算法分析与设计

51CTO学院：邹月平

概述

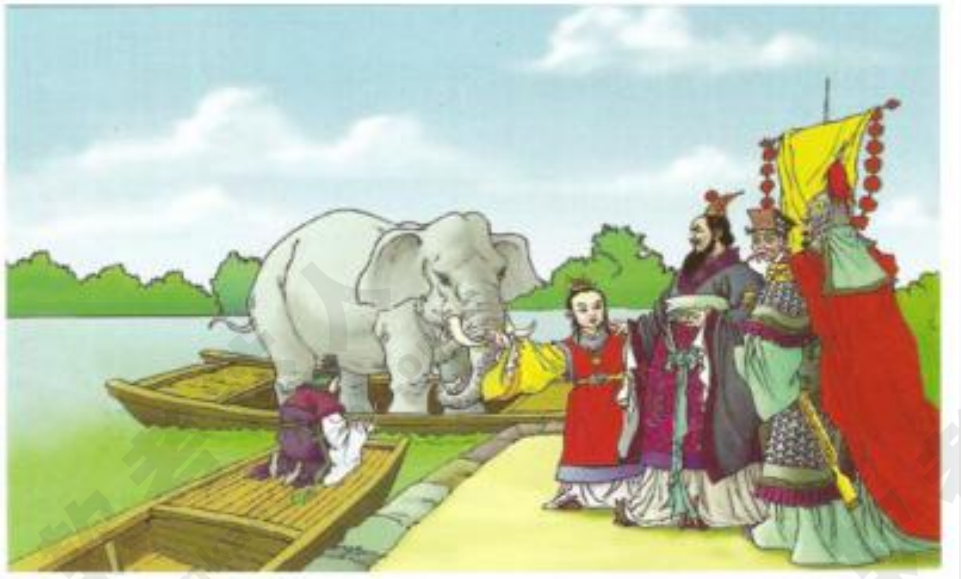
算法是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作。算法具有如下特征：

有穷性	在执行有穷步骤后结束，每个步骤都在有穷时间内完成。
确定性	每条指令必须有确切的含义，没有二义性。
可行性	算法是可行。
输入	一个算法有零个或多个输入。
输出	一个算法有零个或多个输出。

概述

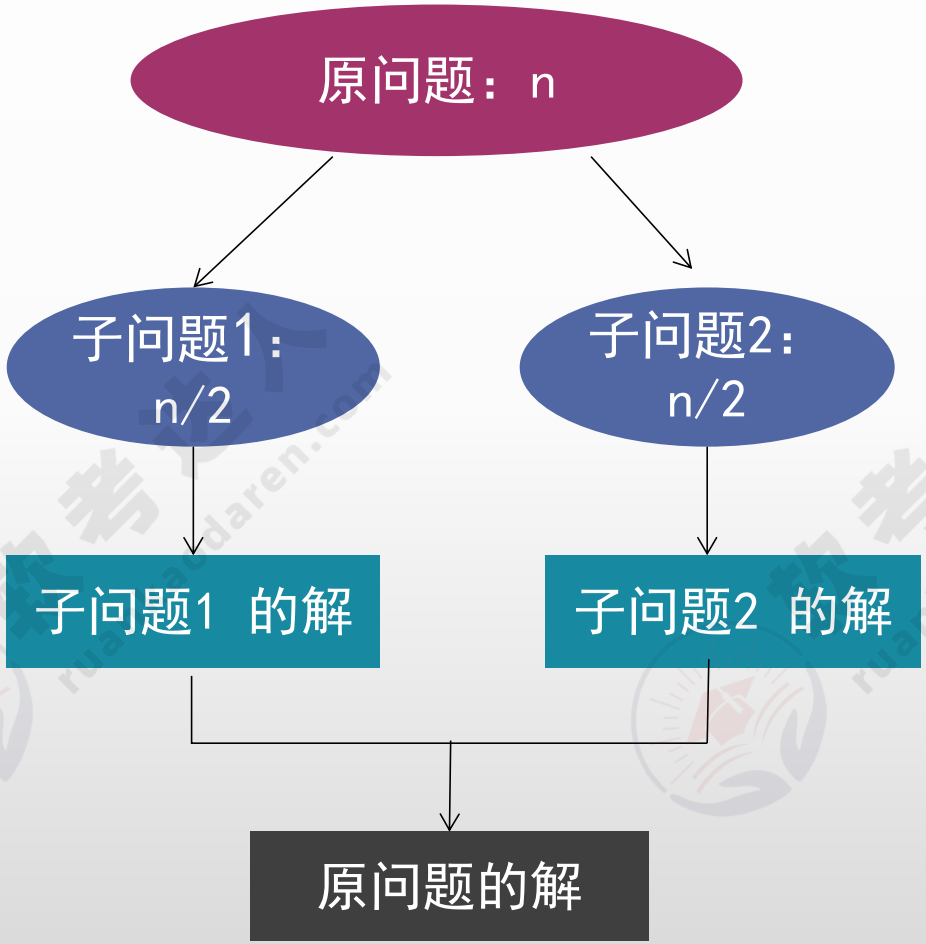
分治	将复杂问题分解成若干规模相同的子问题。
动态规划	类似于分治法。但具有最优子结构性质和重叠子问题性质。
贪心	不从整体考虑只求当前局部最优解。
回溯	达不到（最优）目标，就退回再走。

分治



对于一个规模为n的问题，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。这种算法设计策略叫做分治法。

分治



分治

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问题。

典型真题

【说明】

假币问题：有 n 枚硬币，其中有一枚是假币，已知假币的重量较轻。现只有一个天平，要求用尽量少的比较次数找出这枚假币。

【分析问题】

将 n 枚硬币分成相等的两部分：(1)当 n 为偶数时，将前后两部分，即 1...n/2 和 n/2+1...n，放在天平的两端，较轻的一端里有假币，继续在较轻的这部分硬币中用同样的方法找出假币；

(2)当 n 为奇数时，将前后两部分，即 1..(n-1)/2 和(n+1)/2+1...n，放在天平的两端，较轻的一端里有假币，继续在较轻的这部分硬币中用同样的方法找出假币

：若两端重量相等，则中间的硬币，即第(n+1)/2 枚硬币是假币。

典型真题

【C 代码】

下面是算法的C 语言实现，其中：

coins[]:硬币数组

first，last:当前考虑的硬币数组中的第一个和最后一个下标

```
#include <stdio.h>
int getCounterfeitCoin(int coins[],int first,int last)
{
    int firstSum = 0,lastSum = 0;
    int i;
    if(first==last-1){      /*只剩两枚硬币*/
        if(coins[first] < coins[last])
            return first;
        return last;
    }
```

