动态规划

1、动态规划 DP 基础

1649　前缀最大值

求一个数列的所有前缀最大值之和。即：给出长度为 n 的数列 a_i，求出对于所有 1 ≤ i ≤ n，max(a_1,a_2,...,a_i) 的和。比如，有数列：666 304 692 188 596，前缀最大值为：666 666 692 692 692，和为 3408。

对于每个位置的前缀最大值解释如下：对于第 1 个数 666 ，只有一个数，一定最大；对于第 2 个数，求出前两个数的最大数，还是 666 ；对于第 3 个数，求出前 3 个数的最大数是 692…… 其余位置依次类推，最后求前缀最大值得和。

由于读入较大，数列由随机种子生成。

其中 a[1]=x，a[i]=(379 × a[i-1]+131) mod 997。（mod 代表求余数）

输入 ：一行两个正整数 n, x ，分别表示数列的长度和随机种子。(n ≤ 100000,x < 997)

输出 ：一行一个正整数表示该数列的前缀最大值之和。

输入：5 666

输出：3408

```
#include<bits/stdc++.h>//1-1649      前缀最大值      w2016010182
using namespace std;
int n,x,sum;
int a[100005],dp[100005];
int main()
{
      cin>>n>>x;
      a[1]=x;
      for(int i=2;i<=n;i++)
            a[i]=(379*a[i-1]+131)%997;
      dp[1]=a[1];
      for(int i=2;i<=n;i++)
            dp[i]=max(dp[i-1],a[i]);
      for(int i=1;i<=n;i++)
            sum=sum+dp[i];
      cout<<sum;
      return 0;
}
```

1650 - 前缀最小值

求一个数列的所有前缀最小值之和。

即：给出长度为 n 的数列 a_i，求出对于所有 1 ≤ i ≤ n，min(a_1,a_2,...,a_i) 的和。

比如，有数列：666 304 692 188 596，前缀最小值为：666 304 304 188 188，和为 1650。

对于每个位置的前缀最小值解释如下：对于第 1 个数 666 ，只有一个数，一定最小；对于第 2 个数，求出前两个数的最小数，是 304 ；对于第 3 个数，求出前 3 个数的最小数是 304 …… 其余位置依次类推，最后求前缀最小值的和。

由于读入较大，数列由随机种子生成。

其中 a[1]=x，a[i]=(379 x a[i-1]+131) mod 997。（mod 代表求余数）

输入：一行两个正整数 n, x ，分别表示数列的长度和随机种子。(n ≤ 100000,x<997)

输出：一行一个正整数表示该数列的前缀最小值之和。

输入：5 666

输出：1650

```
#include<bits/stdc++.h>//2-1650      前缀最小值
using namespace std;
int n,x,sum;
int a[100005],dp[100005];
int main()
{
	cin>>n>>x;
	a[1]=x;
	for(int i=2;i<=n;i++)
		a[i]=(379*a[i-1]+131)%997;
	dp[1]=a[1];
	for(int i=2;i<=n;i++)
		dp[i]=min(dp[i-1],a[i]);
	for(int i=1;i<=n;i++)
		sum=sum+dp[i];
	cout<<sum;
	return 0;
}
```

1651 - 跳格子

地面上有一排长度为 n 的格子 1-n，每个格子上都有一个数 x_i，开始时你在位置 0 ，每次你可以向前跳 1-2 格，然后取走格子上的数，直到跳到位置 n+1。

取走的数的和就是你的得分，现在你想知道你可能的最大得分是多少。

输入 ：一行四个整数 n,A,B,C （$n \leqslant 100000, 0 \leqslant A,B,C \leqslant 10000$），其中 n 表示格子的数量。

x[i] 由如下方式生成：

```
for (int i = 1; i <= n; i++){
  int tmp = ((long long)A * i * i + B * i + C) % 20000;
  x[i] = tmp - 10000;
}
```

输出 ：一行一个整数 ans 表示可能的最大得分。

输入 :3 1 1 1

输出 :-9993

```
#include<bits/stdc++.h>//3-1651  跳格子  hasome
using namespace std;
const int N=100010;
int ge[N],dp[N],n,i,j;
int main()
{
      int a,b,c;
      cin>>n;
      cin>>a>>b>>c;
      for(i=1;i<=n;i++)
      {
            int tmp=((long long)a*i*i+b*i+c)%20000;
            ge[i]=tmp-10000;
      }
      dp[1]=ge[1];
      for(i=2;i<=n;i++)
      {
            dp[i]=max(dp[i-2]+ge[i],dp[i-1]+ge[i]);
      }
      cout<<max(dp[i-1],dp[i-2]);
      return 0;
}
```

1652 - 跳格子 2

地面上有一排长度为 n 的格子 1 、n，每个格子上都有一个数 x_i，开始时你在位置 0，每次你可以向前跳 1 ~ 2 格，然后取走格子上的数，直到跳到位置 n+1。

取走的数的和就是你的得分，现在你想知道你可能的最小得分是多少。

输入 ：一行四个整数 n,A,B,C(n ≤ 100000，0 ≤ A,B,C ≤ 10000)，其中 n 表示格子的数量。x[i] 由如下方式生成。

```
for (int i = 1; i <= n; i++){
int tmp = ((long long)A * i * i + B * i + C) % 20000;
x[i] = tmp - 10000;
}
```

输出 ：一行一个整数 ans 表示可能的最小得分。

输入 :3 1 1 1

输出 :-29977

```
#include<iostream>//4-1652    跳格子 2
#include<cstdio>
using namespace std;
int dp[100005],x[100005];
int main()
{
	int n,a,b,c;
	cin>>n>>a>>b>>c;
	for (int i = 1; i <= n; i++)
	{
		int tmp = ((long long)a * i * i + b * i + c) % 20000;
		x[i] = tmp - 10000;
	}
	dp[0] = 0;
	dp[1] = x[1];
	for(int i=2;i<=n+1;i++)
	{
		dp[i] = min(dp[i-1],dp[i-2]) + x[i];
	}
	cout<<dp[n+1];
	return 0;
}
```

1589 最大部分和

有 n 个整数（1 ≤ n ≤ 100），排成一排，例如：n=7,

-2 13 12 9 14 -10 2(7 个整数），其最大的部分和为 48(即 13+12+9+14）。

输入：第一行一个整数 n；n 个整数 -100 ≤ x ≤ 100；第二行的数之间有一个空格；

输出：一个整数（即最大的连续的部分和）。

输入

7

-2 13 12 9 14 -10 2

输出

48

```
#include <bits/stdc++.h>    //5-1589     最大部分和（连续部分和）
using namespace std;
int a[110],dp[110],n,i,j;
int ma;
int main()
{
	cin>>n;
	for(i=1;i<=n;i++)
	{
		cin>>a[i];
	}
	dp[1]=a[1];
	ma=dp[1];

	for(int i=2;i<=n;i++)
	{
		dp[i]=max(dp[i-1]+a[i],a[i]);
		ma=max(dp[i],ma);
	}
	cout<<ma;
	return 0;
}
```

1653. 取数

设有 N 个正整数（1 ≤ N ≤ 50），其中每一个均是大于等于 1、小于等于 300 的数。从这 N 个数中任取出若干个数（不能取相邻的数），要求得到一种取法，使得到的和为最大。例如：当 N=5 时，有 5 个数分别为：13,18,28,45,21；此时，有许多种取法，如：

13,28,21 和为 62；

13, 45 和为 58；

18, 45 和为 63；

……

输入

第一行是一个整数 N；

第二行有 N 个符合条件的整数。

输出

一个整数，即最大和。

输入

5

13 18 28 45 21

输出

63

解法一：dp 数组存储前 i 个数不选连续的数的最大和。对于每个数而言，求前 i 个数不选连续的最大和，分两种情况讨论：

情况一：留下第 i-1 个数，第 i 个数就不能留下，问题就转换成，求前 i-1 个数不选连续的最大和。

情况二：不留第 i-1 个数，第 i 个数留下，问题转换成，求 a[i] + 前 i-2 个数不选连续的最大和。

推导出状态转移方程如下：

dp[i] = max(dp[i-1],a[i] + dp[i-2]) 边界：dp[1] = a[1] dp[2] = max(a[1],a[2])

```cpp
#include <bits/stdc++.h>//6-1653-1 取数    javacn
using namespace std;
/*
  推导出动态转移方程如下：
  dp[i] = max(dp[i-1],a[i] + dp[i-2])
  边界：dp[1] = a[1]  dp[2] = max(a[1],a[2])
*/
int i,n,a[100],dp[100];
int main()
{
	cin>>n;
	for(i = 1;i <= n;i++)
	{
		cin>>a[i];
	}

	dp[1] = a[1];	//交代边界的值
	dp[2] = max(a[1],a[2]);

	for(i = 3;i <= n;i++)	//递推
	{
		dp[i] = max(dp[i-1],a[i]+dp[i-2]);
	}

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

解法二：每个元素有 2 种状态，选或者不选，将 2 种状态的最优解都记录一下。

f[i][0]：代表第 i 个元素不选的最优解。

f[i][1]：代表选择第 i 个元素的最优解。

分析： f[i][0]：第 i 个元素不选的状态，可以来自第 i-1 个元素不选，或者选，取最大。

f[i][1]：第 i 个元素选，只能来自第 i-1 个元素不选。

因此得出状态转移方程：

f[i][0] = max(f[i-1][0],f[i-1][1]);

f[i][1] = f[i-1][0] + a[i];

边界：//0 个数选 0 个收益是 0，0 个数选 1 个数的收益是不合法的

f[0][0] = 0,f[0][1] = -INT_MAX;

最终答案：ans = max(f[n][1],f[n][0]);

