中国计算机学会
China Computer Federation

动态规划算法及应用

东北师范大学附属中学 王晓光

讲课的主要内容

一、培养计算思维，提升解决问题能力

二、动态规划问题的理论基础

三、背包问题及其应用

四、线性动规与区间动归算法及其应用

五、数位统计动态规划算法及其应用

六、状态压缩动态规划算法及其应用

七、树形动态规划算法及其应用

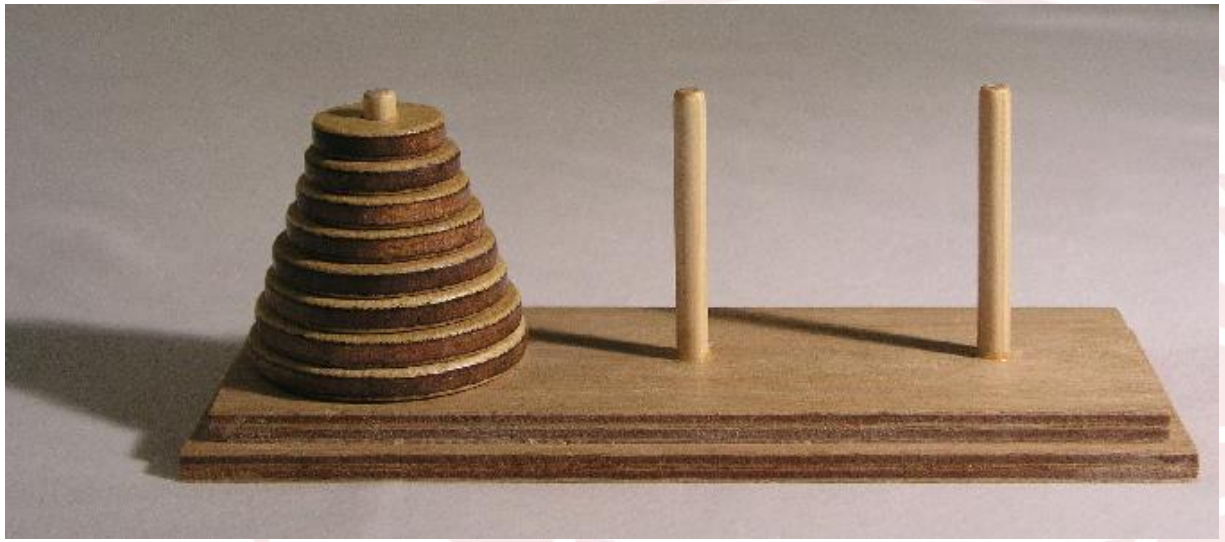
信息技术学科核心素养之一——计算思维

计算思维是运用计算机科学的基础概念进行问题求解、系统设计、以及人类行为理解等涵盖计算机科学之广度的一系列思维活动，由周以真于2006年3月首次提出。2010年，周以真教授又指出计算思维是与形式化问题及其解决方案相关的思维过程，其解决问题的表示形式应该能有效地被信息处理代理执行。

应用计算思维解决问题的步骤

分解——模式——抽象——算法

例1汉诺塔



A B C

1.有三根杆子A,B,C。A杆上有若干碟子
2.每次移动一块碟子,小的只能叠在大的上面
3.把所有碟子从A杆全部移到C杆上

问题：一共n个盘子，最少移动多少次，第一步往B杆上挪，还是往C杆上挪?

演绎计算思维解决问题步骤：　分解——模式——抽象——算法

例2爬楼梯

小明爬楼梯，他可以每次走1级或者2级，输入楼梯的级数，求不同的走法数。
例如：楼梯一共有3级，他可以每次都走一级，或者第一次走一级，第二次走两级，也可以第一次走两级，第二次走一级，一共3种方法。求一共有楼梯n阶，共有多少种爬法。(n<=30)

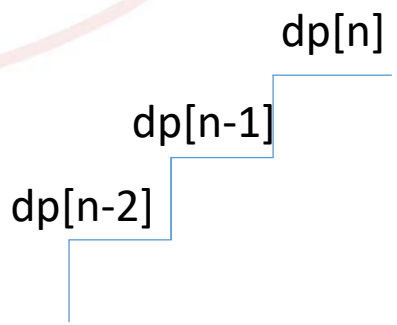
演绎计算思维解决问题步骤：　分解——模式——抽象——算法

1、分解
n=1 有1 共1种方法
n=2 有1 1、2 共2种方法
n=3 有1 1 1、1、2、2、1共3种方法
n=4 有1 1 1 1、1 1 2、1 2 1、2 1 1、2 2 共5种方法
n=5 . . . . . .
n=6 . . . . . .

2、模式识别

发现一个数列1、2、3、5、8.....
猜想：斐波那契数列

3、抽象化
定义dp[n]，表示为共有n阶台阶，一共的爬法为dp[n]种方法。

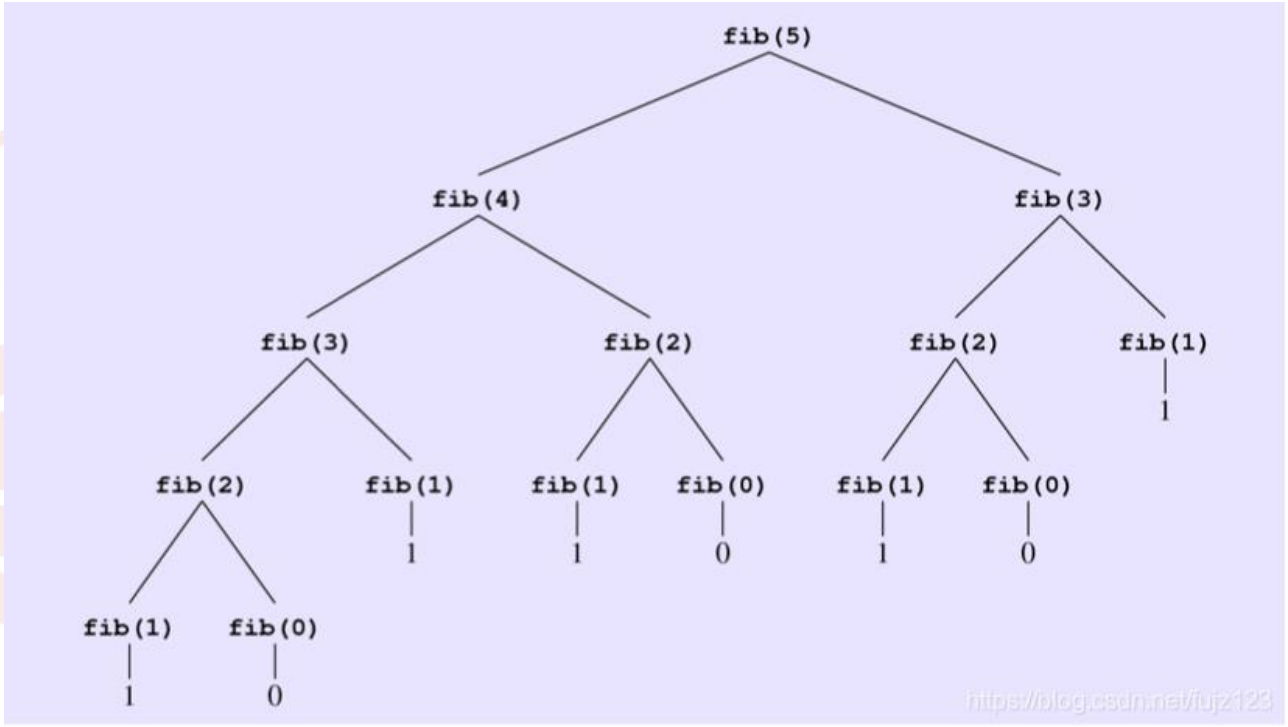


dp[n]=dp[n-1]+dp[n-2]

4、算法

递归写法

```
int fib(int n)
{
  if (n==1) return 1;
  if (n==2) return 2;
  return (fib(n-1)+fib(n-2));
}
```



当计算 fib(5) 时，我们共需要计算1次 fib(4)，2次 fib(3)，3次 fib(2)，5次 fib(1) 和3次 fib(0)。这些无意义的重复计算使得递归效率极低。

代码中递归以2的倍数递增，复杂度是$O(2^n)$

4、算法

优化1：自顶向下记忆化

```
int dp[50];
int fib(int n)
{
   if (n==1) return 1;
   if (n==2) return 2;
   if (dp[n]!=0) return dp[n];
   dp[n]=fib(n-1)+fib(n-2);
   return dp[n];
}
```

优化2：自底向上与制表递推

```
dp[1]=1;
dp[2]=2;
for (int i=3;i<=31;i++)
 dp[i]=dp[i-1]+dp[i-2];
while(cin>>n)
cout<<dp[n]<<endl;
```

2个算法中每个菲波那切数列只计算了一次，时间代价是O(n)

1850	*正确 619369(92%)	1712	0	C++/Edit
1850	正确	1712	0	C++/Edit
1850	正确	1712	10	C++/Edit

二、动态规划问题的理论基础

动态规划是一种在数学、计算机科学和经济学中使用的，通过把原问题分解为相对简单的子问题的方式求解复杂问题的方法。 动态规划常常适用于有重叠子问题和最优子结构性质的问题，动态规划方法所耗时间往往远少于朴素解法。

