一、　填空题

1. 下面程序段的所需要的计算时间为（　　$O(n^2)$　　）。

```
int MaxSum(int n, int *a, int &besti, int &bestj)
{
 int sum=0;
 for(int i=1;i<=n;i++){
     int thissum=0;
     for(int j=i;j<=n;j++){
          thissum+=a[j];
         if(thissum>sum)   {
             sum=thissum;
             besti=i;
             bestj=j;
         }
     }
 }
     return sum;
}
```

2. 有 11 个待安排的活动，它们具有下表所示的开始时间与结束时间，如果以贪心算法求解这些活动的最优安排（即为活动安排问题：在所给的活动集合中选出最大的相容活动子集合），得到的最大相容活动子集合为活动（ {1，4，8，11} ）。

i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
S[i]	1	3	0	5	3	5	6	8	8	2	12
f[i]	4	5	6	7	8	9	10	11	12	13	14

6. 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。如果解空间树中从根结点到叶结点的最长路径的长度为 h(n)，则回溯法所需的计算空间通常为（O(h(n)）。

10. 用回溯法解 0/1 背包问题时，计算结点的上界的函数如下所示，请在空格中填入合适的内容：

```
Typep Knap<Typew, Typep>::Bound(int i)
{// 计算上界
   Typew cleft = c - cw;  // 剩余容量
   Typep b = cp;          // 结点的上界
   // 以物品单位重量价值递减序装入物品
   while (i <= n && w[i] <= cleft) {
      cleft -= w[i];
      b += p[i];
      i++;
      }
   // 装满背包
   if (i <= n) （b += p[i]/w[i] * cleft）;
   return b;
}
```

11. 用回溯法解布线问题时，求最优解的主要程序段如下。如果布线区域划分为 n×m 的方格阵列，扩展每个结点需 O(1)的时间，L 为最短布线路径的长度，则算法共耗时 （ O(mn) ），构造相应的最短距离需要（O(L)）时间。

```
for (int i = 0; i < NumOfNbrs; i++) {
         nbr.row = here.row + offset[i].row;
         nbr.col = here.col + offset[i].col;
         if (grid[nbr.row][nbr.col] == 0) {
             // 该方格未标记
             grid[nbr.row][nbr.col]
                = grid[here.row][here.col] + 1;
             if ((nbr.row == finish.row) &&
                (nbr.col == finish.col)) break; // 完成布线
         Q.Add(nbr);}
         }
```

12. 用 结点的每一个儿子所相应的颜色的可用性，则需耗时（渐进时间上限）（O（mn））。

```
Bool Color::OK(int k)
{//
  for(int j=1;j<=n;j++)
    if((a[k][j]= =1)&&(x[j]= =x[k])) return false;
  return true;
}
```

二、

1. 机器调度问题。

问题描述：现在有 n 件任务和无限多台的机器，任务可以在机器上得到处理。每件任务的开始时间为 s_i，完成时间为 f_i，$s_i<f_i$ 。$[s_i，f_i]$为处理任务 i 的时间范围。两个任务 i，j 重叠指两个任务的时间范围区间有重叠，而并非指 i，j 的起点或终点重合。例如：区间[1，4]与区间[2，4]重叠，而与[4，7]不重叠。一个可行的任务分配是指在分配中没有两件重叠的任务分配给同一台机器。因此，在可行的分配中每台机器在任何时刻最多只处理一个任务。最优分配是指使用的机器最少的可行分配方案。

问题实例：若任务占用的时间范围是{[1，4]，[2，5]，[4，5]，[2，6]，[4，7]}，则按时完成所有任务最少需要几台机器？（提示：使用贪心算法）

画出工作在对应的机器上的分配情况。

2. 已知非齐次递归方程：$\begin{cases} f(n)=bf(n-1)+g(n) \\ f(0)=c \end{cases}$，其中，b、c 是常数，g(n)是 n 的某一个函数。则 f(n)的非递归表达式为：$f(n)=cb^n+\sum_{i=1}^{n} b^{n-i}g(i)$。

现有 Hanoi 塔问题的递归方程为：$\begin{cases} h(n)=2h(n-1)+1 \\ h(1)=1 \end{cases}$，求 h(n)的非递归表达式。

解：利用给出的关系式，此时有：b=2, c=1, g(n)=1, 从 n 递推到 1，有：

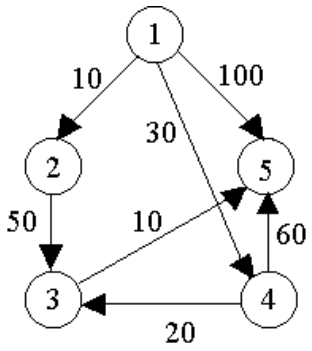
$$\begin{aligned} h(n) &= cb^{n-1}+\sum_{i=1}^{n-1} b^{n-1-i}g(i) \\ &= 2^{n-1}+2^{n-2}+\ldots+2^2+2+1 \\ &= 2^n-1 \end{aligned}$$