```
#include<bits/stdc++.h>//6-1653-2    javacn
using namespace std;
const int N = 110;
int f[N][2],a[N];
int n;
int main()
{
      cin>>n;
      for(int i = 1;i <= n;i++)
      {
        cin>>a[i];
      }

      f[0][0] = 0,f[0][1] = -INT_MAX;     //边界

      for(int i = 1;i <= n;i++)     //递推
      {
            f[i][0] = max(f[i-1][0],f[i-1][1]);//第 i 个数不取
            f[i][1] = f[i-1][0] + a[i];
      }
      cout<<max(f[n][0],f[n][1]);
      return 0;
}
```

1794. 最长不下降子序列（LIS）

问题描述

设有由 n 个不相同的整数组成的数列，记为: a_1、a_2、...、a_n 且 $a_i \neq a_j$ $(i \neq j)$。

例如 3，18，7，14，10，12，23，41，16，24。

若存在 $i_1 < i_2 < i_3 < \cdots < i_e$ 且有$a_{i1} < a_{i2} < \cdots < a_{ie}$，则称为长度为 e 的不下降序列。

如上例中 3，18，23，24 就是一个长度为 4 的不下降序列，同时也有 3，7，10，12，16，24 长度为 6 的不下降序列。

程序要求，当原数列给出之后，求出最长的不下降序列。

输入

第一行为 n，表示 n 个数（$10 \leq n \leq 10000$）；

第二行 n 个整数，数值之间用一个空格分隔（$1 \leq a_i \leq n$）；

输出

最长不下降子序列的长度。

样例

输入	复制
3 1 2 3	
输出	**复制**
3	

```
/*
  dp[i]=1
dp[i]=max(dp[j]+1,dp[i])  j 是 1..i-1 之间的数，a[j]<a[i]
*/
//dp: 存储以每个数结尾的最长不下降子序列的长度
```

```cpp
#include <bits/stdc++.h>//7-1794  最长不下降子序列（LIS）      javacn
using namespace std;
int a[10100],dp[10100],n,ma;
int main()
{
	int i,j;
	cin>>n;
	for(i = 1;i <= n;i++)
	{
		cin>>a[i];
	}


	for(i = 1;i <= n;i++)    //求以每个数结尾的最长不下降子序列的长度
	{
		dp[i] = 1;

		for(j = 1;j < i;j++)//将a[i]尝试续到每个数后面，看dp[i]能否增加
		{
			if(a[j] < a[i])
			{
				dp[i] = max(dp[j]+1,dp[i]);
			}
		}

		ma = max(dp[i],ma);
	}

	cout<<ma;
	return 0;
}
```

1277 - 合唱队形求解

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 位同学不交换位置就能排成合唱队形。

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 $1,2,\ldots,K$，他们的身高分别为 $T_1,T_2,\ldots,T_K$，则他们的身高满足 $T_1<T_2<\cdots<T_i>T_{i+1}>\cdots>T_K\ (1\le i\le K)$。

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

输入的第一行是一个整数 N $(2\le N\le 100)$，表示同学的总数。

第二行有 N 个整数，用空格分隔，第 i 个整数 T_i $(130\le T_i\le 230)$ 是第 i 位同学的身高（厘米）。

输出

输出包括一行，这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 复制

```
8
186 186 150 200 160 130 197 220
```

输出 复制

```
4
```

a 数组：存放元素

dpa 数组：存储状态，存储以每个点结尾的最长不下降子序列的长度（对于第 i 个人，左边加他自己，可以构成的递增的序列最多有几个数，最多留下几个人）

dpb 数组：存储状态，存储以每个点结尾从右至左的最长不下降子序列的长度（对于第 i 个人，包括自己的情况下，右侧最多留下几个人）

对于每个人而言，以他为中心，最多留下的人数 dpa[i]+dpb[i]-1。

因此，最少出去的人数 = n - 最多留下的人数。

```
/*
以第 i 个数为中心点能留下几个人：
计算出从左向右到第 i 个数的最长自增子序列的长度：第 i 个数左侧能留下的人数（含 i）
计算出从右向左到第 i 个数的最长自增子序列的长度：第 i 个数右侧能留下的人数（含 i）
最终能留下的人数 = dpa[i]+dpb[i]-1
留下的人数求最大，n- 最大，得到最少出去的人数
*/
```

```cpp
#include <bits/stdc++.h>//8-1277    合唱队形求解  javacn
using namespace std;
int n,a[110],dpa[110],dpb[110];
int i,j;
int main(){
	cin>>n;
	for(i = 1;i <= n;i++)
	{
		cin>>a[i];
		dpa[i] = 1;		//边界
		dpb[i] = 1;
	}
	for(i = 2;i <= n;i++)	//从左向后计算出最长递增子序列的长度
	{
		for(j = 1;j < i;j++)	//循环每个数前面的数
		{
			if(a[i] > a[j])	//a[i]比它前面的数大，才能递增
				dpa[i] = max(dpa[j]+1,dpa[i]);
		}
	}

	for(i = n - 1;i >= 1;i--)	//从右向左求出最长递增子序列的长度
	{
		for(j = n;j > i;j--)		//反过来循环第i个数后面的数
		{
			if(a[i] > a[j])
				dpb[i] = max(dpb[j]+1,dpb[i]);
		}
	}
	int ma = 0; //计算出最多能留下几个人
	for(i = 1;i <= n;i++)
		ma = max(dpa[i]+dpb[i]-1,ma);
	cout<<n - ma;	//最少要出去的人数
	return 0;
}
```

1795. 拦截导弹

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹的枚数和导弹依次飞来的高度（雷达给出的高度数据是不大于 30000 的正整数，每个数据之间至少有一个空格），计算这套系统最多能拦截多少导弹。

输入：第 1 行有 1 个整数 n，代表导弹的数量。（n<=1000）

第 2 行有 n 个整数，代表导弹的高度。（雷达给出的高度数据是不大于 30000 的正整数）

输出：输出这套系统最多能拦截多少导弹。

输入

8

389 207 155 300 299 170 158 65

输出

6

```
#include<bits/stdc++.h>//9-1795  拦截导弹    求最长递减子序列的长度 javacn
using namespace std;
int n,a[1010],dp[1010],ma;
int main() {
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
		dp[i] = 1;
		for(int j = 1;j < i;j++)
		{
			if(a[j] > a[i])
			{
				dp[i] = max(dp[j]+1,dp[i]);
			}
		}
		ma = max(dp[i],ma);
	}
	cout<<ma;
}
```

```cpp
#include<bits/stdc++.h>//10-1216      数塔问题    javacn
using namespace std;
int a[110][110];
// 思路：将数塔存入二维数组，从倒数第 2 层开始，递推计算出走到每个点最多能够累计的最大数字和，直到第 1 层，就能求出所经过结点的数字和的最大值
int main()
{
	int n;
	cin>>n;
	for(int i=1;i<=n;i++)
	{
		for(int j=1;j<=i;j++)
		{
			cin>>a[i][j];
		}
	}
    // 从倒数第 2 层开始逆推，每个点累加下方的值和下方右边的值中的较大值
	for(int i=n-1;i>=1;i--)
	{
		for(int j=1;j<=i;j++)
		{
			if(a[i+1][j]>a[i+1][j+1])
			{
				a[i][j] = a[i][j] + a[i+1][j];
			}
			else
			{
				a[i][j] = a[i][j] + a[i+1][j+1];
			}
		}
	}
	cout<<a[1][1];
	return 0;
}
```

1282. 简单背包问题

有一个背包能装的重量 maxw（正整数，0 ≤ maxw ≤ 20000），同时有 n 件物品（1 ≤ n ≤ 100），每件物品有一个重量 w_i（正整数）和一个价值 p_i（正整数）。要求从这 n 件物品中任取若干件装入背包内，使背包的物品价值最大。

输入

第 1 行：背包最大载重 maxw，物品总数 n ；

第 2 行到第 n+1 行：每个物品的重量和价值；

输出

一个数字即背包内物品最大价值；

输入

10 3

4 5

3 4

6 9

输出

14

dp[i][j]：代表有 i 个物品，背包容量为 j 时，能够承载的最大价值。

w[i]：代表第 i 个物品的重量。

v[i]：代表第 i 个物品的价值。

二维求解状态转移方程：dp[i][j]=max(dp[i-1][j],v[i]+dp[i-1][j-w[i]])