●重叠子问题

首先，子问题是原大问题的小版本，计算步骤一样；其次，计算大问题时，需要多次重复计算小问题。

●最优性原理

首先，大问题的最优解包含着小问题的最优解；其次可以通过小问题的最优解，推导出大问题的最优解。

●无后效性原则

给定某一阶段的状态，则在这一阶段以后过程的发展不受这阶段以前各段状态的影响，所有各阶段都确定时，整个过程也就确定了。这个性质意味着过程的历史只能通过当前的状态去影响它的未来的发展，这个性质称为无后效性。

未来与过去无关

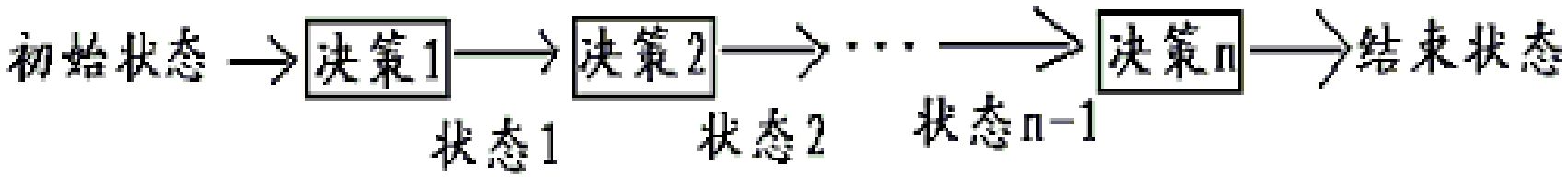
无后效性是应用动态规划解决问题的必要条件，只有这样，才能降低算法的复杂度。

二、动态规划问题的理论基础

动态规划所处理的问题是一个“多阶段决策问题”

目的是得到一个最优解（方案）

大概思想如下图所示：



三、背包问题及其应用

01背包
完全背包
多重背包
混合三种背包
二维费用背包
分组背包
有依赖的背包
.....

三、背包问题及其应用

例3 01背包

给出一个容量为m的背包，n个物品，每个物品有一个体积v[i]和价值c[i]，求这个背包里能装下的物品价值最大是多少。

输入

第一行：两个整数，n(物品数量n<=1000)和m(背包容量，m<=1000)；

第2..N+1行：每行二个整数v_i，c_i，表示每个物品的重量和价值。

输出

仅一行，一个数，表示最大总价值。

样例输入:

```
4 10
2 1
3 3
4 5
7 9
```

样例输出：

```
12
```

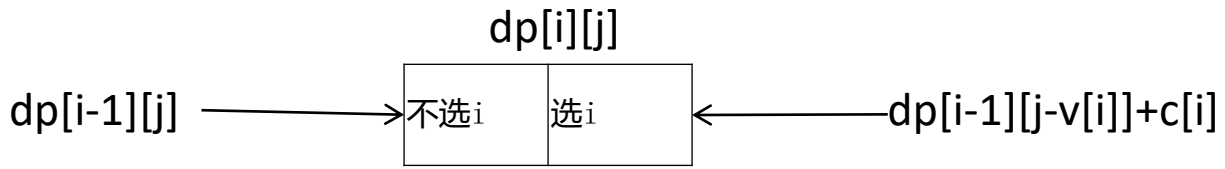
1、划分阶段：按照物品数和包的容量划分；

2、定义状态：dp[i][j]表示从前i件物品中选取若干个放入容量为j的背包中可以获取的最大价值；

3、状态转移方程：

若j>=w[i]，则 dp[i][j]=max(dp[i-1][j],dp[i-1][j-v[i]]+c[i]);

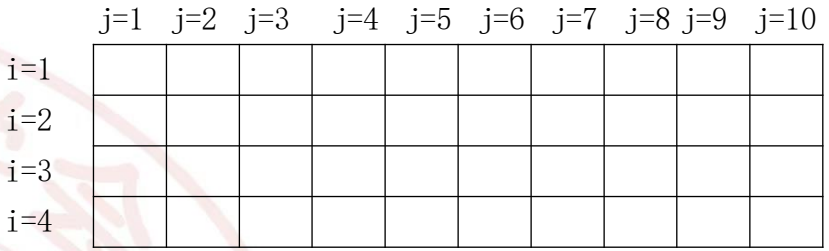
否则 dp[i][j]=dp[i-1][j];



三、背包问题及其应用

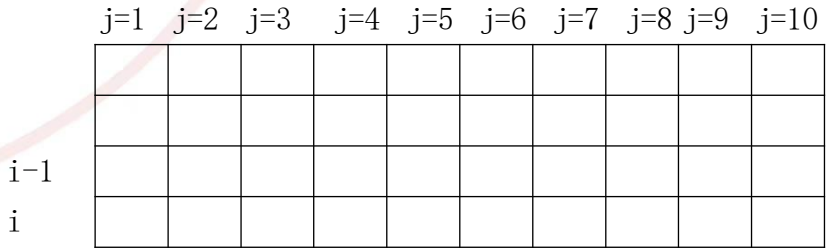
01背包二维空间核心代码

```
cin>>n>>m;
for (int i=1;i<=n;i++) cin>>v[i]>>c[i];
for (int i=1;i<=n;i++)
  for (int j=0;j<=m;j++)
  {
        dp[i][j]=dp[i-1][j];//没选第i个物品
     if (j>=v[i]) dp[i][j]=max(dp[i][j],dp[i-1][j-v[i]]+c[i]);
  }
cout<<dp[n][m]<<endl;
```



01背包一维空间优化核心代码（滚动数组）

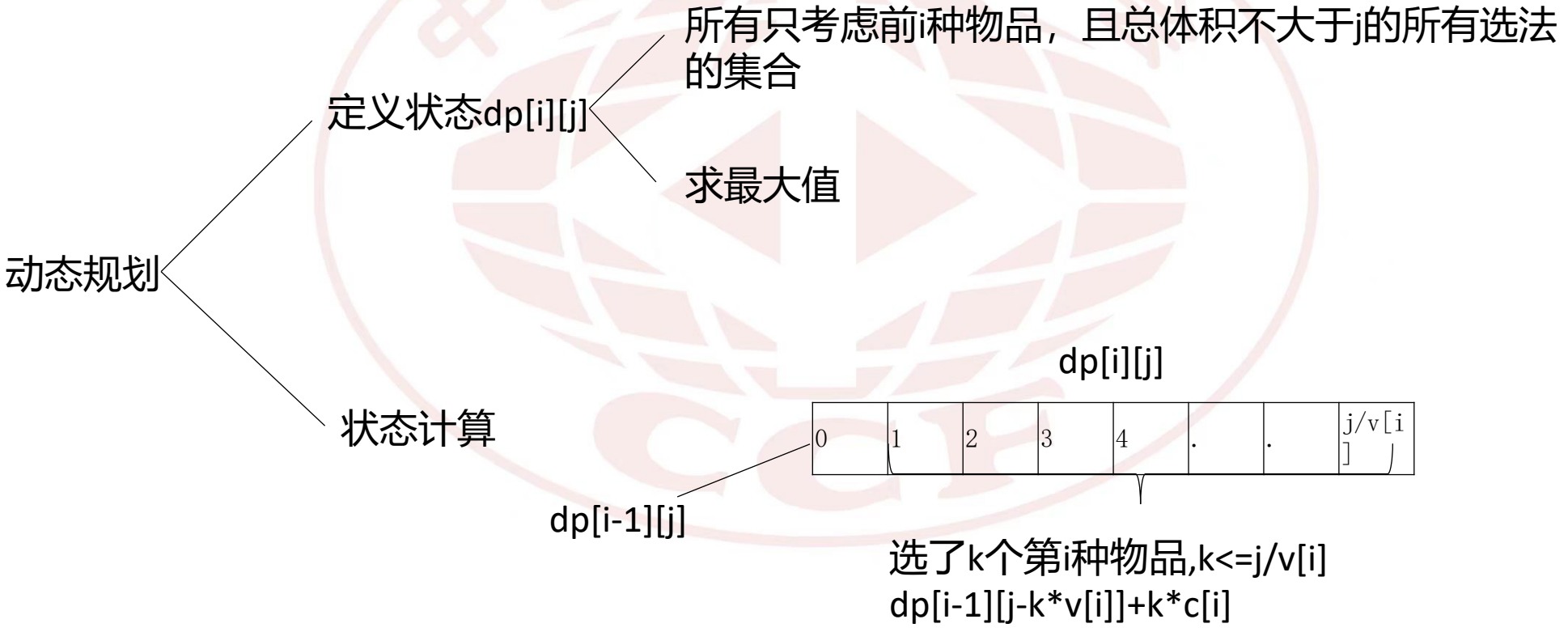
```
cin>>n>>m;
for (int i=1;i<=n;i++) cin>>v[i]>>c[i];
for (int i=1;i<=n;i++)
  for (int j=m;j>=v[i];j--)
  {
     dp[j]=max(dp[j],dp[j-v[i]]+c[i]);
  }
cout<<dp[m]<<endl;
```



