一维动规 2——最长不下降子序列

最长不下降子序列是非常经典的动态规划类型题。我们先看例题：

oiClass1226: 最长不下降子序列

【题目描述】
设有由 n 个不相同的整数组成的数列，记为: a(1)、a(2)、……、a(n)，若存在 i1<i2<i3< … < ie 且有 a(i1)<a(i2)< …<a(ie)的子序列则称该序列为长度为 e 的不下降序列。求一个序列中的最长不下降子序列。
如下序列：

下标	1	2	3	4	5	6	7	8	9
值	13	7	9	16	38	24	37	18	4

存在子序列 13、7、4 是不下降子序列，长度为 3，但不是最长的不下降子序列，最长的不下降子序列应该为 38、24、18、4，长度为 4，也可以是 38、37、18、4，长度也是 4。
在给定数字序列中，最长子序列的值是固定的，最长子序列的方案可以有多种。
给定一个数字序列，求其最长子序列的长度。
【样例输入】
14
13 7 9 16 38 24 37 18 4 19 21 22 63 15
【样例输出】
8
【分析】
我们可以暴搜解决这个问题，枚举每个数 a[i]，以 a[i]为起点，向后搜索，如果有一个数大于等于 a[i]，则继续递归进行，这样解决的时间复杂度接近 O（n!），显然不是最优解法。
我们考虑动态规划的方式解决。
1、划分阶段：在线性序列里，我们可以考虑以每个数为一个阶段。
2、定义状态变量：对于当前阶段，我们定义状态变量 f[i]，表示以 a[i]结尾所能组成的最长不下降子序列的个数，显然，问题所求的是 max(f[i])。
3、状态转移方程：考虑 f[i]怎么由前面的阶段转移过来。假设在 i 前面存在下标 j（j<i），且 a[j]<=a[i]，那么 a[j]和 a[i]就能构成不下降子序列，原先以 a[j]结尾所构成的最长不下降子序列长度为 f[j]，现在可以接上 a[i]，则新构成的不下降子序列长度为 f[j]+1。在 i 前面，满足 i<j 前 a[j]<=a[i]的 j 可能有多个，在这多个转移方案中，选择最长的那个，即 f[i]=max(f[j]+1)|(1<=j<i)&&(a[j]<=a[i])。
4、边界：最坏情况下，每个数所组成的最长不下降子序列为它自身，即 f[i]=1
5、时间复杂度：O（n^2）
【代码】

```
#include<iostream>
using namespace std;
int n,a[1000],f[1000],maxn;
int main(){
    cin>>n;
```

```
    for(int i=1;i<=n;i++){
        cin>>a[i];
    }
    for(int i=1;i<=n;i++){
        f[i]=1; //初始化边界
        for(int j=1;j<i;j++){  //往 i 前面找一个满足条件的 j
            if(a[i]>=a[j]){  //如果满足条件
                f[i]=max(f[i],f[j]+1); //选择最优的转移方案
            }
        }
        maxn=max(f[i],maxn); //求出最大的 f[i]
    }
    cout<<maxn;
}
```

==========网站练习==========

合唱队形

【题目描述】
N 位同学站成一排，音乐老师要请其中的(N-K)位同学出列，使得剩下的 K 位同学排成合唱队形。
合唱队形是指这样的一种队形：设 K 位同学从左到右依次编号为 1，2…，K，他们的身高分别为 T1，T2，…，TK， 则他们的身高满足 T1<...<Ti>Ti+1>…>TK(1<=i<=K)。
你的任务是，已知所有 N 位同学的身高，计算最少需要几位同学出列，可以使得剩下的同学排成合唱队形。
【输入】
输入文件第一行是一个整数 N(2<=N<=100)，表示同学的总数。第一行有 n 个整数，用空格分隔，第 i 个整数 Ti(130<=Ti<=230)是第 i 位同学的身高(厘米)。
【输出】
输出一行只包含一个整数，就是最少需要几位同学出列。
【样例输入】
8
186 186 150 200 160 130 197 220
【样例输出】
4
【分析】
若构成合唱队形的人最多，则需要出列的人最少，所以问题可以转换成求怎样形成最多人的合唱队形。
合唱队形可以看出左边是最长上升子序列，右边是最长下降子序列。为了使合唱队形最长，我们可以枚举每个人，对于当前枚举到的第 i 人，以 i 为最高点，求出以 i 为结尾的最长上升子序列 f[i]和以 i 开始的最长下降子序列 g[i]，最长合唱队形即为 max(f[i]+g[i])。特别注意，求 g[i]，可以看成反向求最长上升序列。

