

codeforces round 638 div.2

不难的一套题。虽然EF两题还没做出来。

开始时间：19:30。坚持了约两个半小时。

通过情况 4/6 165min

题目	A	B	C	D	E	F
通过	√	√	√	√		
补题						

A

题面

菲尼克斯有 n 个硬币，重量 $2^1, 2^2, \dots, 2^n$ 他知道 n 是偶数。

他想把硬币分成两堆，这样每堆硬币正好有 $n/2$ 个硬币，两堆硬币之间的重量差最小化。形式上 a 表示第一堆中的权重之和 b 表示第二堆中的权重之和。帮助菲尼克斯最小化 $a-b$ 的绝对值。

输入

输入由多个测试用例组成。第一行包含整数 t ($1 \leq t \leq 100$) - 测试用例数。

每个测试用例的第一行包含一个整数 n ($2 \leq n \leq 30$ (n 为偶数)) - 菲尼克斯拥有的硬币数量。

输出

对于每个测试用例，输出一个整数 - 两个桩之间可能的最小权重差。

样例

输入

2

2

4

输出

2

6

注意

在第一个测试案例中，凤凰有两个硬币，重量分别为2和4。无论他如何划分硬币，差别将是 $4-2=2$ 。

在第二个测试案例中，凤凰有四枚重量分别为2、4、8和16的硬币。菲尼克斯最好把重量为2和16的硬币放

在一堆里，把重量为4和8的硬币放在另一堆里。差别是 $(2+16) - (4+8) = 6$ 。

分析

这个题很显然。如果将最大的 2^n 放到一堆，那么剩余所有的硬币加起来也不如 2^n 大。

因此，差别最小的，一定是最大的 2^n 和较小的 $1+n/2$ 个硬币放在一起，剩余硬币放在一起。根据等比数列求和，最小的差为 $2^{\lceil 1+n/2 \rceil} - 2$ 个。

代码

```
#include<stdio.h>
#include<math.h>

void solve()
{
    int n;
    scanf("%d",&n);
    n/=2;
    long long ans=pow(2,n+1);
    printf("%lld\n",ans-2);
}

int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        solve();
    }
}
```

B

题面

凤凰喜欢美丽的阵列。如果所有长度为 k 的子数列具有相同的和，则数组是美丽的。数组的子数列是任何连续元素的序列。

菲尼克斯现在有一个长度为 n 的数组 a 。他想在数组中插入一些整数，插入个数可能是0，这样数组就变得漂亮了。插入的整数必须介于1和 n 之间（包括1和 n ）。整数可以插入任何位置（甚至在第一个元素之前或之后），并且他并没有试图最小化插入的整数的数量。

输入

输入由多个测试用例组成。第一行包含整数 t ($1 \leq t \leq 50$) - 测试用例数。

每个测试用例的第一行包含两个整数 n 和 k ($1 \leq k \leq n \leq 100$)

每个测试用例的第二行包含 n 个空格分隔的整数 $1 \leq a_i \leq n$ - Phoenix 当前拥有的数组。这个数组可能很漂亮,也可能不漂亮。

输出

对于每个测试用例, 如果不可能创建一个漂亮的数组, 请打印 -1。否则, 打印两行。

第一行应包含美丽数组的长度 m ($n \leq m \leq 10^4$) 你不需要最小化 m

第二行应该包含 m 个空格分隔的整数 $1 \leq b_i \leq n$ - 菲尼克斯在其数组 a 中插入一些整数 (可能为零) 后可以获得的漂亮数组。您可以打印最初不在数组 a 中的整数。

如果有多种解决方案, 请打印一个解决方案。它保证了如果我们能使数组变得漂亮, 我们总是可以使它的长度不超过 10^4 。

样例

输入

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

输出

```
5
1 2 1 2 1
4
1 2 2 1
-1
7
4 3 2 1 4 3 2
```

注意

在第一个测试用例中，我们可以通过在索引3处插入整数1（在两个现有的2之间）使数组变漂亮。现在，长度 $k=2$ 的所有子数组的和都是3。还有许多其他可能的解决方案，例如：

2,1,2,1,2,1 1,2,1,2,1,2

在第二个测试用例中，数组已经很漂亮了：长度为 $k=3$ 的所有子数组都有相同的和5。在第三个测试用例中，可以证明我们不能插入数字来使数组变得漂亮。在第四个测试用例中，所示的阵列 b 是美丽的，并且长度 $k=4$ 的所有子阵列具有相同的和10。还有其他的解决办法。