三、背包问题及其应用

例4 完全背包

给出一个容量为m的背包，n种物品，每种物品有一个体积v[i]和价值c[i]，一种物品可以放任意个，求这个背包里能装下的物品价值最大是多少。n,m<=1000



完全背包朴素算法核心代码：

```
for (int i=1; i<=n; i++)
    for (int j=0; j<=m; j++)
       for (int k=0;k*v[i]<=j;k++)
         dp[i][j] = max(dp[i][j], dp[i-1][j-k*v[i]]+k*c[i]);
cout<<dp[n][m];
```

时间复杂度0(nm^2)

完全背包算法优化：

dp[i-1][j-k*v[i]]+k*c[i])

dp[i][j]=max(dp[i-1][j],dp[i-1][j-v[i]]+c[i],dp[i-1][j-2*v[i]]+2*c[i]，dp[i-1][j-3*v[i]+3*c[i].....)
dp[i][j-v[i]]=max(　　dp[i-1][j-v[i]],　　dp[i-1][j-2*v[i]]+c[i],　dp[i-1][j-3*v[i]]+2*c[i].....)

dp[i][j]=max(dp[i-1][j],dp[i][j-v[i]]+c[i])

完全背包算法代码优化：

```
for (int i=1; i<=n; i++)
    for (int j=0; j<=m; j++)
      {
           dp[i][j]=dp[i-1][j];
           if (j>=v[i])  dp[i][j] = max(dp[i][j], dp[i][j-v[i]]+c[i]);
      }
cout<<dp[n][m];
```

完全背包算法代码空间优化：

```
for (int i=1; i<=n; i++)
    for (int j=0; j<=m; j++)
        {
        dp[j] = max(dp[j], dp[j-v[i]]+c[i]);
        }
cout<<dp[m];
```

例5 多重背包

给出一个容量为m的背包，n种物品，第i种物品有s[i]件，每件物品体积v[i]和价值c[i]，求这个背包里能装下的物品价值最大是多少。n, m<=1000, s[i]<=1000

动态规划

定义状态dp[i][j]

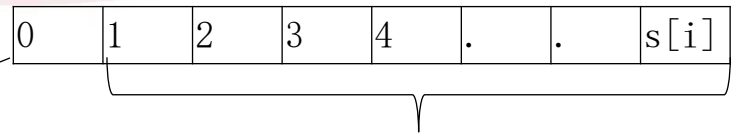
所有只考虑前i种物品中选，且总体积不大于j的所有选法的集合

求最大值

状态计算

dp[i-1][j]

dp[i][j]



选了k个第i种物品,k<=s[i]

dp[i-1][j-k*v[i]]+k*c[i]

多重背包朴素算法核心代码：

```
for (int i=1;i<=n;i++)
    for (int j=0;j<=m;j++)
      for (int k=0;k<=s[i]&&k*v[i]<=j;k++)
        dp[i][j]=max(dp[i][j],dp[i-1][j-k*v[i]]+k*c[i]);
  cout<<dp[n][m]<<endl;
```

多重背包朴素算法优化：

用完全背包的方法发现不行（自己去证明）

二进制优化多重背包

100

1 2 4 8 16 32 37

三、背包问题及其应用

例6 NOIP2014提高组 飞扬的小鸟

题目链接

任务：

1、给出一个长为n、高为m的平面，其中有k组管道（线段）。

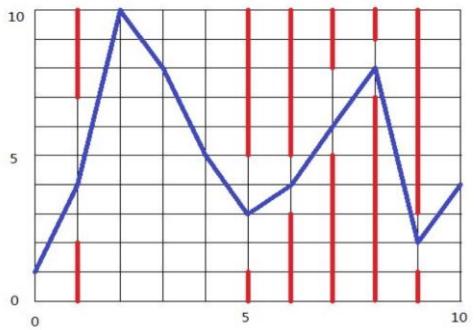
2、小鸟从第0列任意整数高度位置出发，到达第n列时，游戏完成。

3、小鸟每个单位时间沿横坐标方向右移的距离为1，竖直移动的距离由玩家控制。如果点击屏幕，小鸟就会上升一定高度X，每个单位时间可以点击多次，效果叠加；如果不点击屏幕，小鸟就会下降一定高度Y。小鸟位于不同列时，X和Y可能互不相同。

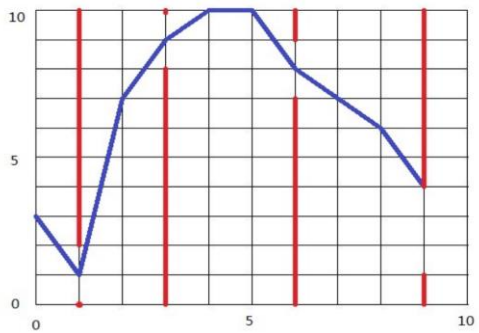
4、小鸟高度等于0或者小鸟碰到管道时，游戏失败 。小鸟高度为m时，无法再上升。注意：当从位置P上升X时，若P+X>m，则新位置P′ =m。

5、现在，请你判断是否可以完成游戏。如果可以，输出最少点击屏幕数；否则，输出小鸟最多可以通过多少个管道缝隙。

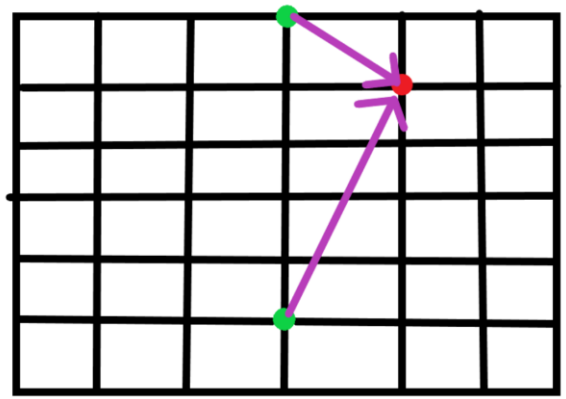
数据范围：5≤n≤10000，5≤m≤1000，0≤k<n。



输入输出样例1说明



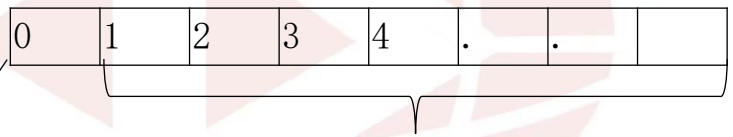
输入输出样例2说明



三、背包问题及其应用

例6 NOIP2014提高组 飞扬的小鸟

动态规划
- 定义状态dp[i][j]
  - 表示小鸟要到横坐标为i ,高度为 j的位置上的点击次数的集合
  - 求小值
- 状态计算
  - dp[i][j]=min(dp[i-1][j+y[i]])
  - dp[i][j]



点了k次

dp[i][j]=min(dp[i-1][j-k*x[i]]+k)

算法优化：

dp[i][j]=min(dp[i-1][j-k*x[i]]+k)

dp[i][j]=min(dp[i-1][j-x[i]]+1,dp[i-1][j-2*x[i]]+2，dp[i-1][j-3*x[i]+3……)

dp[i][j-x[i]]=min(dp[i-1][j-2*x[i]]+1, dp[i-1][j-3*x[i]]+2……)

dp[i][j]=min(dp[i-1][j-x[i]]+1,dp[i][j-x[i]]+1)

时间复杂度O(nm)

三、背包问题及其应用

例6 NOIP2014提高组 飞扬的小鸟

细节的处理

1、dp[i][j]的初始值

首先初始化每一层DP数组成INF:

2、然后就开始转移从下面跳上来的情况:注意处理超过高度m的情况

```
for(int j=x[i]+1;j<=x[i]+m;j++)dp[i][j]=min(dp[i-1][j-x[i]]+1,dp[i][j-x[i]]+1);
```

注意，这里的j要循环到x[i]+m

因为跳到m外的情况也要转移出来，方便下面处理跳到m就不能再往上跳了的情况。

```
for(int j=m+1;j<=m+x[i];j++)dp[i][m]=min(dp[i][m],dp[i][j]);
```

3、处理掉下来的情况:

```
for(int j=1;j<=m-y[i];j++)dp[i][j]=min(dp[i][j],dp[i-1][j+y[i]]);
```

4、一定要在处理完跳起来的情况后在处理掉下去的情况 !!!