3. 单源最短路径的求解。

问题的描述：给定带权有向图（如下图所示）G =(V, E)，其中每条边的权是非负实数。另外，还给定 V 中的一个顶点，称为源。现在要计算从源到所有其它

各顶点的最短路长度。这里路的长度是指路上各边权之和。这个问题通常称为单源最短路径问题。

解法：现采用 Dijkstra 算法计算从源顶点 1 到其它顶点间最短路径。请将此过程填入下表中。



迭代	S	u	dist[2]	dist[3]	dist[4]	dist[5]
初始	{1}	-	10	maxint	30	100
1	{1,2}	2	10	60	30	100
2	{1,2,4}	4	10	50	30	90
3	{1,2,3,4}	3	10	50	30	60
4	{1,2,3,4,5}	5	10	50	30	60

4. 请写出用回溯法解装载问题的函数。

装载问题：有一批共 n 个集装箱要装上 2 艘载重量分别为 c1 和 c2 的轮船，其中集装箱 i 的重量为 wi，且 $\sum_{i=1}^{n} w_i \le c_1 + c_2$。装载问题要求确定是否有一个合理的装载方案可将这 n 个集装箱装上这 2 艘轮船。如果有，找出一种装载方案。

```
解：void backtrack (int i)
    {// 搜索第 i 层结点
       if (i > n)  // 到达叶结点
       更新最优解 bestx,bestw;return;
       r -= w[i];
       if (cw + w[i] <= c) {// 搜索左子树
          x[i] = 1;
          cw += w[i];
          backtrack(i + 1);
          cw -= w[i];       }
       if (cw + r > bestw)  {
```

```
            x[i] = 0;  // 搜索右子树
            backtrack(i + 1);       }
        r += w[i];
    }
```

5. 用分支限界法解装载问题时，对算法进行了一些改进，下面的程序段给出了改进部分；试说明斜线部分完成什么功能，以及这样做的原因，即采用这样的方式，算法在执行上有什么不同。

```
// 检查左儿子结点
  Type wt = Ew + w[i];   // 左儿子结点的重量
      if (wt <= c) {     // 可行结点
         if (wt > bestw) bestw = wt;
         // 加入活结点队列
         if (i < n) Q.Add(wt);
      }
  // 检查右儿子结点
      if (Ew + r > bestw && i < n)
          Q.Add(Ew);     // 可能含最优解
         Q.Delete(Ew);     // 取下一扩展结点
```

解答：斜线标识的部分完成的功能为：提前更新 bestw 值；

这样做可以尽早的进行对右子树的剪枝。具体为：算法 Maxloading 初始时将 bestw 设置为 0，直到搜索到第一个叶结点时才更新 bestw。因此在算法搜索到第一个叶子结点之前，总有 bestw=0，r>0 故 Ew+r>bestw 总是成立。也就是说，此时右子树测试不起作用。

为了使上述右子树测试尽早生效，应提早更新 bestw。又知算法最终找到的最优值是所求问题的子集树中所有可行结点相应重量的最大值。而结点所相应得重量仅在搜索进入左子树是增加，因此，可以在算法每一次进入左子树时更新 bestw 的值。

7. 最长公共子序列问题：给定 2 个序列 X={x1,x2,…,xm}和 Y={y1,y2,…,yn}，找出 X 和 Y 的最长公共子序列。

由最长公共子序列问题的最优子结构性质建立子问题最优值的递归关系。用 c[i][j]记录序列 Xi 和 Yj 的最长公共子序列的长度。其中，Xi={x1,x2,…,xi}；Yj={y1,y2,…,yj}。当 i=0 或 j=0 时，空序列是 Xi 和 Yj 的最长公共子序列。故此时 C[i][j]=0。其它情况下，由最优子结构性质可建立递归关系如下：

$$c[i][j]=\begin{cases}0 & i=0,j=0\\ c[i-1][j-1]+1 & i,j>0;x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0;x_i\neq y_j\end{cases}$$

在程序中，b[i][j]记录 C[i][j]的值是由哪一个子问题的解得到的。

（1） 请填写程序中的空格，以使函数 LCSLength 完成计算最优值的功能。

```
void LCSLength(int m, int n, char *x, char *y, int **c, int **b)
{
        int i, j;
        for (i = 1; i <= m; i++) c[i][0] = 0;
        for (i = 1; i <= n; i++) c[0][i] = 0;
        for (i = 1; i <= m; i++)
           for (j = 1; j <= n; j++) {
              if (x[i]==y[j]) {
                   c[i][j]=c[i-1][j-1]+1; b[i][j]=1;}
             else if (c[i-1][j]>=c[i][j-1]) {
                   c[i][j]=c[i-1][j]; b[i][j]=2;}
              else { c[i][j]=c[i][j-1]; b[i][j]=3; }
              }
}
```