```
/*
dp[i][j]：代表有 i 个物品，背包容量为 j 时，能够存储的最大价值
对于第 i 个物品：
情况一：背包容量 j< 物品重量 w[j]，放不下，能够存放的价值 dp[i-1][j]
情况二：背包容量 j>= 物品重量 w[j]，放得下
能够得到的最大价值 = max(dp[i-1][j],v[i]+dp[i-1][j-w[i]])
*/
```

```cpp
#include <bits/stdc++.h>//11-1282-1  简单背包问题        javacn
using namespace std;
int dp[110][20010];
int n,w[110],v[110],maxw,i,j;
int main()
{
	cin>>maxw>>n;

	for(i = 1;i <= n;i++)    // 读入每个物品的重量和价值
	{
		cin>>w[i]>>v[i];
	}
	for(i = 1;i <= n;i++)    // 推导       // 循环 n 个物品
	{
		for(j = 1;j <= maxw;j++)              // 循环背包容量从 1~maxw
		{
			if(j < w[i])       // 如果放不下：背包容量 < 物品重量
			{
				dp[i][j] = dp[i-1][j];
			}
			else
			{
				// 放得下，就看放进来价值高，还是不放价值高
				dp[i][j] = max(dp[i-1][j],v[i]+dp[i-1][j-w[i]]);
			}
		}
	}

	cout<<dp[n][maxw];       // 输出 n 个物品，背包容量为 maxw 的情况下最大价值
	return 0;
}
```

一维求解状态转移方程：dp[j]=max(dp[j], dp[j-w[i]]+v[i])

```
#include <bits/stdc++.h>//11-1282-2  简单背包问题        javacn
using namespace std;
int maxw;// 背包承重
int w,v;// 物品的重量和价值
int dp[20010]; // 讨论：有 i 个物品背包容量为 j 时能够存放的最大价值
int n;

int main()
{
	cin>>maxw>>n;

	for(int i = 1;i <= n;i++)	// 读入 n 个物品的重量和价值，并计算
	{
		cin>>w>>v;		// 读入重量和价值

		for(int j = maxw;j >= w;j--)  // 逆序从背包容量循环到当前物品的重量
		{
			dp[j] = max(dp[j],v+dp[j-w]);
		}
	}

	cout<<dp[maxw];
	return 0;
}
```

2、DP 进阶

上体育课的时候，小蛮的老师经常带着同学们一起做游戏。这次，老师带着同学们一起做传球游戏。

游戏规则是这样的：n个同学站成一个圆圈，其中的一个同学手里拿着一个球，当老师吹哨子时开始传球，每个同学可以把球传给自己左右的两个同学中的一个（左右任意），当老师再次吹哨子时，传球停止，此时，拿着球没有传出去的那个同学就是败者，要给大家表演一个节目。

聪明的小蛮提出一个有趣的问题：有多少种不同的传球方法可以使得从小蛮手里开始传的球，传了m次 以后，又回到小蛮手里。两种传球方法被视作不同的方法，当且仅当这两种方法中，接到球的同学按接球顺序组成的序列是不同的。比如有三个同学 1 号、 2 号、 3 号，并假设小蛮为 1 号，球传了 3 次回到小蛮手里的方式有 1->2->3->1 和 1->3->2->1 ，共 2 种。

输入

一行，有两个用空格隔开的整数$n, m(3 \leq n \leq 30, 1 \leq m \leq 30)$。

输出

1个整数，表示符合题意的方法数。

样例

输入	复制
3 3	
输出	复制
2	

我们可以发现，任何一个位置都只能从左边和右边传过来，这样我们就可以列出我们的方程：
dp[i][j]=dp[i-1][j-1]+dp[i-1][j+1]。
dp[i][j]：代表第 i 次传球，第 j 个人获得球的机会数
j!=1&&j!=n：dp[i][j] = dp[i-1][j-1] + dp[i-1][j+1]

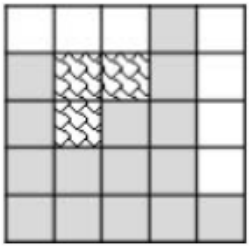
j=1 dp[i][j] = dp[i-1][j+1] + dp[i-1][n]，这是计算第 i 次传球，第 1 个人获得球的机会数
j=n dp[i][j] = dp[i-1][j-1] + dp[i-1][1]，这是计算第 i 次传球，第 n 个人获得球的机会数

```cpp
#include <bits/stdc++.h>//1-1801    传球游戏    javacn
using namespace std;
/*
dp[i][j]：代表第 i 次传球，第 j 个人获得球的机会数
j!=1&&j!=n：dp[i][j] = dp[i-1][j-1] + dp[i-1][j+1]
j=1 dp[i][j] = dp[i-1][j+1] + dp[i-1][n]
j=n dp[i][j] = dp[i-1][j-1] + dp[i-1][1]

边界：dp[0][1] = 1
*/
int n,m;

int dp[40][40];// 代表第 i 次传球，第 j 个人获得球的机会数
int main()
{
	cin>>n>>m;

	dp[0][1] = 1;	// 边界：第 0 次传球，第 1 个人获得球的机会数

	for(int i = 1;i <= m;i++)	// 传 m 次球
	{
		for(int j = 1;j <= n;j++)
		{
			if(j==1) dp[i][j] = dp[i-1][j+1] + dp[i-1][n];
			else if(j==n) dp[i][j] = dp[i-1][j-1] + dp[i-1][1];
				else dp[i][j] = dp[i-1][j-1] + dp[i-1][j+1];
		}
	}

	// 经过 m 次传球，球回到第 1 个人手中的机会数
	cout<<dp[m][1];
	return 0;
}
```

乌龟家的屋顶是凹凸不平的，所以每次雨后都会积水。为了知道屋顶是否会在暴雨后塌掉，他把屋顶的形 状给了你，希望你帮他计算暴雨后屋顶的积水总量。

乌龟的屋顶由顺次排在同一水平线上的 n 个宽度为 1、高度为整数 (分别给出) 的瓦片组成。例如给定 $n = 5$ ，瓦片的高度分别为 $4, 2, 3, 5, 1$ ，屋顶可以画在下图所示的网格中，灰色格子为瓦片。

暴雨过后，如果一个方格向左右两侧延伸都能到达瓦片占据的方格，它就会积水。所以图中波浪线格子在暴雨后会积水，屋顶的积水方格总数为 3 。



输入

两个整数 n, R_1，表示屋顶的宽度和生成数列的首项。从左向右数第 i ($1 \leq i \leq n$)个瓦片的高度 $a_i = R_i \mod 10$ 。

试题中使用的生成数列 R 定义如下：整数 $0 \leq R_1 < 201701$ 在输入中给出。对于 $i > 1$，$R_i = (R_{i-1} \times 6807 + 2831) \mod 201701$。

输出

一个整数，表示暴雨后屋顶积水方格的总数。

样例

输入	复制
10 1	
输出	复制
23	

动态规划！

对于每个点，计算出其左侧的最高值和右侧的最高值，那么积水量 = min(最侧最高值，右侧最高值) - 该点瓦片高度，以第 2 列为例，左侧最高值是 4，右侧最高值是 5，瓦片高度为 2，那么积水量 = min(4,5)-2=2。

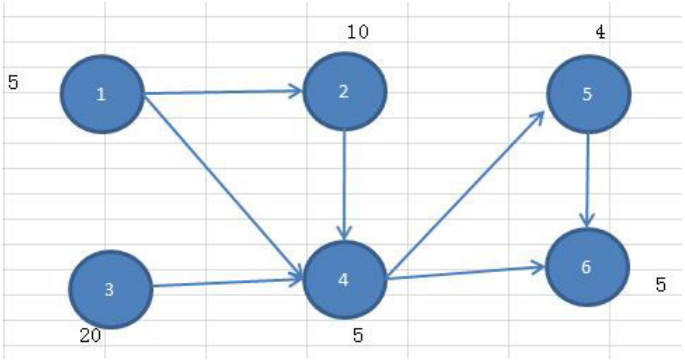
```cpp
#include <bits/stdc++.h>//2-1550-1  房屋积水  javacn
using namespace std;
int n,t;
int a[110];
int l[110];// 求从左向右的前缀最大值
int r[110];// 求从右向左的前缀最大值
int main()
{
	cin>>n>>t;
	for(int i = 1;i <= n;i++)
	{
		a[i] = t % 10;
		t =  (t * 6807 + 2831) % 201701;
	}

	l[1] = a[1];		// 求从左向右的前缀最大值（前 i 个数的最大数）
	for(int i = 2;i <= n;i++)
	{
		l[i] = max(l[i-1],a[i]);
	}

	r[n] = a[n];
	for(int i = n - 1;i >= 1;i--)
	{
		r[i] = max(r[i+1],a[i]);
	}

	int s = 0;  // 求每个位置的积水量
	for(int i = 2;i <= n - 1;i++)
	{
		s = s + min(l[i],r[i]) - a[i];
	}
	cout<<s;
	return 0;
}
```

