问题 A: 北极熊的游戏

题目描述

米什卡是一只小北极熊。众所周知，小熊喜欢把空闲时间花在玩巧克力骰子上。有一次在一个阳光明媚的早晨，米什卡在冰块上散步，遇到了她的朋友克里斯，他们开始玩游戏

游戏规则非常简单：两人准备玩骰子比大小的游戏，共玩 n 轮。在每一轮中，每个玩家都会掷出一个 6 面立方体骰子，骰子的每一面分别写着从 1 到 6 的不同数字。掷骰子后点数较大的玩家赢得该回合。如果玩家骰子值相等，则没有人是赢家。

平均而言，赢得大部分回合的玩家是游戏的赢家。如果两个玩家赢得相同数量的回合，则游戏结果为平局。

米什卡还很小，无法计算输赢，所以她让你观看他们的比赛并确定结果。请帮助她！

输入格式

输入的第一行包含单个整数 n

n（$1 \le n \le 100$）——游戏回合数。

接下来的 n 行包含回合说明。其中第 i 个包含一对整数 m_i 和 c_i（$1 \le m_i, c_i \le 6$），分别对应第 i 轮中 Mishka 和 Chris 投掷后骰子上面的值。

输出格式

如果米什卡是游戏的赢家，请在唯一的行中打印“Mishka”（不带引号）。

如果克里斯是游戏的赢家，请在唯一的行中打印“Chris”（不带引号）。

如果游戏结果是平局，请在唯一的行中打印“Friendship is magic!^^”（不带引号）。

输入样例 1：

3

3 5

2 1

4 2

输出样例 1：

Mishka

输入样例 2

2

6 1

1 6

输出样例 2

Friendship is magic!^^

输入样例 3

3

1 5

3 3

2 2

输出样例 3

Chris

在第一个例子中，米什卡输掉了第一轮比赛，但赢得了第二轮和第三轮比赛，因此她是比赛的获胜者。
在第二个例子中，米什卡赢得了第一轮，克里斯赢得了第二轮，比赛以 1:1 的比分结束。
在第三个例子中，克里斯赢得了第一轮比赛，但接下来的两轮比赛没有赢家。这场比赛的获胜者是克里斯。

题解：
掷骰子比大小，问谁大比分领先，模拟即可。
代码如下：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    while (cin >> n)
    {
        int p1 = 0, p2 = 0;
        while (n--)
        {
            int P1, P2;
            cin >> P1 >> P2;
            p1 += (P1 > P2);
            p2 += (P2 > P1);
        }
        if (p1 > p2)
```

```
            cout << "Mishka" << endl;
        else if (p1 < p2)
            cout << "Chris" << endl;
        else
            cout << "Friendship is magic!^^" << endl;
    }
    return 0;
}
```

问题 B: 编程比赛

题目描述

陈教授喜欢编程竞赛，现在，他决定准备一个新的比赛。

总共有 n 名学生参加，在开始之前，学生从 1 到 n 编号，即为每个学生的初始排名。让我们将第 i 名学生的评分表示为一个 ai。比赛结束后，每个学生最终都会得到一个评分，分值越高排名也靠前。陈教授希望他的学生根据分值进行重新排名。如果学生 A 的评分严格低于学生 B，则 A 的排名在 B 后面，如果两个学生的评分相同，他们将共享相同的排名。陈教授希望你按照他的期望重建结果。如果一切按预期进行，请在比赛结束后确定每个学生的位置。

输入格式

第一行包含整数 n（1≤n≤2000），表示陈教授 的学生人数。

第二行包含 n 个数字 a1,a2,... an (1≤ai≤2000)， 其中 ai 是第 i 个学生的评分 （1≤i≤n）。

输出格式

在一行中，按照输入中显示的顺序打印 n 名学生在比赛结束后的排名。

Examples

Input

3

1 3 3

Output

3 1 1

Input

1

1

Output

1

Input

5

3 5 3 4 5

Output

4 1 4 3 1

Note

在第一个样本中，学生 2 和 3 分数最高， 排名第一（没有其他学生评分更高），学生 1 排名第三，因为有两个学生评分更高。

在第二个样本中，第一个学生是唯一参加比赛的学生。

在第三个样本中，学生 2 和 5 共享评分最高的第一位置，学生 4 紧随其后，排名第三，学生 1 和 3 是最后一个并列第四的位置。

题解：

题目要求输出每个数的排序，因此统计有几个数比它大就可以。

代码如下：