因为掉完了之后不能再跳,如果顺序错了会WA的很惨。

5、处理遇到柱子的情况:

```
//now:下一个跳到的柱子编号
if(i==g[now].x){//遇到柱子
        for(int j=g[now].l;j>=0;j--)dp[i][j]=INF;//从下边界一直到0都不能被跳到了
        for(int j=g[now].r;j<=m;j++)dp[i][j]=INF;//从上边界一直到m都不能被跳到了
        ans=INF;//检查能不能跳出这个柱子
        for(int j=1;j<=m;j++)ans=min(ans,dp[i][j]);
        if(ans==INF){puts("0");printf("%d\n",now-1);return 0;}//如果不能跳出就输出now-1(上一次跳过的)
        now++;//跳过去
}
```

三、背包问题及其应用

例6 NOIP2014提高组 飞扬的小鸟

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=10005,M=1005,INF=0x3FFFFFFF;
int dp[N][M],x[N],y[N],n,m,k,now=1,ans=INF;
struct guandao{//管道结构体
    int x,l,r;
}g[N];
bool cmp(guandao a,guandao b){return a.x<b.x;}
int main(){
    scanf("%d%d%d",&n,&m,&k);
    for(int i=1;i<=n;i++)scanf("%d%d",&x[i],&y[i]);
    for(int i=1;i<=k;i++)scanf("%d%d%d",&g[i].x,&g[i].l,&g[i].r);//输入
    sort(g+1,g+k+1,cmp);//先排序
    for(int i=1;i<=n;i++){
        for(int j=0;j<=m;j++)dp[i][j]=INF;//初始化
        for(int j=x[i]+1;j<=x[i]+m;j++)dp[i][j]=min(dp[i-1][j-x[i]]+1,dp[i][j-x[i]]+1);//转移从下面跳上来的情况
        for(int j=m+1;j<=m+x[i];j++)dp[i][m]=min(dp[i][m],dp[i][j]);//处理跳到了m就不能再往上跳了的情况
        for(int j=1;j<=m-y[i];j++)dp[i][j]=min(dp[i][j],dp[i-1][j+y[i]]);//处理掉下来的情况
        //now:下一个跳到的柱子编号
        if(i==g[now].x){//遇到柱子
            for(int j=g[now].l;j>=0;j--)dp[i][j]=INF;//从下边界一直到0都不能被跳到了
            for(int j=g[now].r;j<=m;j++)dp[i][j]=INF;//从上边界一直到m都不能被跳到了
            ans=INF;//检查能不能跳出这个柱子
            for(int j=1;j<=m;j++)ans=min(ans,dp[i][j]);
            if(ans==INF){puts("0");printf("%d\n",now-1);return 0;}//如果不能跳出就输出now-1(上一次跳过的)
            now++;//跳过去
        }
    }
    ans=INF;
    for(int i=1;i<=m;i++)ans=min(ans,dp[n][i]);
    puts("1");
    printf("%d\n",ans);//输出
    return 0;
}
```

三、背包问题及其应用

例6 NOIP2018提高组 货币系统

题目链接

题意：

1、有n种不同面值的货币，第 i 种面值为a[i]，每种无限张。将一个有n种货币、面值数组为a[1,n]的货币系统记作(n,a)。

2、两个货币系统（n,a）和（m,b）是等价的，当且仅当对于任意非负整数x，它要么均可以被两个货币系统表出，要么不能被其中任何一个表出。

3、给定一个货币系统（n,a），找到一个最小的m使得存在货币系统(m,b)与(n,a)等价。

(n≤100,a[i]≤25000)

分解——模式——抽象——算法

1、分解：

(4,[3,19,10,6]) 与 (2,[3,10])等价

(5,[11,29,13,19,17])只能与自己等价

(2,[2,3])只能与自己等价

(3,[2,3,4])与(2,[2,3])等价

(3,[3,6,9])与(1,[3])等价

(4,[3,10,15,20])与(3,[3,10,15]等价

2、模式识别：

1、将货币面额排序（因为给的面额是无序的）

2、每一个面额考查它能不能被它之前的面额描述出来，如果能，它就没有存在的必要。

三、背包问题及其应用

例6 NOIP2018提高组 货币系统

3、抽象：

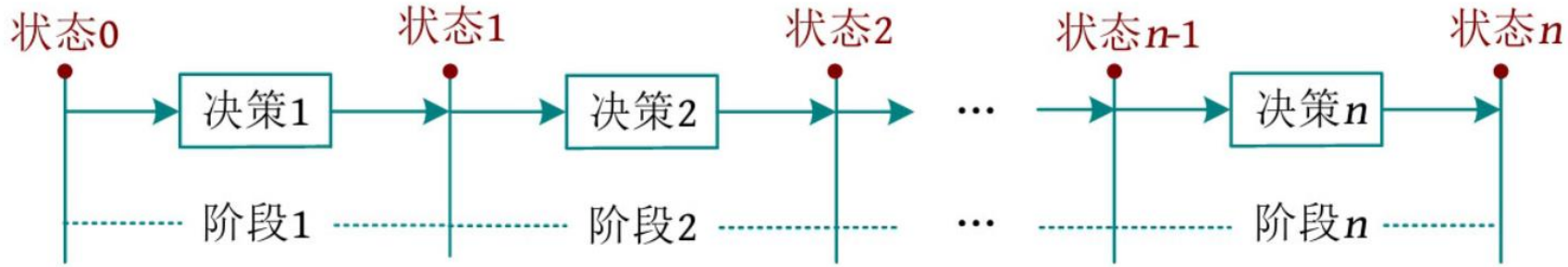
问题转化为判断ai能不能被表示出来就是判断能不能从a1——ai-1组成恰好值为ai的完全背包背包。

4、算法

```
memset(dp,0,sizeof dp);
dp[0]=1;
for(int i=1;i<=n;i++)cin>>a[i];
sort(a+1,a+1+n);
m=a[n];
int ans=0;
for(int i=1;i<=n;i++)
{
    if(!dp[a[i]]) ans++;
    for(int j=0;j<=m;j++)
    {
       if(j>=a[i])dp[j]+=dp[j-a[i]];
    }
}
 cout<<ans<<endl;
}
```

四、线性动规与区间动归算法及其应用

具有线性阶段划分的动态规划算法叫作线性动态规划（简称线性DP）。若状态包含多个维度，则每个维度都是线性划分的阶段，也属于线性DP，如下图所示：



例7 最长上升子序列

给定一个长度为n的数列，求数值严格单调递增的子序列的长度最长是多少。1≤n≤100000

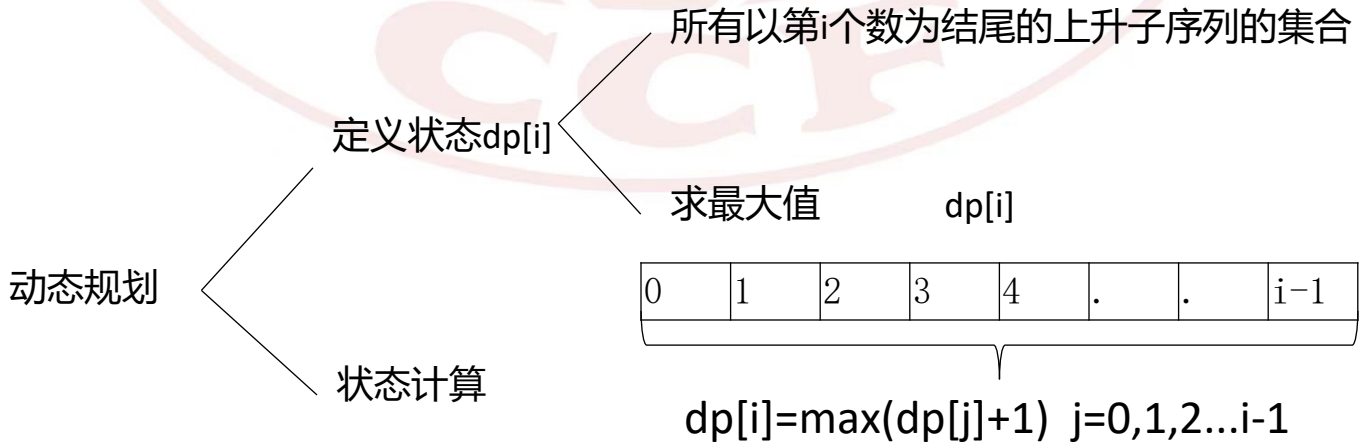
输入样例

7

3 1 2 1 8 5 6

输出样例

4



例7 最长上升子序列O(n²)核心代码：