```cpp
#include <bits/stdc++.h>//2-1550-2    房屋积水 remedy1314
using namespace std;
int n, r;
int a[105], lmax[105], rmax[105],t;
int main()
{
	cin>>n>>r;
	a[1] = r % 10;
	for (int i = 2; i <= n; i++)
	{
		r = (r * 6807 % 201701 + 2831) % 201701;
		a[i] = r % 10;
	}
	for (int i = 1; i <= n; i++)
	{
		lmax[i] = max(lmax[i - 1], a[i]);
	}

	for (int i = n; i >= 1; i--)
	{
		rmax[i] = max(rmax[i + 1], a[i]);
	}

	int ans = 0;
	for (int i = 2; i < n; i++)
	{
		t = min(lmax[i - 1], rmax[i + 1]);

		if(a[i] >= t)    continue;

		ans += t - a[i];
	}
	cout<<ans<<endl;
	return 0;
}
```

1276 - 挖地雷的算法

在一个地图上有 n 个地窖（n<=200），每个地窖中埋有一定数量的地雷。同时，给出地窖之间的连接路径，并规定路径都是单向的，且保证都是小序号地窖指向大序号地窖，也不存在可以从一个地窖出发经过若干地窖后又回到原来地窖的路径。某人可以从任一处开始挖地雷，然后沿着指出的连接往下挖（仅能选择一条路径），当无连接时挖地雷工作结束。设计一个挖地雷的方案，使他能挖到最多的地雷。

如下图所示：圆圈内的 1 2 3 4 5 6，代表 6 个地窖的编号，地窖编号旁边的数字代表这个地窖地雷的数量！



输入

第一行：地窖的个数；

第二行为依次每个地窖地雷的个数；

下面若干行：

xi yi　// 表示从 xi 可到 yi，xi<yi。

最后一行为 "0 0" 表示结束。

输出

第一行输出挖地雷的地窖编号的顺序：k1-k2-…-kv

第二行输出一个整数，代表最多能挖到的地雷的数量

输入：	输出：
6	3-4-5-6
5 10 20 5 4 5	34
1 2	
1 4	
2 4	
3 4	
4 5	
4 6	
5 6	
0 0	

dp[i]=a[i]+max(dp[j])　j=i+1 ... n　ij 有通路

dp[n]=a[n]　边界

dp: 存储从每个地窖开始最多能挖到的地雷数量

a：存储每个地窖的地雷数量

r：存储第 i 号地窖去了哪一号地窖

f：存储地窖之间的通路关系

```cpp
#include <bits/stdc++.h>//3-1276   挖地雷的算法     javacn
using namespace std;
int n,i,j,a[210],dp[210],r[210];
bool f[210][210];
int main()
{
	cin>>n;
	for(i = 1;i <= n;i++)		cin>>a[i];  // 读入地雷数量
	int x,y;    // 读入通路关系
	while(true)
	{
		cin>>x>>y;
		if(x == 0 && y == 0) break;
		f[x][y] = true;//x y 之间有通路
	}
	// 边界：从最后一个地窖开始最多能挖到的地雷数就是最后一个地窖的地雷数
	dp[n] = a[n];
	//index：存放 i 号地窖应该去哪个地窖
//count：编号 >i 的地窖中，有通路的情况下，哪个地窖开始挖到的地雷最多，存地雷数
	int index,count;
	// 逆推
	for(i = n - 1;i >= 1;i--)// 从第 i 个地窖开始应该去哪个地窖取决于：
	{    // 从 i+1~n 之间的地窖中有通路的地窖，从哪个地窖开始挖，得到的地雷最多
		count = 0;
		for(j = i + 1;j <= n;j++)
		{
			if(f[i][j] == true && dp[j] > count)
			{                  // 如果 i j 有通路，且从 j 开始挖到的地雷更多
				index = j;
				count = dp[j];
			}
		}
		dp[i] = a[i] + count; // 记录从 i 开始最多能挖到的地雷数量
		r[i] = index;           // 存储第 i 号地窖，去了哪个编号的地窖
	}
```

```
	//求从哪个地窖出发：求 dp 数组最大值的下标
	index = 1;
	for(i = 2;i <= n;i++)
	{
		if(dp[i] > dp[index]) index = i;
	}
	int ans = dp[index];//最多地雷数

	//打印路径
	while(index != 0)
	{
		cout<<index;
		index = r[index];//求出 index 编号的地窖去了哪里
		if(index != 0) cout<<"-";
	}

	cout<<endl<<ans;
	return 0;
}
```

问题描述

总公司拥有高效设备 M 台，准备分给下属的 N 个分公司。各分公司若获得这些设备，可以为国家提供一定的盈利。问：如何分配这 M 台设备才能使国家得到的盈利最大？求出最大盈利值。

其中 $M \leq 15$，$N \leq 10$。分配原则：每个公司有权获得任意数目的设备，但总台数不超过设备数 M 。

输入

第一行有两个数，第一个数是分公司数 N ，第二个数是设备台数 M 。

接下来是一个 $N \times M$ 的矩阵，表明了第 i 个公司分配 j 台机器的盈利。

输出

第一行为一个整数，代表最大盈利的值。

接下来 N 行，每行两个数，用空格隔开，第一个数代表了第 i 个公司的序号，第二个数代表该公司分得机器的数量。

样例

输入 复制

```
3 3
30 40 50
20 30 50
20 25 30
```

输出 复制

```
70
1 1
2 1
3 1
```

按公司顺序分配机器，第一个阶段把 M 台设备分配给第一个公司，记录下获得的各个盈利，然后把 M 台设备分给前两个公司，和第一个阶段比较记录下来更优各个盈利值，一直到第 N 个阶段把 M 台机器全部分给了 N 个公司。两个阶段之间关系如下讨论。

f[i-1][k]： 表示前 i-1 个公司分配 k 台机器的最大盈利，用 c[i][j] 表示第 i 个公司分配 j 台机器盈利；

f[i-1][k]+c[i][j-k] // 前 i-1 公司分配 k 台机器最大盈利 + 第 i 个公司分配 j-k 台机器的盈利。

状态转移方程：f[i][j]=max(f[i-1][k]+c[i][j-k])，边界条件：f[0][0]=0。

通过递归输出每个公司分配机器的方案数的解法：

```cpp
#include <bits/stdc++.h>//4-1378-1    机器分配    javacn
#define N 20
using namespace std;
int f[N][N],  c[N][N], n, m, ans;
void print(int i, int j){           //打印分配情况
	if(i==0) return;
	for(int k = 0; k <= j; k++)
	{
		if(ans == f[i-1][k] + c[i][j-k])
		{
			ans = f[i-1][k];
			print(i-1, k);
			printf("%d %d\n", i, j-k);
			break;
		}
	}
}
int main() {
	scanf("%d%d", &n, &m);
	for(int i = 1; i <= n; i++)
		for(int j = 1; j <= m; j++)
			scanf("%d", &c[i][j]);
	for(int i = 1; i <= n; i++)
		for(int j = 1; j <= m; j++)
		{
			int maxx = 0;
			for(int k = 0; k <= j; k++) //递归求各公司分配机器数量
				maxx = max(f[i-1][k] + c[i][j-k], maxx);
			f[i][j] = maxx;
		}
	printf("%d\n", f[n][m]);
	ans = f[n][m];
	print(n, m);
	return 0;
}
```

直接递推求出每个公司分配机器方案数的解法：

```
#include <bits/stdc++.h>//4-1378-2    机器分配    javacn
using namespace std;

/*
f[i][j]：i 个公司在共有 j 台设备的最大利润
f[i][j] = max(f[i-1][k]+c[i][j-k]);
*/
const int N = 20;
int f[N][N];
int n, m, ans;
int c[N][N],r[N];//r：每个公司分到的机器数量

int main()
{
	scanf("%d%d", &n, &m);
	for(int i = 1; i <= n; i++)
		for(int j = 1; j <= m; j++)
			scanf("%d", &c[i][j]);

	for(int i = 1; i <= n; i++)	// 循环每个公司
	{
		for(int j = 1; j <= m; j++)		// 循环机器的额数量
		{
			int maxx = 0;
			for(int k = 0; k <= j; k++)  // 递归求各公司分配机器数量
			{
				maxx = max(f[i-1][k] + c[i][j-k], maxx);
			}
			f[i][j] = maxx;
		}
	}
```