```
#include<iostream>
using namespace std;
int n,a[2000],t;
int main()
{
  scanf("%d",&n);
  for(int i=0;i<n;i++)scanf("%d",&a[i]);
  for(int i=0;i<n;i++)
  {
    t=1;//统计有几个比当前元素大的就可以
    for(int j=0;j<n;j++)if(a[j]>a[i])t++;
    printf("%d ",t);
  }
  return 0;
}
```

问题 C: Kefa 与伴侣

题目描述

Kefa 想去餐厅庆祝他的第一份高薪。他有 n 个朋友，每个朋友有一定的友谊值和工资。没人想觉得自己穷，所以 Kefa 邀请的朋友中两两工资差小于 d。现在给出朋友的信息，请求出最大友谊值是多少。

输入格式

输入的第一行包含两个空格分隔的整数，n 和 d（$(1\le n\le 10^5, 1\le d\le 10^9)$ ——分别是 Kefa 的朋友数量和令人感到贫穷的最小金额差。

接下来的 n 行包含对 Kefa 的朋友的描述，第(i+1)行包含对第 i 个朋友的 mi,si 的描述(0≤mi, si≤109) – 钱的数量和友谊值。

输出格式

输出可达到的最大总友谊值。

示例

Input1

4 5
75 5
0 100
150 20

75 1

Output1
100

Input2
5 100
0 7
11 32
99 10
46 8
87 54

Output2
111
在第一个样本测试中，最有效的策略是仅由第二个朋友组建伴侣。在所有其他变体中，友谊的总体程度会更糟。

在第二个样本测试中，我们可以带走所有的朋友。

题解：两两的差其实转化为最大和最小的差。那么我们先对所有人按工资排序，然后枚举最小的人是谁，在他工资上加 d，然后找到小于这个值的人并求和。这可以用二分查找找到小于这个值的最大工资的人，求和的话可以前缀和。
代码如下：

```
#include<bits/stdc++.h>
using namespace std;

const int maxn = 1e5 + 10;
long long n, d, ans, sum[maxn];
struct node {
    long long x, y; // x is money;
}a[maxn];
bool cmp(node a, node b) {
    return a.x < b.x;
}
int main(){
    cin >> n >> d;
    for(int i = 1; i <= n; ++i)
        cin >> a[i].x >> a[i].y;
    sort(a + 1, a + n + 1, cmp);
    for(int i = 1; i <= n; ++i)
        sum[i] = sum[i - 1] + a[i].y; //获取前缀和，注意一定要在排序之后
    int r = 1;
    for(int l = 1; l <= n; ++l) {
```

```
            while(r <= n && a[r].x - a[l].x < d) { //不满足退出
                ans = max(ans, sum[r] - sum[l - 1]);
                r ++; //满足条件，继续寻找
            }
        }
        cout << ans;
        return 0;
}
```

方法 2：

```
#include <bits/stdc++.h>
using namespace std;
int n;
long long d;
struct point
{
    long long x, y;
} a[100005];
long long s[100005], ans;
bool cmp(point x, point y)
{
    return x.x < y.x;
}
int find(long long x)
{
    int l = 1, r = n;
    while (l <= r)
    {
        int mid = (l + r) / 2;
        if (a[mid].x >= x)
        {
            r = mid - 1;
        }
        else
            l = mid + 1;
    }
    return r;
}
int main()
{
    cin >> n >> d;
    for (int i = 1; i <= n; i++)
        cin >> a[i].x >> a[i].y;
    sort(a + 1, a + n + 1, cmp);
```

```
    for (int i = 1; i <= n; i++)
        s[i] = a[i].y + s[i - 1];
    for (int i = 1; i <= n; i++)
    {
        long long p = a[i].x + d;
        int u = find(p);
        ans = max(ans, s[u] - s[i - 1]);
    }
    cout << ans;
    return 0;
}
```

问题 D: 英雄联盟

题目描述

正在上大学的小皮球热爱英雄联盟这款游戏，而且打的很菜，被网友们戏称为「小学生」现在，小皮球终于受不了网友们的嘲讽，决定变强了，他变强的方法就是：买皮肤！

小皮球只会玩 N 个英雄，因此，他也只准备给这 N 个英雄买皮肤，并且决定，以后只玩有皮肤的英雄。

这 N 个英雄中，第 i 个英雄有 Ki 款皮肤，价格是每款 Ci Q 币（同一个英雄的皮肤价格相同）。

为了让自己看起来高大上一些，小皮球决定给同学们展示一下自己的皮肤，展示的思路是这样的：对于有皮肤的每一个英雄，随便选一个皮肤给同学看。比如，小皮球共有 5 个英雄，这 5 个英雄分别有 0,0,3,2,4 款皮肤，那么，小皮球就有 3×2×4=24 种展示的策略。

现在，小皮球希望自己的展示策略能够至少达到 M 种，请问，小皮球至少要花多少钱呢？

输入格式

第一行，两个整数 N,M。

第二行，N 个整数，表示每个英雄的皮肤数量 Ki。

第三行，N 个整数，表示每个英雄皮肤的价格 Ci。

输出格式

一个整数，表示小皮球达到目标最少的花费。

输入样例

3 24

4 4 4

2 2 2

输出样例

18

每一个英雄都只有 4 款皮肤，每款皮肤 2 Q 币，那么每个英雄买 3 款皮肤，3×3×3≥24，共花费 6×3Q 币。

数据范围

第 i 组数据满足：N≤max(5,log24i)

100%的数据：M<=10^17,1≤Ci≤199。保证有解。

题解：对于每一个皮肤，都有一个数量、一个购买的 Q 币数(花费)，那么这种有限物品数量的背包问题就是多重背包问题。

根据题目要求的量，我们来设计状态。我们看到，方案数量是价值，Q 币数量是我们的花费，于是我们用 dp[i][j]来表示前 i 个皮肤的 j 个花费的最大方案数

状态转移方程：dp[i][j]=max(dp[i][j],dp[i−1][j−p∗ c[i]]∗ p)

其中 p 是当前英雄选的皮肤数量，c[i]是题目中的意思。

用一维数组来表示状态:用 dp[j]表示花费 j 个 Q 币的最大方案数量

dp[j]=max(dp[j−p∗ c[i]]∗ p,dp[j])

代码如下：

