

Dynamic Programming

# 基础动态规划

第3课 经典题目选讲

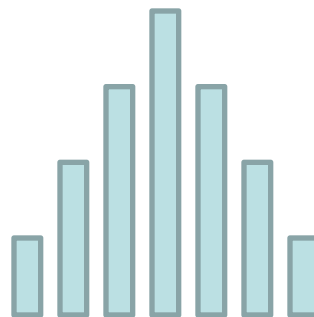
## 合唱队形(NOIP)

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 $T_1, T_2, \dots, T_K$ , 则他们的身高满足

$T_1 < \dots < T_i > T_{i+1} > \dots > T_K, (1 \leq i \leq K)$ 。

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。



# 合唱队形(NOIP)

阶段：从左到右上依次讨论。

状态：fup[i]表示以第i个数结尾的最长上升序列的长度，  
fdown[i]表示以第i个数开头的最长下降序列的长度。（可以看成从末尾开始的上升子序列）

决策：第i个数与它左侧小于i的数中，fup[]值最大那一个组合成上升序列。

求上升序列最长长度：

$$\text{fup}[i]=1+\max\{\text{fup}[j]\}$$

求下降序列最长长度：

$$\text{fdown}[i]=1+\max\{\text{fdown}[j]\}$$

边界条件：  $i < j \leq n$ , 且  $a[i] < a[j]$

//到每个点为止的最长上升子序列（当前点是最高点）

```
for(i=2;i<=n;++i)
{
    for(j=1;j<i;++j)
    {
        if(a[j] < a[i] && fup[j] >= fup[i])
            fup[i]=fup[j]+1;
    }
}
```

//从每个点到最后一个点的最长下降子序列（当前点是最高点）

```
for(i=n;i>=1;--i)
{
    for(j=n;j>i;--j)
    {
        if(a[j] < a[i] && fdown[j] >= fdown[i])
            fdown[i]=fdown[j]+1;
    }
}
```

# 最佳游览

有一座旅游城，它的街道成网格状（如下表）。其中东西向(左右方向)的街道是“风景线”上面分布着许多景点。南北向（上下方向）的街道都是林荫道，两旁看不到任何建筑物。由于游客众多，“游览线”被规定为单行道，游客在风景线上只能从西走到东（从左往右），林荫道上则可以任意行走。

一名游客将到这座旅游城旅游。他根据自己对景观的喜好给所有的景点打了分，分值是从-100到+100的整数，分值越大表示我们的旅游者越喜欢这里的景致。显然这位游客不可能给这座旅游城的所有景点都打负分。如下图，该城市共有三条从西往东的风景区，每条线上的每个景点都被标上了分值。竖列间的南北走向的空白区域就是林荫道

-50	-47	36	-30	-23
17	-19	-34	-13	-8
-42	-3	-43	34	-45

游客可以从旅游城的任一个景点开始游览，在任一个景点结束游览。我们的旅游者希望一路上游览的所有景点的分值之和能够尽可能地大。请你写一个程序，帮助这位游客寻找一条最佳的游览路线。

## 样例输入 1

3	6			
-50	-47	36	-30	-23
17	-19	-34	-13	-8
-42	-3	-43	34	-45

## 样例输出 1

84

# 最佳游览

按**阶段**讨论问题：从右到左依次讨论每个数字

确定要求解的**状态**： $f[i]$ 表示以第 $i$ 个数字作为开头的连续序列的最大和

做出最佳**决策**：

如果  $f[i+1] \leq 0$ ，那么  $f[i] = a[i]$ ；

如果  $f[i+1] > 0$ ，那么  $f[i] = f[i+1] + a[i]$ ；

最后得出**方程**：

$$f[i] = \begin{cases} f[i+1] + a[i]; & (f[i+1] > 0) \\ a[i]; & (f[i+1] \leq 0) \end{cases}$$

**边界条件**： $1 \leq i \leq n$

-50	-47	36	-30	-23
17	-19	-34	-13	-8
-42	-3	-43	34	-45



-50	-47	36	-30	-23
17	-19	-34	-13	-8
-42	-3	-43	34	-45



17	-3	36	34	-8
----	----	----	----	----

原题转换成“最大连续和”问题

```

int p[105][20005],a[20005],f[20005],n,m;
int main(){
    cin>>m>>n;
    //初始化状态
    for(int i=1;i<=n+5;i++)
        f[i]=-105;
    //输入
    for(int i=1;i<=m;i++)
        for(int j=1;j<=n-1;j++)
            cin>>p[i][j];
    //每列的最大值
    for(int i=1;i<n;i++){
        a[i]=p[1][i];
        for(int j=1;j<=m;j++){
            a[i]=max(p[j][i],a[i]);
        }
    }
}

