的原理进行解题。

明确题目所采用的算法后，解题的关键就在于对算法原理的运用。从历年的考试真题来看，对算法的考查方式主要有算法的表示（程序流程图、伪代码或程序代码补全）和算法复杂度的计算。因此，对常用算法的原理的掌握十分重要。在表5-7中，对常用的算法原理和解题步骤做了对比分析。

表5-7 常用的算法原理和解题步骤

算法名称	算法原理	解题步骤
递归法	递归的能力在于用有限的语句来定义对象的无限集合。一般来说，递归需要有边界条件、递归前进段和递归返回段。当边界条件不满足时，递归前进；当边界条件满足时，递归返回。	递归思想是一种典型的通过逆向思维求解问题的方法，其解题过程主要分为 2 个步骤： 1、分析递归关系，得出递归式； 2、确定终止条件，防止出现死循环。
贪心法	贪心法通常以自顶向下的方式进行，分阶段工作，以迭代的方式作出相继的贪心选择，每做一次贪心选择就将所求问题简化为规模更小的子问题。在每一个阶段总是选择认为当前最好的方案，然后从小的方案推广到大的方案的解决办法，它只需随着过程的进行保持当前的最好方案，采用“有好处就先占着”的贪心者的策略。	其解题过程主要分为 3 个步骤： 1、从问题的某一初始解出发； 2、循环求解，求出可行解的一个解元素； 3、由所有解元素组合成问题的一个可行解。
回溯法	回溯法是一种满足某些约束条件的穷举搜索法。它要求设计者找出所有可能的方法，然后选择其中的一种方法，若该方法不可行则试探下一种可能的方法。	其解题过程主要分为 3 个步骤： 1、针对所给问题，定义问题的解空间； 2、确定易于搜索的解空间结构； 3、以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。
分治法	将一个规模较大的问题分解为若干规模较小的子问题，找出各子问题的解，然后把各子问题的解组合成整个问题的解。在求解子问题时，往往继续采用同样的策略进行，即继续分解问题，逐个求解，最后合并解。这种不断用同样的策略求解规模较小的子问题，在程序设计语言实现时往往采用递归调用的方式实现。	其解题过程主要分为 3 个步骤： 1、分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题； 2、解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题； 3、合并：将各个子问题的解合并为原问题的解。
动态规划法	动态规划法通常以自底向上的方式解各子问题，分多阶段进行决策，其基本思路是：按时空特点将复杂问题划分为相互联系的若干个阶段，在选定系统行进方向之后，逆着这个行进方向，从终点向始点计算，逐次对每个阶段寻找某种决策，使整个过程达到最优，故又称为逆序决策过程。	其解题过程主要分为 2 个步骤： 1、划分阶段：按照问题的时间或空间特征，把问题分为若干个阶段。注意这若干个阶段一定要是有序的或者是可排序的（即无后向性）。 2、选择状态：将问题发展到各个阶段时所处于的各种客观情况用不同的状态表示出来。

在实际的软件程序设计过程中，不同算法之间的联系往往非常紧密。如分治法与递归法就经常同时出现，分治法产生的子问题往往是原问题的较小模式，这为递归法的使用提供了方便。比如归并排序算法就是成功应用分治法的一个典型例子，其基本思想是将待排序元素分成大致相同的两个子序列，分别对其进行排序，最终将排好序的子序列合并为所求的序列。按照上表中所述的分治法的3个步骤完成归并排序的过程是：

（1）分解。将N个元素分解为两个分别含有N/2个元素的子序列。

（2）求解。用归并排序对两个子序列递归地排序。

（3）合并。合并两个已经排好序的子序列以得到排序结果。

又如动态规划法和贪心法都有最优子结构的性质，但是动态规划法的关键是存在重叠子问题；贪心法的关键是存在贪心选择（即通过问题的整体最优解可以通过一系列局部最优的选择来得到）。如活动选择问题，即若干个具有竞争性的活动要求互斥使用某一公共资源（如会议室、教室等）时，如何选择最大的相容活动集合。这类问题具有最优子结构，可以用动态规划法求解。但同时该类问题还具有贪心选择性质，因此贪心法也是适用的。然而，有些动态规划法可以求解的问题，贪心法并不适用，如前面讲到的0-1背包问题适合用动态规划法来求解，但不适合用贪心法。

总之，算法设计的实践性要求比较强，在计算机软件资格水平考试下午的“软件设计师”考试中，对算法设计的大纲要求也正是要考查考生对常用算法的综合运用能力。

为此，广大考生在备考过程中，需要通过对历年真题的反复钻研、练习，做到“温故知新、举一反三”，进而在真正掌握常用算法设计技巧的同时，使得算法设计方面的考试题目不丢分。

作者：希赛教育软考学院　来源：希赛网　2014年05月06日

考情分析

本章主要围绕软件设计师大纲进行C++、Java语言程序设计和设计模式进行讲解，其中设计模式和软件设计师历年的真题的UML结构图全部采用Visio 2007绘制。本章知识全面，对C++、Java语言和设计模式知识进行了精炼的总结与概括，并对典型真题进行解析，考前必练选取了2007-2010年软件设计师历年的真题进行讲解与分析，是参加软件设计师考试的好帮手。

C++、Java语言程序设计往往与设计模式结合出现在下午的考题中，分值为15分。根据考试大纲要求，要求考生掌握C语言和C++、Java中的一种面向对象的程序语言。往年必做题有四道题，选做题有三道题，分别为C语言题、C++题和Java题，整个卷面共七道题。由于考试中许多考生在选做题中只选C语言题应答，对不会或不熟C++和Java语言的考生来说，一样可以应考，这与软件设计师考试大纲的要求是相背离的。因此，2010年上半年软件设计师下午试题中，必做题数量不变，选做题里去掉了C语言题，即选做题为二道题，整个卷面共六道题。这样，考生在选做题中就只能在C++和Java中选择一题作答，达到了考试大纲所要求的掌握一门面向对象的程序设计语言目的。在今后的软件设计师考试中，这种倾向将会维持一段时间。