分析

如果任意连续 k 个数都具有相同的和，那么这个数列一定是循环的，循环节为 k

这样一来，如果原数列中已经出现了超过 k 个不同的数字，一定不可能实现，否则应该可以实现。

既然不需要保证最短，我们只需要想各种办法让数列变成 k 位循环就行了。

假设这个数列一共用到了 c 个数字 $c \leq k$ 首先从左往右读，直到 c 个数字均在数列中出现。假设这 c 个数字出现的次序依次是 $d_1 d_2 \dots d_c$

那么就可以构造一个循环数列，循环节为 $d_1 d_2 \dots d_c d_c d_c \dots d_c$ 共 k 位。

接下来再从头开始读。每当读下一个数字的时候，先判断是不是循环节中下一个数。如果不是，就把后面一整个循环节续进去，直到是为止。这样就构造完了这个冗长的数列。

反正题目没有要求最优解，只要可行解就行了

有 c 个数字 $c \leq k$ 的话，直接 $1 2 3 \dots k$ 行不？循环几遍到 $m \geq n$ 停止？

也有道理。反正暴力解决，保证有子串就行了。

看了眼数据范围 $n < 100$ 这…直接输出自然数就行，要保证出现过，用个散列表统计一下。

代码

```
#include<stdio.h>

int a[1000000];
int b[1000000];
int ans[1000000];

int t,n,i,j,k;

void solve()
{
    scanf("%d%d",&n,&k);
    for(i=0;i<n;i++)
    {
```

```
    int tmp;
    scanf("%d",&tmp);
    a[tmp]++;
}
int cnt=0;
int max=0;
for(i=1;i<=n;i++)
{
    if(a[i]>0)
    {
        b[cnt++]=i;
    }
}
if(cnt>k)
{
    printf("-1\n");
}
else
{
    printf("%d\n",n*k);
    while(n--)
    {
        for(i=0;i<k;i++)
        {
            if(b[i]>0)
            {
                printf("%d ",b[i]);
            }
            else
            {
                printf("1 ");
            }
        }
        printf("\n");
    }
    memset(a,0,1000000);
    memset(b,0,1000000);
    memset(ans,0,1000000);
}

int main()
{
    int t;
    scanf("%d",&t);
    while(t--)
    {
        solve();
    }
}
```

C

题面

菲尼克斯有一个由小写拉丁字母组成的字符串 s 。他想把字符串中的所有字母都分配到 k 个非空字符串 $a_1a_2\dots a_k$ 中，这样 s 的每个字母都正好指向其中一个字符串 a_i 。字符串 a_i 不需要是 s 的子字符串。Phoenix可以分发 s 的字母，并在每个字符串 a_i 中重新排列字母。

例如，如果 $s=baba$ 和 $k=2$ ，Phoenix可能会以多种方式分发字符串中的字母，例如：

ba 和 ba a 和 abb ab 和 ab aa 和 bb

但这些方法是无效的：

baa 和 ba b 和 ba $baba$ 和空字符串 a_i 应为非空)

菲尼克斯想把字符串 s 的字母分布成 k 个字符串 $a_1a_2\dots a_k$ 以最小化其中词汇最大字符串，即最小化 $\max\{a_1a_2\dots a_k\}$ 帮助他找到最优分布并打印 $\max\{a_1a_2\dots a_k\}$ 的最小可能值。

String x 的字形小于 y 如果 x 是 y 的前缀并且 x 不等于 y 或者存在下标 $i \leftarrow \min(|x|, |y|)$ 使得 $x_i < y_i$ 对于每个 $j \in [1, i]$ 均有 $x_j = y_j$ 这里 $|x|$ 表示字符串 x 的长度。

总之就是字典序的意思。

输入

输入由多个测试用例组成。第一行包含整数 t $1 \leq t \leq 1000$ - 测试用例数。每个测试用例由两行组成。

每个测试用例的第一行包含两个整数 n 和 k $1 \leq k \leq n \leq 105$ - 字符串 s 的长度和非空字符串的数目。Phoenix希望将 s 的字母分别分配到这两个整数中。

