[编程题] 好多鱼！

时间限制：1秒

空间限制：32768K

牛牛有一个鱼缸。鱼缸里面已经有n条鱼，每条鱼的大小为fishSize[i] (1 ≤ i ≤ n,均为正整数)，牛牛现在想把新捕捉的鱼放入鱼缸。鱼缸内存在着大鱼吃小鱼的定律。经过观察，牛牛发现一条鱼A的大小为另外一条鱼B大小的2倍到10倍(包括2倍大小和10倍大小)，鱼A会吃掉鱼B。考虑到这个，牛牛要放入的鱼就需要保证：  
1、放进去的鱼是安全的，不会被其他鱼吃掉  
2、这条鱼放进去也不能吃掉其他鱼  
鱼缸里面已经存在的鱼已经相处了很久，不考虑他们互相捕食。现在知道新放入鱼的大小范围[minSize,maxSize](考虑鱼的大小都是整数表示),牛牛想知道有多少种大小的鱼可以放入这个鱼缸。

**输入描述:**

输入数据包括3行. 第一行为新放入鱼的尺寸范围minSize,maxSize(1 ≤ minSize,maxSize ≤ 1000)，以空格分隔。  
 第二行为鱼缸里面已经有鱼的数量n(1 ≤ n ≤ 50)  
 第三行为已经有的鱼的大小fishSize[i](1 ≤ fishSize[i] ≤ 1000)，以空格分隔。

**输出描述:**

输出有多少种大小的鱼可以放入这个鱼缸。考虑鱼的大小都是整数表示

**输入例子1:**

1 12 1 1

**输出例子1:**

3

[编程题] 循环单词

时间限制：1秒

空间限制：32768K

如果一个单词通过循环右移获得的单词，我们称这些单词都为一种循环单词。 例如：picture 和 turepic 就是属于同一种循环单词。 现在给出n个单词，需要统计这个n个单词中有多少种循环单词。

**输入描述:**

输入包括n+1行：

第一行为单词个数n(1 ≤ n ≤ 50)

接下来的n行，每行一个单词word[i]，长度length(1 ≤ length ≤ 50)。由小写字母构成

**输出描述:**

输出循环单词的种数

**输入例子1:**

5

picture

turepic

icturep

word

ordw

**输出例子1:**

2

[编程题] DNA合成

时间限制：1秒

空间限制：32768K

DNA分子是以4种脱氧核苷酸为单位连接而成的长链，这4种脱氧核苷酸分别含有A,T,C,G四种碱基。碱基互补配对原则：A和T是配对的，C和G是配对的。如果两条碱基链长度是相同的并且每个位置的碱基是配对的，那么他们就可以配对合成为DNA的双螺旋结构。现在给出两条碱基链，允许在其中一条上做替换操作：把序列上的某个位置的碱基更换为另外一种碱基。问最少需要多少次让两条碱基链配对成功

**输入描述:**

输入包括一行： 包括两个字符串,分别表示两条链,两个字符串长度相同且长度均小于等于50。

**输出描述:**

输出一个整数，即最少需要多少次让两条碱基链配对成功

**输入例子1:**

ACGT TGCA

**输出例子1:**

0

[编程题] 连续整数

时间限制：1秒

空间限制：32768K

牛牛的好朋友羊羊在纸上写了n+1个整数，羊羊接着抹除掉了一个整数，给牛牛猜他抹除掉的数字是什么。牛牛知道羊羊写的整数神排序之后是一串连续的正整数，牛牛现在要猜出所有可能是抹除掉的整数。例如：  
10 7 12 8 11 那么抹除掉的整数只可能是9  
5 6 7 8 那么抹除掉的整数可能是4也可能是9

**输入描述:**

输入包括2行：  
 第一行为整数n(1 <= n <= 50)，即抹除一个数之后剩下的数字个数  
 第二行为n个整数num[i] (1 <= num[i] <= 1000000000)

**输出描述:**

在一行中输出所有可能是抹除掉的数,从小到大输出,用空格分割,行末无空格。如果没有可能的数，则输出mistake

**输入例子1:**

2 3 6

**输出例子1:**

mistake

[编程题] 超级素数幂

时间限制：1秒

空间限制：32768K

如果一个数字能表示为p^q(^表示幂运算)且p为一个素数,q为大于1的正整数就称这个数叫做超级素数幂。现在给出一个正整数n,如果n是一个超级素数幂需要找出对应的p,q。