典型真题

```
if((last - first + 1) % 2 ==0){   /*偶数枚硬币*/
    for(i = first;i < __(1)__; i++){
      firstSum += coins[i];
    }
    for(i=first+(last-first)/2+1;i<last+1;i++){
      lastSum += coins[i];
    }
    if( __(2)__ ){
      return getCounterfeitCoin(coins,first,first+(last-first)/2);
    }else{
       return getCounterfeitCoin(coins,first+(last-first)/2+1,last);
    }
}
```

典型真题

```
else{/*奇数枚硬币*/
    for(i=first;i<first+(last-first)/2;i++){
        firstSum+=coins[i];

    }
    for(i=first+(last-first)/2+1;i<last+1;i++){
        lastSum+=coins[i];
    }
    if(firstSum<lastSum){
        return getCounterfeitCoin(coins,first,first+(last-first)/2-1);
    }else if(firstSum>lastSum){
        return getCounterfeitCoin(coins,first+(last-first)/2+1,last);
    }else{
        return  (3)  ;
    }
  }
}
```

典型真题

【问题一】（6 分）

根据题干说明，填充 C 代码中的空（1）-（3）。

【问题二】（6 分）

根据题干说明和 C 代码，算法采用了（4）设计策略。

函数 getCounterfeitCoin 的时间复杂度为（5 ）（用 O 表示）。

【问题三】（3 分）

若输入的硬币数为 30，则最少的比较次数为（6），最多的比较次数为（7）。

典型真题

参考答案：

(1)first + (last - first) / 2 + 1 或 (last + first) / 2 + 1

(2)firstSum<lastSum

(3)first + (last - first) / 2 或(last + first) / 2

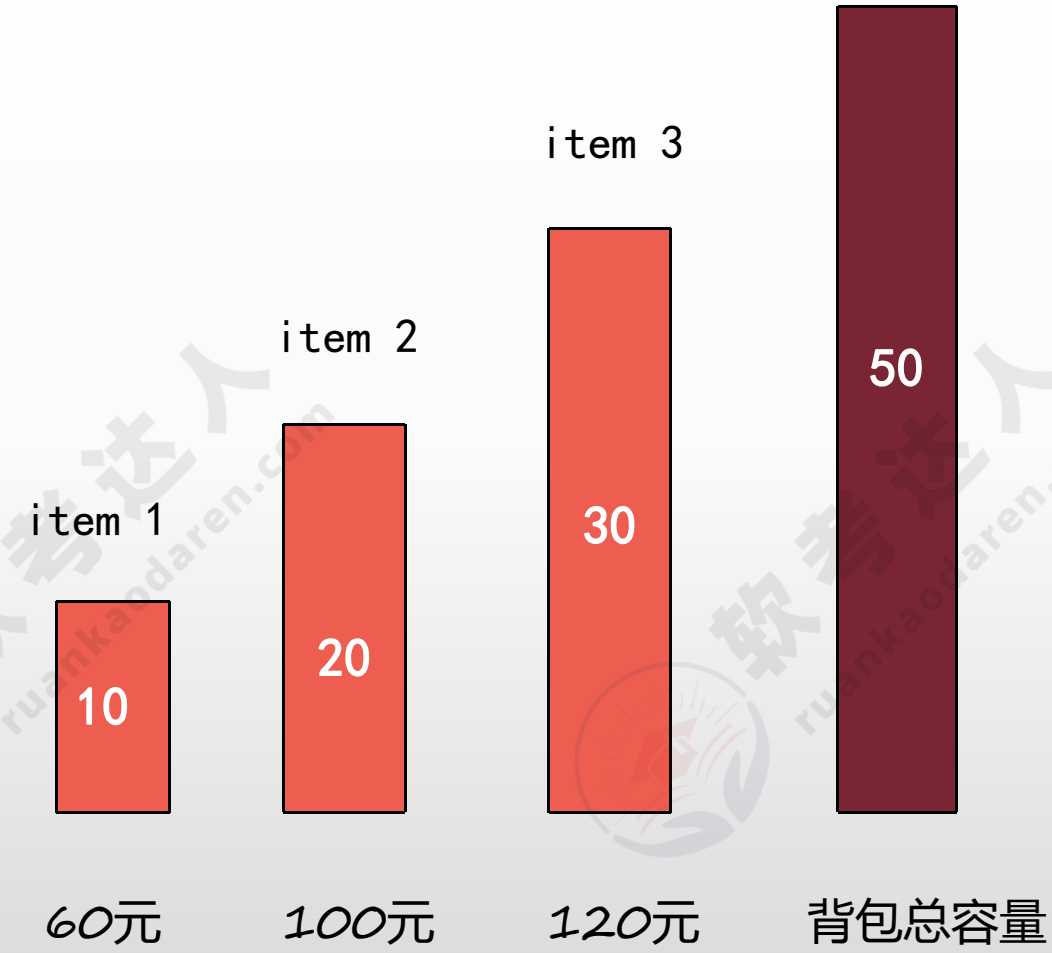
(4)分治法

(5)O(nlgn)

(6)2

(7)4

动态规划



动态规划

动态规划法与分治法类似，也是将要求解的问题一层一层地分解成一级一级、规模逐步缩小的子问题，直到可以直接求出其解的子问题为止。动态规划法的特征：

（1）最优子结构。当问题的最优解包含了其子问题的最优解时，称该问题具有最优子结构性质。问题的最优子结构性质使我们能够以自底向上的方式递归地从子问题的最优解构造出整个问题的最优解。

（2）重叠子问题。有些子问题被反复计算多次，这就是重叠子问题性质。动态规划算法对每一个子问题只解一次，而后将其解保存在一个表格中，当再次需要解此问题时，只是简单地利用常数时间查看一下结果。

典型真题

某汽车加工工厂有两条装配线L1和L2，每条装配线的工位数均为n（Sij，i=1或2，j=1，2，...，n），两条装配线对应的工位完成同样的加工工作，但是所需要的时间可能不同（aij，i=1或2，j=1，2，...，n）。汽车底盘开始到进入两条装配线的时间(e1，e2)以及装配后到结束的时间(X1X2)也可能不相同。从一个工位加工后流到下一个工位需要迁移时间(tij，i=1或2，j=2，...n）。现在要以最快的时间完成一辆汽车的装配，求最优的装配路线。

分析该问题，发现问题具有最优子结构。以L1为例，除了第一个工位之外，经过第j个工位的最短时间包含了经过L1的第j-1个工位的最短时间或者经过L2的第j-1个工位的最短时间，如式(1)。装配后到结束的最短时间包含离开L1的最短时间或者离开L2的最短时间如式（2）。

典型真题

$$f_{1,j}=\begin{cases} e_1+a_{1,j} & \text{若}j=1 \\ \min(f_{1,j-1}+a_{1,j}+t_{1,j-1}, f_{2,j-1}+a_{1,j}+t_{2,j-1}) & \text{其他} \end{cases} \quad (1)$$