每个测试用例的第二行包含一个长度为 n 的字符串 s 。该字符串仅由小写拉丁字母组成。

保证所有测试用例的 n 之和 ≤ 105 。

输出

打印 t 答案-每个测试用例一个。第 i 个答案应该是第 i 个测试用例中 $\max\{a_1a_2\dots a_k\}$ 的最小可能值。

样例

输入

6

4 2

baba

5 2

baacb

5 3

baacb

5 3

aaaaa

6 4

aaxxzz

7 1

phoenix

输出

ab

abbc

b

aa

x

ehinopx

注意

在第一个测试用例中，一个最优的解决方案是将baba分配到ab和ab[] 在第二个测试用例中，一个最优的解决方案是将baacb分配到abbc和a中。在第三个测试用例中，一个最佳解决方案是将baacb分配到ac[]ab和b中。在第四个测试用例中，一个最优的解决方案是将a aa aa分布到aa[]aa和a中。在第五个测试用例中，一个最佳解决方案是将aaxxzz分布到az[]az[]x和x中。在第六个测试用例中，一个最佳的解决方案是将phoenix分配到ehinopx中。

分析

先排序，然后分这几种情况：

比较s[0]和s[k-1]如果不同直接输出s[k-1]因为这时在首字母中s[k-1]已经是最大的了，后面什么都不加就是最小的，其他的加到别的地方。

如果s[0]和s[k-1]相同，说明首字母相同，注意样例中的aaaaa[]这就是一种特殊情况，这个情况a要均分，才是最小，比较s[k]和s[n-1]相同就是尽量均分。

s[k]和s[n-1]不同的话，只能把s[k]到s[n-1]全都放在一个的后面，因为如果分给了其他的，一定会导致字典序增大。如abbc变成abc[]

代码

```
#include<iostream>
#include<algorithm>

using namespace std;

void solve()
{
    int n, k;
    cin >> n >> k;
    string s;
    cin >> s;
    sort(s.begin(), s.end());
    if(s[0] != s[k-1])
    {
        cout<<s[k-1]<<endl;
        return;
    }
    cout<<s[k-1];
    if(s[k] != s[n-1])
    {
        int i;
        for(i = k; i < n; i++)
        {
            cout<<s[i];
        }
    }
    else
    {
        int i;
        for(i = 0; i < (n - 1) / k; i++)
        {
            cout<<s[k];
        }
    }
    cout <<endl;
    return;
}

int main()
{
    int t;
    cin >> t;
    while(t--)
    {
        solve();
    }
}
```



```
}
```

D

题面

菲尼克斯决定成为一名科学家！他目前正在调查细菌的生长。

最初，在第1天，有一种细菌的质量为1。

每天，一些细菌会分裂（可能为零或全部）。当一个质量为 m 的细菌分裂时，它变成两个质量为 $m/2$ 的细菌。例如，一个质量为3的细菌可以分裂为两个质量为1.5的细菌。