**输入描述:**

输入一个正整数n(2 ≤ n ≤ 10^18)

**输出描述:**

如果n是一个超级素数幂则输出p,q,以空格分隔,行末无空格。 如果n不是超级素数幂，则输出No

**输入例子1:**

27

**输出例子1:**

3 3

[编程题] 序列和

时间限制：1秒

空间限制：32768K

给出一个正整数N和长度L，找出一段长度大于等于L的连续非负整数，他们的和恰好为N。答案可能有多个，我我们需要找出长度最小的那个。  
例如 N = 18 L = 2：  
5 + 6 + 7 = 18   
3 + 4 + 5 + 6 = 18  
都是满足要求的，但是我们输出更短的 5 6 7

**输入描述:**

输入数据包括一行： 两个正整数N(1 ≤ N ≤ 1000000000),L(2 ≤ L ≤ 100)

**输出描述:**

从小到大输出这段连续非负整数，以空格分隔，行末无空格。如果没有这样的序列或者找出的序列长度大于100，则输出No

**输入例子1:**

18 2

**输出例子1:**

5 6 7

[编程题] 页码统计

时间限制：1秒

空间限制：32768K

牛牛新买了一本算法书，算法书一共有n页，页码从1到n。牛牛于是想了一个算法题目：在这本算法书页码中0~9每个数字分别出现了多少次？

**输入描述:**

输入包括一个整数n(1 ≤ n ≤ 1,000,000,000)

**输出描述:**

输出包括一行10个整数，即0~9这些数字在页码中出现的次数，以空格分隔。行末无空格。

**输入例子1:**

999

**输出例子1:**

189 300 300 300 300 300 300 300 300 300

[编程题] 01翻转

时间限制：1秒

空间限制：32768K

牛牛正在挑战一款名为01翻转的游戏。游戏初始有A个0,B个1，牛牛的目标就是把所有的值都变为1，每次操作牛牛可以任意选择恰好K个数字，并将这K个数字的值进行翻转(0变为1，1变为0)。牛牛如果使用最少的操作次数完成这个游戏就可以获得奖品，牛牛想知道最少的操作次数是多少？  
例如:A = 4 B = 0 K = 3   
0000 -> 1110 -> 1001 -> 0100 -> 1111   
需要的最少操作次数为4

**输入描述:**

输入为一行： 一共三个整数A(0 ≤ A ≤ 100,000),B(0 ≤ B ≤ 100,000),K(1 ≤ K ≤100,000).以空格分隔

**输出描述:**

输出一个整数，表示最少需要的操作次数。如果不能完成，则输出-1

**输入例子1:**

4 0 3

**输出例子1:**

4

1. #include<iostream>

#include<cmath>

using namespace std;

int main(){

    int minSize, maxSize, n, fishsize, count = 0;

    int size[1001] = { 0 };

    cin >> minSize >> maxSize >> n;

    for (int i = 1; i <= n; i++){

        cin >> fishsize;

        for (int j = 2 \* fishsize; j <= maxSize && j <= 10 \* fishsize; j++)

            size[j] = 1;

        for (int j = fishsize / 2; j >= minSize && j >= ceil(fishsize / 10.0); j--)

            size[j] = 1;

    }

    for (int i = minSize; i <= maxSize; i++)

        if (size[i] == 0)

            count++;

    cout << count;

    return 0;

}

2. #include <string>

#include <iostream>

#include <vector>

using namespace std;

int main()

{

int n,num=0;

vector<string> twords;

vector<bool> checks;//判断是否已属于某种循环单词

cin>>n;

for (int i = 0; i < n; i++)

{

string t\_w;

cin >> t\_w;

twords.push\_back(t\_w);

checks.push\_back(false);

}

for (int j = 0; j < n; j++)

{

if (!checks[j]){

string tt;

tt = twords[j] + twords[j];

for (int k = j + 1; k < n; k++)

{

if (!checks[k]){

if (tt.find(twords[k]) != string::npos&&twords[k].length() == twords[j].length()){ checks[k] = true; }

}

}

num++;

}

}

cout << num << endl;

return 0;

}

3. #include<iostream>

#include<string>

using namespace std;

int main(){

    string a, b;

    cin >> a >> b;

    int len = a.length();

    int count = 0;

    for (int i = 0; i < len; i++)

    if ((a[i] == 'A' && b[i] == 'T') || (a[i] == 'T' && b[i] == 'A') || (a[i] == 'C' && b[i] == 'G') || (a[i] == 'G' && b[i] == 'C'))

        count++;

    cout << len - count;

    return 0;

