Dynamic Programming

# 基础动态规划

## 第3课 经典题目选讲

# 例题1：回文词 1036

回文词是一种对称的字符串——也就是说，一个回文词，从左到右读和从右到左读得到的结果是一样的。任意给定一个字符串，通过插入若干字符，都可以变成一个回文词。你的任务是写一个程序，求出将给定字符串变成回文词所需插入的最少字符数。

比如字符串“Ab3bd”，在插入两个字符后可以变成一个回文词（“dAb3bAd”或“Adb3bdA”）。

给出一个单词，将它变为回文词，求需要插入的最少字符数。

## 解法：

假设S是给出的字符串，S'是S的逆串；

求出S与S'的**最长公共子序列**的长度；

最后用字符串的长度减去最长公共子序列的长度即是解。

# 例题2：锁妖塔1011

小A爬锁妖塔，该塔的建造很特别,塔总共有n层,但是高度却不相同,这造成了小A爬过每层的时间也不同.小A会用仙术,每用一次可以让他**向上跳一层或两层**,但是**每次跳跃后必须爬过至少一层**才能再次跳跃(你可以认为小A需要跳两次一层才休息),小A想用最短的时间爬到塔顶,请你告诉他最短时间是多少.他可以最后跳到塔外即超过塔高.

**输入格式**

第一行一个数n (n<=10000),表示塔的层数.

接下来的n行每行一个数(<=100),表示从下往上每层的高度.

**输出格式**

一个数,表示最短时间

# 例题2：锁妖塔1011

阶段：从下往上按每层楼依次讨论。

状态：p[i]表示到达第i层的最短时间，并且到达第i层的方式是爬。
t[i]表示到达第i层的最短时间，并且到达第i层的方式是跳。

决策：到达第i层的方式采用爬还是采用跳。

情况1. 到达第i层的方式是爬：

那么到达第i-1层的方式可以使爬也可以是跳，从两者中选最小。

**p[i]=min{p[i-1],t[i-1]}+a[i]**

情况2. 到达第i层的方式是跳：

那么可以从第i-1层起跳，也可以从第i-2层起跳。并且到达第i-1层和i-2层的方式只能选爬（到第i层是跳），所以在两者中选最小。

**t[i]=min{p[i-1],p[i-2]}**

最后在**p[n]**和**t[n]**中选最小者做结果

# 例题2：锁妖塔1011

阶段：从下往上按每层楼依次讨论。

状态：f[i]表示到达第i层所需最小时间

决策：

行动的方式有三种

1. **爬一层；** 等价于一次**跨一层**，耗时为爬行那一层的时间
2. **跳一层，爬一层；** 等价于一次**跨两层**，耗时为爬行那一层的时间
3. **跳两层，爬一层；** 等价于一次**跨三层**，耗时为爬行那一层的时间

**于是，我们到第达i的方式有三种：**

1. **从i-1层一口气跨来，**
2. **从i-2层一口气跨来，**
3. **从i-3层一口气跨来，**

**我们从这三种方式中，选最优的一种即可。**

方程：**f[i]=min{f[i-1],f[i-2],f[i-3]}+a[i]**

# 例题3：护卫队(1006) 分组类DP

## 分析：如何分组

第1辆车通过的最短时间

time[1]=length/speed[1]

前2辆车通过的最短时间

time[2]=min

方案1=time[1]+length/speed[2]

weight[1]+weight[2]<=bridge
方案2=length/min(speed[1],speed[2])

前3辆车通过的最短时间

time[3]=min

方案1=time[2]+length/speed[3]

weight[2]+weight[3]<=bridge
方案2=time[1]+length/min(speed[2],speed[3])

weight[1]+weight[2]+weight[3]<=bridge
方案3=length/min(speed[1],speed[2],speed[3])

在桥的承重允许的情况下，依次讨论第i两车与前面哪些车能够分在一组，讨论每种分组方案的情况。

# 例题3：护卫队(1006) 分组类DP

**阶段**：按每辆车划分一个阶段

**状态**：f[i]表示前 i 辆车过桥所需的最短时间

**决策**：第i辆车与前面哪些车分为一组过桥时间最短

**方程**：f[i]=min{ f[k] + length/MinSpeed(k+1,k+2,..,i) }

weight[k+1]+weight[k+1]+...+weight[i]<=bridge

MinSpeed(k+1,k+2,..i)表示找出第k+1辆到第i辆这些车中最小的速度

**边界条件:** 1<=i<=n, 0<=k<=i-1

# 滑雪路径 NKOI1009

Michael 喜欢滑雪。可是为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，你不得不再次走上坡或者等待升降机来载你。 Michael想知道在一个区域中最长的滑坡。区域由一个二维数组给出。数组的每个数字代表该点的高度（类似于等高线）。

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 16 | 17 | 18 | 19 | 6 |
| 15 | 24 | 25 | 20 | 7 |
| 14 | 23 | 22 | 21 | 8 |
| 13 | 12 | 11 | 10 | 9 |

图 A

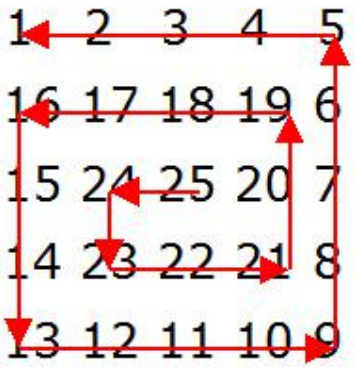


图 B

一个人**可以从某个点滑向上下左右相邻四个点之一，但只能由高向低滑。**在上面的例子中，一条可滑行的滑坡为24-17-16-1。当然25-24-23-...-3-2-1更长，事实上，这是最长的一条。你的任务就是要在给出的地图中，帮Michael找一条最长的滑坡。