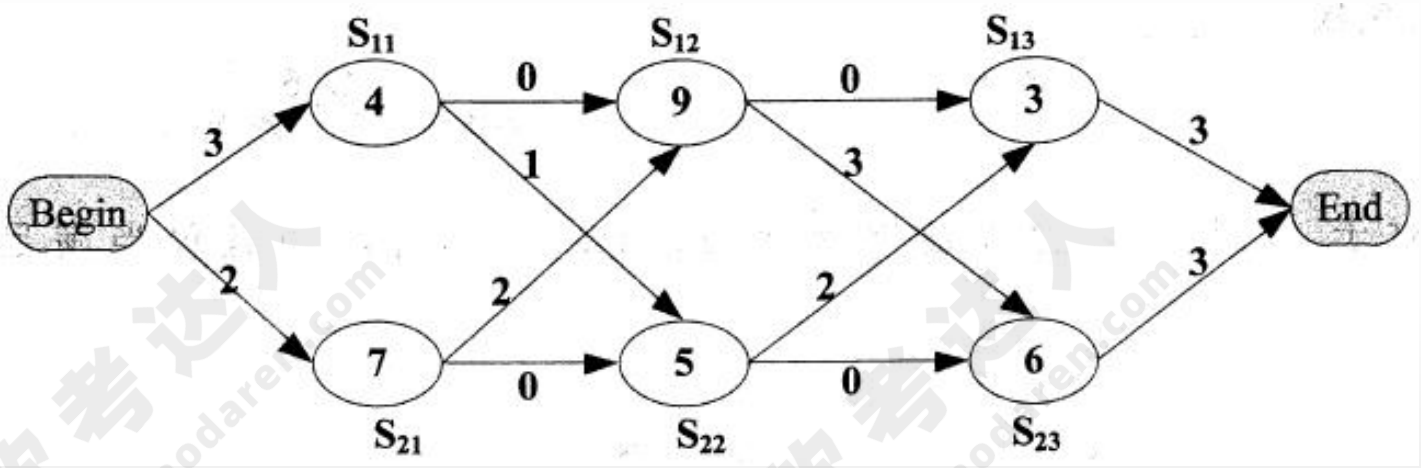
$$f_{min}=\min(f_{1,n}+x_1, f_{2,n}+x_2) \quad (2)$$

由于在求解经过L1和L2的第j个工位的最短时间均包含了经过L1的第j-1个工位的最短时间或者经过L2的第j-1个工位的最短时间，该问题具有重复子问题的性质，故采用迭代方法求解。

该问题采用的算法设计策略是（1），算法的时间复杂度为（2）。

典型真题

以下是一个装配调度实例，其最短的装配时间为（3），装配路线为（4）。



（1）A. 分治　　B. 动态规划　　C. 贪心　　D. 回溯

（2）A. O(lgn)　　B. O(n)　　C. $O(n^2)$　　D. O(nlgn)

（3）A. 21　　B. 23　　C. 20　　D. 26

（4）A. S11→S12→S13　　B. S11→S22→S13

典型真题

试题分析

动态规划算法与分治法不同的是，适合于用动态规划求解的问题，经分解得到子问题往往不是互相独立的。若用分治法来解这类问题，则分解得到的子问题数目太多，有些子问题被重复计算了很多次。如果能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间。可以用一个表来记录所有已解的子问题的答案。不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中。这就是动态规划法的基本思路。本题中的时间复杂度为0(n) 。

贪心选择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。这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。

典型真题

回溯算法实际上一个类似枚举的搜索尝试过程，主要是在搜索尝试过程中寻找问题的解，当发现已不满足求解条件时，就“回溯”返回，尝试别的路径。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”。

求最短的装配时间与装配路线只需要将选项按照公式带入计算（将图上每条路径上的所有数字相加）可得最短路线为S11→S22→S13 ，时间为21。

参考答案：B、B、A、B

典型真题

已知矩阵Am*n和Bn*p相乘的时间复杂度为O(m*n*p)。矩阵相乘满足结合律，如三个矩阵A、B、C相乘的顺序可以是(A*B)*C)，也可以是A*(B*C)。不同的相乘序所需进行的乘法次数可能有很大的差别，因此确定n个矩阵相乘的最优计算顺序是一个非常重要的问题。已知确定n个矩阵$A_1, A_2 \ldots\ldots A_n$相乘的计算顺序具有最优子结构，即$A_1A_2 \ldots\ldots\ldots A_n$的最优计算顺序包含其子问题$A_1A_2 \ldots\ldots A_k$和$A_{k+1}A_{k+2} \ldots\ldots A_n$($1 \leqslant k < n$)的最优计算顺序。可以列出其递归式为

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

典型真题

其中，Ai的维度为$p_{i-1}*p_i$，m[i，j]表示$A_iA_{i+1}\cdots A_j$最优计算顺字的相乘次数，先采用自底向上的方法求n个矩阵相乘的最优计算顺序。则该问题的算法设计策略为（ ），算法的时间复杂度为（ ），空间复杂度为（ ）。给定一个实例，$(P_0P_1.\ldots\ldots P_5)=(20，15，4，10，20，25)$最优计算顺序为（ ）。

A. 分治法　　B. 动态规划法　　C. 贪心法　　D. 回溯法

A. $O(n^2)$　　B. $O(n^2lgn)$　　C. $O(n^3)$　　D. $O(2^n)$

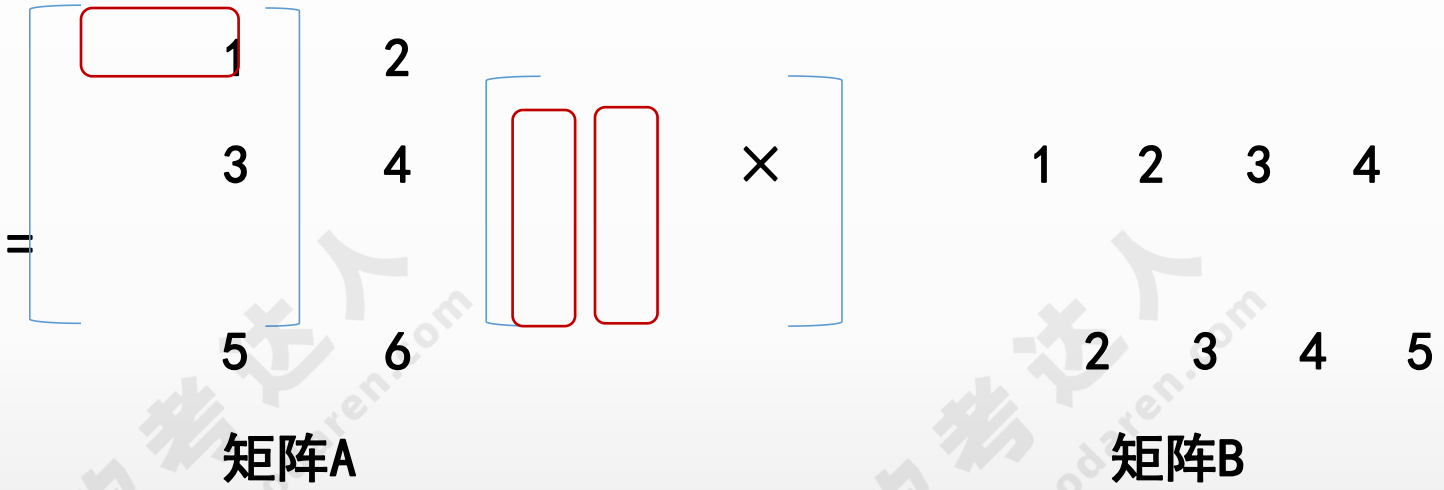
A. $O(n^2)$　　B. $O(n^2lgn)$　　C. $O(n^3)$　　D. $O(2^n)$