而且，每天晚上，所有细菌的质量增加1。

菲尼克斯想知道所有细菌的总质量是否有可能精确到 n 。如果可能的话，他感兴趣的是如何使用尽可能少的夜数来获得这个质量。帮助他成为最好的科学家！

输入

输入由多个测试用例组成。第一行包含整数 t $1 \leq t \leq 1000$ - 测试用例数。

每个测试用例的第一行包含一个整数 n $2 \leq n \leq 10^9$ - 菲尼克斯感兴趣的细菌总数。

输出

对于每个测试用例，如果细菌无法精确达到总质量 n 请打印-1。否则，打印两行。

第一行应该包含一个整数 d - 所需的最少夜数。

下一行应该包含 d 个整数，第 i 个整数表示第 i 天应该分裂的细菌数量。

如果有多种解决方案，请打印任何解决方案。

例子

输入复制

3

9

11

2

输出

3

1 0 2

3

1 1 2

1

0

注意

在第一个测试用例中，以下过程产生总质量为9的细菌：

第1天：质量为1的细菌分裂。现在有两种细菌，每种质量为0.5。

夜1：所有细菌的质量增加一个。现在有两种质量为1.5的细菌。

第二天：没有分裂。

第二晚：现在有两种细菌的质量是2.5。

第三天：两种细菌都分裂了。现在有四种质量为1.25的细菌。

第三晚：现在有四种细菌的质量是2.25。

总质量为 $2.25+2.25+2.25+2.25=9$ 。可以证明，3是所需的最少夜数。还有其它方法可以在3晚内获得总质量9。

在第二个测试用例中，以下过程产生总质量为11的细菌：

第1天：质量为1的细菌分裂。现在有两种质量为0.5的细菌。

夜1：现在有两种细菌的质量是1.5。

第二天：一个细菌分裂。现在有三种细菌的质量分别为0.75、0.75和1.5。

第二夜：现在有三种细菌，质量分别为1.75、1.75和2.5。

第3天：质量为1.75和2.5的细菌分裂。现在有5种细菌的质量分别为0.875、0.875、1.25、1.25和1.75。

第三夜：现在有五种细菌，它们的质量分别是1.875、1.875、2.25、2.25和2.75。

总质量为 $1.875+1.875+2.25+2.25+2.75=11$ 。可以证明，3是所需的最少夜数。还有其它方法可以在3晚内获得总质量11。

在第三个测试案例中，细菌在第1天不会分裂，然后在第1天晚上生长到第2块。

分析

因为最后要求的是总质量，显然我们完全不用关心细菌究竟质量多少，只要看总个数就行了。相当于要求每次增加的质量为当前细菌个数，并且只增不减，要求从1开始，最少步骤精确达到给定值 n 。

这个 n 就是细菌个数数列某个部分和+1（初始质量）。如果将1置于数列最前面（第0天），那就是部分和。要求数列下一项在前一项1倍到2倍之间。

可以想象，如果是 2^{n-1} 这样的数，只需要 $n-1$ 次全体分裂就行了，也就是说次数的大致增长量级是对数量级。

那么方法也就呼之欲出了。

需要的最少晚上数，就是 $\log_2 n$ 即二进制位数-1。例如对于质量1，需要0个晚上。

细菌总个数和总质量无关……

对于二进制数1xxxxxxxx.....考虑后面xxxxxxxx.....部分：

从左往右数第一个1出现在哪里，代表现在正在修改第几位。

10000 0 1 2 4——初始数列0 1 2 4。

10001 1 0 2 4——从初始数列0 1 2 4到最终数列1 2 4 8过渡中，正在修改12两位。

10010 1 1 1 4——从初始数列0 1 2 4到最终数列1 2 4 8过渡中，正在修改23两位。

10011 1 2 0 4——从初始数列0 1 2 4到最终数列1 2 4 8过渡中，正在修改23两位。

10100 1 2 1 3——从初始数列0 1 2 4到最终数列1 2 4 8过渡中，正在修改34两位。

10101 1 2 2 2——从初始数列0 1 2 4到最终数列1 2 4 8过渡中，正在修改34两位。

10110 1 2 3 1——从初始数列0 1 2 4到最终数列1 2 4 8过渡中，正在修改34两位。

10111 1 2 4 0——从初始数列0 1 2 4到最终数列1 2 4 8过渡中，正在修改34两位。

11000 1 2 4 1——修改1 2 4 8的末位。…… 11111 1 2 4 8

100000 0 1 2 4 8——以此类推

被修改的两位，和始终保持不变，为2的{位数减一}次幂，前一个数是从左往右数第一个1后面部分减1。

代码

```
#include<iostream>
#include<algorithm>
#include<vector>

using namespace std;

void solve()
{
    vector<int>inc;
    int N;
    cin >> N;
    int i;
    for(i = 1; i <= N; i*=2)
    {
        inc.push_back(i);
        N-=i;
    }
    if(N>0)
    {
        inc.push_back(N);
    }
    sort(inc.begin(),inc.end());
```

```
cout << inc.size()-1<<endl;
for(i = 1; i<inc.size(); i++)
{
    cout<<inc[i]-inc[i-1]<<' ';
}
cout <<endl;
}

int main()
{
    int t;
    cin >> t;
    while(t--)
    {
        solve();
    }
}
```

E

题面

菲尼克斯在后院摘浆果。有 n 灌木，每个灌木有 a_i 红色浆果和 b_i 蓝色浆果。

每个篮子可以装 k 浆果。但是，菲尼克斯决定每个篮子只能装同一种灌木的浆果或同一颜色的浆果（红色或蓝色）。换句话说，篮子里的所有浆果必须来自同一种灌木或/和具有相同的颜色。

例如，如果有两个灌木，第一个灌木中有5红色和2蓝色浆果，第二个灌木中有2红色和1蓝色浆果，则Phoenix可以完全填充2篮容量4：

第一个篮子将包含来自第一个灌木的3红色和1蓝色浆果；

第二个篮子将包含来自第一个灌木的2剩余红色浆果和来自第二个灌木的2红色浆果。

帮助菲尼克斯确定他能完全填满的最大篮数！

输入

第一行包含两个整数 n 和 k ($1 \leq n \leq 500$) - 分别是灌木的数量和篮子容量。

接下来的 n 行中的 i -th包含两个整数 a_i 和 b_i ($0 \leq a_i, b_i \leq 10^9$) - 分别是 i -th灌木中红色和蓝色浆果的数量。