```
for (int i=1;i<=n;i++)
    {
      dp[i]=1;
      for (int j=1;j<i;j++)
        {
          if (a[j]<a[i]) dp[i]=max(dp[j]+1,dp[i]);
        }
    }
    int res=0;
    for (int i=1;i<=n;i++) res=max(res,dp[i]);
    cout<<res<<endl;
```

优化：

7

3 1 2 1 8 5 6

分析：a[]为原数列数组，f[]为严格上升子序列，并且每个位置上的数最小的序列

1、贪心：长度为r的一个严格上升子序列，每个位置上的数值越小越有利于后面数进入到答案数列。

2、策略：遍历a数组的每个元素，判断它是否大于f数组的尾元素，如果大于，则把它也加入到f数组之中，f数组增长；如果不大于，则从头遍历f数组，如果该数小于其中一个数，将其替换，因为f数组中的数越小越好，其越小，后面的数才容易进入。由于单调上升，所以我们可以使用二分去找。由于单调上升，所以我们可以使用二分去找。

时间代价：O(nlogn)

四、线性动规与区间动归算法及其应用

区间DP：属于线性DP的一种，它以“区间长度”作为DP的“阶段”，使用两个坐标（区间的左、右端点）描述每个维度。一个状态通常由被他包含且比它更小的区间状态转移而来。

例8 NOI1995 石子合并

设有N堆沙子排成一排，其编号为1，2，3，…，N（N<=300）。每堆沙子有一定的数量，可以用一个整数来描述，现在要将这N堆沙子合并成为一堆，每次只能合并相邻的两堆，合并的代价为这两堆沙子的数量之和，合并后与这两堆沙子相邻的沙子将和新堆相邻，合并时由于选择的顺序不同，合并的总代价也不相同，如有4堆沙子分别为 1 3 5 2 我们可以先合并1、2堆，代价为4，得到4 5 2 又合并 1，2堆，代价为9，得到9 2 ，再合并得到11，总代价为4+9+11=24，如果第二步是先合并2，3堆，则代价为7，得到4 7，最后一次合并代价为11，总代价为4+7+11=22；问题是：找出一种合理的方法，使总的代价最小。输出最小代价。

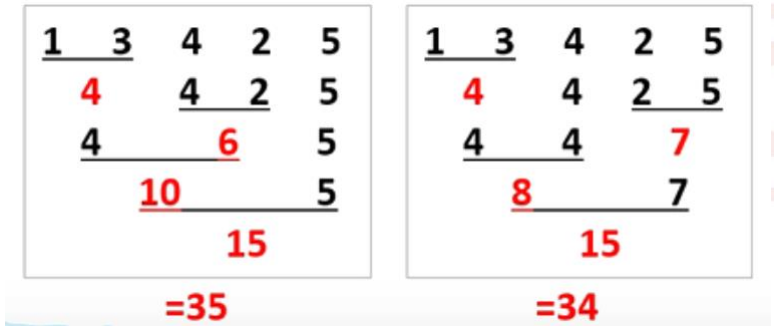
输入样例：

4

1 3 5 2

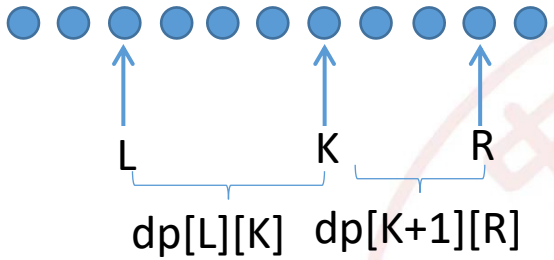
输出样例：

22



四、线性动规与区间动归算法及其应用

例8 NOI1995 石子合并



1、把区间[L,R]切分成2部分[L,K]和[K+1,R],K是切分点
2、再把两部分[L,K]和[K+1,R]合并到一起

1、定义状态：dp[L][R]表示把从L到R的石子合并成一堆的最小代价

2、状态计算：dp[L][R]=min(dp[L][R],dp[L][K]+dp[K+1][R]+S[R]-S[L-1])

3、初值：dp[i][i]=0,其余为正无穷。

4、目标：dp[1][n]

例8 NOI1995 石子合并O(n³)核心代码：

```
for (int i=1;i<=n;i++) scanf("%d",&a[i]);
  for (int i=1;i<=n;i++) s[i]=s[i-1]+a[i];
  for (int len=2;len<=n;len++)  //阶段：枚举区间长度
    for (int i=1;i+len-1<=n;i++)//状态：枚举区间起点
      {
       int l=i,r=i+len-1;
       dp[l][r]=1e8;
       for (int k=l;k<r;k++)//决策：枚举分割点
         dp[l][r]=min(dp[l][r],dp[l][k]+dp[k+1][r]+s[r]-s[l-1]);
      }
     printf("%d\n",dp[1][n]);
```

阶段：若干状态
Len=2: [1, 2], [2, 3], [3, 4], [4, 5]
Len=3: [1, 3], [2, 4], [3, 5]
Len=4: [1, 4], [2, 5]
Len=5: [1, 5]

状态：若干决策（分割点k）
Len=4: 对[2, 5]切分，
k=2: [2, 2]+[3, 5]
k=3: [2, 3]+[4, 5]
k=4: [2, 4]+[5, 5]

例9 NOI2009 二叉查找树

【问题描述】

已知一棵特殊的二叉查找树。根据定义，该二叉查找树中每个结点的**数据值**都比它左子树结点的**数据值**大，而比它右子树结点的**数据值**小。

另一方面，这棵查找树中每个结点都有一个**权值**，每个结点的**权值**都比它的儿子结点的**权值**要小。

已知树中所有结点的**数据值**各不相同；所有结点的**权值**也各不相同。这时可得出这样一个有趣的结论：如果能够确定树中每个结点的**数据值**和**权值**，那么树的形态便可以唯一确定。因为这样的一棵树可以看成是按照**权值**从小到大顺序插入结点所得到的、按照**数据值**排序的二叉查找树。

一个结点在树中的**深度**定义为它到树根的距离加 1。因此树的根结点的深度为 1。

每个结点除了**数据值**和**权值**以外，还有一个**访问频度**。我们定义一个结点**在树中的访问代价**为它的**访问频度**乘以它在树中的**深度**。**整棵树的访问代价**定义为所有结点**在树中的访问代价**之和。

现在给定每个结点的**数据值**、**权值**和**访问频度**，你可以根据需要修改某些结点的**权值**，但每次修改你会付出 K 的**额外修改代价**。你可以把结点的**权值**改为任何实数，但是修改后所有结点的**权值**必须仍保持互不相同。现在你要解决的问题是，**整棵树的访问代价**与**额外修改代价**的和最小是多少？

【输入文件】

输入文件名为 treapmod.in。

输入文件第一行包含两个正整数 N 和 K。N 为结点的个数，K 为每次修改所需的**额外修改代价**。

接下来一行包含 N 个非负整数，是每个结点的**数据值**。

再接下来一行包含 N 个非负整数，是每个结点的**权值**。

再接下来一行包含 N 个非负整数，是每个结点的**访问频度**。

所有的数据值、权值、访问频度均不超过 400000。每两个数之间都有一个空格分隔，且行尾没有空格。

【输出文件】

输出文件名为 treapmod.out。输出文件只有一个数字，即你所能得到的**整棵树的访问代价**与**额外修改代价**之和的最小值。

【输入样例】

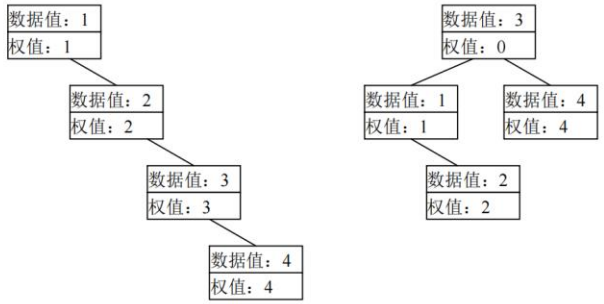
```
4 10
1 2 3 4
1 2 3 4
```


【输出样例】

```
29
```

【样例说明】

输入的原图是左图，它的**访问代价**是 1×1+2×2+3×3+4×4=30。最佳的修改方案是把输入中的第 3 个结点的权值改成 0，得到右图，**访问代价**是 1×2+2×3+3×1+4×2=19，加上**额外修改代价** 10，一共是 29。



【数据规模】

40%的数据满足 $N \le 30$;
70%的数据满足 $N \le 50$;
100%的数据满足 $N \le 70$, $1 \le K \le 30000000$。

例9 NOI2009 二叉查找树

发掘一下本题的细节与性质

1、每个节点的权值可以改，甚至可以改成实数，那么权值的作用是什么？

2、结点在树中的访问代价为它的访问频度乘以它在树中的深度，与数值、权值均无关？

Treap

3、二叉查找树又有一个性质：二叉查找树的中序遍历是其代表的序列从小到大排序的结果。

4、平衡树难以抓住形态的变化，也不方便记录深度之类。所以必须抓住不变的量当做突破口。
不变的是无论Treap如何旋转怎么转，根据二叉树的定义，它的中序遍历一定是不变的。

5、这棵树的中序遍历，就把这棵树变成了一个静态的区间，只不过每个区间所代表的点的优先级可能会变。

6、区间，序列，最优值——区间DP

7、n的大小给我们的启示 N<=70

例9 NOI2009 二叉查找树

定义状态dp[i][j][k]

表示i到j节点组成了一棵树，其中权值都>=k，所需要的代价为多少集合

求最小值 dp[i][j][k]

动态规划

状态计算 dp[i][j][k] = min(dp[i][j][k], dp[i][t - 1][k] + dp[t + 1][j][k] + K + i到j的访问频率之和

dp[i][j][k]=min(dp[i][j][k],dp[i][t−1][t的权值]+dp[t+1][j][t的权值]+i到j的访问频率之和) 注意此时t的权值>=k

1、阶段：一个小区间就是一个阶段，一个小区间是一个子树，逐渐扩展到大区间，最后扩展到整棵树
2、目标：dp[1][n][1]
3、访问代价 = 深度 × 访问频率是怎么体现的。
当我们一层层计算时，每次都会将当前子树内的访问频率加一遍，累积起来就是访问代价。
4、边界：dp[i][i][k]?还是其他
dp[i][i-1][k]=0,因为当只有一个节点时，左右子树没有时空，空时应为0
5、dp[i][j][k],k值题目中给定的是400000，离散化。
6、i到j的访问频率之和：前缀和

