信息学

动态规划

西南大学附属中学校

信息奥赛教练组

动态规划的关键点

西南大学附属中学
High School Affiliated to Southwest University

状态的表示

阶段的划分

状态转移方程

边界

目标

最长不下降子序列 LIS

给出一串序列，求出该序列中最长的不下降（即**非严格递增顺序**）的子序列长度。

状态的表示

f [i]表示以a [i] 作为结尾的LIS 的长度

阶段的划分

子序列的结尾为止（从前到后）

状态转移方程

f [i]= max(f [j] +1)

边界

f [0]= 0

目标

Max(f [i])

```
for(i = 1;i< N;i++){
    for(j = 0;j < i;j++){//从i的下一个数开始
        if(arr[i] > arr[j]) {
            if(dp[i] < dp[j] + 1){//如果
                dp[i] = dp[j] + 1;//更新
            }
        }
    }
}
```

$O(n^2)$

n log(n)做法

tail [i]　表示的是长度为 i 的最长不下降子序列的结尾元素的最小值。

对于序列（1 2 5 2 5 4）中，对于前3位分别有如下的最长不下降子序列：

长度为1 的：（1）　（2）　（5）　=>但是1,2,3中1最小，所以 **tail[1] = 1**
长度为2 的：（1 2）　（1 5）　（2 5）=> **tail[2] = 2**
长度为3 的：（1 2 5）=> **tail[3] = 5**

继续读取到第四位，　2 比 tail [3] = 5 小　　怎么办?

找到 tail 中第一个大于 2 的数替换掉　　因为我们用1 2 2就可以构成一个长度为3的子序列

tail数组的值是单调递增　　二分

tail数组的长度就是最长不下降子序列的长度

```
void getLIS(){
    int tail[maxN ];//tail[i]表示长度为i的LIS中结尾的元素
    fill(tail,tail+maxN,0);
    int cnt = 1;
    tail[cnt] = arr[0];//第一个长度为1的最长不下降子序列的结尾元素是arr[0]
    for(int i = 1;i< n;i++){
        if(arr[i] >= tail[cnt]){
            tail[++cnt] = arr[i];
        }
        else{//二分找出最小的
            int idx= lower_bound(tail+1,tail+cnt+1,arr[i]) - tail;
            tail[idx] = arr[i];//换掉这个元素
        }
    }
    cout << cnt<<"\n";
}
```

尝试提交一下 [NOIP1999 普及组] 导弹拦截https://www.luogu.com.cn/problem/P1020

最长公共子序列 LCS

给出两个字符串序列，求出同时是两个序列的子序列的字符串长度最长是多少。

状态的表示

f [i][j]表示前缀子串a [1-i] 与 b[1-j] LCS的长度

阶段的划分

已处理的前缀长度

状态转移方程

$$f[i][j]=\max\begin{cases} f[i][j-1] \\ f[i-1][j] \\ f[i-1][j-1]+1 \end{cases}$$

边界

f [i] [0]= f [0][j] =0

目标

f [N][M]

```
for(int i=1;i<=na;i++){
    for(int j=1;j<=nb;j++){
        dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        if(a[i]==b[j]){
            dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);
        }
    }
}
```

最大子段和

对于给定序列 a1,a2,a3……an 寻找它的连续的最大和子数

状态的表示

f [i] 表示到当前位置i的最大的连续子段

阶段的划分

已处理的前缀长度

状态转移方程

$$f[i]=\begin{cases} f[i-1]+a[i] & f[i-1]>=0 \\ a[i] & f[i-1]<0 \end{cases}$$

边界

f [1]=a[1]

目标

f [N]

```
ans = dp[1] = a[1];     // 初始化第一个 d[0]
for(int i=2;i<=n;i++) // 遍历从第二数开始
{
    if(dp[i-1]<0)      // i 之前的最大连续子数组小于零，则抛弃它，自己从开始
        dp[i] = a[i];
    else                // i 之前的最大连续子数组不小于零，则加上他们
        dp[i] = a[i]+dp[i-1];
    ans = max(dp[i],ans);  // 保存d[i] 中的最大值
}
```

最大子矩阵

枚举开始行和结束行，边枚举，边在列上做区域和并更新数据到数组b中

	0	1	2	3	4
A	2	−5	7	3	−2
B	−5	9	−8	−4	6
C	3	3	−2	4	5

b	2	−5	7	3	−2

此时对b数组求最大子序列和，实质上就是在求A行上的最大子矩阵和。（ans=10）

	0	1	2	3	4
A	2	-5	7	3	-2
B	-5	9	-8	-4	6
C	3	3	-2	4	5

b	-3	4	-1	-1	4

可以发现此时A～B行的最大子矩阵和，只有6（4+-1+-1+4

	0	1	2	3	4
A	2	−5	7	3	−2
B	−5	9	−8	−4	6
C	3	3	−2	4	5

b	0	7	−3	3	9

此时A～C的最大子矩阵为16（7+-3+3+9），16>ans 所以ans=16

依此类推从B行开始枚举

	0	1	2	3	4
A	2	−5	7	3	−2
B	−5	9	−8	−4	6
C	3	3	−2	4	5

b	−5	9	−8	−4	6

```
int ans =0;// 如果最大值为负数，则输出0
 for(int i=0;i<m;i++){ //开始行
    int b[n]={0}; //初始化b数组
    for(int j=i;j<m;j++){ //结束行
        for(int k=0;k<n;k++){ //按列计算
            b[k] += a[j][k]
        }
        //对b[k]求 最大子序列和 并得到答案temp
        if(temp>ans)ans = temp;
    }
 }
 cout<<ans;
```