【代码】

```
#include<iostream>
using namespace std;
int n,a[101],f[101],g[101];
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        f[i]=g[i]=1;
    }
    for(int i=2;i<=n;i++){  //求最长上升子序列
        for(int j=1;j<i;j++){
            if(a[i]>a[j])f[i]=max(f[i],f[j]+1);
        }
    }
    for(int i=n-1;i>0;i--){  //反向求最长上升子序列
        for(int j=n;j>i;j--){
            if(a[i]>a[j])g[i]=max(g[i],g[j]+1);
        }
    }
    int maxn=0;
    for(int i=1;i<=n;i++){  //枚举最大的 f[i]+g[i]
        if(f[i]+g[i]>maxn)maxn=f[i]+g[i];
    }
    cout<<n-maxn+1;
}
```

渡轮问题

【题目描述】

某国家被一条河划分为南北两部分，在南岸和北岸总共有 N 对城市，每一城市在对岸都有唯一的友好城市，任何两个城市都没有相同的友好城市。每一对友好城市都希望有一条航线来往，于是他们向政府提出了申请。由于河终年有雾。政府决定允许开通的航线就互不交叉（如果两条航线交叉，将有很大机会撞船）。兴建哪些航线以使在安全条件下有最多航线可以被开通。

【输入】

第一行两个由空格分隔的整数 x，y，10〈=x〈=6000，10〈=y〈=100。x 表示河的长度而 y 表示宽。

第二行是一个整数 N(1<=N<=5000)，表示分布在河两岸的城市对数。接下来的 N 行每行有两个由空格分隔的正数 C，D（C、D〈=x〉，描述每一对友好城市与河起点的距离，C 表示北岸城市的距离而 D 表示南岸城市的距离。在河的同一边，任何两个城市的位置都是不同的。

【输出】

在安全条件下能够开通的最大航线数目。

【样例输入】

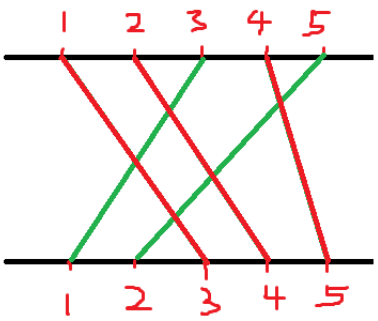
30 4
5
4 5
2 4
5 2
1 3
3 1

【样例输出】

3

【分析】

本题中，航线能开通，必须这些航线不能交叉，那么对于目前的一条航线（x1，y1），存在另一条安全航线（x2，y2）的条件是 x1<x2 且 y1<y2。如果按航线的一端 x 进行排序，问题即转化为在另一端序列 y 中求最长上升序列的个数。



【代码】

```
#include<iostream>
#include<algorithm>
using namespace std;
struct data{
    int x,y;
}a[5001];
int x,y,n,f[5001],maxn;
bool cmp(data a,data b){
    return a.x<b.x;
}
int main(){
    cin>>x>>y>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i].x>>a[i].y;
        f[i]=1;
    }
    sort(a+1,a+n+1,cmp);  //按左端点从小到大排序
    for(int i=2;i<=n;i++){  //在右端点中找最长上升序列
        for(int j=1;j<i;j++){
            if(a[i].y>a[j].y){
```

```
                f[i]=max(f[i],f[j]+1);
                maxn=max(f[i],maxn);
            }
        }
    }
    cout<<maxn;
}
```

词链

【题目描述】
给定一个仅包含小写字母的英文单词表，其中每个单词最多包含 50 个字母。
如果一张由一个词或多个词组成的表中，每个单词（除了最后一个）都是排在它后面的单词的前缀，则称此表为一个词链。例如下面的单词组成了一个词链：

i
int
integer

而下面的单词不组成词链：

integer
intern

请在给定的单词表中取出一些词，组成最长的词链。最长的词链就是包含单词数最多的词链。数据保证给定的单词表中，单词互不相同。
【输入】
第一行一个整数(n<=1000)，表示单词表中单词数
下接 n 行每行一个单词。
【输出】
一个整数，表示最长词链长度。
【样例输入】
5
i
integer
internet
intern
int
【样例输出】
4
【分析】
按字符串从小到大排序，这就保证了对于字符串 s[i]，能被它包含的字符串都在 i 的前面，无后效性问题。设状态变量 f[i]表示以 s[i]结尾的最长词链个数，则 f[i]=max(f[j]+1)|s[j]是 s[i]的前缀。
【代码】

```
#include<iostream>
#include<string>
```

oiClass

```
#include<algorithm>
using namespace std;
string wd[1001];
int n,f[1001],ans;
bool cmp(string a,string b){
    return a<b;
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>wd[i];
        f[i]=1;
    }
    sort(wd+1,wd+n+1,cmp);  //字符串从小到大排序
    for(int i=2;i<=n;i++){  //求 f[i]
        for(int j=1;j<i;j++){  //枚举前面的 j
            if(wd[i].substr(0,wd[j].size())==wd[j]){ //如果 wd[j]是 wd[i]的前缀
                f[i]=max(f[i],f[j]+1);  //选择最长的词链
                ans=max(ans,f[i]);
            }
        }
    }
    cout<<ans;
}
```