```
    printf("%d\n", f[n][m]);
    ans = f[n][m];
    for(int i = n; i >= 1; i--) //逆序推导出每个公司实际分配了几台机器
    {
        for(int k = 0; k <= m; k++)//枚举第 i 个公司可能分到的机器总数
        {
            //第 i 个公司分配 j-k 台机器可以获得最大总利润
            if(ans == f[i-1][k] + c[i][m-k])
            {
                r[i] = m - k;
                ans = f[i-1][k];//接下来计算 i-1 个公司有 m-k 台机器的
                m = k;
                break;
            }
        }
    }

    for(int i = 1;i <= n;i++)
    {
        printf("%d %d\n",i,r[i]);
    }
    return 0;
}
```

1781. 乌龟棋

问题描述

乌龟棋的棋盘是一行 N 个格子，每个格子上一个分数（非负整数）。棋盘第 1 格是唯一的起点，第 N 格是终点，游戏要求玩家控制一个乌龟棋子从起点出发走到终点。

乌龟棋中 M 张爬行卡片，分成 4 种不同的类型（M 张卡片中不一定包含所有 4 种类型的卡片，见样例），每种类型的卡片上分别标有 $1, 2, 3, 4$ 四个数字之一，表示使用这种卡片后，乌龟棋子将向前爬行相应的格子数。游戏中，玩家每次需要从所有的爬行卡片中选择一张之前没有使用过的爬行卡片，控制乌龟棋子前进相应的格子数，每张卡片只能使用一次。

游戏中，乌龟棋子自动获得起点格子的分数，并且在后续的爬行中每到达一个格子，就得到该格子相应的分数。玩家最终游戏得分就是乌龟棋子从起点到终点过程中到过的所有格子的分数总和。

很明显，用不同的爬行卡片使用顺序会使得最终游戏的得分不同，小明想要找到一种卡片使用顺序使得最终游戏得分最多。

现在，告诉你棋盘上每个格子的分数和所有的爬行卡片，你能告诉小明，他最多能得到多少分吗?

输入

每行中两个数之间用一个空格隔开。

第 1 行 2 个正整数 N, M ，分别表示棋盘格子数和爬行卡片数。

第 2 行 N 个非负整数，$a_1, a_2, \ldots, a_N$，其中 a_i 表示棋盘第 i 个格子上的分数。

第 3 行 M 个整数，$b_1, b_2, \ldots, b_M$，表示 M 张爬行卡片上的数字。

输入数据保证到达终点时刚好用光 M 张爬行卡片。

输出

1个整数，表示小明最多能得到的分数。

样例

输入 复制

```
9 5
6 10 14 2 8 8 18 5 17
1 3 1 2 1
```

输出 复制

```
73
```

说明

样例 1 解释

小明使用爬行卡片顺序为 $1,1,3,1,2$ ，得到的分数为 $6+10+14+8+18+17=73$ 。注意，由于起点是 1 ，所以自动获得第 1 格的分数 6。

数据范围

对于 30% 的数据有 $1 \leq N \leq 30, 1 \leq M \leq 12$ 。

对于 50% 的数据有 $1 \leq N \leq 120, 1 \leq M \leq 50$，且 4 种爬行卡片，每种卡片的张数不会超过 20 。

对于 100% 的数据有 $1 \leq N \leq 350, 1 \leq M \leq 120$，且 4 种爬行卡片，每种卡片的张数不会超过 40 ； $0 \leq a_i \leq 100, 1 \leq i \leq N, 1 \leq b_i \leq 4, 1 \leq i \leq M$ 。

dp[a][b][c][d]：表示你出了a张爬行牌1，b张爬行牌2，c张爬行牌3，d张爬行牌4时的得分。

起始状态 dp[0][0][0][0]=num[1]，即不出任何爬行卡；之后对于每一张卡片，我都可以选择放与不放， 设当前放的卡 1 数量为 a，卡 2 数量为 b，卡 3 数量为 c，卡 4 数量为 d（以下出现 a~d 均为这个意思），则对于卡一：

比较卡一的放与不放，只需决策卡一的放与不放，即取 dp[a][b][c][d],dp[a-1][b][c][d]+num[r] 的最大值。又由于 a 有一定数量，所以我们可以得出关于 a 的转移方程：

dp[a][b][c][d]=max(dp[a][b][c][d],F[a-1][b][c][d]+num[r])

对于 bcd 以此类推：

```cpp
#include<bits/stdc++.h> //5-1781    乌龟棋    javacn
using namespace std;
int dp[41][41][41][41],num[351],g[5],n,m,x;
int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++) cin>>num[i];
    dp[0][0][0][0]=num[1];
    for(int i=1;i<=m;i++)
    {
        cin>>x;
        g[x]++;
    }
    for(int a=0;a<=g[1];a++)
        for(int b=0;b<=g[2];b++)
            for(int c=0;c<=g[3];c++)
                for(int d=0;d<=g[4];d++)
                {
                    int r=1+a+b*2+c*3+d*4;
         if(a!=0)    dp[a][b][c][d]=max(dp[a][b][c][d],dp[a-1][b][c][d]+num[r]);
         if(b!=0)    dp[a][b][c][d]=max(dp[a][b][c][d],dp[a][b-1][c][d]+num[r]);
         if(c!=0)    dp[a][b][c][d]=max(dp[a][b][c][d],dp[a][b][c-1][d]+num[r]);
         if(d!=0)    dp[a][b][c][d]=max(dp[a][b][c][d],dp[a][b][c][d-1]+num[r]);
                }
    cout<<dp[g[1]][g[2]][g[3]][g[4]];
    return 0;
}
```

1796 - 奶牛沙盘队

Farmer Han 开始玩飞盘之后，YDS 也打算让奶牛们享受飞盘的乐趣。

他要组建一只奶牛飞盘队，他的 N (1 ≤ N ≤ 2000) 只奶牛，每只奶牛有一个飞盘水准指数 R_i(1 ≤ R_i ≤ 100000)。

YDS 要选出 1 只或多于 1 只奶牛来参加他的飞盘队。由于 YDS 的幸运数字是 F (1 ≤ F ≤ 1000)，他希望所有奶牛的飞盘水准指数之和是幸运数字的倍数。

帮 YDS 算算一共有多少种组队方式，组队方式数模 10^8 取余的结果。

输入

第 1 行输入 N 和 F ，之后 N 行输入 R_i。

输出

组队方式数模 10^8 取余的结果。

输入

4 5

1

2

8

2

输出

3

```cpp
#include <bits/stdc++.h>//6-1796    奶牛沙盘队    javacn
using namespace std;
const int MOD = 100000000; // 余数
const int N = 2010;
long long a[N], dp[N][N]; //cow[i] 指第 i 头奶牛的能力,
int n, f;
int main()
{
	scanf("%d%d", &n, &f);
	for (int i = 1; i <= n; i++)  // 读入每个数
	{
		scanf("%d", &a[i]);
		a[i] %= f;// 提前取余
	}

    // 边界条件
	for (int i = 1; i <= n; i++)
	{
		dp[i][a[i]] = 1;
	}

	for (int i = 1; i <= n; i++)     // 枚举每个数    //01 背包求方案数
	{

		for (int j = 0; j <= f - 1; j++)          // 枚举余数
		{
		dp[i][j]=((dp[i][j]+dp[i-1][j])%MOD + dp[i-1][(j-a[i]+f)%f])%MOD;
		}
	}

	printf("%d", dp[n][0]);

	return 0;
}
```

1800 - 小朋友的数字

有 n 个小朋友排成一列。每个小朋友手上都有一个数字，这个数字可正可负。规定每个小朋友的特征值等于排在他前面（包括他本人）的小朋友中连续若干个（最少有一个）小朋友手上的数字之和的最大值。

作为这些小朋友的老师，你需要给每个小朋友一个分数，分数是这样规定的：第一个小朋友的分数是他的特征值，其它小朋友的分数为排在他前面的所有小朋友中（不包括他本人），小朋友分数加上其特征值的最大值。

请计算所有小朋友分数的最大值， 输出时保持最大值的符号，将其绝对值对 p 取模后输出。

【输入输出样例 1】

number.in	number.out
5 997	21
1 2 3 4 5	

【输入输出样例说明】小朋友的特征值分别为 1、 3、 6、 10、 15，分数分别为 1、 2、 5、 11、 21，最大值 21 对 997 的模是 21。

【输入输出样例 2】

number.in	number.out
5 7	-1
-1 -1 -1 -1 -1	

【输入输出样例说明】

小朋友的特征值分别为 -1、 -1、 -1、 -1、 -1，分数分别为 -1、 -2、 -2、 -2、 -2，最大值 -1 对 7 的模为 -1，输出 -1。

【数据范围】

对于 50% 的数据， 1 ≤ n ≤ 1,000， 1 ≤ p ≤ 1,000 所有数字的绝对值不超过 1000；

对于 100% 的数据， 1 ≤ n ≤ 1,000,000， 1 ≤ p ≤ 109， 其他数字的绝对值均不超过 109。

输入：第一行包含两个正整数 n、 p，之间用一个空格隔开。

第二行包含 n 个数，每两个整数之间用一个空格隔开，表示每个小朋友手上的数字。

输出：输出只有一行， 包含一个整数， 表示最大分数对 p 取模的结果。

输入

5 997

1 2 3 4 5

输出

21