A.　(((A1*A2)*A3)*A4)*A5　　B.　A1*(A2*(A3*(A4*A5)))

C.　((A1*A2)*A3)*(A4*A5)　　D .(A1*A2）*((A3*A4)*A5)

矩阵连乘法

矩阵A3*2，矩阵B 2*4相乘，总的计算量是：3*2*4=24次



$$= \begin{bmatrix} 5 & 8 & 11 & 14 \\ 11 & 18 & 25 & 32 \\ 17 & 28 & 39 & 50 \end{bmatrix}$$

典型真题

试题分析

根据题干有A1-A5五个矩阵，根据题干“给定一个实例，$(P_0P_1.....P_5)=(20，15，4，10，20，25)$”这5个矩阵分别为：20*15、15*4、4*10、10*20、20*25，分别带入65题各个选项，得到选项D是计算次数最少的选项。

具体计算结果为：

选项A：A1*A2=20*15*4=1200，(A1*A2)*A3)=20*4*10=800，

(((A1*A2)*A3)*A4)=20*10*20=4000，

(((A1*A2)*A3)*A4)*A5=20*20*25=10000，总的计算次数为1200+800+4000+10000=16000次。

典型真题

选项B：A4*A5=10*20*25=5000，A3*(A4*A5)=4*10*25=1000，
A2*(A3*(A4*A5))=15*4*25=1500，A1*(A2*(A3*(A4*A5)))=20*15*25=7500，
总的计算次数为：5000+1000+1500+7500=15000次。

选项C：A1*A2=20*15*4=1200，(A1*A2)*A3)=20*4*10=800，
A4*A5=10*20*25=5000，

((A1*A2)*A3)*(A4*A5)=20*10*25=5000，总的计算次数为
1200+800+5000+5000=12000次。

典型真题

选项D：A1*A2=20*15*4=1200，A3*A4=4*10*20=800，

(A3*A4)*A5=4*20*25=2000，

(A1*A2）*((A3*A4)*A5)=20*4*25=2000，总的计算次数为

1200+800+2000+2000=6000次。

该算法的，pi-1，pk，pj的值需要三重循环解决，因此时间复杂度为O（n^3），空间复杂度为O（n^2）。

参考答案：B、C、A、D

典型真题

【说明】

某公司购买长钢条，将其切割后进行出售。切割钢条的成本可以忽略不计，钢条的长度为整英寸。已知价格表 P，其中中 P_i（i＝1，2，...，m）表示长度为 i 英寸的钢条的价格。现要求解使销售收益最大的切割方案。

求解此切割方案的算法基本思想如下：

假设长钢条的长度为 n 英寸，最佳切割方案的最左边切割段长度为 i 英寸，则继续求解剩余长度为 n－i 英寸钢条的最佳切割方案。考虑所有可能的 i，得到的最大收益 r_n对应的切割方案即为最佳切割方案。rn的递归定义如下：

$$r_n = \max_{1 \leqslant i \leqslant n}(p_i + r_{n-i})$$

对此递归式，给出自顶向下和自底向上两种实现方式

典型真题

【C 代码】

```
/*常量和变量说明
n：长钢条的长度
P[]：价格数组
*/
#define LEN 100
int Top_Down_ Cut_Rod(int P[]，int n) {/*自顶向下*/
int r=0;
int i;
if(n==0){
return 0;
}
for(i=1;（1）;i++){
int tmp=p[i]+Top_Down_ Cut_Rod(p,n-i);
r=(r>=tmp)?r：tmp;
}
return r;
}
```

典型真题

P0	P1	P2	P3	P4	P5
0	1	3	5	4	8

递过程：

① i=1,n-i=4 代入p[i]+Top_Down_ Cut_Rod(p,n-i)，传递参数进行递归调用。

② i=1,n-i=3 代入p[i]+Top_Down_ Cut_Rod(p,n-i)，传递参数进行递归调用。

③ i=1,n-i=2 代入p[i]+Top_Down_ Cut_Rod(p,n-i)，传递参数进行递归调用。

④ i=1,n-i=1 代入p[i]+Top_Down_ Cut_Rod(p,n-i)，传递参数进行递归调用。

⑤ i=1,n-i=0 代入p[i]+Top_Down_ Cut_Rod(p,n-i)，传递参数进行递归调用。

典型真题

P0	P1	P2	P3	P4	P5
0	1	3	5	4	8

归过程：

当i=1，n-i=0时，tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[1]+(p,0)=1，返回结果1到递4：

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[1]+(p,1)=2，i++。

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[2]+(p,0)=3，返回结果3到递3：

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[1]+(p,2)=4，i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[2]+(p,1)=4，i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[3]+(p,0)=5，，返回结果5到递2：

典型真题

P0	P1	P2	P3	P4	P5
0	1	3	5	4	8

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[1]+(p,3)=6, i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[2]+(p,2)=6, i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[3]+(p,1)=6, i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[4]+(p,0)=4, , 返回结果6到递1：

典型真题

P0	P1	P2	P3	P4	P5
0	1	3	5	4	8

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[1]+(p,4)=5， i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[2]+(p,3)=8， i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[3]+(p,2)=8， i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[4]+(p,1)=5， i++

tmp=p[i]+Top_Down_ Cut_Rod(p,n-i)=p[5]+(p,0)=8，返回结果8。

典型真题

```
int Bottom_Up_Cut_Rod(int p[],int n){ /*自底向上*/
int r[LEN]={0};
int temp=0;
int i,j;
for(j=1;j<=n;j++){
temp=0;
for(i=1;(2);i++){
temp=(3);
}
 (4)
}
return r[n];
}
```

典型真题

【问题 1】（8 分）

根据说明，填充 C 代码中的空（1）～（4）。

【问题 2】（7 分）

根据说明和 C 代码，算法采用的设计策略为（5）。

求解 r_n时，自顶向下方法的时间复杂度为（6）；自底向上方法的时间复杂度为（7）（用 O 表示）。

典型真题

参考答案：

【问题 1】

(1) i<=n

(2) i<=j

(3) temp= (temp>=p[i]+r[j-1]) ?temp: (p[i]+r[j-1])

