技巧提示：

对于大多数的动态规划问题来说，尽可能地减少状态的存储量，是优化存贮空间的一个基本思考方法。对于状态的存贮，还有一种经常使用的处理方法，就是对状态进行“滚动式”存贮，这一方法的基本思想是：对于先前（上一阶段）产生的状态，如果后面的处理过程中不再需要使用时，就可以用后面的状态去覆盖先前的状态，或者释放其所占有的空间，以达到空间优化的效果。

5、最佳航线问题。你在加拿大航空公司组织的一次竞赛中获奖，奖品是一张免费机票，可在加拿大旅行，从最西的一个城市出发，单方向从西向东经若干城市到达最后一个城市（必须到达最东的城市），然后在单方向从东向西飞回起点（可途径若干城市）。除起点城市外，任何城市只能访问一次。起点城市被访问两次：出发一次，返回一次除指定的航线外，不允许乘其他航线，也不允许使用其他交通工具。求解的问题是：给定城市表列及两城市之间的直通航线，亲请找出一条旅行航线，在满足上述限制条件下，使访问的城市尽可能多。

【分析】本题可以采用动态规划方法求解。假设城市已按从西到东编号，数组dist[i,j]表示城市i到城市n,再到城市j的所有可行方案中，最多能够访问的城市数目。这时可以得到一个递推关系式（状态转移方程）为：

```
dist[n,n]=1;
dist[i,j]=dist[j,i];
dist[i,j]=max(dist[i,k])+1;  (j<i,j<k<=n,且存在航线k─j)
```

如果要记录中间过程，就必须再开辟一个二维数组，就会导致程序可完成的数据规模降低。而采用递归求出路径后再输出，编程并未增加多少难度，而可处理的数据量却大大增加了。求路径的过程完全按照倒推的方法，利用dist数组得出往返的路线。

对于大多数用动态规划方法解决的问题，都可以采用递归输出的方法，以降低问题的空间需求。下面给出本题的源程序。

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
{$M 16384,0,655360}
program  ex7-12;{ 最佳航线问题}
var a, {航线表}
dist:array[1..100,1..100] of byte;
   {dist[i,j]表示从i出发到n再到j最多可经过的城市数}
   n:integer;
procedure readp;{读入数据}
var f:text;
   st:string;    i,j:integer;
begin
    write('File name:');   readln(st);   assign(f,st);   reset(f);
    readln(f,n);
    for i:=1 to n do
       for j:=1 to n do
             read(f,a[i,j]);
    close(f);
end;
procedure print(i,j:integer);{递归输出往返路线}
var k:byte;
begin
    if (i=1) and (j=1) then
         exit;
    if i<j then  {输出往程线路}
         begin
           write(i,'-->');
           for k:=i+1 to n do
              if (a[k,i]=1) and (dist[k,j]+1=dist[i,j]) then  begin
                    print(k,j);
                    break;
                end;
       end
```

```
        else {输出返程线路}
             begin
              for k:=j+1 to n do
                 if (a[k,j]=1) and (dist[k,i]+1=dist[i,j]) then  begin
                         print(i,k);
                         break;         end;
              write(j,'-->');  end;  end;
    procedure main; {动态规划求解}
    var i,j,k:integer;
    begin
       fillchar(dist,sizeof(dist),0);
       dist[n,n]:=1;
       for i:=n downto 1 do
          for j:=i-1 downto 1 do
             begin
                for k:=j+1 to n do
                   if (a[j,k]=1) and (dist[i,k]<>0) and (dist[i,j]<dist[i,k]+1)
                      then dist[i,j]:=dist[i,k]+1;
                dist[j,i]:=dist[i,j];
             end;
       k:=2;
       for i:=3 to n do {寻找最佳航线}
          if (a[1,i]=1) and (dist[1,i]>dist[1,k]) then
             k:=i;
       print(1,k); {输出途径城市}
       writeln(1);
       writeln('Max=',dist[1,k]);
    end;
    begin {主程序}
       readp;
       main;
end.
```

【赛前优化训练】

6、最长非降子序列问题。给定一个由N个正整数组成的序列，从中间删除M个数，使剩下的序列非降，求M的最小值（N<=1000）。

7、资源分配问题。现在有某种资源共n个单位，准备将它们分配到m个项目上。设在第i个项目上分配j个单位的资源可以获得的收益为a[i,j]，求可以获得的最大总收益。

8、一个n*m的矩阵，每一个方格上都有一个值，每列上最多可以取一个数值，求数值和最大一串连续的列。

【参考答案】

6、通过分析题意，求M的最小值，实际上是求包含在这N个数中的最长非降序列。对于这一类最优化问题的求解，考虑用贪心策略求解，无法找到一种能够证明正确的贪心标准；考虑采用搜索算法，问题的数据规模N的值又太大，肯定无法满足问题的时间效率。但是可以从搜索中发现，每当搜索到第i个数（即不删除第i个数），再继续搜索时，获得的从第i个数算起的最长非降子序列，总是相同的，因此，就可以考虑采用动态规划的方法求解，假设a[i]表示第i到第n个数中，保留第i个数可获得的最长非降子序列的长度，这样就可以用倒推的方法，依次求出a[n],a[n-1]　　a[1]。而max(a[i]) (1<=i<=n)即为原问题的解，其动态转移方程为：a[i]=max(a[j]+1,1) (i<j<=n,且第j个数要大于或等于i第个数)。

为更方便于编程，还可以在序列中加上两个数，编号分别为佳0和N+1，数值分别为maxint和0，a[0]便是所求的结果。

参考程序：

