题目：

<http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&page=show_problem&problem=36>

# The 3n + 1 problem

Time limit: 3.000 seconds

## Background

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

## The Problem

Consider the following algorithm:

1. input *n*

2. print *n*

3. if *n* = 1 then STOP

4. if *n* is odd then tex2html_wrap_inline44

5. else tex2html_wrap_inline46

6. GOTO 2

Given the input 22, the following sequence of numbers will be printed 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers *n* such that 0 < *n* < 1,000,000 (and, in fact, for many more numbers than this.)

Given an input *n*, it is possible to determine the number of numbers printed (including the 1). For a given *n* this is called the cycle-length of *n*. In the example above, the cycle length of 22 is 16.

For any two numbers *i* and *j* you are to determine the maximum cycle length over all numbers between *i* and *j*.

## The Input

The input will consist of a series of pairs of integers *i* and *j*, one pair of integers per line. All integers will be less than 1,000,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including *i* and *j*.

You can assume that no operation overflows a 32-bit integer.

## The Output

For each pair of input integers *i* and *j* you should output *i*, *j*, and the maximum cycle length for integers between and including *i* and *j*. These three numbers should be separated by at least one space with all three numbers on one line and with one line of output for each line of input. The integers *i* and *j* must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

## Sample Input

1 10

100 200

201 210

900 1000

## Sample Output

1 10 20

100 200 125

201 210 89

900 1000 174

# 注意

* 中间的计算结果会大于INT\_MAX，所以应该使用long long 来存储n
* 输入的i,j，不一定i < j

# 分析

本体考查的是穷举法+DP，还有能否优化程序提高效率的能力。

# 解法1-[笨法]

## 参考

## 耗时

0.965s

## 分析

直接穷举，不用动脑子

## 源码

#include <algorithm>

#include <iostream>

**using** **namespace** std**;**

int main**(**int argc**,** char **\***argv**[])**

**{**

**while** **(**1**)**

**{**

int i**,** j**,** \_max **=** 0**;**

cin **>>** i **>>** j**;**

**if** **(**cin**.**fail**())**

**break;**

int temp\_min **=** min**(**i**,** j**);**

int temp\_max **=** max**(**i**,** j**);**

**for** **(**int k **=** temp\_min**;** k **<=** temp\_max**;** **++**k**)**

**{**

long long n **=** k**;**

int count **=** 1**;**

**while** **(**1 **!=** n**)**

**{**

count**++;**

**if** **(**1 **==** n **%** 2**)**

n **=** 3 **\*** n **+** 1**;**

**else**

n **=** n **/** 2**;**

**}**

\_max **=** max**(**\_max**,** count**);**

**}**

cout **<<** i **<<** " " **<<** j **<<** " " **<<** \_max **<<** endl**;**

**}**

**return** 0**;**

**}**

# 解法2-[穷举法+DP]

## 参考

<http://blog.csdn.net/metaphysis/article/details/6431937>

## 耗时

0.045s

## 分析

* 中间计算结果可以保存下来，供后续计算使用。
* n=3n+1可以优化为n += (n << 1) + 1;
* n=n/2 可以优化为n = n >> 1;

## 源码

#include <iostream>

#include <algorithm>

#define MAXSIZE 1000000

**using** **namespace** std**;**

int cache**[**MAXSIZE**];**

int recursion**(**long long \_n**);**

int

main**(**int argc**,** char **\***argv**[])**

**{**

int i**,** j**;**

**while** **(**cin **>>** i **>>** j**)**

**{**

int result **=** 0**,** \_max **=** 0**;**

const int begin **=** min**(**i**,** j**);**

const int end **=** max**(**i**,** j**);**

**for** **(**long long k **=** begin**;** k **<=** end**;** **++**k**)**

**{**

result **=** recursion**(**k**);**

\_max **=** max**(**\_max**,** result**);**

**}**

cout **<<** i **<<** " " **<<** j **<<** " " **<<** \_max **<<** endl**;**

**}**

**return** 0**;**

**}**

int

recursion**(**long long \_n**)**

**{**

int count **=** 0**;**

**if** **(**1 **==** \_n**)**

count **=** 1**;**

**else**

**{**

count**++;**

**if** **(**\_n **&** 1**)**

\_n **+=** **(**\_n **<<** 1**)** **+** 1**;**

**else**

\_n **=** \_n **>>** 1**;**

**if** **(**MAXSIZE **<=** \_n**)**

**{**

// \_n is too large to record the result in cache

count **+=** recursion**(**\_n**);**

**}**

**else**

**{**

// DP, cache the result to speed up

**if** **(**0 **==** cache**[**\_n**])**

**{**

count **+=** recursion**(**\_n**);**

cache**[**\_n**]** **=** count**;**

**}**

**else**

count **=** cache**[**\_n**];**

**}**

**}**

**return** count**;**

**}**

# 解法3-[穷举法+DP]

## 参考

<http://blog.csdn.net/metaphysis/article/details/6431937>

## 耗时

0.052s

## 分析

## 源码

// The 3n+1 problem （3n+1 问题）

// PC/UVa IDs: 110101/100, Popularity: A, Success rate: low Level: 1

// Verdict: Accepted

// Submission Date: 2011-05-22

// UVa Run Time: 0.032s

//

// 版权所有（C）2011，邱秋。metaphysis # yeah dot net。

//

// [问题描述]