(4) r[j]=temp

典型真题

【问题 2】

(5) 动态规划

（6）0（2^n）

(7) 0（n^2）

动态规划法

阅读下列说明和 C 代码，回答问题 1至问题 3，将解答写在答题纸的对应栏内。

【说明】

0-1背包问题定义为:给定1个物品的价值v[1....i]、重量w[1....i]和背包容量T,每个物品装到背包里或者不装到背包里,求最优的装包方案,使得所得到的价值最大。

0-1背包问题具有最优子结构性质,定义c为最优装包方案所获得的最大价值。

$$c[i][T]=\begin{cases}0 & \text{若 } i=0 \text{ 或 } T=0\\ c[i-1][T] & \text{若 } T<w[i]\\ \max(c[i-1][T-w[i]]+v[i].c[i-1][T]) & \text{若 } i>0 \text{ 且 } T\geq w[i]\end{cases}$$

典型真题

$$V[i,W]=\max\{v_i+V[i-1,W-w_i]+,V[i-1,W]\}$$

i \ w	0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	1	1	1	1	1	1	1
2	0	1	6	7	7	7	7	7	7	7	7	7
3	0	1	6	7	7	18	19	24	25	25	25	25
4	0	1	6	7	7	18	22	24	28	29	29	40
5	0	1	6	7	7	18	22	28	29	34	35	40

物品	重量	价值
1	1	1
2	2	6
3	5	18
4	6	22
5	7	28

典型真题

$V[1,1]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,0],V[0,1]\}=\max\{1+0,0\}=1$

$V[1,2]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,1],V[0,2]\}=\max\{1+0,0\}=1$

$V[1,3]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,2],V[0,3]\}=\max\{1+0,0\}=1$

$V[1,4]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,3],V[0,4]\}=\max\{1+0,0\}=1$

$V[1,5]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,4],V[0,5]\}=\max\{1+0,0\}=1$

$V[1,6]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,5],V[0,6]\}=\max\{1+0,0\}=1$

$V[1,7]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,6],V[0,7]\}=\max\{1+0,0\}=1$

$V[1,8]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,7],V[0,8]\}=\max\{1+0,0\}=1$

$V[1,9]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,8],V[0,9]\}=\max\{1+0,0\}=1$

$V[1,10]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,9],V[0,10]\}=\max\{1+0,0\}=1$

$V[1,11]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_1+V[0,10],V[0,11]\}=\max\{1+0,0\}=1$

典型真题

$V[2,1]=V[i-1,w]=V[1,1]=1$

$V[2,2]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,0],V[1,2]\}=\max\{6+0,1\}=6$

$V[2,3]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,1],V[1,3]\}=\max\{6+1,1\}=7$

$V[2,4]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,2],V[1,4]\}=\max\{6+1,1\}=7$

$V[2,5]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,3],V[1,5]\}=\max\{6+1,1\}=7$

$V[2,6]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,4],V[1,6]\}=\max\{6+1,1\}=7$

$V[2,7]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,5],V[1,7]\}=\max\{6+1,1\}=7$

$V[2,8]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,6],V[1,8]\}=\max\{6+1,1\}=7$

$V[2,9]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,7],V[1,9]\}=\max\{6+1,1\}=7$

$V[2,10]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,8],V[1,10]\}=\max\{6+1,1\}=7$

$V[2,11]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_2+V[1,9],V[1,11]\}=\max\{6+1,1\}=7$

典型真题

V[3, 1]=V[i−1, w]=V[2, 1]=1

V[3, 2]=V[i−1, w]=V[2, 2]=6

V[3, 3]=V[i−1, w]=V[2, 3]=7

V[3, 4]=V[i−1, w]=V[2, 4]=7

$V[3,5]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_3+V[2,0],V[2,5]\}=\max\{18+0,7\}=18$

$V[3,6]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_3+V[2,1],V[2,6]\}=\max\{18+1,7\}=19$

$V[3,7]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_3+V[2,2],V[2,7]\}=\max\{18+6,7\}=24$

$V[3,8]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_3+V[2,3],V[2,8]\}=\max\{18+7,7\}=25$

$V[3,9]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_3+V[2,4],V[2,9]\}=\max\{18+7,7\}=25$

$V[3,10]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_3+V[2,5],V[2,10]\}=\max\{18+7,7\}=25$

$V[3,11]=\max\{v_i+V[i-1,w-w_i],V[i-1,w]\}=\max\{v_3+V[2,6],V[2,11]\}=\max\{18+7,7\}=25$

典型真题

V[4, 1]=V[i−1, w]=V[3, 1]=1

V[4, 2]=V[i−1, w]=V[3, 2]=6

V[4, 3]=V[i−1, w]=V[3, 3]=7

V[4, 4]=V[i−1, w]=V[3, 4]=7

V[4, 5]=V[i−1, w]=V[3, 5]=18

$V[4, 6]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_4+V[3, 0], V[3, 6]\}=\max\{22+0, 19\}=22$

$V[4, 7]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_4+V[3, 1], V[3, 7]\}=\max\{22+1, 24\}=24$

$V[4, 8]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_4+V[3, 2], V[3, 8]\}=\max\{22+6, 25\}=28$

$V[4, 9]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_4+V[3, 3], V[3, 9]\}=\max\{22+7, 25\}=29$

$V[4, 10]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_4+V[3, 4], V[3, 10]\}=\max\{22+7, 25\}=29$

$V[4, 11]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_4+V[3, 5], V[3, 11]\}=\max\{22+18, 25\}=40$

典型真题

V[5, 1]=V[i−1, w]=V[4, 1]=1

V[5, 2]=V[i−1, w]=V[4, 2]=6

V[5, 3]=V[i−1, w]=V[4, 3]=7

V[5, 4]=V[i−1, w]=V[4, 4]=7

V[5, 5]=V[i−1, w]=V[4, 5]=18

V[5, 6]=V[i−1, w]=V[4, 6]=22

$V[5, 7]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_5+V[4, 0], V[4, 7]\}=\max\{28+0, 24\}=28$

$V[5, 8]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_5+V[4, 1], V[4, 8]\}=\max\{28+6, 28\}=34$

$V[5, 9]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_5+V[4, 2], V[4, 9]\}=\max\{28+6, 29\}=34$

$V[5, 10]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_5+V[4, 3], V[4, 10]\}=\max\{28+7, 29\}=35$

$V[5, 11]=\max\{v_i+V[i-1, w-w_i], V[i-1, w]\}=\max\{v_5+V[4, 4], V[4, 11]\}=\max\{28+7, 40\}=40$

典型真题

阅读下列说明和 C 代码，回答问题 1至问题 3，将解答写在答题纸的对应栏内。

【说明】

0-1背包问题定义为:给定1个物品的价值v[1....i]、重量w[1....i]和背包容量T,每个物品装到背包里或者不装到背包里,求最优的装包方案,使得所得到的价值最大。