```
{$A+,B-,D+,E+,F-,G-,I+,L+,N-,O-,P-,Q-,R-,S+,T-,V+,X+}
```

```
{$M 16384,0,655360}
program ex6;{求解最长非降子序列}
var a, {记录n个整数}
   b:array[0..1001] of word; {b[i]表示第i到第n个数中非降子序列的最多元素个数}
   n:word;
procedure readp; {读入数据}
var f:text;
   st:string;
   i:integer;
begin
    write('File name:');
    readln(st);
    assign(f,st);
    reset(f);
    readln(f,n);
    for i:=1 to n do
       read(f,b[i]);
    close(f);
end;
procedure main; {求解并输出}
var i,j:word;
begin
    b[0]:=0;
    b[n+1]:=65535; {设置边界}
    {逆推求解}
    for i:=n downto 0 do
       for j:=i+1 to n+1 do
          if (b[j]>=b[i]) and (a[j]>=a[i]) then
             a[i]:=a[j]+1; {寻找中间过程输出}
    i:=0;
    while i<n+1 do
        begin
           j:=i+1;
           while (b[j]<b[i]) or (a[j]+1<a[i]) do
                inc(j);
           if i<>0 then write(b[i]:5);
           i:=j;
        end;
    writeln;
    writeln('Max=',a[0]-1);
end;
begin
    readp;
    main;
end.
```

7、用两个状态变量描述问题的状态：状态[i,j]表示在1至i这i个项目上共分配总量为j的资源，要求解这i个项目上能获得的最大总收益。这样原问题状态就是（m,n）,而状态之间存在转化关系（设p[i,j]表示子问题(i,j)的解）： p[i,j] = max(p[i-1,j-k])+a[i,k] (0≤k≤j)。问题的终止条件为当i=1时，p[1,j]=a[1,j]。

```
program ex7;
const
 nn=10;
 mm=100;
```

```
type
  arr=array[1..nn,0..mm] of real;
  brr=array[1..nn,0..mm] of 0..mm;
  rrr=array[1..nn] of 0..mm;
var
  a,c:arr;
  b:brr;
  d:rrr;
  i,j,k:integer;
  n:1..nn;
  m:0..mm;

procedure readn;
var
  t:text;
  st:string;
begin
  write('File name:');readln(st);
  assign(t,st);
  reset(t);
  readln(t,n,m);
  for i:=1 to n do
    begin
      for j:=0 to m do
        read(t,c[i,j]);
      readln(t);
    end;
  close(t);
end;

begin
  readn;
  for i:=0 to m do
    begin
      a[1,i]:=c[1,i];
      b[1,i]:=i;
    end;
  for i:=2 to n do
    for j:=0 to m do
      begin
        a[i,j]:=0;
        for k:=0 to j do
          if a[i,j]<a[i-1,k]+c[i,j-k] then
            begin
              a[i,j]:=a[i-1,k]+c[i,j-k];
              b[i,j]:=j-k;
            end;
      end;
  writeln('Total=',a[n,m]:1:2);
  j:=m;
  for i:=n downto 1 do
    begin
      d[i]:=b[i,j];
      j:=j-b[i,j];
```

```
    end;
  for i:=1 to n do
    writeln(i,': ',d[i]);
end.
```

8、这是一个经典的动态规划题目。首先对于每一列如果我们要取其值的话，肯定取这一列中的最大的值。这样就转化为在一个数值串中，求一个连续的数值和最大的子串。

我们很容易写出规划方程：

f[I]：=max（f[I-1]，f[I-1]+a[I]）

边界：f[0]：=0

参考程序：

```
program Get_max_one;
const
  inf   ='input.txt';
  outf  ='output.txt';
var
  sum   :array[1..100] of integer;
  max   :integer;
  n,m   :integer;

procedure init;
var
  x,i,j   :integer;
begin
  assign(input,inf);
  reset(input);
  readln(n,m);
  for i:=1 to m do sum[i]:=-101;
  for i:=1 to n do begin
    for j:=1 to m do begin
      read(x);
      if x>sum[j] then sum[j]:=x;
    end;
  end;
  close(input);
end;

procedure main;
var
  i,now     :integer;
begin
  now:=0;
  for i:=1 to m do begin
    now:=now+sum[i];
    if now>max then max:=now;
    if now<0 then now:=0;
  end;
end;

procedure out;
begin
  assign(output,outf);
  rewrite(output);
  writeln(max);
```

```
  close(output);
end;

Begin
  init;
  main;
  out;
End.
```