（2） 函数 LCS 实现根据 b 的内容打印出 Xi 和 Yj 的最长公共子序列。请填写程序中的空格，以使函数 LCS 完成构造最长公共子序列的功能（请将 b[i][j]的取值与（1）中您填写的取值对应，否则视为错误）。

```
void LCS(int i, int j, char *x, int **b)
{
      if (i ==0 || j==0) return;
      if (b[i][j]== 1) {
            LCS(i-1, j-1, x, b);
            cout<<x[i];
            }
      else if (b[i][j]== 2) LCS(i-1, j, x, b);
      else LCS(i, j-1, x, b);
}
```

8.对下面的递归算法，写出调用 f(4)的执行结果。

```
    void f(int k)
    { if( k>0 )
            { printf("%d\n  ",k);
              f(k-1);
              f(k-1);
            }
}
```

15、 设计一个算法在一个向量 A 中找出最大数和最小数的元素。

解：设计一个算法在一个向量 A 中找出最大数和最小数的元素。

```
Void maxmin(A,n)
Vector A;
int n;
{
int max,min,i;
```

```
max=A[1];min=A[1];
for(i=2;i<=n;i++)
if(A[i]>max)max=A[i];
else if(A[i]<min)min=A[i];
printf(“max=%d,min=%d\n”,max,min);
}
```

四、设计算法

1. 设有 n 项独立的作业{1,2,…, n},由 m 台相同的机器加工处理。作业 i 所需要的处理时间为 t_i。约定：任何一项作业可在任何一台机器上处理，但未完工前不准中断处理；任何作业不能拆分更小的子作业。

多机调度问题要求给出一种调度方案,使所给的 n 个作业在尽可能短的时间内由 m 台机器处理完。设计算法，并讨论是否可获最优解。

解：对于处理机 j，用 S[j] 表示处理机 j 已有的作业数，用 P[j,k]表示处理机 j 的第 k 个作业的序号 。

1）将作业按照 t[1]≥t[2]≥……≥t[n]排序

2）S[1:m]清零 j←0　　//从第一个处理机开始安排

3) for i←1 to n do　　//安排 n 个作业

　　j←j mod m +1　　//选下一个处理机

　　S[j]←S[j]+1;

　　P[j,S[j]]←i ;

　Repeat

3. 有 n 个物品，已知 n=7, 利润为 P=(10,5,15,7,6,18,3)，重量 W=(2,3,5,7,1,4,1)，背包容积 M=15,物品只能选择全部装入背包或不装入背包，设计贪心算法，并讨论是否可获最优解。

解：定义结构体数组 G，将物品编号、利润、重量作为一个结构体：例如 G[k]={1,10,2}

求最优解，按利润/重量的递减序，有

{5,6,1,6} {1,10,2,5}{6,18,4,9/2} {3,15,5,3} {7,3,1,3}{2,5,3,5/3} {4,7,7,1}

算法

```
procedure  KNAPSACK(P, W, M, X, n)
 //P(1：n)和 W(1；n)分别含有按
   P(i)/W(i)≥P(i+1)/W(i+1)排序的 n 件物品的效益值
   和重量。M 是背包的容量大小，而 x(1：n)是解向量//
 real P(1：n), W(1：n), X(1：n), M, cu;
 integer i, n;
 X←0  //将解向量初始化为零//
 cu←M  //cu 是背包剩余容量//
 for i←1 to n do
   if  W(i)>cu  then  exit  endif
   X(i) ←1
    cu←cu-W(i)
 repeat
 end GREEDY-KNAPSACK
```

根据算法得出的解：

X=（1,1,1,1,1,0,0）获利润 52， 而解

（1,1,1,1， 0， 1,0）可获利润 54

因此贪心法不一定获得最优解。

法分析考试试卷（A 卷）

。

1、对于下列各组函数 f(n)和 g(n)，确定 f(n)=O(g(n))或 $f(n)=\Omega(g(n))$ 或 $f(n)=\theta(g(n))$，并简述理由。（12 分）

(1) $f(n)=\log n^2; g(n)=\log n+5;$

(2) $f(n)=2^n; g(n)=100n^2;$

(3) $f(n)=2^n; g(n)=3^n;$

解：简答如下：

（1）$\log n^2=\theta(\log n+5)$，（2）$2^n=\Omega(100n^2)$，（3）$2^n=o(3^n)$

2、试用分治法实现有重复元素的排列问题：设 $R=\{r_1,r_2,...,r_n\}$ 是要进行排列的 n 个元素，其中元素 $r_1,r_2,...,r_n$ 可能相同，试计算 R 的所有不同排列。（13 分）