0-1背包问题具有最优子结构性质,定义c为最优装包方案所获得的最大价值。则可得到如下所示的递归式。

$$c[i][T]=\begin{cases}0 & \text{若 } i=0 \text{ 或 } T=0\\ c[i-1][T] & \text{若 } T<w[i]\\ \max(c[i-1][T-w[i]]+v[i],c[i-1][T]) & \text{若 } i>0 \text{ 且 } T\geq w[i]\end{cases}$$

典型真题

【C 代码】

下面是算法的 C 语言实现

(1)常量和变量说明

T:背包容量

V[]:价值数组

W[]:重量数组

C[][]:c[i][j]表示前 i 个物品在背包容量为 j 的情况下最优装包方案所能获得的最大价值

典型真题

```
#include <stdio. h>
#include <math. h>
#define N 6
#define maxT 1000
int c[N][maxT]={0}:
int Memoized_Knapsack(int v[N], int w[N], int T) {
   int i;
   int j:
   for(i=0;i<N;i++){
   for(j=0;j<=T;j++){
         c[i][j]= -1;
      }
   }
   return Calculate_Max_Value(v, w, N-l, T);
}
```

典型真题

```
[N], int w[N], int i , int j){
    int temp =0;
    if(c[i][j] !=-1){
return（1） ;
    }
     if(i=0||j=0){
      c[i][j]=0;
      }else{
            c[i] [j]= Calculate_Max_Value (v, w, i-1, j);
            if（（2））{
                  temp=（3）;
                  if(c[i] [j]<temp) {
                        （4）;
                  }
            }
      }
      return c [i][j];
}
```

典型真题

【问题1】(8分)

根据说明和C代码，填充C代码中的空(1)~(4)。

【问题2】(4分)

根据说明和C代码，算法采用了(5)设计策略。在求解过程中，采用了(6)(自底向上或者自顶向下)的方式。

【问题3】(3分)

若5项物品的价值数组和重量数组分别为v[]={0, 1, 6, 18, 22, 28}和w[]={0, 1, 2, 5, 6, 7}，背包容量为T=11，则获得的最大价值为(7)。

典型真题

参考答案：

问题1.

1： c[i][j]

2: j >= w[i]

3: Calculate_Max_Value(v, w, i - 1, j - w[i]) + v[i]

4: c[i][j] = temp

问题2.

5：动态规划

6:自底向上

问题3.

典型真题

● 考虑一个背包问题，共有n=5个物品，背包容量为W= 10，物品的重量和价值分别为：w={2，2，6，5，4}，v={6，3，5，4，6}，求背包问题的最大装包价值。若此为0-1背包问题，分析该问题具有最优子结构，定义递归式为

$$c(i,j)=\begin{cases}0 & \text{若 } i=0 \text{ 或 } j=0\\ c(i-1,j) & \text{若 } w[i]>j\ ,\\ \max\{c(i-1,j),c(i-1,j-w[i])+v[i]\} & \text{其他}\end{cases}$$

其中c（i，j）表示i个物品、容量为j的0-1背包问题的最大装包价值，最终要求解c（n，W）。

采用自底向上的动态规划方法求解，得到最大装包价值为（ ），算法的时间复杂度为（ ）。

若此为部分背包问题，首先采用归并排序算法，根据物品的单位重量价值从大到小排序，然后依次将物品放入背包直至所有物品放入背包中或者背包再无容量，则得到的最大装包价值为（ ），算法的时间复杂度为（ ）。

典型真题

A. 11　　B. 14　　C. 15　　D. 16.67

A. Θ(nW)　　B. Θ(nlgn)　　C. Θ(n^2)　　D. Θ(nlgnW)

A. 11　　B. 14　　C. 15　　D. 16.67

A. Θ(nW)　　B. Θ(nlgn)　　C. Θ(n2)　　D. Θ(nlgnW)

典型真题

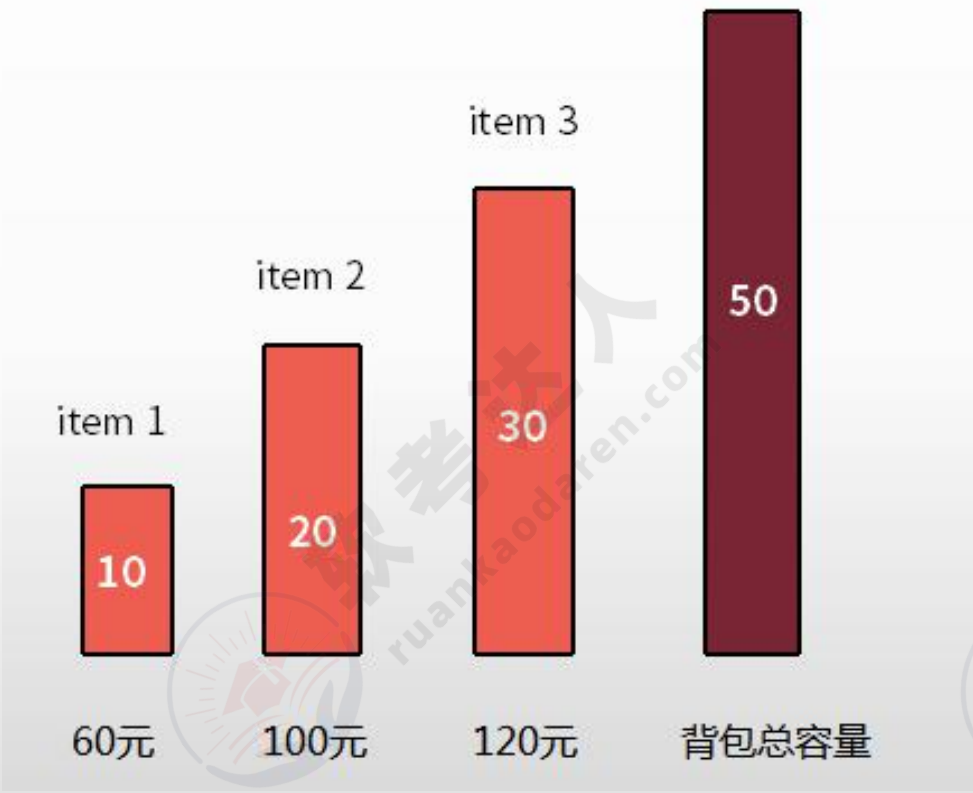
试题分析

这是典型的01背包问题，动态规划算法中，自底向上（递推）：从小范围递推计算到大范围，可以看到装第一个和第五个物品价值是最高的，这时候V=12了，然后占了6的重量了，只能装物品2了，价值15，第二个问题是部分背包，部分背包的时候计算每个物品单位重量价值多少，单位重量v={3 1.5 5/6 0.8 1.5}，可以看到1 2 5的单位价值最高，选择125后背包重量还只有8，还有2个重量可以选择3号物品，得5*1/3的 价值，就是1.67，所以第三问为16.67；复杂度，都没有进行指数级别的运算，问题1只需要找n个物品与价值W相乘，问题3计算单位物品价值然后考虑背包大小就可以了。