输出

输出一个整数-菲尼克斯可以完全填充的最大篮数。

样例

输入

2 4

5 2

2 1

输出

2

输入

1 5

2 3

输出

1

输入

2 5

2 1

1 3

输出

0

输入

1 2

1000000000 1

输出

500000000

注意

上面描述了第一个示例。

在第二个例子中，菲尼克斯可以用第一个（也是唯一一个）灌木的所有浆果填满一个篮子。

在第三个示例中，Phoenix无法完全填充任何篮子，因为每个灌木中的浆果少于5，总红色浆果少于5，总蓝色浆果少于5。

在第四个例子中，菲尼克斯可以把所有的红色浆果放进篮子里，留下一个额外的蓝色浆果。

分析

大致意思是只能从矩阵同行或同列取值，要求每次取值必须填满篮子 k 个），问最多填多少个篮子。

n 棵树上 a_i 个红果 b_i 个蓝果，每个篮子容量为 k 可以装同一棵树上的果实（条件一）或者颜色相同的果实（条件二），问最多使多少篮子恰好填满

$1 \leq n, k \leq 500, 1 \leq a_i, b_i \leq 1e9$

以下搬运自其他页面的本题题解：

考虑暴力 dp ， $dp[i][j][k]$ 表示考虑到第 i 种物品，红色的篮子还可以放 j 个，蓝色的篮子还可以放 k 个的最大值。

这样如果直接的转移是 n^5 的，显然不行。可以考虑一下 j 是否能推出 k

发现有果实总数- $dp[i][j]*$ 每框个数= k

转移就是枚举之前红色的框剩几个，现在红色的框剩几个。剩下的差就是本次与蓝色单独装框的即可。

考虑到对于一棵树如果装在 t 个篮子中，一定可以使大于等于 $t-1$ 个篮子满足条件一，因而所有最优解中存在一种方案使得每棵树最多提供一个满足条件二的篮子

因而记 $f(i, j)$ 表示前 i 棵树装完剩余 j 个红果的最大篮子数， S_i 表示前 i 棵树果子总和，可以推得剩余蓝果数量为 $S_i - f(i, j) * k - j$

下面只简述存在条件二篮子的转移

枚举该篮子中红果数量 num_1 蓝果为 $k - num_1$ 令 $t_1 = j - num_1 + a_i, t_2 = S_i - f(i, j) * k - j - k + num_1 + b_i$

转移方程 $f(i+1, t_1 \% k) = \max(f(i+1, t_1 \% k), f(i, j) + 1 + t_1 / k + t_2 / k)$

F

题面

菲尼克斯正在给他的 n 个朋友拍照，他们的标签是 $1 \ 2 \ \dots \ n$ 按特殊顺序排成一排。但他还没来得及拍照，他的朋友们就被一只鸭子弄得心烦意乱，把秩序搞得一团糟。

现在，菲尼克斯必须恢复秩序，但他记不清了！他只记得左边的第 i 个朋友在 a_i 和 b_i -inclusive之间有一个标签。有没有一种独特的方法可以根据他的记忆来安排他的朋友？

输入

第一行包含一个整数 $n \ 1 \leq n \leq 2 \cdot 10^5$ -朋友数。

接下来 n 行中的第 i 行包含两个整数 a_i 和 $b_i \ 1 \leq a_i \leq b_i \leq n$ -菲尼克斯从左侧对第 i 个位置的记忆。

菲尼克斯的内存是有效的，所以至少有一个有效的顺序。

输出

如果菲尼克斯可以按照唯一的顺序重新排列他的朋友，请在“是”后面加上 n 个整数-第 i 个整数应该是第 i

个朋友在左侧的标签。

否则，请打印“否”。然后，在以下两行上打印任意两个不同的有效订单。如果是多个解决方案，请打印任意两个。

样例

输入

4

4 4

1 3

2 4

3 4

输出

YES

4 1 2 3

输入

4

1 3

2 4

3 4

2 3

输出

NO

1 3 4 2

1 2 4 3

分析

（这个题也没思路）

大致意思是找一个合法的排列，使得每个数恰好落进给定的区间。

如果合法排列唯一，是YES；合法排列数大于等于2；NO并输出任意两个。

From:
<https://wiki.cvbbacm.com/> - **CVBB ACM Team**

Permanent link:
https://wiki.cvbbacm.com/doku.php?id=2020-2021:teams:namespace:codeforces_round_638_div_2 

Last update: **2020/07/10 16:29**