// 考虑如下的序列生成算法：从整数 n 开始，如果 n 是偶数，把它除以 2；如果 n 是奇数，把它乘 3 加

// 1。用新得到的值重复上述步骤，直到 n = 1 时停止。例如，n = 22 时该算法生成的序列是：

//

// 22，11，34，17，52，26，13，40，20，10，5，16，8，4，2，1

//

// 人们猜想（没有得到证明）对于任意整数 n，该算法总能终止于 n = 1。这个猜想对于至少 1 000 000

// 内的整数都是正确的。

//

// 对于给定的 n，该序列的元素（包括 1）个数被称为 n 的循环节长度。在上述例子中，22 的循环节长度

// 为 16。输入两个数 i 和 j，你的任务是计算 i 到 j（包含 i 和 j）之间的整数中，循环节长度的最大

// 值。

//

// [输入]

// 输入每行包含两个整数 i 和 j。所有整数大于 0，小于 1 000 000。

//

// [输出]

// 对于每对整数 i 和 j，按原来的顺序输出 i 和 j，然后输出二者之间的整数中的最大循环节长度。这三

// 个整数应该用单个空格隔开，且在同一行输出。对于读入的每一组数据，在输出中应位于单独的一行。

// [样例输入]

// 1 10

// 100 200

// 201 210

// 900 1000

// 900 1000

//

// [样例输出]

// 1 10 20

// 100 200 125

// 201 210 89

// 900 1000 174

//

// [解题方法]

// 计算每个数的循环节长度，求给定区间的最大值。

//

// 需要注意：

// 1. 中间计算过程会超过 int 或 long （如果 int 或 long 型均为 4 字节存储空间） 型数据所能

// 表示的范围，故需要选择 long long （8 字节存储空间）型整数（除非你使用的算法在做乘的时候不

// 使用一般的乘法，而是使用替代方法实现原数的三倍加一）。

// 2. 输入时可能较大的数在前面，需要调整顺序，这个是导致算法正确却 WA 的重要原因。

// 3. 采用填表的方法保存既往计算结果，可以显著减少计算时间。

//

// 从网络上看了许多别人的解题方案，大多数都是忽略了第一点，求循环节长度的过程中，选择了 int 或

// long （按 32 位 CPU 来假定，4 字节存储空间）类型的数据，当计算 （n \* 3 + 1） 时会超出 32

// 位整数的表示范围而得到错误答案，只不过 Programming Challenges 和 UVa 上的测试数据不是很强，

// 所以尽管不完善但都会获得 AC。在 1 - 999999 之间共有 41 个数在中间计算过程中会得到大于 32 位

// 无符号整数表示范围的整数，当测试数据包含这些数时，选用 int 或 long 类型有可能会得到错误的答案。

//

// 在中间计算过程中会超过 32 位整数表示范围的整数（括号内为循环节长度）：

// 159487（184） 270271（407） 318975（185） 376831（330） 419839（162）

// 420351（242） 459759（214） 626331（509） 655359（292） 656415（292）

// 665215（442） 687871（380） 704511（243） 704623（504） 717695（181）

// 730559（380） 736447（194） 747291（248） 753663（331） 763675（318）

// 780391（331） 807407（176） 822139（344） 829087（194） 833775（357）

// 839679（163） 840703（243） 847871（326） 859135（313） 901119（251）

// 906175（445） 917161（383） 920559（308） 937599（339） 944639（158）

// 945791（238） 974079（383） 975015（321） 983039（290） 984623（290）

// 997823（440）

#include <iostream>

**using** **namespace** std**;**

#define min(a, b) ((a) <= (b) ? (a) : (b))

#define max(a, b) ((a) >= (b) ? (a) : (b))

#define MAXSIZE 1000000

int cache**[**MAXSIZE**];**

// 计算循环节长度。

int counter**(**long long number**)**

**{**

**if** **(**number **==** 1**)**

**return** 1**;**

// 模 2 计算可用与计算代替，除 2 计算可用右移计算代替。

**if** **(**number **&** 1**)**

number **+=** **(**number **<<** 1**)** **+** 1**;**

**else**

number **>>=** 1**;**

// 若 number 在缓存范围内则根据情况取用。

**if** **(**number **<** MAXSIZE **)**

**{**

**if** **(!**cache**[**number**])**

cache**[**number**]** **=** counter**(**number**);**

**return** 1 **+** cache**[**number**];**

**}**

**return** 1 **+** counter**(**number**);**

**}**

int main**(**int ac**,** char **\***av**[])**

**{**

// 对于 GUN C++ 编译器，使用默认参数，在编译时会自动将全局数组 cache 中未初始化

// 的元素初始化为 0，故可以不需要显式的进行初始化的工作。对于其他编译器应该根据情况调整。

//

// memset(cache, 0, sizeof(cache));

//

int first**,** second**,** start**,** end**;**

**while** **(**cin **>>** first **>>** second**)**

**{**

// 得到给定范围的上下界。

start **=** min**(**first**,** second**);**

end **=** max**(**first**,** second**);**

// 查找最大步长值。

int result **=** 0**,** steps**;**

**for** **(**int i **=** start**;** i **<=** end**;** i**++)**

**if** **((**steps **=** counter**(**i**))** **>** result**)**

result **=** steps**;**

// 输出。

cout **<<** first **<<** " " **<<** second **<<** " " **<<** result **<<** endl**;**

**}**

**return** 0**;**

**}**