```cpp
#include <bits/stdc++.h>//7-1800    小朋友的数字    javacn
using namespace std;
typedef long long LL;
const int N = 1e6 + 10;
LL n,p,a[N];
//f：存放特征值
LL s[N],f[N],score[N],r,ma = LONG_LONG_MIN;//r 存放每个人的分数
int main()
{
	cin>>n>>p;
	// 读入 n 个值
	for(int i = 1; i <= n; i++)
	{
		cin>>a[i];
		// 以每个点结束的最大和
		s[i] = max(s[i-1]+a[i],a[i]);
		ma = max(ma,s[i]);
		f[i] = ma % p;// 特征值
	}

	// 第一个人的分数是特征值
	score[1] = f[1];
	LL ans = score[1];
	ma = LONG_LONG_MIN;
	// 计算每个人的得分
	for(int i = 2; i <= n; i++)
	{
		ma = max(ma,f[i-1]+score[i-1]);
		score[i] = ma;
		ans = max(ans,ma) % p;
	}

	cout<<ans;
	return 0;
}
```

先求出以每个数结尾的最大和，再取和的前缀最大值作为每个人的特征值（因为不一定以第 i 个数结尾的最大和，是前 i 个数取连续数的最大和）。

接下来根据题意算分数，再计算分数的最大值。

3、背包基础

1282. 简单背包问题

有一个背包能装的重量 maxw（正整数，0 ⩽ maxw ⩽ 20000），同时有 n 件物品（1 ⩽ n ⩽ 100），每件物品有一个重量 w_i（正整数）和一个价值 p_i（正整数）。要求从这 n 件物品中任取若干件装入背包内，使背包的物品价值最大。

输入

第 1 行：背包最大载重 maxw，物品总数 n ；

第 2 行到第 n+1 行：每个物品的重量和价值；

输出

一个数字即背包内物品最大价值；

输入

10 3

4 5

3 4

6 9

输出

14

dp[i][j]：代表有 i 个物品，背包容量为 j 时，能够承载的最大价值。

w[i]：代表第 i 个物品的重量。

v[i]：代表第 i 个物品的价值。

二维求解状态转移方程：dp[i][j]=max(dp[i-1][j],v[i]+dp[i-1][j-w[i]])

```
/*
dp[i][j]：代表有 i 个物品，背包容量为 j 时，能够存储的最大价值
对于第 i 个物品：
情况一：背包容量 j< 物品重量 w[j]，放不下，能够存放的价值 dp[i-1][j]
情况二：背包容量 j>= 物品重量 w[j]，放得下
能够得到的最大价值 = max(dp[i-1][j],v[i]+dp[i-1][j-w[i]])
*/
```

```cpp
#include <bits/stdc++.h>//1-1282-1  简单背包问题        javacn
using namespace std;
int dp[110][20010];
int n,w[110],v[110],maxw,i,j;
int main()
{
	cin>>maxw>>n;

	for(i = 1;i <= n;i++)    //读入每个物品的重量和价值
	{
		cin>>w[i]>>v[i];
	}
	for(i = 1;i <= n;i++)    //推导       //循环 n 个物品
	{
		for(j = 1;j <= maxw;j++)              //循环背包容量从 1~maxw
		{
			if(j < w[i])       //如果放不下：背包容量 < 物品重量
			{
				dp[i][j] = dp[i-1][j];
			}
			else
			{
				//放得下，就看放进来价值高，还是不放价值高
				dp[i][j] = max(dp[i-1][j],v[i]+dp[i-1][j-w[i]]);
			}
		}
	}

	cout<<dp[n][maxw];       //输出 n 个物品，背包容量为 maxw 的情况下最大价值
	return 0;
}
```

一维求解状态转移方程：dp[j]=max(dp[j], dp[j-w[i]]+v[i])

```
#include <bits/stdc++.h>//1-1282-2  简单背包问题        javacn
using namespace std;
int maxw;// 背包承重
int w,v;// 物品的重量和价值
int dp[20010]; // 讨论：有 i 个物品背包容量为 j 时能够存放的最大价值
int n;

int main()
{
	cin>>maxw>>n;

	for(int i = 1;i <= n;i++)	// 读入 n 个物品的重量和价值，并计算
	{
		cin>>w>>v;		// 读入重量和价值

		for(int j = maxw;j >= w;j--)  // 逆序从背包容量循环到当前物品的重量
		{
			dp[j] = max(dp[j],v+dp[j-w]);
		}
	}

	cout<<dp[maxw];
	return 0;
}
```

2-1780. 采灵芝

仙岛上种了无数的不同种类的灵芝，小芳跟着爷爷来到仙岛采摘灵芝。由于他们带的食物和饮用水有限，必须在时间 t 内完成采摘。假设岛上有 m 种不同种类的灵芝，每种灵芝都有无限多个，已知每种灵芝采摘需要的时间，以及这种灵芝的价值；请你编程帮助小芳计算，在有限的时间 t 内，能够采摘到的灵芝的最大价值是多少？

输入：输入第一行有两个整数 T（1 ≤ T ≤ 100000）和 M（1 ≤ M ≤ 2000），用一个空格隔开，T 代表总共能够用来采灵芝的时间，M 代表岛上灵芝的种类数。

接下来的 M 行每行包括两个在 1 到 10000 之间（包括 1 和 10000）的整数，分别表示采摘某种灵芝的时间和这种灵芝的价值。

输出：一行，这一行只包含一个整数，表示在规定的时间内，可以采到的灵芝的最大总价值。

完全背包状态转移方程：

二维写法：f[i][j] = max(f[i-1][j], f[i][j-w[i]]+v[i])

一维写法：f[j]=max(f[j],f[j-w[i]]+v[i])

一维状态转义方程和 01 背包一致，要注意，完全背包要从前往后推导。

```
#include<bits/stdc++.h>//2-1780    采灵芝
using namespace std;
int t,m;
int f[100010];
int ti,vi;// 每个物品的采摘时间和价值
int main()
{
	cin>>t>>m;
	for(int i = 1;i <= m;i++)
	{
		cin>>ti>>vi;
		// 从当前物品的重量（采摘时间）~ 背包容量（最大时间）循环
		for(int j = ti;j <= t;j++)
		{
			f[j] = max(f[j],f[j-ti]+vi);
		}
	}
	cout<<f[t];// 背包能够存储的最大价值
	return 0;
}
```

输入

70 3

71 100

69 1

1 2

输出

140

1888. 多重背包（1）

有 N 种物品和一个容量是 V 的背包。第 i 种物品最多有 s_i 件，每件体积是 v_i，价值是 w_i。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。

输出最大价值。

输入

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 v_i, w_i, s_i，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

数据范围：$0<N,V \leqslant 100, 0<v_i, w_i, s_i \leqslant 100$。

输出

输出一个整数，代表最大价值。

输入

4 10

3 2 2

4 3 2

2 2 1

5 3 4

输出

8

解法一：将多重背包的 si 个物品分别装入 w 和 v 数组，直接转换为 01 背包！

```
/*
01 背包：每种物品有 1 件
完全背包：每种物品有无限件数
多重背包：每种物品有 Si 件
解题思路：将多重背包转换为 01 背包
将 Si 件物品都存起来，转换为有 Si 个物品，每个物品有 1 件
*/
```

```cpp
#include <bits/stdc++.h>//3-1888-1    多重背包（1）   javacn
using namespace std;
int n,c;          //c 背包容量
int v[10010],w[10010];
int dp[110];
int vi,wi,si,k;   //k 代表数组下标
int main()
{
	cin>>n>>c;
	for(int i = 1;i <= n;i++)
	{
		cin>>vi>>wi>>si;

		for(int j = 1;j <= si;j++)          // 第 i 个物品有 si 件，都存入数组
		{
			k++;
			v[k] = vi;
			w[k] = wi;
		}
	}


	for(int i = 1;i <= k;i++)      //01 背包
	{
		for(int j = c;j >= v[i];j--)  // 逆序从背包容量循环到当前物品体积
		{
			dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
		}
	}

	cout<<dp[c];
	return 0;
}
```

解法二：在做 01 背包时，体现一下有 Si 件物品这个条件。

```
#include <bits/stdc++.h>//3-1888-2     多重背包（1）    javacn
using namespace std;
01 背包：每种物品有 1 件
完全背包：每种物品有无限件数
多重背包：每种物品有 Si 件
解题思路：将多重背包转换为 01 背包
将 Si 件物品都存起来，转换为有 Si 个物品，每个物品有 1 件
int n,c;//c 背包容量
int v[110],w[110],s[110];
int dp[110];
int main()
{
	cin>>n>>c;
	for(int i = 1; i <= n; i++)
	{
		cin>>v[i]>>w[i]>>s[i];
	}