```
#include <iostream>
#include <cstdlib>
#include <cstdio>
using namespace std;
long long int dp[1000001];                    // 状态数组
long long int k[1000001], c[1000001]; // 数组意义如题目所述
long long int n, m, qb;
int main()
{
    int i, j, p;
    cin >> n >> m;
    for (i = 1; i <= n; i++) // 输入
    {
        cin >> k[i];
    }
    for (i = 1; i <= n; i++)
    {
        cin >> c[i];
        qb += c[i] * k[i]; // 将 Q 币总量记录下来
    }
    dp[0] = 1;                          // 初始化
    for (i = 1; i <= n; i++) // 枚举皮肤的种类
    {
        for (j = qb; j >= 0; j--) // 枚举每一“格”Q 币数量
        {
            for (p = 0; p <= k[i] && p * c[i] <= j; p++) // 枚举当前皮肤的数量
            {
                dp[j] = max(dp[j], dp[j - p * c[i]] * p); //状态转移方程
            }
        }
```

```
    }
    long long int ans = 0;
    while (ans <= qb && dp[ans] < m)
        ans++; // 找到最小、同时大于 M 的那一“格”Q 币数量
    cout << ans;
    return 0;
}
```

问题 E: 午餐费用

题目描述

Watashi 是 ZJU-ICPC 团队的队长，他非常善良。在 ZJU-ICPC 夏令营中，学生被分成几个小组，每天其中一个小组将设计一些题目来举行比赛。今天 C 组的学生被要求设计问题，他们花了一整晚的时间来检查测试数据，这让他们非常疲劳。Watashi 决定给 C 组一些钱作为奖励，以便他们可以免费购买午餐。

培训日程中有 N 天，所有学生都预订了 N 天的午餐，所以我们知道他们每天要花多少钱。现在，C 组组长需要决定如何使用 Watashi 的钱。由于钱有限，他们不可能每天都有免费的午餐。因此，每天领导者可以选择自己支付整个团队的午餐费用，或者该日午餐费用全部使用 Watashi 的钱支付。当然，领导希望尽可能多地花 Watashi 的钱，但他太忙了，没有时间编写程序来计算他可以从瓦塔西奖励中花费的最大金额。你能帮他吗？

输入格式

输入包含多个测试用例（不超过 50 个测试用例）。

在每个测试案例中，首先有两个整数，N（1<=N<=30），这是训练天数，M（0<=M<=100000），是 Watashi 的奖金。

然后有一行包含 N 个正整数，其中第 i 个整数表示 C 组需要支付第 i 天的午餐。所有这些整数都不超过 10000000，整数之间用空格隔开。

输出格式

对于每个测试用例，输出一行整数，这是 C 组可以从 Watashi 的奖励中花费的最大金额

输入样例

3 10

8 4 5

输出样例

9

题解：要用 dfs+剪枝来写。就剪枝来说，主要有几个方面：一是满足条件则返回；二是每次都更新最优值；三是花费和已经超过奖金数，则返回；四是下标超过则返回；五是加当前的值小于奖金，则下标和值下移，否则回溯。具体在注释已经给出，详见 code。

代码如下：

```
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

const int MAXN = 30+5;
int n,m,ans,tmp,flag;
```

```
int c[MAXN];

void dfs(int i,int d){
	if(flag) return ; //已经满足，则不需要遍历了
	if(d==m){ans=d;flag=1;return ;} //满足则返回
	if(ans<d) ans=d; //更新最优值
	if(d>m) return ; //已经超过 m，则后面的不需遍历了
	if(i>=n) return ; //下标已经超过 n，则后面的不需遍历了
	if(d+c[i]<=m) //当相加和小于 m，则下标和值都加
		dfs(i+1,d+c[i]);
	dfs(i+1,d); //否则回溯，移动下标进行下一次判断
}

int main(){
	while(~scanf("%d%d",&n,&m)){
		ans=0;tmp=0;flag=0;
		for(int i=0;i<n;i++){
			scanf("%d",&c[i]);
			tmp+=c[i];
		}
		if(tmp<=m){ //所有数之和小于等于 m，则说明必须全部每天都要花掉
			printf("%d\n",tmp);
			continue;
		}
		dfs(0,0);
		printf("%d\n",ans);
	}
	return 0;
}
```