解：解答如下：

```
Template<class Type>
void Perm(Type list[],int k,int m)
{
   if(k= =m){
      for(int i=0;i<=m;i++) cout<<list[i]; ……..（4 分）
      cout<<endl;
   }
   else for(int i=k;i<=m;i++)
        if(ok(list,k,i)){
          swap(list[k],list[i]);
          Perm(list,k+1,m);
          swap(list[k],list[i]); ……..（8 分）
          };
  }
```

其中 ok 用于判别重复元素。

```
Template<class>
int ok(Type list[],int k,int i)
{
   if(i>k)
       for(int t=k;t<I;t++)
          if(list[t]= =list[i]) return 0;
   return 1;
```

}……..（13 分）

3、试用分治法对一个有序表实现二分搜索算法。（12 分）

解：解答如下：

```
Template<class>
int BinarySearch(Type a[],const Type& x,int n)
{//假定数组 a[]已按非递减有序排列，本算法找到 x 后返回其在数组 a[]中的位置，//否则返回-1
 int left=0,right=n-1;
 while(left<=right){
    int middle=(left+right)/2; ……..（4 分）
    if(x= =a[middle]) return middle+1;
    if(x>a[middle]) left=middle+1; ……..（8 分）
    else right=middle-1;
  }
  return -1;
 }……..（12 分）
```

4、试用动态规划算法实现 0-1 闭包问题。（15 分）

解：解答如下：

```
Template<class>
void Knapsack(Type v,int w,int c,int n,Type **m)
{
   Int jMax=min(w[n]-1,c);
   for(int j=0;j<=jMax;j++) m[n][j]=0;
   for(int j=w[n];j<=c;j++) m[n][j]=v[n]; ……..（5 分）
   for(int i=n-1;i>1;i--){
      jMax=min(w[i]-1,c);
      for(int j=0;j<=jMax;j++) m[i][j]=m[i+1][j];
      for(int j=w[i];j<=c;j++) m[i][j]=max(m[i+1][j],m[i+1][j-w[i]]+v[i]); ……..（8 分）
   };
   m[1][c]=m[2][c];
   if(c>=w[1]) m[1][c]=max(m[1][c],m[2][c-w[1]]+v[1]); ……..（10 分）
}
Template<class>
Void Traceback(Type **m,int w,int c,int n,int x)
{
   for(int i=1;i<n;i++)
      if(m[i][c]= =m[i+1][c]) x[i]=0; ……..（12 分）
      else { x[i]=1,c-=w[i];};
   x[n]=(m[n][c])?1:0;
}……..（15 分）
```

5、试用贪心算法求解下列问题：将正整数 n 分解为若干个互不相同的自然数之和，使这些自然数的乘积最大。（15 分）

解：解答如下：

```
void dicomp(int n,int a[])
{
  k=1;
  if(n<3){ a[1]=0;return;};
  if(n<5){ a[k]=1;a[++k]=n-1;return;}; ……..（5 分）
  a[1]=2;n-=2;
while(n>a[k]){
   k++;
   a[k]=a[k-1]+1;
   n-=a[k];
  };……..（10 分）
  if(n= =a[k]){ a[k]++;n--;};
  for(int i=0;i<n;i++) a[k-i]++;
}……..（15 分）
```

6、试用动态规划算法实现最大子矩阵和问题：求 $m \times n$ 矩阵 A 的一个子矩阵，使其各元素之各为最大。（15 分）

解：解答如下：

```
int MaxSum2(int m,int n,int **a)
{
  int sum=0;
  int *b=new int[n+1];
  for(int i=1;i<=m;i++){
    for(int k=1;k<=n;k++) b[k]=0; ……..（5 分）
    for(int j=i;j<=m;j++){
      for(int k=1;k<=n;k++) b[k]+=a[j][k];
      int max=MaxSum(n,b);
      if(max>sum) sum=max;
    }
  }
  return sum; ……..（10 分）
 }

 int MaxSum(int n,int *a)
 {
    int sum=0,b=0;
    for(int i=1;i<=n;i++){
      if(b>0) b+=a[i];
      else b=a[i];
      if(b>sum) sum=b;
```

```
        }
        Return sum; ……（15 分）
    }
```

7、试用回溯法解决下列整数变换问题：关于整数 i 的变换 f 和 g 定义如下：$f(i)=3i; g(i)=\lfloor i/2 \rfloor$。对于给定的两个整数 n 和 m，要求用最少的变换 f 和 g 变换次数将 n 变为 m。（18 分）

解：解答如下：

```
    void compute()
    {
      k=1;
      while(!search(1,n)){
        k++;
        if(k>maxdep) break;
        init();
      };……（6 分）
      if(found) output();
   else cout<<”No Solution!”<<endl;
 }……（9 分）

 bool search(int dep,int n)
{
  If(dep>k) return false;
  for(int i=0;i<2;i++){
     int n1=f(n,i);t[dep]=i; ……（12 分）
     if(n1= =m||search(dep+1,n1)){
       Found=true;
       Out();
       return true;
     }
  return false; ……（18 分）
}
```