}

4. #include<iostream>

using namespace std;

int main() {

    int n;

    cin>>n;

    int min = 1000000001;

    int max = 0;

    int res = 0;

    int num = 0;

    for(int i = 0; i < n; i++) {

        cin>>num;

        if(num < min) min = num;

        if(num > max) max = num;

        res ^= num;

    }

    for(int i = min; i <= max; i++) {

        res ^= i;

    }

    if(res == 0) {

        if(min - 1> 0)

            cout<<min-1<<" "<<max+1;

        else cout<<max+1;

    }

    else if(res <= min || res >= max) {

        cout<<"mistake"<<endl;

    }

    else cout<<res;

}

5. #include<iostream>

#include<cmath>

using namespace std;

bool isprime(int n){

    int s = sqrt(n);

    for (int i = 2; i <= s; i++)

    if (n % i == 0)

        return false;

    return true;

}

int main(){

    long long int n;

    while (cin >> n){

        int p, q;

        int ceil = log10(n) / log10(2);

        for (q = 2; q <= ceil; q++){

            p = pow(n, 1.0 / q);

            //double转换为int会丢失，所以下面要再次判断

            if (pow(p, q) == n && isprime(p)){

                cout << p << " " << q;

                break;

            }

        }

        if (q > ceil)

            cout << "No";

    }

    return 0;

}

//时间复杂度O（log2(N) \* sqrt(sqrt(N))），也就是 以2为底N的对数 乘以 N的4分之1次方

6. #include<iostream>

using namespace std;

int main() {

int N, L;

cin>>N>>L;

for(int i = L; i <= 100; i++) {

if((2\*N+i-i\*i)%(2\*i) == 0) {

int start = (2\*N+i-i\*i)/(2\*i);

for(int j = 0; j < i-1; j++) {

cout<<start+j<<" ";

}

cout<<start+i-1;

return 0;

}

}

cout<<"No"<<endl;

}

7. #include<iostream>

using namespace std;

int count(int n, int x) {

    int res = 0, j;

    for (int i = 1; j = n / i; i \*= 10) {

        int high = j / 10;

        if (x == 0) {

            if (high)

                high--;

            else

                break;

        }

        res += high \* i;

        int tem = j % 10;

        if (tem > x)

            res += i;

        else if (tem == x)

            res += n - j \* i + 1;

    }

    return res;

}

int main(){

    int n;

    while (cin >> n){

        cout << count(n, 0);

        for (int i = 1; i <= 9; i++)

            cout << " " << count(n, i);

    }

    return 0;

}

//最优的解法了，时间复杂度O(log10(N))，以10为底N的对数，也就是N的位数

8. #include<iostream>

#include<cmath>

using namespace std;

int function(int A, int B, int K){

    int remainder = A % K;//直接翻转后的剩余待翻转个数

    int count = A / K;//直接翻转

    int S = A + B;//总个数

    if (A == 0 || remainder == 0)

        ;

    else if ((S <= K) || (remainder % 2 == 1 && K % 2 == 0))

        count = -1;//无法翻成功的输出-1

    else if ((K + remainder) % 2 == 0 && count > 0 && B + K \* count >= 2 \* K - (remainder + K) / 2)

        count++;//一种特殊情况，还剩下K+remainder个0时直接翻两次即可完成

    else if (remainder % 2 == 0)

        count += 2 \* ceil(remainder / double(2 \* (S - K)));

       //每翻两次最多能把remainder中的2\*(S-K)个0翻成1，注意这里指的是最多，当翻最后2次或者S-K>remainder/2时，只需翻两次，所以这里用到了ceil（）

    else

        count += 2 \* ceil((K - remainder) / double(2 \* (S - K))) + 1;

       //当remainder是奇数时，相当于先把所有1中的K-remainder个翻成0，这样加上remainder一共K个0，只需额外再翻一次即可，K-remainder是奇数时，永远不能翻成功，是偶数时，翻转方法同上面

    return count;

}

int main(){

    int A, B, K;

    cin >> A >> B >> K;

    cout << function(A, B, K);

    return 0;

}

//没有循环，只用到了条件语句，时间复杂度O(1),空间复杂度O(1)