# 方法一：记忆化搜索

阶段:以每个点作为起点，依次讨论。
状态:f[x][y]表示从点(x,y)出发，能够得到的最长滑雪路径的长度。

```
main()
{
        ......
        for(x=1;x<=m;x++)
            for(j=1;y<=n;y++)
                if(visited[x][y]==false) //若从(x,y)出发的最长路径未被讨论过
                {
                    FindWay(x,y); //找出从(x,y)出发的最长路径，求f[x][y]
                    ans=max(ans,f[x][y]); //更新结果
                }
        cout<<ans;
        ......
}
```

# 方法一：记忆化搜索

FindWay(x,y)//找出从(x,y)出发的最长滑雪路径长度
深搜讨论从(x,y)出发的走向上下左右四个方向，每个方向能走的最长长度。
假设算出的四个方向能走的最长长度分别为len[1],len[2],len[3],len[4],
那么f[x][y]=max(len[1],len[2],len[3],len[4])+1
若某个方向(xx,yy)已被其它点搜索时讨论过了，则f[xx][yy]已被算好，不用再次讨论

```
void FindWay(int x,int y)  //找出从(x,y)出发的最长化学路径长度
{
    int len[5],i,xx,yy,MaxLen;  //len[1..4]记录从(x,y)出发，四个方向能到达的最远距离的长度。
    for(i=1;i<=4;i++)   //依次讨论四个方向
    {
        len[i]=0;
        xx=x+dx[i];   yy=y+dy[i];   //dx[ ],dy[ ]为讨论四个方向的增量数组
        if(map[xx][yy]<map[x][y]) //(dx,dy)的高度小于(x,y)，从(x,y)可以走向(dx,dy)
            if(visited[xx][yy]) len[i]=f[xx][yy]; //若(dx,dy)已被讨论过，记下最优值，避免重复讨论(记忆化)
            else {   FindWay(xx,yy);   len[i]=f[xx][yy];   } //若(dx,dy)未被讨论过，则深搜求长度
    }
    //找出从(x,y)出发往四个方向走的路径，谁最长。也就是在len[1..4]中找出最大者。
    MaxLen=max(len[1],len[2]);
    MaxLen=max(len[3],MaxLen);
    MaxLen=max(len[4],MaxLen);
    f[x][y]=max(MaxLen+1,f[x][y]);  //记录下从(x,y)出发能走出的最优长度(记忆化)。
    visited[x][y]=true;  //将(x,y)标记为已讨论
}
```

# 方法一：记忆化搜索

时间复杂度？O(nm)

记忆化搜索的特点：

在搜索的时候还是按普通搜索顺序进行，但是每搜索一个状态，就将它的解保存下来。以后再次遇到这个状态的时候，就不必重新搜索了。

记忆化搜索=搜索的形式+动态规划的思想

# 方法二：动态规划

**阶段:**以每个点作为起点，依次讨论。
**状态:**f[x][y]表示从点(x,y)出发，能够得到的最长滑雪路径的长度。
**决策:**

对于点(x,y)，它下一步只能去跟它相邻的，且高度比它低的点。

**f[x][y]=max{ f[xx][yy] }+1　　map[xx][yy]<map[x][y]且相邻**

必须事先算好f[xx][yy]的值。

怎样保证先算高度比map[x][y]低的点呢？

**先由小到大排序！**

# 方法二：动态规划

```
//tot=n*m，也就是点的总数。
sort(1,tot);  //按高度由低到高排序
 ......初始化.....此处已省略若干代码

 //dp
 for(i=1;i<=tot;i++)//阶段，由低到高依次讨论每个点
 {
      //假设i号点的坐标为(x,y)
      for(j=1;j<=4;j++) //讨论四个方向
      {
           xx=x+dx[j];
           yy=y+dy[j];
          if( (xx,yy)没越界 && (x,y)的高度>(xx,yy)的高度 && (f[xx][yy]+1>f[x][y])   )
          f[x][y]=f[xx][yy]+1;
     }
 }
```

最后输出f[ ][ ]数组中值最大的一个。

时间复杂度？ O(nm)

相似题目：1183

# 例题：咒语1035

求两个数列的最长上升公共子序列.

数列长度 1<=M<=500

F[ i ][ j ]来表示A的前i项和B的前j项可以组成的最长上升公共子序列。
a[i]!=b[j]时: F[i][j]=F[i-1][j]
a[i]==b[j]时: F[i][j]=max(F[i-1][k])+1
(1=<k<j)&&b[j]>b[k]//保证上升

```
for(i=1;i<=n1;i++)
{
        for(j=1;j<=n2;j++)
        {
          if(a[i]!=b[j])f[i][j]=f[i-1][j];
          else
           {
            for(k=1;k<j;k++)
             if(b[j]>b[k])f[i][j]=max(f[i][j],f[i-1][k]);//dp

             f[i][j]+=1;
           }
        }
}
         Max=0;
        for(i=1;i<=n1;i++)
         for(j=1;j<=n2;j++)
         Max=max(f[i][j],Max);    //取出答案
```