《算法设计与分析》考试试卷

排序和查找是经常遇到的问题。按照要求完成以下各题：（20 分）

（1） 对数组 A={15，29，135，18，32，1，27，25，5}，用快速排序方法将其排成递减序。

（2） 请描述递减数组进行二分搜索的基本思想，并给出非递归算法。

（3） 给出上述算法的递归算法。

（4） 使用上述算法对（1）所得到的结果搜索如下元素，并给出搜索过程：18，31，135。

答案：（1）第一步：15 29 135 18 32 1 27 25 5

第二步：29 135 18 32 27 25 15 1 5 【1 分】

第三步：135 32 29 18 27 25 15 5 1 【1 分】

第四步：135 32 29 27 25 18 15 5 1 【1 分】

（2）基本思想：首先将待搜索元素 v 与数组的中间元素 $A\left[\frac{n}{2}\right]$ 进行比较，如果 $v > A\left[\frac{n}{2}\right]$，则在前半部分元素中搜索 v；若 $v = A\left[\frac{n}{2}\right]$，则搜索成功；否则在后半部分数组中搜索 v。【2 分】

非递归算法：

输入：递减数组 A[left:right]，待搜索元素 v。【1 分】

输出：v 在 A 中的位置 pos，或者不在 A 中的消息（-1）。【1 分】

步骤：【3 分】

```
int BinarySearch(int A[],int left,int right,int v)
{
    int mid;
    while (left<=right)
    {
        mid=int((left+right)/2);
        if (v==A[mid]) return mid;
        else if (v>A[mid]) right=mid-1;
        else left=mid+1;
    }
    return -1;
}
```

（3）递归算法：

输入：递减数组 A[left:right]，待搜索元素 v。【1 分】

输出：v 在 A 中的位置 pos，或者不在 A 中的消息（-1）。【1 分】

步骤：【3 分】

```
int BinarySearch(int A[],int left,int right,int v)
{
    int mid;
    if (left<=right)
    {
        mid=int((left+right)/2);
        if (v==A[mid]) return mid;
```

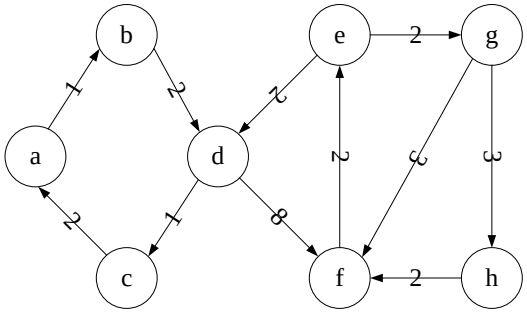
```
        else if (v>A[mid]) return BinarySearch(A,left,mid-1,v);
        else return BinarySearch(A,mid+1,right,v);
    }
    else
        return -1;
}
```

（4）搜索 18：首先与 27 比较，18<27，在后半部分搜索；再次与 18 比较，搜索到，返回 5。【1 分】

搜索 31：首先与 27 比较，31>27，在前半部分搜索；再次 32 比较，31<32，在后半部分搜索，与 29 比较，31>29，此时只有一个元素，未找到，返回-1。【2 分】

搜索 135：首先与 27 比较，135>27，在前半部分搜索；再次 32 比较，135>32，在前半部分搜索；与 135 比较，相同，返回 0。【2 分】

二、 对于下图使用 Dijkstra 算法求由顶点 a 到顶点 h 的最短路径。（20 分）。



答案：用 V_1表示已经找到最短路径的顶点，V_2表示与 V_1中某个顶点相邻接且不在 V_1中的顶点；E_1表示加入到最短路径中的边，E_2为与 V_1中的顶点相邻接且距离最短的路径。【1 分】

步骤	V_1	V_2	E_1	E_2
1.	{a}	{b}	{}	{ab}
2.	{a,b}	{d}	{ab}	{bd}
3.	{a,b,d}	{c,f}	{ab,bd}	{dc,df}
4.	{a,b,d,c}	{f}	{ab,bd}	{df}
5.	{a,b,c,d,f}	{e}	{ab,bd,dc,df}	{fe}
6.	{a,b,c,d,e,f}	{g}	{ab,bd,dc,df,fe}	{eg}
7.	{a,b,c,d,e,f,g}	{h}	{ab,bd,dc,df,fe,eg}	{gh}
8.	{a,b,c,d,e,f,g,h}	{}	{ab,bd,de,df,fe,eg,gh}	{}