```cpp
const int maxn = 80;
const int maxm = 4e5 + 10;
int n, K, val[maxn], sum[maxn], dp[maxn][maxn][maxn];
struct node{
    int num, val, sum;
}a[maxn];
bool cmp(node a, node b)
{
    return a.num < b.num;
}
int main()
{
    cin>>n>>K;
    for(int i = 1; i <= n; i++) cin>>a[i].num;
    for(int i = 1; i <= n; i++)
    {
       cin>>a[i].val;
       val[i]=a[i].val;
    }
    for(int i = 1; i <= n; i++) cin>>a[i].sum;
    sort(a + 1, a + n + 1, cmp);
    sort(val + 1, val + n + 1);
    for(int i = 1; i <= n; i++)
    {
        a[i].val = lower_bound(val + 1, val + n + 1, a[i].val) - val;//离散化
        sum[i] = sum[i - 1] + a[i].sum;
    }
    memset(dp, 63, sizeof dp);
     for(int i = 1; i <= n + 1; i++)
        for(int j = 1; j <= n; j++) dp[i][i - 1][j] = 0;
    for(int i = n; i >= 1; i--)
        for(int j = i; j <= n; j++)
            for(int k = 1; k <= n; k++)
                for(int t = i; t <= j; t++)
                {
                    if(a[t].val >= k)
                    dp[i][j][k] = min(dp[i][j][k], dp[i][t - 1][a[t].val] + dp[t + 1][j][a[t].val] + sum[j] - sum[i - 1]);
                    dp[i][j][k] = min(dp[i][j][k], dp[i][t - 1][k] + dp[t + 1][j][k] + K + sum[j] - sum[i - 1]);
                }
    printf("%d", dp[1][n][1]);
    return 0;
}
```

五、数位统计动态规划算法及其应用

数位统计动态规划用于数字的数位统计。

例10 [ZJOI2010]数字计数

给定两个正整数 a 和 b，求在[a,b]中的所有整数中，每个数码(digit)各出现了多少次。

输入格式：仅包含一行两个整数 a,b含义如上所述。 $1 \leqslant a \leqslant b \leqslant 10^{12}$

输出格式：包含一行十个整数，分别表示 0~9 在 [a,b]中出现了多少次。

输入样例：1 99

输出样例：9 20 20 20 20 20 20 20 20 20

分析：

1、由于a和b的值太大，用暴力方法直接统计[a,b]内每个整数显然会超时

2、转化题目要求：区间[a,b]等于[1,a-1]与[1,b]的差分，把问题转化成[1,n]区间内，统计0~9出现的次数

3、定义dp[i],为i位数每种数字有都少个

dp[1]=1

dp[2]=20

$dp[i]=dp[i-1]*10+10^{(i-1)}$

五、数位统计动态规划算法及其应用

例10 [ZJOI2010]数字计数

n=abcdefg

dp[i]为i位数每种数字有都少个

ten[i]为10的i次方

num[i]为数字n每个位置的值

cnt[j]为j这个数出现的次数

从高位到低位进行处理，当前处理到最高位i=7,num[7]=a

1、j为0~9　　cnt[j]+=dp[i-1]*num[i]

2、j为0~num[i]-1　　cnt[j]+=ten[i-1]

3、j为num[i]　　cnt[j]+=bcdefg+1

4、处理前导0　　cnt[0]-=ten[i-1]

例如以n=324为例，先从最高位开始，每种数字出现的次数cnt计算如下：

1、普通情况：例如00~99共出现了3次，cnt[j]=dp[i-1]*num[i]=dp[2]*3=60

2、特判最高位，即“数位限制”，数字0,1,2分别在000~099,100~199,200~299出现了100次

3、特判最高位，数字3，在300~324中出现了25次

4、处理前导0,需要减去ten[i-1]个，共100个

例10 [ZJOI2010]数字计数

```
const int N=15;
long long ten[N],dp[N],cnta[N],cntb[N];
int num[N];
void init()
{
  ten[0]=1;
  for (int i=1;i<=N;i++)
    {
        dp[i]=dp[i-1]*10+ten[i-1];
        ten[i]=10*ten[i-1];
    }
}
void solve(long long n,long long *cnt)
{
      int len=0;
      while(n)
       {
        num[++len]=n%10;
        n=n/10;
       }
       for (int i=len;i>=1;i--)
       {
          for(int j=0;j<=9;j++)  cnt[j]+=dp[i-1]*num[i];
          for(int j=0;j<num[i];j++) cnt[j]+=ten[i-1];
          long long d=0;
          for (int j=i-1;j>=1;j--)  d=d*10+num[j];
          cnt[num[i]]+=d+1;
          cnt[0]-=ten[i-1];
       }
}
int main()
{
    init();
    long long a,b;
    cin>>a>>b;
    solve(a-1,cnta),solve(b,cntb);
    for (int i=0;i<=9;i++) cout<<cntb[i]-cnta[i]<<" ";
}
```

五、数位统计动态规划算法及其应用

例10 [ZJOI2010]数字计数(数学解法)

n=abcdefg

分别求出1在每一个位置上出现的次数

比如想求出1在第4位上出现的次数

1<=xxx1yyy<=abcdefg

分2种情况：

1、xxx=000~abc-1 yyy=000~999 1出现次数为：abc*1000

2、xxx=abc

(1)d<1 abc1yyy>abcdefg 1出现次数为：0

(2)d=1 yyy=000~efg 1出现次数为：efg+1

(3)d>1 yyy=000~999 1出现次数为：1000

以为例3234中1在十位上出现的次数

1、前两位 00~31 第四位 0~9 1出现次数为：32*10=320

2、当前两位是32 十位上的数字3>1 第四位0~9 1出现次数为：10

因此3234中1在十位上出现的次数330次

五、数位统计动态规划算法及其应用

例10 [ZJOI2010]数字计数(数学解法)

```
#include<bits/stdc++.h>
using namespace std;
long long a,b;
long long work(long long k,long long num){
        if(k==0&&num==0)return 0;   //work(a-1,i)会出现work(0,0)
        long long x=10,ans=0;
        while(1){
                long long l=k/x,r=k%x;  //把一个数分离左右两部分
                ans+=l*x/10;          //对于左侧部分值是0000~l-1时，方法数
                if(r/(x/10)==num)ans+=r%(x/10)+1;    //对于左侧部分值是l时,当前位的值恰好等于num,方法数
                else{
                        if(num==0&&l==0)break;
                        if(r/(x/10)>num)ans+=x/10;  //对于左侧部分值是l时,当前位的值恰好大于num,方法数

                }
                if(l==0)break;
                if(num==0)ans-=x/10;  //当统计的num值是0时，把前导0的方法数去掉
                x*=10;
        }
        return ans;
}
int main(){
        cin>>a>>b;
        for(int i=0;i<=9;i++)cout<<work(b,i)-work(a-1,i)<<" ";
}
```

五、数位统计动态规划算法及其应用

例11 [SCOI2009]windy数

windy定义了一种windy数。不含前导零且相邻两个数字之差至少为2的正整数被称为windy数。 windy想知道，在A和B之间，包括A和B，总共有多少个windy数？

六、状态压缩动态规划算法及其应用

状态压缩动态规划算法是一个小技巧，一般应用在集合问题中。当DP状态是集合时，把集合的组合或者排列用一个二进制数表示。

例12 最短哈密顿路径

给定一个有权无向图，包括n个点，标记为0 ~ n-1，以及连接n个点的边，求从起点0到终点n-1的最短路径。要求必须经过所有点，而且只经过一次。1 ≤ n ≤ 20。

输入格式：

第一行输入整数n。接下来n行每行n个整数，其中第i行第j个整数表示点i到j的距离（记为a[i，j]）。0 ≤ a[i，j] ≤ 10^7

对于任意的x，y，z，数据保证 a[x，x]=0，a[x，y]=a[y，x] 并且 a[x，y]+a[y，z]>=a[x，z]。

输出格式：

输出一个整数，表示最短哈密顿路径的长度。

输入样例：

```
5
0 2 4 5 1
2 0 6 5 3
4 6 0 8 3
5 5 8 0 5
1 3 3 5 0
```

输出样例：

```
18
```

暴力的方案，n!*n

六、状态压缩动态规划算法及其应用

例12 最短哈密顿路径

定义状态dp[i][j]

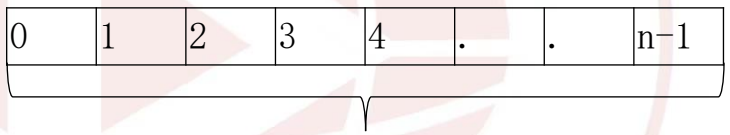
所有从0点走到j,经过i中所有点的路径的集合,i是一个二进制压缩数
例如i=00110101,表示集合包含0、2、4、5几个点

求最小值

动态规划

状态计算

dp[i][j]



k，枚举集合i-j所有的点

dp[i][j]=min(dp[i-{j}][k]+dis[k][j]) k=0,1,2...n-1

六、状态压缩动态规划算法及其应用

例12 最短哈密顿路径