饥饿的牛

【题目描述】
牛在饲料槽前排好了队。饲料槽依次用 1 到 N(1<=N<=2000)编号。每天晚上，一头幸运的牛根据约翰的规则，吃其中一些槽里的饲料。 约翰提供 B 个区间的清单。一个区间是一对整数 start-end,1<=start<=end<=N，表示一些连续的饲料槽，比如 1-3,7-8,3-4 等等。牛可以任意选择区间，但是牛选择的区间不能有重叠。
当然，牛希望自己能够吃得越多越好。给出一些区间，帮助这只牛找一些区间，使它能吃到最多的东西。 在上面的例子中，1-3 和 3-4 是重叠的；聪明的牛选择{1-3，7-8}，这样可以吃到 5 个槽里的东西。
【输入】
第一行，整数 B(1<=B<=1000)
第 2 到 B+1 行，每行两个整数，表示一个区间，较小的端点在前面。
【输出】
仅一个整数，表示最多能吃到多少个槽里的食物。
【样例输入】
3
1 3
7 8

3 4
【样例输出】
5
【分析】
这道题是典型的区间最长不下降子序列问题，对于这类问题，我们需要首先进行排序，保证无后效性。对于区间[s,t]，我们按照结束位置从小到大排序，对于第 i 段区间，我们设状态变量 f[i]表示以结束区间 t 为止所能获得的最多食物，因为在 i 之前的区间，没有一个比 t[i]更大的。那么 f[i]=max(f[j]+t[i]-s[i]+1)|t[j]<s[i]。
【代码】

```
#include<iostream>
#include<algorithm>
using namespace std;
struct data{
    int s,t,sum;
}a[1001];
int n,ans,f[1001];
bool cmp(data a,data b){
    return a.t<b.t;
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i].s>>a[i].t;
        a[i].sum=a[i].t-a[i].s+1;
    }
    sort(a+1,a+n+1,cmp);
    f[1]=a[1].sum;
    for(int i=2;i<=n;i++){
        f[i]=a[i].sum;
        for(int j=1;j<i;j++){
            if(a[j].t<a[i].s){
                f[i]=max(f[i],f[j]+a[i].sum);
            }
        }
        ans=max(ans,f[i]);
    }
    cout<<ans;
}
```

打鼹鼠

【题目描述】
鼹鼠是一种很喜欢挖洞的动物，但每过一定的时间，它还是喜欢把头探出到地面上来透透气

的。
根据这个特点阿 Q 编写了一个打鼹鼠的游戏：在一个 n*n 的网格中，在某些时刻鼹鼠会在某一个网格探出头来透透气。你可以控制一个机器人来打鼹鼠，如果 i 时刻鼹鼠在某个网格中出现，而机器人也处于同一网格的话，那么这个鼹鼠就会被机器人打死。而机器人每一时刻只能够移动一格或停留在原地不动。机器人的移动是指从当前所处的网格移向相邻的网格，即从坐标为（i,j）的网格移向(i-1, j),(i+1, j),(i,j-1),(i,j+1)四个网格，机器人不能走出整个 n*n 的网格。游戏开始时，你可以自由选定机器人的初始位置。
现在你知道在一段时间内，鼹鼠出现的时间和地点，希望你编写一个程序使机器人在这一段时间内打死尽可能多的鼹鼠。
【输入】
文件第一行为 n（n<=1000）, m（m<=10000），其中 m 表示在这一段时间内出现的鼹鼠的个数，接下来的 m 行每行有三个数据 time,x,y 表示有一只鼹鼠在游戏开始后 time 个时刻，在第 x 行第 y 个网格里出现了一只鼹鼠。Time 按递增的顺序给出。注意同一时刻可能出现多只鼹鼠，但同一时刻同一地点只可能出现一只鼹鼠。
【输出】
输出文件中仅包含一个正整数，表示被打死鼹鼠的最大数目。
【样例输入】
2 2
1 1 1
2 2 2
【样例输出】
1
【分析】
题目中鼹鼠是按时间顺序递增给出的。我们可以以每只鼹鼠为一个阶段，设 f[i]表示以第 i 只鼹鼠出现的时间为止能打到的最多鼹鼠数，那么，如果 i 前面的第 j 只鼹鼠，可以在 a[i].t-a[j].t 这个时间内，从(a[j].x,a[j].y)走到(a[i].x,a[i].y)，即 abs(a[i].x-a[j].x)+abs(a[i].y-a[j].y)，那么就可以在打死第 j 只鼹鼠后接着打死第 i 只鼹鼠，当然，如果提前可以到，也可以等待的。所以，对于第 i 只鼹鼠，如果存在 abs(a[i].x-a[j].x)+abs(a[i].y-a[j].y)<=a[i].t-a[j].t，那么 f[i]=f[j]+1，可能存在多个这样的 j，所以最优决策为 f[i]=max(f[j]+1)，题目即求最大的 f[i]，即 max(f[i])。时间复杂度为 O(m^2)，m 最大为 10000，可能会超时，不过本题数据比较水，这个方法可以过。不过也可以采取一些剪枝优化的方法，可以思考如果优化。
【代码】

```
#include<iostream>
using namespace std;
struct ys{
    int t,x,y;
}a[10001];
int n,m,f[10001],ans;
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>a[i].t>>a[i].x>>a[i].y;
    }
    for(int i=1;i<=m;i++){
```