【以上每步 2 分】

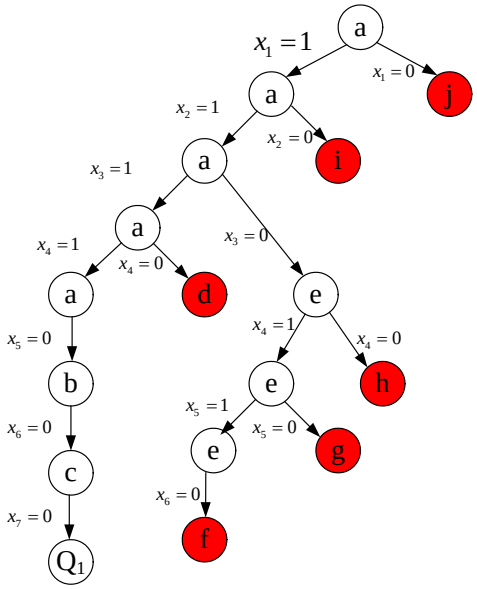
结果：从 a 到 h 的最短路径为 $a \to b \to d \to f \to e \to g \to h$，权值为 18。【1 分】

三、 假设有 7 个物品，它们的重量和价值如下表所示。若这些物品均不能被分割，且背包容量 M＝150，使用回溯方法求解此背包问题。请写出状态空间搜索树（20 分）。

物品	A	B	C	D	E	F	G
重量	35	30	60	50	40	10	25
价值	10	40	30	50	35	40	30

答案：求所有顶点对之间的最短路径可以使用 Dijkstra 算法，使其起始节点从 a 循环到 h，每次求起始节点到其他节点的最短路径，最终可以求得所有顶点对之间的最短路径。【2 分】

三、按照单位效益从大到小依次排列这 7 个物品为：FBGDECA。将它们的序号分别记为 1～7。则可生产如下的状态空间搜索树。其中各个节点处的限界函数值通过如下方式求得：【排序 1 分】



【状态空间搜索树及其计算过程 17 分，每个节点 1 分】

a. $40+40+30+50+35\times\frac{150-115}{40}=190.625$ $(1,1,1,1,\frac{7}{8},0,0)$

b. $40+40+30+50+30\times\frac{150-115}{60}=177.5$ $(1,1,1,1,0,\frac{7}{12},0)$

c. $40+40+30+50+10=170$ $(1,1,1,1,0,0,1)$

d. $40+40+30+35+30\times\frac{150-105}{60}=167.5$ $(1,1,1,0,1,\frac{3}{4},0)$

e. $40+40+50+35+30\times\frac{150-130}{60}=175$ $(1,1,0,1,1,\frac{1}{3},0)$

f. $40+40+50+35+10\times\frac{150-130}{35}=170.71$ $(1,1,0,1,1,0,\frac{4}{7})$

g. $40+40+50+30=160$ $(1,1,0,1,0,1,0)$

h. $40+40+35+30+10\times\frac{150-140}{35}=146.85$ $(1,1,0,0,1,1,\frac{2}{7})$

i. $40+30+50+35+30\times\frac{150-125}{60}=167.5$ $(1,0,1,1,1,\frac{5}{12},0)$

j. $40+30+50+35+30\times\frac{150-145}{60}=157.5$ $(0,1,1,1,1,\frac{1}{12},0)$

在 Q_1 处获得该问题的最优解为 $(1,1,1,1,0,0,1)$，背包效益为 170。即在背包中装入物品 F、B、G、D、A 时达到最大效益，为 170，重量为 150。【结论 2 分】

四、 已知 $A_k=(a_{ij}^{(k)})_{r_i*r_{i+1}}$，$k$=1，2，3，4，5，6，$r_1$=5，$r_2$=10，$r_3$=3，$r_4$=12，$r_5$=5，$r_6$=50，$r_7$=6，求矩阵链积 $A_1\times A_2\times A_3\times A_4\times A_5\times A_6$的最佳求积顺序。（要求：给出计算步骤）（20分）

答案：使用动态规划算法进行求解。

求解矩阵为：【每个矩阵 18 分】

	1	2	3	4	5	6
1	0	150	330	405	1655	2010
2		0	360	330	2430	1950
3			0	180	930	1770
4				0	3000	1860
5					0	1500
6						0

	1	2	3	4	5	6
1	0	1	2	2	4	2
2		0	2	2	2	2
3			0	3	4	4
4				0	4	4
5					0	5
6						0

因此，最佳乘积序列为（A_1A_2）（（A_3A_4）（A_5A_6））， 共执行乘法 2010 次。【结论 2 分】

五、回答如下问题：（20 分）

（1） 什么是算法？算法的特征有哪些？

（2） 什么是 P 类问题？什么是 NP 类问题？请描述集合覆盖问题的近似算法的基本思想。

答案：（1）算法是解决某类问题的一系列运算的集合【2 分】。具有有穷行、可行性、确定性、0 个或者多个输入、1 个或者多个输出【8 分】。

（2）用确定的图灵机可以在多项式实践内可解的判定问题称为 P 类问题【2 分】。用不确定的图灵机在多项式实践内可解的判定问题称为 P 类问题。【2 分】

集合覆盖问题的近似算法采用贪心思想：对于问题<X,F>，每次选择 F 中覆盖了尽可能多的未被覆盖元素的子集 S，然后将 U 中被 S 覆盖的元素删除，并将 S 加入 C 中，最后得到的 C 就是近似最优解。【6 分】