```
using namespace std;

const int N = 20, M = 1 << N;

int n;
int w[N][N];
int f[M][N];

int main()
{
    cin >> n;
    for (int i = 0; i < n; i ++ )
        for (int j = 0; j < n; j ++ )
            cin >> w[i][j];

    memset(f, 0x3f, sizeof f);
    dp[1][0] = 0;

    for (int i = 0; i < 1 << n; i ++ )
        for (int j = 0; j < n; j ++ )
            if (i >> j & 1)
                for (int k = 0; k < n; k ++ )
                    if (i >> k & 1)
                        dp[i][j] = min(dp[i][j], dp[i - (1 << j)][k] + w[k][j]);

    cout << dp[(1 << n) - 1][n - 1];

    return 0;
}
```

例13 NOIP2017宝藏

【问题描述】

参与考古挖掘的小明得到了一份藏宝图，藏宝图上标出了 n 个深埋在地下的宝藏屋，也给出了这 n 个宝藏屋之间可供开发的 m 条道路和它们的长度。

小明决心亲自前往挖掘所有宝藏屋中的宝藏。但是，每个宝藏屋距离地面都很远，也就是说，从地面打通一条到某个宝藏屋的道路是很困难的，而开发宝藏屋之间的道路则相对容易很多。

小明的决心感动了考古挖掘的赞助商，赞助商决定免费赞助他打通一条从地面到某个宝藏屋的通道，通往哪个宝藏屋则由小明来决定。

在此基础上，小明还需要考虑如何开凿宝藏屋之间的道路。已经开凿出的道路可以任意通行不消耗代价。每开凿出一条新道路，小明就会与考古队一起挖掘出由该条道路所能到达的宝藏屋的宝藏。另外，小明不想开发无用道路，即两个已经被挖掘过的宝藏屋之间的道路无需再开发。

新开发一条道路的代价是：

这条道路的长度 × 从赞助商帮你打通的宝藏屋到这条道路起点的宝藏屋所经过的宝藏屋的数量（包括赞助商帮你打通的宝藏屋和这条道路起点的宝藏屋）。

请你编写程序为小明选定由赞助商打通的宝藏屋和之后开凿的道路，使得工程总代价最小，并输出这个最小值。

【输入格式】

输入文件名为 treasure.in。

第一行两个用空格分离的正整数 n 和 m，代表宝藏屋的个数和道路数。

接下来 m 行，每行三个用空格分离的正整数，分别是由一条道路连接的两个宝藏屋的编号（编号为 1~n），和这条道路的长度 v。

【输出格式】

输出文件名为 treasure.out。

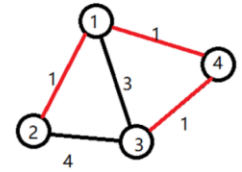
输出共一行，一个正整数，表示最小的总代价。

【输入输出样例 1】

treasure.in	treasure.out
4 5 1 2 1 1 3 3 1 4 1 2 3 4 3 4 1	4

见选手目录下的 treasure/treasure1.in 与 treasure/treasure1.ans

【输入输出样例 1 说明】



小明选定让赞助商打通了 1 号宝藏屋。小明开发了道路 1→2，挖掘了 2 号宝藏。开发了道路 1→4，挖掘了 4 号宝藏。还开发了道路 4→3，挖掘了 3 号宝藏。工程总代价为：1×1 + 1×1 + 1×2 = 4

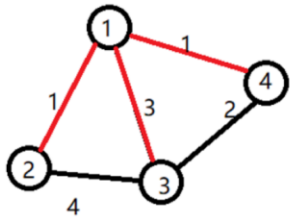
(1→2) (1→4) (4→3)

【样例输入输出 2】

treasure.in	treasure.out
4 5 1 2 1 1 3 3 1 4 1 2 3 4 3 4 2	5

见选手目录下的 treasure/treasure2.in 与 treasure/treasure2.ans。

【输入输出样例 2 说明】



小明选定让赞助商打通了 1 号宝藏屋。小明开发了道路 1→2，挖掘了 2 号宝藏。开发了道路 1→3，挖掘了 3 号宝藏。还开发了道路 1→4，挖掘了 4 号宝藏。工程总代价为：1×1 + 3×1 + 1×1 = 5

(1→2) (1→3) (1→4)

例13 NOIP2017宝藏

简单概括题意：给出一副无向图，每条边都有一个权值且均未开通，先可以随便取一个起点，要开通一些边，使它成为一个连通图，开通一条边的代价为这条边的权值*起点到它的点的个数(起点也算)，求最小代价。

数据范围N<=12，基本上可以确定是状压DP了。

1、我们直接最简单地设置状态：dp[S]代表以及取的节点的状态为S的最小代价，那么目标状态就是dp[(1<<n)-1]。

2、如果考虑递推求解的话，我们发现需要计算花费的深度会很难枚举，这就导致了DP顺序的问题，如果选择用记忆化搜索求解的话，会方便很多。

3、由于起点不是确定的，我们需要用一重循环来枚举起点，然后以1<<(root-1)为初始状态，对每一种情况进行一次记忆化搜索，记搜的大体思路如下：

（1）对于状态S，枚举一个点i，满足i∈S

（2）在枚举一个点j，满足i,j有边相连且j∉S

（3）状态转移方程：dp[S′]=minf[S]+depth[i]*dis[i][j]

（4）更新新加入的节点j的深度

（5）搜索下一个状态S′

（6）回溯还原节点j的深度

六、状态压缩动态规划算法及其应用

```
int n,m,dis[N][N],dp[Smax],depth[N],ans=INF;
inline void input(void)
{
    memset(dis,0x3f,sizeof dis);
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++)
    {
        int u,v,val;
        scanf("%d%d%d",&u,&v,&val);
        dis[u][v]=dis[v][u]=min(dis[u][v],val);
    }
}
inline void Search(int S)
{
    for(int i=1;i<=n;i++)
    {
        if((1<<(i-1))&S)
        {
            for(int j=1;j<=n;j++)
            {
                if(not ((1<<(j-1))&S)&&(dis[i][j]<INF))
                {
                    if(dp[S|(1<<(j-1))]>dp[S]+depth[i]*dis[i][j])
                    {
                        int temp=depth[j];
                        depth[j]=depth[i]+1;
                        dp[S|(1<<(j-1))]=dp[S]+depth[i]*dis[i][j];
                        Search(S|(1<<(j-1)));
                        depth[j]=temp;
                    }
                }
            }
        }
    }
}
```

```
void solve(void)
{
    for(int root=1;root<=n;root++)
    {
        memset(dp,0x3f,sizeof dp);
        memset(depth,0x3f,sizeof depth);
        dp[1<<(root-1)]=0;
        depth[root]=1;
        Search(1<<(root-1));
        ans=min(ans,dp[(1<<n)-1]);
    }
}
int main(void)
{
    input();
    solve();
    printf("%d\n",ans);
}
```

七、树形动态规划算法及其应用

例14 NOIP2003 加分二叉树

【问题描述】

设一个n个节点的二叉树tree的中序遍历为（1,2,3,…,n），其中数字1,2,3,…,n为节点编号。每个节点都有一个分数（均为正整数），记第j个节点的分数为di，tree及它的每个子树都有一个加分，任一棵子树subtree（也包含tree本身）的加分计算方法如下：

subtree的左子树的加分× subtree的右子树的加分＋subtree的根的分数

若某个子树为主，规定其加分为1，叶子的加分就是叶节点本身的分数。不考虑它的空子树。

试求一棵符合中序遍历为（1,2,3,…,n）且加分最高的二叉树tree。要求输出；

（1）tree的最高加分

（2）tree的前序遍历

【输入格式】

第1行：一个整数n（n＜30），为节点个数。

第2行：n个用空格隔开的整数，为每个节点的分数（分数＜100）。

【输出格式】

第1行：一个整数，为最高加分（结果不会超过4,000,000,000）。

第2行：n个用空格隔开的整数，为该树的前序遍历。

【输入样例】

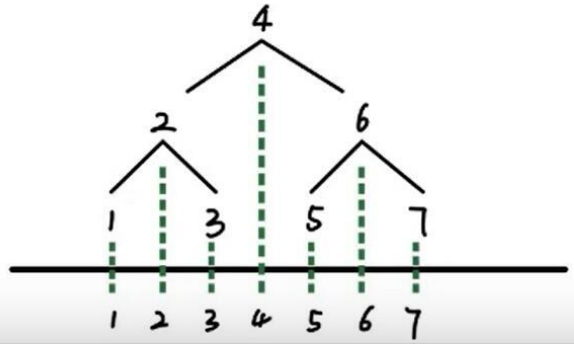
5

5 7 1 2 10

【输出样例】

145

3 1 2 4 5



七、树形动态规划算法及其应用

例14 NOIP2003 加分二叉树

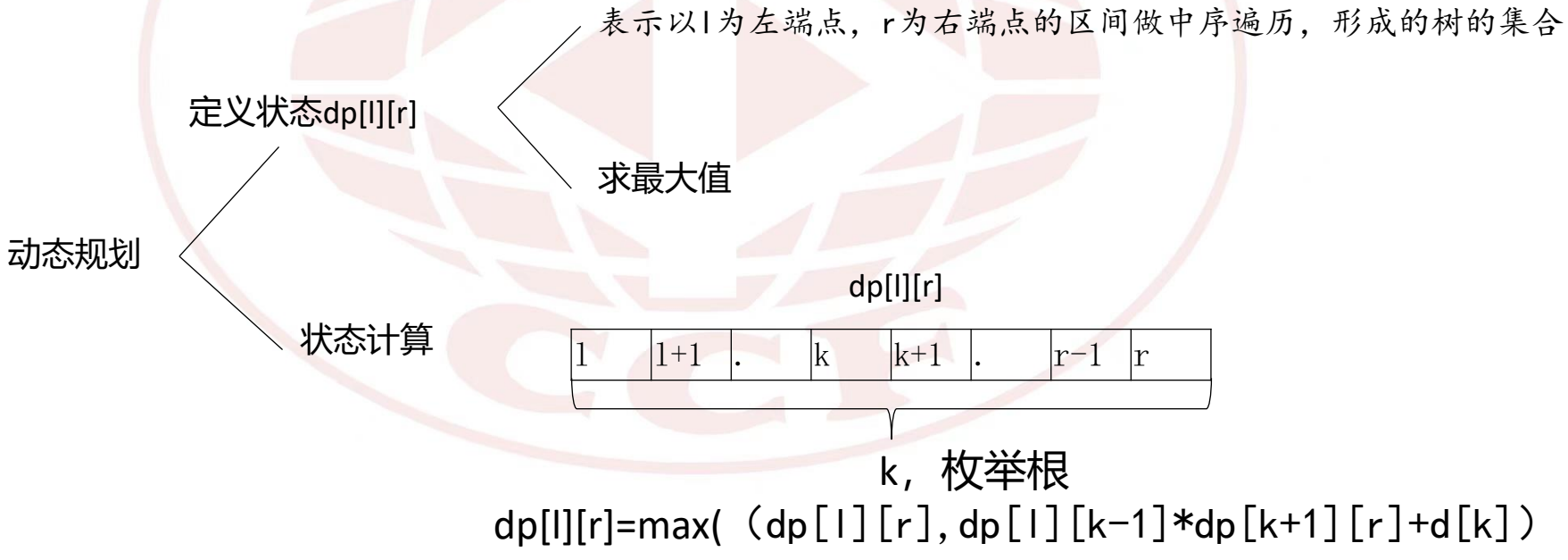
1、我们发现，任选中序遍历的某一段区间，枚举根，可以生成多棵子树

2、定义状态：dp[l][r],表示以l为左端点，r为右端点的区间做中序遍历，形成的最高加分

3、状态计算：dp[l][r]=max（dp[l][r],dp[l][k-1]*dp[k+1][r]+d[k]）

4、初始状态：dp[i][i]=d[i]

5、目标状态：dp[1][n]



七、树形动态规划算法及其应用

例14 NOIP2003 加分二叉树