消化一下

最长公共上升子序列 LCIS

先求出LCS再求出LCS的LIS?

a: 7 1 5 6 4 2 7
b: 7 1 5 4 6 7 2

按照递归的取“最长公共子序列”,取出:
7 1 5 6 2

此序列的“最长上升子序列”为:

1 5 6 (len=3)

但原序列的“最长公共上升子序列”为:

1 5 6 7 (len=4)

最长公共上升子序列 LCIS

状态的表示

f [i][j]表示表示a串前i个字符和b串前j个字符且以b[j]为结尾的LCIS

阶段的划分

已处理的前缀长度

状态转移方程

$$f[i][j]=\begin{cases} f[i-1][j] & a[i]\neq b[j] \\ \max(f[i-1][k]+1) & b[k]<b[j] \end{cases}$$

边界

f [i] [0]= f [0][j] =0

目标

max(f[N][j])

```
for(int i=1;i<=n;i++) {
    for(int j=1;j<=n;j++){
        f[i][j]=f[i-1][j];  //a[i]不存在
        //a[i]存在
        if(a[i]==b[j]){
            for(int k=0;k<j;k++){
                if(b[k]<b[j])
                    f[i][j]=max(f[i][j],f[i][k]+1);
            }
        }
    }
}
```

```
for(int i=1;i<=n;i++) {
    for(int j=1;j<=n;j++){
        f[i][j]=f[i-1][j];  //a[i]不存在
        //a[i]存在
        if(a[i]==b[j]){
            for(int k=0;k<j;k++){
                if(b[k]<b[j])
                    f[i][j]=max(f[i][j],f[i][k]+1);
            }
        }
    }
}
```

b[k]<a[i]

f[i][k]的k前缀最大值

求的是一个与 k 无关的前缀最值!

设前缀最值为maxv，表示b [j] < a [i]是f [i] [j] 的最大值，maxv初始化为1。

当a [i] = = b [j] 时，f [i] [j]直接由maxv更新即可!

```
for(int i=1;i<=n;i++) {
    maxv=1;
    for(int j=1;j<=n;j++){
        f[i][j]=f[i-1][j];  //a[i]不存在
        //a[i]存在
        if(a[i]==b[j]){
            f[i][j]=max(f[i][j],maxv);
        }
        if(b[j]<a[i])
            maxv=max(maxv,f[i][j]+1);
    }
}
```

消化一下

例题：Mobile Service

一个公司有三个移动服务员。如果某个地方有一个请求，某个员工必须赶到那个地方去（那个地方没有其他员工），某一时刻只有一个员工能移动。被请求后，他才能移动，不允许在同样的位置出现两个员工。从p到q移动一个员工，需要花费c(p,q)。这个函数没有必要对称，但是c(p,p)=0。公司必须满足所有的请求。目标是最小化公司花费。

输入

第一行有两个整数L,N(3< =L< =200，1< =N< =1000)。L是位置数；N是请求数。每个位置从1到L编号。下L行每行包含L个非负整数。第i+1行的第j个数表示c(i,j)，并且它小于2000。最后一行包含N个数，是请求列表。一开始三个服务员分别在位置1，2，3。

输出

一个数M，表示最小服务花费。

样例

样例输入1

```
5 9
0 1 1 1 1
1 0 2 3 2
1 1 0 4 1
2 1 5 0 1
4 2 3 4 0
4 2 4 1 5 4 3 2 1
```

样例输出1

```
5
```

阶段的划分

已经完成的请求数量

状态的表示

最朴素的想法　　状态转移时每个服务员的位置都要知道

f [i] [x] [y] [z] 完成了前 i 个请求 三个员工分别位于 x y z

状态的转移

$f[i+1][P_{i+1}][y][z] = \min(f[i+1][P_{i+1}][y][z], f[i][x][y][z] + c(x, P_{i+1}))$

$f[i+1][x][P_{i+1}][z] = \min(f[i+1][x][P_{i+1}][z], f[i][x][y][z] + c(y, P_{i+1}))$

$f[i+1][x][y][P_{i+1}] = \min(f[i+1][x][y][P_{i+1}], f[i][x][y][z] + c(z, P_{i+1}))$

一个位置不会出现两个员工，加 if 判断合法性

状态的转移

$f[i+1][P_{i+1}][y][z] = \min(f[i+1][P_{i+1}][y][z], f[i][x][y][z] + c(x, P_{i+1}))$

$f[i+1][x][P_{i+1}][z] = \min(f[i+1][x][P_{i+1}][z], f[i][x][y][z] + c(y, P_{i+1}))$

$f[i+1][x][y][P_{i+1}] = \min(f[i+1][x][y][P_{i+1}], f[i][x][y][z] + c(z, P_{i+1}))$

复杂度 N*L^3 1000*200^3

怎么优化 完成前 i 个请求后，一定有一个员工在 Pi，所以我们只需要表示两外两个员工的位置

状态的转移

$f[i+1][x][y] = \min(f[i+1][x][y], f[i][x][y] + c(P_i, P_{i+1}))$

$f[i+1][P_i][y] = \min(f[i+1][P_i][y], f[i][x][y] + c(y, P_{i+1}))$

$f[i+1][x][P_i] = \min(f[i+1][x][P_i], f[i][x][y] + c(x, P_{i+1}))$

边界　　目标

$P_0 = 3$

f [0] [1] [2] = 0　　f [N] [?] [?]

线性DP，一般先明确“阶段”，

如果“阶段”不足以表示一个状态，可以把所需附加信息也作为状态的维度

如果DP状态由多个维度构成，先看看这些维度可否相互导出，尽量减少冗余