分别用贪心算法、动态规划法、回溯法设计 0-1 背包问题。要求：说明所使用的算法策略；写出算法实现的主要步骤；分析算法的时间。

1、 **分治法的基本思想**是将一个规模为 n 的问题分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题相同；对这 k 个子问题分别求解。如果子问题的规模仍然不够小，则再划分为 k 个子问题，如此递归的进行下去，直到问题规模足够小，很容易求出其解为止；将求出的小规模的问题的解合并为一个更大规模的问题的解，自底向上逐步求出原来问题的解。

2、 **“最优化原理”**用数学化的语言来描述：假设为了解决某一优化问题，需要依次作出 n 个决策 D1，D2，…，Dn，如若这个决策序列是最优的，对于任何一个整数 k，1 < k < n，不论前面 k 个决策是怎样的，以后的最优决策只取决于由前面决策所确定的当前状态，即以后的决策 Dk+1，Dk+2，…，Dn 也是最优的。

3、 某个问题的最优解包含着其子问题的最优解。这种性质称为**最优子结构性质**。

4、 **回溯法的基本思想**是在一棵含有问题全部可能解的状态空间树上进行深度优先搜索，解为叶子结点。搜索过程中，每到达一个结点时，则判断该结点为根的子树是否含有问题的解，如果可以确定该子树中不含有问题的解，则放弃对该子树的搜索，退回到上层父结点，继续下一步深度优先搜索过程。在回溯法中，并不是先构造出整棵状态空间树，再进行搜索，而是在搜索过程，逐步构造出状态空间树，即边搜索，边构造。

5、 P(Polynomial 问题)：也即是多项式复杂程度的问题。

NP 就是 Non-deterministic Polynomial 的问题，也即是多项式复杂程度的非确定性问题。

NPC(NP Complete)问题，这种问题只有把解域里面的所有可能都穷举了之后才能得出答案，这样的问题是 NP 里面最难的问题，这种问题就是 NPC 问题。

（1）贪心算法 O（nlog（n））

- 首先计算每种物品单位重量的价值 Vi/Wi，然后，依贪心选择策略，将尽可能多的单位重量价值最高的物品装入背包。若将这种物品全部装入背包后，背包内的物品总重量未超过 C，则选择单位重量价值次高的物品并尽可能多地装入背包。依此策略一直地进行下去，直到背包装满为止。
- 具体算法可描述如下：

```
void Knapsack(int n,float M,float v[],float w[],float x[])
{Sort(n,v,w);
int i;
for (i=1;i<=n;i++) x[i]=0;
float c=M;
for (i=1;i<=n;i++)
{if (w[i]>c) break;
x[i]=1;
c-=w[i];
}
if (i<=n) x[i]=c/w[i];
}
```

（2）动态规划法 O(nc)

m(i，j)是背包容量为 j，可选择物品为 i，i+1，…，n 时 0-1 背包问题的最优值。由 0-1

背包问题的最优子结构性质，可以建立计算 m(i，j)的递归式如下。

$$m(i,j)=\begin{cases}\max\{m(i+1,j),m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

$$m(n,j)=\begin{cases}v_n & j\geq w_n\\ 0 & 0\leq j<w_n\end{cases}$$

```
void KnapSack(int v[],int w[],int c,int n,int m[][11])
{int jMax=min(w[n]-1,c);
for (j=0;j<=jMax;j++)     /*m(n,j)=0     0=<j<w[n]*/
m[n][j]=0;
for (j=w[n];j<=c;j++)      /*m(n,j)=v[n]     j>=w[n]*/
m[n][j]=v[n];
for (i=n-1;i>1;i--)
{ int jMax=min(w[i]-1,c);
for (j=0;j<=jMax;j++)     /*m(i,j)=m(i+1,j)     0=<j<w[i]*/
    m[i][j]=m[i+1][j];
for (j=w[i];j<=c;j++)/*m(n,j)=v[n]     j>=w[n]*/
    m[i][j]=max(m[i+1][j],m[i+1][j-w[i]]+v[i]);
}
m[1][c]=m[2][c];
if(c>=w[1])
m[1][c]=max(m[1][c],m[2][c-w[1]]+v[1]);
}
```