参考答案：C、A、D、B

贪心法

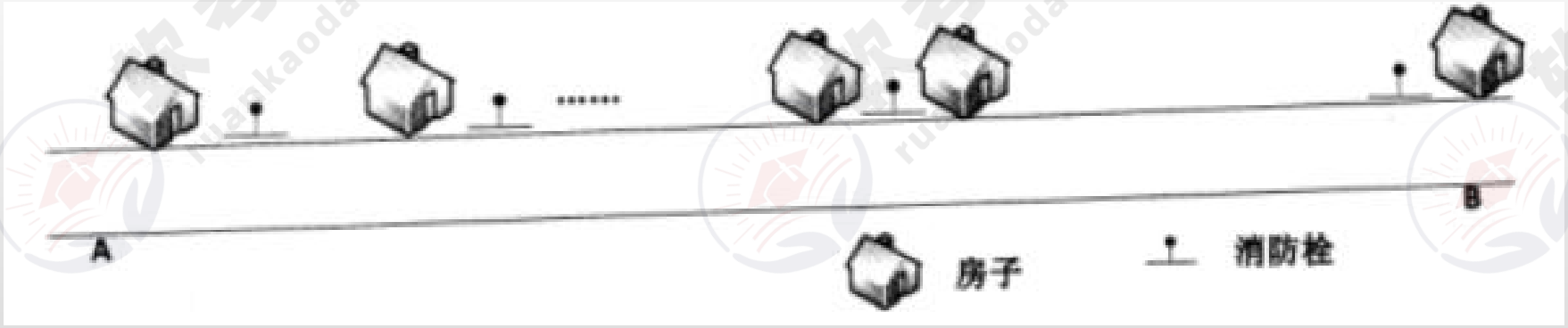
贪心法总是做出在当前来说是最好的选择，而并不从整体上加以考虑，它所做的每步选择只是当前步骤的局部最优选择，但从整体来说不一定是最优的选择。



典型真题

在一条笔直公路的一边有许多房子，现要安装消防栓，每个消防栓的覆盖范围远大于房子的面积，如下图所示。现求解能覆盖所有房子的最少消防栓数和安装方案（问题求解过程中，可将房子和消防栓均视为直线上的点。

该问题求解算法的基本思路为：从左端的第一栋房子开始，在其右侧m米处安装一个消防栓，去掉被该消防栓覆盖的所有房子。在剩余的房子中重复上述操作，直到所有房子被覆盖。算法采用的设计策略为（1）；对应的时间复杂度为（2）。



典型真题

假设公路起点A的坐标为0，消防栓的覆盖范围（半径）为20米，10栋房子的坐标为（10，20，30，35，60，80，160，210，260，300），单位为米。根据上述算法，共需要安装（ ）个消防栓。以下关于该求解算法的叙述中，正确的是（ ）。

A. 分治　　B. 动态规划　　C. 贪心　　D. 回溯

A. lgn　　B. n　　C. nlgn　　D. n^2

A. 4　　B. 5　　C. 6　　D. 7

A. 肯定可以求得问题的一个最优解

B. 可以求得问题的所有最优解

C. 对有些实例，可能得不到最优解

D. 只能得到近似最优解

典型真题

试题解析：

对于第一空，本题使用的是贪心法。

由于本题的算法过程，是依次与各个房子进行判断，当所有房子都被比较之后，则问题结束，因此时间复杂度与房子的个数相关，本问题的时间复杂度应该趋于现象，为0(n)。

典型真题

对于第三空，关于对应序列（10，20，30，35，60，80，160，210，260，300）

第一轮放置：在第一座房子x=10的右侧20米处安装一个消防栓，可以覆盖10，20，30，35这4栋房子；

2、第二轮放置：去掉前4栋房子，在第5栋房子x=60的右侧20米处安装一个消防栓，可以覆盖60、80这2栋房子；

3、第三轮放置：去掉前面已覆盖的房子，在第7栋房子x=160的右侧20米处安装一个消防栓，只可以覆盖160这一栋房子；

4、第四轮放置：去掉前面已覆盖的房子，在第8栋房子x=210的右侧20米处安装一个消防栓，可以覆盖210这一栋房子

第五轮放置：去掉前面已覆盖的房子，在第9栋房子x=260的右侧20米处安装一个消防栓，可以覆盖260、300这2栋房子；

房子全部覆盖完毕，因此共需安装5个消防栓。

典型真题

对于第四空，对于得到一个最优解是动态规划的特点，可以得到问题所有的最优解，是回溯法的特征，可以排除A、B选项。选择C。有些实例，可能得不到最优解。

参考答案:C.B.B.C

典型真题

现需要申请一些场地举办一批活动，每个活动有开始时间和结束时间。在同一个场地，如果一个活动结束之前，另一个活动开始，即两个活动冲突。若活动A从1时间开始，5时间结束，活动B从5时间开始，8时间结束，则活动A和B不冲突。现要计算n个活动需要的最少场地数。

求解该问题的基本思路如下（假设需要场地数为m，活动数为n，场地集合为P1，P2，…，Pm），初始条件Pi均无活动安排：

（1）采用快速排序算法对n个活动的开始时间从小到大排序，得到活动a1，a2，…，an。对每个活动ai，i从1到n，重复步骤（2）、（3）和（4）；

（2）从p1开始，判断ai与P1的最后一个活动是否冲突，若冲突，考虑下一个场地P2，…；

（3）一旦发现ai与某个Pj的最后一个活动不冲突，则将ai安排到Pj，考虑下一个活动；

典型真题

(4) 若ai与所有已安排活动的Pj的最后一个活动均冲突，则将ai安排到一个新的场地，考虑下一个活动；

(5) 将n减去没有安排活动的场地数即可得到所用的最少场地数算法首先采用了快速排序算法进行排序，其算法设计策略是(1)；后面步骤采用的算法设计策略是(2)。整个算法的时间复杂度是(3)。下表给出了n=11的活动集合，根据上述算法，得到最少的场地数为(4)。

i	1	2	3	4	5	6	7	8	9	10	11
开始时间 s_i	0	1	2	3	3	5	5	6	8	8	12
结束时间 f_i	6	4	13	5	8	7	9	10	11	12	14

典型真题

（1）A. 分治　　B. 动态规划　　C. 贪心　　D. 回溯

（2）A. 分治　　B. 动态规划　　C. 贪心　　D. 回溯

（3）A. $\Theta(\lg n)$　　B. $\Theta(n)$　　C. $\Theta(n\lg n)$　　D. $\Theta(n^2)$

（4）A. 4　　B. 5　　C. 6　　D. 7

典型真题

试题分析

快速排序由C. A. R. Hoare在1962年提出。它的基本思想是：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。快速排序采用的思想是分治思想。