```
int dp[N][N],g[N][N],d[N];
int n;
//递归进行前序遍历
void dfs(int l,int r){
    if(l<=r){
        int root = g[l][r];
        cout<<root<<" ";
        dfs(l,root-1);
        dfs(root+1,r);
    }
}
int main(){
    cin >> n;
    for(int i=1;i<=n;i++)  cin >>d[i];
    for(int len = 1;len<=n;len++){
        for(int l=1;l+len-1<=n;l++){
            int r = l+len-1;
            if(len == 1){
                dp[l][r] = d[l];//如果是叶子节点就等于 叶子节点本身分数
                g[l][r] = l;//他的根节点就是本身
            }
            else
            for(int k = l;k<=r;k++){
                int left = l==k?1:dp[l][k-1];//如果左子树为空加分为1，否则为左边最大值
                int right = r==k?1:dp[k+1][r];//如果右子树为空加分为1，否则为右边的最大值
                int socre = left*right + d[k];
                if(dp[l][r]<socre){
                    dp[l][r] = socre;
                    g[l][r] = k;//更新该区间的根节点
                }
            }
        }
    }
    cout<< dp[1][n]<<endl;
    dfs(1,n);
    return 0;
}
```

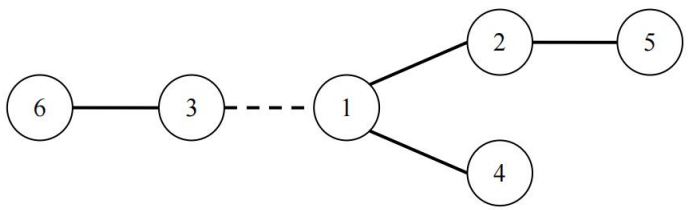
七、树形动态规划算法及其应用

例15 NOI2011 道路修建

【问题描述】

在 W 星球上有 n 个国家。为了各自国家的经济发展，他们决定在各个国家之间建设双向道路使得国家之间连通。但是每个国家的国王都很吝啬，他们只愿意修建恰好 $n-1$ 条双向道路。

每条道路的修建都要付出一定的费用，这个费用等于道路长度乘以道路两端的国家个数之差的绝对值。例如，在下图中，虚线所示道路两端分别有 2 个、4 个国家，如果该道路长度为 1，则费用为 $1\times|2-4|=2$。图中圆圈里的数字表示国家的编号。



由于国家的数量十分庞大，道路的建造方案有很多种，同时每种方案的修建费用难以用人工计算，国王们决定找人设计一个软件，对于给定的建造方案，计算出所需要的费用。请你帮助国王们设计一个这样的软件。

【输入格式】

从文件 ***road.in*** 中读入数据。

输入的第一行包含一个整数 n，表示 W 星球上的国家的数量，国家从 1 到 n 编号。

接下来 $n-1$ 行描述道路建设情况，其中第 i 行包含三个整数 a_i、b_i 和 c_i，表示第 i 条双向道路修建在 a_i 与 b_i 两个国家之间，长度为 c_i。

【输出格式】

输出到文件 ***road.out*** 中。

输出一个整数，表示修建所有道路所需要的总费用。

【样例输入】

```
6
1 2 1
1 3 1
1 4 2
6 3 1
5 2 1
```

【样例输出】

```
20
```

【数据规模与约定】

所有测试数据的范围和特点如下表所示

测试点编号	n 的规模（注意是等于号）	约定
1	$n=2$	
2	$n=10$	
3	$n=100$	
4	$n=200$	
5	$n=500$	
6	$n=600$	
7	$n=800$	
8	$n=1000$	
9	$n=10,000$	
10	$n=20,000$	$1\le a_i, b_i\le n$
11	$n=50,000$	$0\le c_i\le 10^6$
12	$n=60,000$	
13	$n=80,000$	
14	$n=100,000$	
15	$n=600,000$	
16	$n=700,000$	
17	$n=800,000$	
18	$n=900,000$	
19	$n=1,000,000$	
20	$n=1,000,000$	

七、树形动态规划算法及其应用

例16 NOI2008 设计路线

【问题描述】

Z 国坐落于遥远而又神奇的东方半岛上，在小 Z 的统治时代公路成为这里主要的交通手段。Z 国共有 n 座城市，一些城市之间由双向的公路所连接。非常神奇的是 Z 国的每个城市所处的经度都不相同，并且**最多只和一个位于它东边的城市直接通过公路相连**。Z 国的首都是 Z 国政治经济文化旅游的中心，每天都有成千上万的人从 Z 国的其他城市涌向首都。

为了使 Z 国的交通更加便利顺畅，小 Z 决定在 Z 国的公路系统中确定若干条**规划路线**，将其中的公路全部改建为铁路。

我们定义每条**规划路线**为一个长度大于 1 的城市序列，每个城市**在该序列中最多出现一次**，序列中相邻的城市之间由公路直接相连(待改建为铁路)。并且，每个城市**最多只能出现在一条规划路线中**，也就是说，任意两条**规划路线**不能有公共部分。

当然在一般情况下是不可能将所有的公路修建为铁路的，因此从有些城市出发去往首都依然需要通过乘坐长途汽车，而**长途汽车只往返于公路连接的相邻的城市之间**，因此从某个城市出发可能需要不断地换乘长途汽车和火车才能到达首都。

我们定义一个城市的“不便利值”为从它出发到首都需要乘坐的长途汽车的次数，而 Z 国的交通系统的“不便利值”为所有城市的不便利值的最大值，很明显首都的“不便利值”为 0。小 Z 想知道如何确定**规划路线**修建铁路使得 Z 国的交通系统的“不便利值”最小，以及有多少种不同的**规划路线**的选择方案使得“不便利值”达到最小。当然方案总数可能非常大，小 Z 只关心这个天文数字 mod Q 后的值。

注意：**规划路线** 1-2-3 和**规划路线** 3-2-1 是等价的，即将一条**规划路线**翻转依然认为是等价的。两个方案不同当且仅当其中一个方案中存在一条**规划路线**不属于另一个方案。

【输入格式】

输入文件 design.in 第一行包含三个正整数 N、M、Q，其中 N 表示城市个数，M 表示公路总数，N 个城市从 1~N 编号，其中编号为 1 的是首都。Q 表示上文提到的设计路线的方法总数的模数。接下来 M 行，每行两个不同的正数 a_i、b_i ($1 \le a_i, b_i \le N$)表示有一条公路连接城市 a_i 和城市 b_i。 输入数据保证一条公路只出现一次。

【输出格式】

输出文件 design.out 应包含两行。第一行为一个整数，表示最小的“不便利值”。 第二行为一个整数，表示使“不便利值”达到最小时不同的设计路线的方法总数 mod Q 的值。

如果某个城市无法到达首都，则输出两行-1。

【输入样例】

```
5 4 100
1 2
4 5
1 3
4 1
```

【输出样例】

```
1
10
```

【样例说明】

以下样例中是 10 种设计路线的方法：

(1) 4-5
(2) 1-4-5
(3) 4-5, 1-2
(4) 4-5, 1-3
(5) 4-5, 2-1-3
(6) 2-1-4-5
(7) 3-1-4-5
(8) 1-4
(9) 2-1-4
(10) 3-1-4

【数据规模和约定】

对于 20%的数据，满足 $N,M \le 10$。
对于 50%的数据，满足 $N,M \le 200$。
对于 60%的数据，满足 $N,M \le 5000$。
对于 100%的数据，满足 $1 \le N,M \le 100000$，$1 \le Q \le 120000000$。

【评分方式】

每个测试点单独评分。对于每个测试点，第一行错则该测试点得零分，否则若第二行错则该测试点得到 40%的分数。如果两问都答对，该测试点得到 100%的分数。

谢谢大家