	for(int i = 1; i <= n; i++)	//01 背包	// 有 n 个物品
	{
		for(int k = 1;k <= s[i];k++)
		{
			for(int j = c; j >= v[i]; j--) // 从背包容量循环到当前物品体积
			{
				dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
			}
		}
	}
    cout<<dp[c];
    return 0;
}
```

1889. 多重背包（2）

有 N 种物品和一个容量是 V 的背包。

第 i 种物品最多有 s_i 件，每件体积是 v_i ，价值是 w_i 。

求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。

输出最大价值。

输入

第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 v_i, w_i, s_i ，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

输出

输出一个整数，表示最大价值。

数据范围：$0<N \leqslant 1000$，$0<V \leqslant 2000$，$0<v_i, w_i, s_i \leqslant 2000$。

输入

4 5

1 2 3

2 4 1

3 4 3

4 5 2

输出

10

关于 DP 要理解的关键点：

1、DP 的本质

求有限的集合中的最值（个数）

本质上，DP 代表了走到阶段 i 的所有路线的最优解；

2、DP 需要思考的点：

（1）DP 的状态是什么？状态要求什么：最大、最小、数量？

（2）DP 的状态计算？

状态转义方程；

求解方法：a、递推 b、考虑阶段 i（最后一个阶段的值）的值是如何得来的；

（3）DP 的边界是什么？

关键术语：阶段、状态、决策（状态转移方程）、边界；

以数塔问题（1216: 【基础】数塔问题）为例，理解 DP 的本质，再理解 01 背包的本质（1282: 【提高】简单背包问题）；

经典的 DP 模板题要熟练掌握，熟记状态转义方程！

本题解题的关键点：二进制优化（类似压缩的思想）

（1）有 n 个不同的物品，要讨论 2n 种选择的可能（每个物品选或者不选）；

（2）一个物品有 n 件，虽然要讨论 2n 种选择的可能，但由于 n 个物品是一样的，那么就减少了讨论数量，比如：有 4 个物品，如果是不同物品的选 2 个，选 1 2、2 3 是不同的选择，但如果是相同的物品，选哪两个就都是一样的了。

因此，n 个物品，要讨论的可能就分别是：选 0 个、选 1 个、选 2 个、选 3 个…选 n 个。

（3）要将 0~n 个不同的选择表达出来，比较简单的方法是将 n 二进制化。

比如：整数 7，只需要用 1 2 4 三个数任意组合，就能组合出 0~7 这 8 种可能。

再比如：整数 10，只需要用 1 2 4 3（注意最后一个数），就能组合出 0~10 这 11 种可能，这样 n 这个值就被二进制化了。

因此如果要讨论 10 个一样的物品，就转化为讨论 4 个不同的物品了；而 n 个一样的物品，就转化为 $\log_2 n$ 个不同的物品进行讨论。

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

```cpp
#include <bits/stdc++.h>//4-1889  多重背包（2）二进制化  javacn
using namespace std;
const int N = 20010;
int v[N],w[N],dp[2010];
int n,m;    //n 种物品，背包容量为 m
int vi,wi,si;
int k = 0;
int main(){
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		cin>>vi>>wi>>si;
		int t = 1;          // 权重，表示 2 的次方
		while(t <= si)
		{	k++;
			v[k] = t * vi;
			w[k] = t * wi;
			si = si - t;
			t = t * 2;
		}
		if(si > 0)          // 如果二进制化有剩余，存入
		{
			k++;
			v[k] = si * vi;
			w[k] = si * wi;
		}
	}
	for(int i = 1;i <= k;i++)     //01 背包
	{
		for(int j = m;j >= v[i];j--)
			dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
	}
	cout<<dp[m];
	return 0;
}
```

对 si 二进制化，比如：有 10 件一样的物品，我们转换为有 4 件不同的物品：1 2 4 3 这 4 种物品的体积分别是：1*vi 2*vi 4*vi 3*vi

1905. 混合背包

有 N 种物品和一个容量是 V 的背包。物品一共有三类：

第一类物品只能用 1 次（01 背包）；

第二类物品可以用无限次（完全背包）；

第三类物品最多只能用 s_i 次（多重背包）；

每种体积是 v_i ，价值是 w_i。求解将哪些物品装入背包，可使物品体积总和不超过背包容量，且价值总和最大。输出最大价值。

输入：第一行两个整数，N，V，用空格隔开，分别表示物品种数和背包容积。

接下来有 N 行，每行三个整数 v_i, w_i, s_i ，用空格隔开，分别表示第 i 种物品的体积、价值和数量。

s_i=-1 表示第 i 种物品只能用 1 次；

s_i=0 表示第 i 种物品可以用无限次；

s_i>0 表示第 i 种物品可以使用 s_i 次；

数据范围 0<N,V ≤ 1000, 0<v_i, w_i ≤ 1000, -1 ≤ si ≤ 1000。

输出：输出一个整数，表示最大价值。

输入

4 5

1 2 -1

2 4 1

3 4 0

4 5 2

输出

8

多重背包转换为 01 背包，接下来分 01 背包、完全背包两种情况讨论：

```
#include <bits/stdc++.h>//5-1905      混合背包      javacn
using namespace std;
const int N = 20000;
int v[N],w[N],s[N];
int vi,wi,si;
int k = 0;          // 表示存入数组的数据量
int dp[1010];
int n,m;
```

```
int main()
{
	cin>>n>>m;
	for(int i = 1;i <= n;i++)
	{
		cin>>vi>>wi>>si;
		if(si > 0)		// 如果是多重背包，做二进制拆分
		{
			int t = 1;
			while(t <= si)
			{
				k++;
				w[k] = t * wi;
				v[k] = t * vi;
				s[k] = -1;		// 转换为 01 背包
				si = si - t;
				t = t * 2;
			}
			if(si > 0)
			{
				k++;
				w[k] = si * wi;
				v[k] = si * vi;
				s[k] = -1;		//01 背包
			}
		}
		else
		{
			k++;
			w[k] = wi;
			v[k] = vi;
			s[k] = si;
		}
	}
```

```
	for(int i = 1;i <= k;i++)	// 计算	// 循环 k 个物品
	{
		if(s[i] == -1)		// 判断是 01 背包还是完全背包
		{
			for(int j = m;j >= v[i];j--)
			{
				dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
			}
		}
		else
		{
			for(int j = v[i];j <= m;j++)
			{
				dp[j] = max(dp[j],dp[j-v[i]]+w[i]);
			}
		}
	}

	cout<<dp[m];
	return 0;
}
```

4、LIC 和 LCS

1794 - 最长不下降子序列（LIS）

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 且 $a_i \neq a_j$ $(i \neq j)$。

例如 3，18，7，14，10，12，23，41，16，24。

若存在 $i_1 < i_2 < i_3 < \cdots < i_e$ 且有$a_{i1} < a_{i2} < \cdots < a_{ie}$，则称为长度为 e 的不下降序列。

如上例中 3，18，23，24 就是一个长度为 4 的不下降序列，同时也有 3，7，10，12，16，24 长度为 6 的不下降序列。

程序要求，当原数列给出之后，求出最长的不下降序列。

输入

第一行为 n，表示 n 个数（$10 \leq n \leq 10000$）；

第二行 n 个整数，数值之间用一个空格分隔（$1 \leq a_i \leq n$）；

输出

最长不下降子序列的长度。

样例

输入 复制

```
3
1 2 3
```

输出 复制

```
3
```

dp 数组：存储状态，以 a[i] 结尾的最长不下降子序列的长度

归纳动态转移方程：

dp[i]=1

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

j 是 1..i-1 之间的数，a[j]<a[i]

```
/*
  dp[i]=1
dp[i]=max(dp[j]+1,dp[i])  j 是 1..i-1 之间的数，a[j]<a[i]
*/
//dp：存储以每个数结尾的最长不下降子序列的长度
```

```cpp
#include <bits/stdc++.h>//1-1794      最长不下降子序列（LIS）      javacn
using namespace std;
int a[10100],dp[10100],n,ma;
int main()
{
    int i,j;
      cin>>n;
      for(i = 1;i <= n;i++)
      {
            cin>>a[i];
      }


      for(i = 1;i <= n;i++)    // 求以每个数结尾的最长不下降子序列的长度
      {
            dp[i] = 1;

            for(j = 1;j < i;j++)// 将 a[i] 尝试续到每个数后面，看 dp[i] 能否增加
            {
                  if(a[j] < a[i])
                  {
                        dp[i] = max(dp[j]+1,dp[i]);
                  }
            }

            ma = max(dp[i],ma);
      }

      cout<<ma;
      return 0;
}
```

1893 - 最长上升子序列LIS（2）

题目描述

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。

输入

第一行包含整数 N 。

第二行包含 N 个整数，表示完整序列。

$1 \leq N \leq 100000$，$-10^9 \leq$ 数列中的数 $\leq 10^9$

输出

输出一个整数，表示最大长度。

样例

输入 复制