```

```

//状态转移
for(int i=1;i<n;i++){
    if(f[i-1]<0) f[i]=a[i];
    else f[i]=a[i]+f[i-1];
}
//找最大值
int t=-105;
for(int i=1;i<n;i++){
    t=max(t,f[i]);
}
cout<<t;
}

```

## 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。



```
#include <iostream>
using namespace std;
int n,f[5005],s,t;
char a[5005];
int main() {
    cin>>n;
    cin>>a+1;
    for(int i=1;i<=n;i++){
        s=0;
        for(int j=1;j<=n;j++){
            t=f[j];
            if(f[j]<f[j-1]){
                f[j]=f[j-1];
            }
            if(a[i]==a[n+1-j] && f[j]<s+1){
                f[j]=s+1;
            }
            s=t;
        }
    }
    cout<<n-f[n]<<endl;
}
```

## 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 $n$  ( $n \leq 10000$ ), 表示塔的层数。

接下来的 $n$ 行每行一个数 ( $\leq 100$ ), 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

```
#include<iostream>
using namespace std;
const int N=10000+5;
int t[N],p[N],a[N],n;
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
    }
    p[1]=a[1],t[1]=0;
    for(int i=2;i<=n;i++){
        p[i]=min(p[i-1],t[i-1])+a[i];
        t[i]=min(p[i-1],p[i-2]);
    }
    cout<<min(p[n],t[n]);
}
```

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

```
#include<cstdio>
#include<algorithm>
using namespace std;
int n,a[10005],f[10005];
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    for(int i=1;i<=n+1;i++)
    {
        f[i]=f[i-1]+a[i];
        if(i-2>=0) f[i]=min(f[i],f[i-2]+a[i]);
        if(i-3>=0) f[i]=min(f[i],f[i-3]+a[i]);
    }
    printf("%d",f[n+1]);
    return 0;
}
```

## 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：** $f[i] = \min \{ f[k] + \text{length} / \text{MinSpeed}(k+1, k+2, \dots, i) \}$   
           $\text{weight}[k+1] + \text{weight}[k+1] + \dots + \text{weight}[i] \leq \text{bridge}$   
           $\text{MinSpeed}(k+1, k+2, \dots, i)$ 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：** $1 \leq i \leq n, 0 \leq k \leq i-1$



## 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

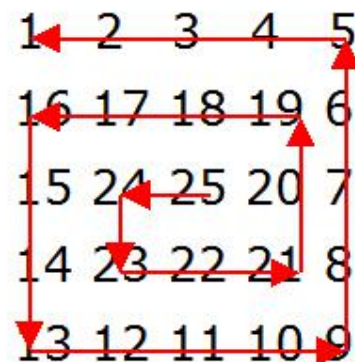


图 B

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 $f[x][y]=\max(\text{len}[1],\text{len}[2],\text{len}[3],\text{len}[4])+1$

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  $1 \leq M \leq 500$

$F[i][j]$  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。

$a[i] \neq b[j]$  时:  $F[i][j] = F[i-1][j]$

$a[i] = b[j]$  时:  $F[i][j] = \max(F[i-1][k]) + 1$

$(1 \leq k < j) \ \&\& \ b[j] > b[k]$  // 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

```



## 例题：母牛的货币系统

### 背包模型：

背包的容积：要构造的数量钱

物品：各种类的货币

这个问题就变成了：装满背包有多少种方案。

**阶段：依次讨论每种货币。**

**状态： $f[i]$ 表示使用前 $i$ 种货币，能构造出该钱的方案数量。**

**状态转移方程： $f[j] += f[j - a[i]];$**

```
#include<bits/stdc++.h>
using namespace std;
long long a[30],f[10005];
int main()
{
    int v,n,i,j;
    cin>>v>>n;
    for(i=1;i<=v;i++)
        cin>>a[i];
    f[0]=1;
    for(i=1;i<=v;i++)
        for(j=a[i];j<=n;j++)
            f[j]+=f[j-a[i]];

    cout<<f[n];
    return 0;
}
```