贪心算法（又称贪婪算法）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，他所做出的是在某种意义上的局部最优解。

整个算法的时间复杂度是O（nlogn）。

场地上可以安排活动1、8、11为一个场地；活动2、6、9一个场地；活动3为一个场地；活动4、7为一个场地；活动5、10为一个场地，共5个场地。

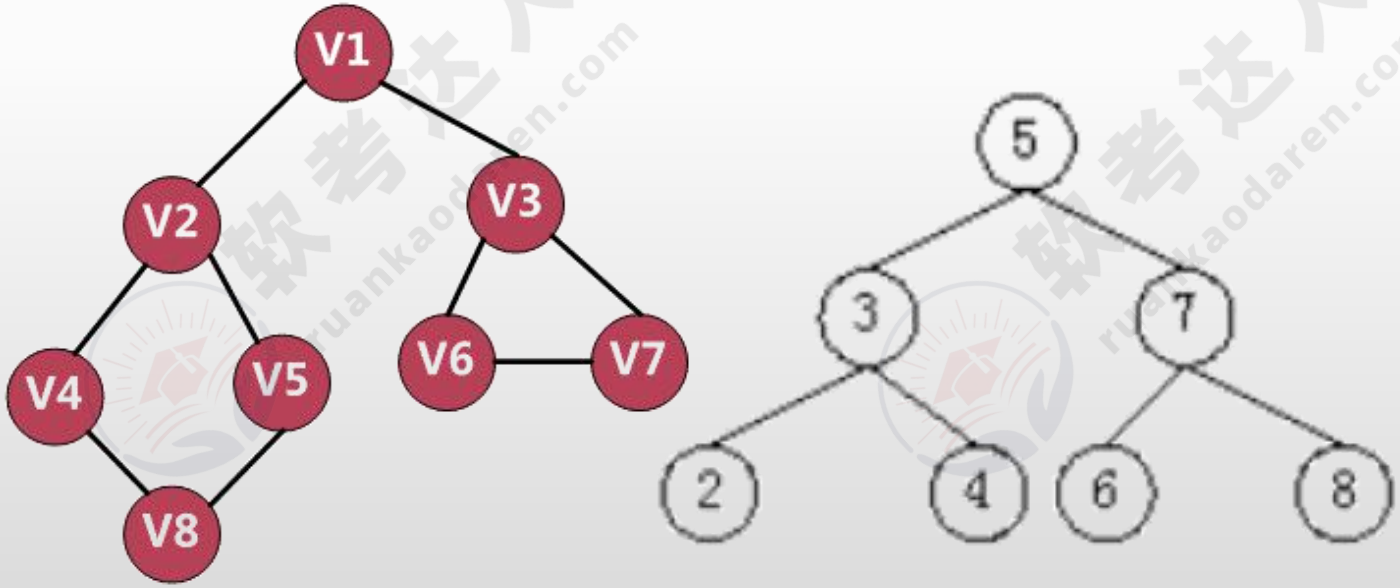
参考答案：A、C、C、B

回溯法

是一种选优搜索法，又称为试探法，按选优条件向前探索，以达到目标。担当探索到某一步时，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称为回溯法。

基本思想类同于：

- 图的深度优先搜索
- 二叉树的后序遍历

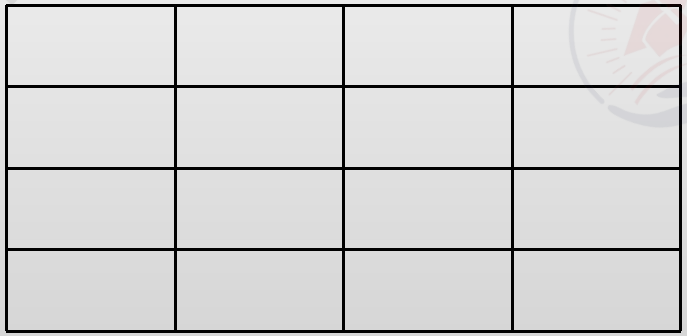


典型真题

n 皇后问题描述为：在一个n×n 的棋盘上摆放 n 个皇后，要求任意两个皇后不能冲突，即任意两个皇后不在同一行、同一列或者同一斜线上。

算法的基本思想如下：

将第 i 个皇后摆放在第 i 行，i 从 1 开始，每个皇后都从第 1 列开始尝试。尝试时判断在该列摆放皇后是否与前面的皇后有冲突，如果没有冲突，则在该列摆放皇后，并考虑摆放下一个皇后；如果有冲突，则考虑下一列。如果该行没有合适的位置，回溯到上一个皇后，考虑在原来位置的下一个位置上继续尝试摆放皇后，……，直到找到所有合理摆放方案。



典型真题

【C 代码】 下面是算法的 C 语言实现。

（1）常量和变量说明

n：皇后数，棋盘规模为n×n

queen[]：皇后的摆放位置数组，

queen[i]表示第i个皇后的位置，

1≤queen[i]≤n

典型真题

（2）C程序

```
#include <stdio.h>
#define n 4
int queen[n+1];
void Show()   /*输出所有皇后摆放方案*/
{
    int i;
    printf("(");
    for(int i=1;i<=n;i++)
    {
        printf("%d",queen[i]);

    }
    printf(")\n");
}
```

典型真题

```
int Place(int j)   /*检查当前列能否放置皇后，不能返回 0，能返回 1*/
{
    int i;
    for(i=1;i<j;i++)     /*检查与已摆放的皇后是否在同一列或同一斜线上*/
    {
        if( （1） ||abs(queen[i]-queen[j])==(j-i))
        {
            return 0;
        }
    }
    return （2） ;
}
```

典型真题

```
void Nqueen(int j)
{
    int i;
    for(i=1;i<=n;i++)
    {
        queen[j]=i;
        if(  (3)  )
        {
            if(j==n)/*如果所有皇后都摆放好,则输出当前摆放方案*/
            {
                Show();
            }
            else/*否则继续摆放下一个皇后*/
            {
                (4) ;
            }
        }
    }
}
```

典型真题

【问题 1】（8分）

根据题干说明，填充 C 代码中的空(1)-（4）。

【问题2】（3分）

根据题干说明和 C 代码，算法采用的设计策略为（5）。

【问题 3】（4分〉

当 n=4 时，有 （6） 种摆放方式，分别为 （7） 。

典型真题

参考答案：

问题1

1. queen[i]==queen[j]或其他等价形式

2. 1

3. Place(j)&&j<=n或其他等价形式

4. Nqueen(j+1)

问题2.

回溯法

问题3.

两种 2413，3142

	2		
			4
1			
		3	

		3	
1			
			4
	2		

技术成就梦想