```
6
1 3 2 8 5 6
```

输出 复制

```
4
```

将原来的 dp 数组的存储以每个数结尾的 LIS 序列的长度，修改为存储上升子序列长度为 i 的上升子序列的最小末尾数值。

原理：LIS 长度如果已经确定，那么如果这种长度的子序列的结尾元素越小，后面可能续的元素会更多！

```cpp
#include <bits/stdc++.h>//2-1893    最长上升子序列 LIS（2）    javacn
using namespace std;
int a[100100],dp[100100];//dp: 长度为 i 的 LIS 的最后一位最小值是多少
int i,n,l,r,mid;
int main()
{
    scanf("%d",&n);    // 读入
    for(i = 1; i <= n; i++)            scanf("%d",&a[i]);
    dp[1] = a[1];      // 边界
    int len = 1;//LIS 的长度
    for(i = 2; i <= n; i++)        // 从第 2 个数开始求解
    {
        // 如果 a[i] 比 dp 最后一位大，a[i] 直接续上去，增加 LIS 的长度
        if(a[i] > dp[len])
        {
            len++;
            dp[len] = a[i];
        }
        else
        {
    // 二分查找到 dp 数组中第 1 个 >=a[i] 的元素下标，替换（dp 数组一定是递增的）
            l = 1;
            r = len;
            while(l <= r)
            {
                mid = (l + r) / 2;
                if(a[i] <= dp[mid]) r = mid - 1;
                else l = mid + 1;
            }
            dp[l] = a[i];      // 替换
        }
    }
    printf("%d",len);
    return 0;
}
```

1821 - 最长公共子序列(LCS)(1)

题目描述

给出 $1 \sim n$ 的两个排列 P_1 和 P_2，求它们的最长公共子序列。

输入

第一行是一个数 n；（n 是 $5 \sim 1000$ 之间的整数）

接下来两行，每行为 n 个数，为自然数 $1 \sim n$ 的一个排列（$1 \sim n$ 的排列每行的数据都是 $1 \sim n$ 之间的数，但顺序可能不同，比如 $1 \sim 5$ 的排列可以是：1 2 3 4 5，也可以是 2 5 4 3 1）。

输出

一个整数，即最长公共子序列的长度。

样例

输入 复制

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

输出 复制

```
3
```

我们可以用 dp[i][j] 来表示第一个串的前 i 位，第二个串的前 j 位的 LCS 的长度，那么递推出状态转移方程：

如果当前的 a [i] 和 b[j] 相同（即是有新的公共元素） 这说明该元素一定位于公共子序列中。因此，现在只需要找：a 数组 1~i-1 和 b 数组 1~j-1 的最长公共子序列：
dp[i][j]=max(dp[i][j],dp[i-1][j-1]+1);

如果不相同，说明最后一个元素肯定不是公共子序列中的元素，那么考虑找 a 数组 1~i-1 和 b 数组 1~j 的 LCS，或者找：a 数组的 1~i 和 b 数组的 1~j-1 的 LCS，那么，状态转义方程如下：dp[i][j]=max(dp[i-1][j],dp[i][j-1]);

```cpp
#include <bits/stdc++.h>//3-1821    最长公共子序列(LCS)(1)    javacn
using namespace std;

/*
a[i]==b[j], 方程：dp[i-1][j-1]+1
a[i]!=b[j], 方程：max(dp[i][j-1],dp[i-1][j])
*/
const int N = 1010;// 常量, 表示数组大小
int a[N],b[N],dp[N][N];
int n,i,j;

int main()
{
	cin>>n;
	for(i = 1;i <= n;i++) cin>>a[i];
	for(i = 1;i <= n;i++) cin>>b[i];

	// 递推
	for(i = 1;i <= n;i++)
	{
		for(j = 1;j <= n;j++)
		{
			if(a[i] == b[j]) dp[i][j] = dp[i-1][j-1] + 1;
			else dp[i][j] = max(dp[i-1][j],dp[i][j-1]);
		}
	}

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

1822 - 最长公共子序列(LCS)(2)

题目描述

给出 $1 \sim n$ 的两个排列 P_1 和 P_2，求它们的最长公共子序列。

和最长公共子序列(LCS)(1)问题不同的是，本题的 n 在 $5 \sim 100000$ 之间。

输入

第一行是一个数 n；（n 是 $5 \sim 100000$ 之间的整数）

接下来两行，每行为 n 个数，为自然数 $1 \sim n$ 的一个排列（$1 \sim n$ 的排列每行的数据都是 $1 \sim n$ 之间的数，但顺序可能不同，比如 $1 \sim 5$ 的排列可以是：1 2 3 4 5，也可以是 2 5 4 3 1）。

输出

一个整数，即最长公共子序列的长度。

样例

输入 复制

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

输出 复制

```
3
```

1、因为两个序列都是 1~n 的全排列，那么两个序列元素互异且相同，也就是说只是位置不同；

2、通过 c 数组将 b 序列的数字在 a 序列中的位置求出；

3、如果 b 序列每个元素在 a 序列中的位置递增，说明 b 中的这个数在 a 中的这个数整体位置偏后，可以考虑纳入 LCS；

4、从而就可以转变成求用来记录新的位置的 c 数组中的 LIS。

```
#include <bits/stdc++.h>//4-1822      最长公共子序列(LCS)(2)     javacn
using namespace std;
const int N = 100100;
int a[N],b[N],c[N],dp[N];
int n,i;
```

```
int main()
{
	cin>>n;
	for(i = 1; i <= n; i++)
	{
		scanf("%d",&a[i]);
		c[a[i]] = i;// 求出 a 数组的每个数的位置
	}
	for(i = 1; i <= n; i++)		scanf("%d",&b[i]);
	dp[1] = c[b[1]];// 边界  // 求 b 数组的每个数在 a 数组的位置 (c[b[i]]) 的 LIS
	int len = 1;
	int l,r,mid;
	for(i = 2; i <= n; i++)		// 从第 2 个数开始讨论
	{
		if(c[b[i]] > dp[len])		// 增加 LIS 的长度
		{
			len++;
			dp[len] = c[b[i]];
		}
		else
		{
			l = 1;
			r = len;
			while(l <= r)
			{
				mid = (l + r) / 2;
				if(c[b[i]] <= dp[mid]) r = mid - 1;
				else l = mid + 1;
			}
			dp[l] = c[b[i]];
		}
	}
	cout<<len;
	return 0;
}
```

1795. 拦截导弹

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。

输入导弹的枚数和导弹依次飞来的高度（雷达给出的高度数据是不大于 30000 的正整数，每个数据之间至少有一个空格），计算这套系统最多能拦截多少导弹。

输入：第 1 行有 1 个整数 n，代表导弹的数量。（n<=1000）

第 2 行有 n 个整数，代表导弹的高度。（雷达给出的高度数据是不大于 30000 的正整数）

输出：输出这套系统最多能拦截多少导弹。

输入

8

389 207 155 300 299 170 158 65

输出

6

```
#include<bits/stdc++.h>//5-1795  拦截导弹    求最长递减子序列的长度 javacn
using namespace std;
int n,a[1010],dp[1010],ma;
int main() {
	cin>>n;
	for(int i = 1;i <= n;i++)
	{
		cin>>a[i];
		dp[i] = 1;
		for(int j = 1;j < i;j++)
		{
			if(a[j] > a[i])
			{
				dp[i] = max(dp[j]+1,dp[i]);
			}
		}
		ma = max(dp[i],ma);
	}
	cout<<ma;
}
```

1902. 最少的修改次数

现有整数 A1,A2,...An，修改最少的数字为实数（整数或者小数），使得数列严格单调递增。

输入：第一行，一个整数 n 。（$n \leqslant 10^5$）。第二行，n 个整数 A_i。（$Ai \leqslant 10^9$）

输出：1 个整数，表示最少修改的数字的数量。

思路：求最少的修改次数， 那就是要找出需要修改的数字，而且越少也好。逆向思维，找最长的上升子序列（LIS）。然后，用总个数减去上升的，即需要修改的数字。

输入	输出
3 1 3 2	1

```
#include<bits/stdc++.h>//6-1902    最少的修改次数    dragoncatter
using namespace std;
const int N = 1e5 + 10 ;
int n, cnt, a[N], dp[N];
int main()
{
	cin >> n;
	for (int i = 1; i <= n; i++)		cin >> a[i];
	dp[1] = a[1];
	cnt = 1;
	for (int i = 2; i <= n; i++)
	{
		if (a[i] > dp[cnt])
		{
			dp[++cnt] = a[i];
		}
		else
		{
			int k = lower_bound(dp+1, dp+cnt + 1, a[i], less<int>()) - dp;
			dp[k] = a[i];
		}
	}
	cout << n - cnt;
	return 0;
}
```