（3）回溯法　$O(2^n)$

cw:当前重量　　cp:当前价值　　bestp：当前最优值

```
void backtrack(int i)
 //回溯法  i 初值 1
{   if(i > n)   //到达叶结点
          { bestp = cp;          return;          }
   if(cw + w[i] <= c) //搜索左子树
      { cw += w[i];
        cp += p[i];
       backtrack(i+1);
      cw -= w[i];
        cp -= p[i];
      }
      if(Bound(i+1)>bestp)
//搜索右子树
      backtrack(i+1);
 }
```

七、算法设计题（本题 10 分）

为了尽可能地逼近目标，我们选取的贪心策略为：**每一步总是选择一个使剩下的数最小的数字删去，即按高位到低位的顺序搜索，若各位数字递增，则删除最后一个数字，否则删除第一个递减区间的首字符。**然后回到串首，按上述规则再删除下一个数字。重复以上过程 s 次，剩下的数字串便是问题的解了。

具体算法如下：

```
输入s, n;
while（ s > 0 ）
 {   i=1;   //从串首开始找
    while (i < length(n)) && (n[i]<n[i+1])
     {i++;}
      delete(n,i,1); //删除字符串n的第i个字符
     s--;
  }
while (length(n)>1)&& (n[1]='0')
  delete(n,1,1); //删去串首可能产生的无用零
 输出n;
```

2．(10 分) 下面是求解矩阵链乘问题的动态规划算法。

矩阵链乘问题：给出 n 个矩阵 $M_1, M_2, \cdots, M_n$, M_i 为 $r_i \times r_{i+1}$阶矩阵，i=1, 2, …, n，求计算 $M_1M_2\cdots M_n$所需的最少数量乘法次数。

记 $M_{i,j}=M_iM_{i+1}\cdots M_j$, i<=j。设 C[i, j], 1<=i<=j<=n, 表示计算 $M_{i,j}$的所需的最少数量乘法次数，则

$$C[i, j]=\begin{cases}0 & , i = j \\ \min\limits_{i<k\le j}\{C[i, k-1]+C[k, j]+r_i r_k r_{j+1}\} & , i < j\end{cases}$$

算法 MATCHAIN

输入：矩阵链长度 n, n 个矩阵的阶 r[1..n+1], 其中 r[1..n]为 n 个矩阵的行数，r[n+1]为第 n 个矩阵的列数。

输出：n 个矩阵链乘所需的数量乘法的最少次数。

```
for i=1 to n    C[i, i]=_____（1）_____
  for d=1 to n-1
    for i=1 to n-d
      j=_____(2)_____
      C[i, j]= ∞
```

```
        for k=i+1 to j
            x= _____(3)_____
            if x<C[i, j] then
                _____(4)_____=x
            end if
        end for
    end for
end for
return _____(5)_____
end MATCHAIN
```

1. 一个旅行者要驾车从 A 地到 B 地，A、B 两地间距离为 s。A、B 两地之间有 n 个加油站，已知第 i 个加油站离起点 A 的距离为 d_i 公里，$0=d_1<d_2<\cdots<d_n\le s$，车加满油后可行驶 m 公里，出发之前汽车油箱为空。应如何加油使得从 A 地到 B 地沿途加油次数最少？给出用贪心法求解该最优化问题的贪心选择策略，写出求该最优化问题的最优值和最优解的贪心算法，并分析算法的时间复杂性。

3. (1) i>=1　　(2)k[i]+1　　(3) 1
   (4) i+1　　(5) k[i]=0　　(6) tag[x, y]=0
   (7) x=x-dx[k[i]];　y=y-dy[k[i]]

一. 算法设计题：

1. 贪心选择策略：从起点的加油站起每次加满油后不加油行驶尽可能远，直至油箱中的油耗尽前所能到达的最远的油站为止，在该油站再加满油。

   算法 MINSTOPS

   输入：A、B 两地间的距离 s，A、B 两地间的加油站数 n，车加满油后可行驶的公里数 m，存储各加油站离起点 A 的距离的数组 d[1..n]。

   输出：从 A 地到 B 地的最少加油次数 k 以及最优解 x[1..k]（x[i]表示第 i 次加油的加油站序号），若问题无解，则输出 no solution。

```
d[n+1]=s;  //设置虚拟加油站第 n+1 站。
for i=1 to n
  if d[i+1]-d[i]>m then
    output “no solution”; return  //无解，返回
  end if
end for
```

```
    k=1; x[k]=1    //在第 1 站加满油。
    s1=m    //s1 为用汽车的当前油量可行驶至的地点与 A 点的距离
    i=2
    while s1<s
       if d[i+1]>s1 then    //以汽车的当前油量无法到达第 i+1 站。
          k=k+1; x[k]=i    //在第 i 站加满油。
          s1=d[i]+m        //刷新 s1 的值
       end if
       i=i+1
    end while
    output k, x[1..k]
MINSTOPS
```

最坏情况下的时间复杂性：$\Theta(n)$