oiClass

```
        f[i]=1;
        for(int j=1;j<i;j++){
            int d=abs(a[i].x-a[j].x)+abs(a[i].y-a[j].y);
            if(d<=a[i].t-a[j].t){
                f[i]=max(f[i],f[j]+1);
            }
        }
        ans=max(ans,f[i]);
    }
    cout<<ans;
}
```

看球的巴士

【题目描述】
两个球队的支持者要一起坐车去看球，他们已经排成了一列。我们要让他们分乘若干辆巴士，同一辆巴士上的人必须在队伍中是连续的。为了在车上不起冲突，希望两队的支持者人数尽量相等，差至多是 D。有一个例外，就是一辆车上的人全部都是一个球队的支持者。问要将这 N 个人全部送至球场，至少要几辆巴士。
【输入】
第一行是整数 N 和 D，1<=N<=2500，1<=D<=N。
接下来的 N 行，按排队的顺序，描述每个人支持的球队，用 H 或 J 表示。
【输出】
至少要几辆巴士。
【样例输入】
14 3
H
J
H
H
H
J
H
J
H
H
H
H
H
H
【样例输出】
2
【提示】
有多种方案，例如让前 9 人做一辆车，差正好是 3；后 5 人做一辆车，因为只有一对的支持

者。

【分析】

设 f[i]表示前 i 人所需的最少乘车数量，那么对于 i 之前的 j，如果 j 到 i 这个区间可以共乘一辆车，则所需乘车数量为 f[i]=f[j-1]+1，枚举 j，对于满足条件的 j，f[i]=min(f[j]+1)。那么，题目的难度在于如何判断 j 到 i 之间的人能否共乘一辆车。设数组 H[i]表示前 i 个人中有 H 队的人数，J[i]表示前 i 个人中有 J 队的人数，那么，如果 j 到 i 全是 H 队，则 H[i]-H[j-1]==i-j+1，如果全是 J 队，则 J[i]-J[j-1]=i-j+1，判断 j 到 i 之间的人数差值在 d 以内可以用 abs((H[i]-H[j-1])-(J[i]-J[j-1]))<=d。时间复杂度为 O(n^2)。

【代码】

```
#include<iostream>
#include<cmath>
using namespace std;
int n,d,H[2501],J[2501],f[2501];
char ch;
bool check(int s,int t){  //判断 s 到 t 之间的人能否共乘一辆车
    if(H[t]-H[s-1]==t-s+1)return true;  //都是 H 队的人
    if(J[t]-J[s-1]==t-s+1)return true;  //都是 J 队的人
    int h=H[t]-H[s-1];
    int j=J[t]-J[s-1];
    if(abs(h-j)<=d)return true;  //H 队和 J 队的差值在 d 以内
    return false;
}
int main(){
    cin>>n>>d;
    for(int i=1;i<=n;i++){
        cin>>ch;
        if(ch=='H')H[i]=H[i-1]+1;  //统计前缀和数组 H
        else H[i]=H[i-1];
        if(ch=='J')J[i]=J[i-1]+1;  //统计前缀和数组 J
        else J[i]=J[i-1];
        f[i]=i;  //假设最大值，每个人一台车
    }
    for(int i=1;i<=n;i++){
        for(int j=1;j<i;j++){
            if(check(j,i)){
                f[i]=min(f[i],f[j-1]+1);
            }
        }
    }
    cout<<f[n];
}
```