A solução direta para este problema está aqui.

```
#include <iostream>

using namespace std;

long long int n,m,a,b,i,cont;

long long int cycle(long long int n){

	long long int cont;

	cont = 1;

	while(n!=1){

		if(n%2==0) n = n/2;

		else n = 3*n+1;

		cont++;

	}

	return cont;

}

int main(){

	long long int max;

	while( cin >> n >> m ){

		a = n < m ? n: m;

		b = n > m ? n: m;

		max = 0;

		for(i=a;i<=b;i++){

			cont = cycle(i);

			if(cont > max) max = cont;

		}

		cout << n <<" " << m << " " << max << endl;

	}

}
```

2010-04-28 19:14:30 The 3n plus 1 problem accepted 0.28 2.6M C++

Na segunda tentativa, tentei otimizar o código aproveitando para calcular o tamanho do ciclo de vários números intermediários durante o cálculo de um número e utilizando memorização. Para isso foi utilizado uma fila e mapeamento. Só que o tiro saiu pela culatra(FAIL). O overhead associado ao mapeamento pesou no algoritmo. Se utilizasse tabela dispersão talvez seria melhor. Neste ponto, Java permite uma maior flexibilidade.

3574580 2010-04-28 19:24:43 The 3n plus 1 problem accepted 3.51 11M C++4.3.2

```
#include <iostream>

#include <stack>

#include <map>

#include <queue>

using namespace std;

long long int n,m,a,b,i,cont;

queue <long long int> q ;

map <long long int, long long int> mem;

long long int cycle(long long int n){

   long long int cont;

   long long int cont2;

   if(mem[n]!=0) return mem[n];

   cont = 1;

   while (!q.empty()) q.pop();

     while(n!=1){

       q.push(n);

       if(n%2==0) n = n/2;

       else n = 3*n+1;

       cont++;

    }

   cont2 = cont;

   while( !q.empty() ){

     mem[q.front()] = cont2--;

     q.pop();

   }

   return cont;

}
```

O jeito foi parar de preguiça e deixar de utilizar o mapeamento e utilizar somente um vetor.

2010-04-28 19:30:00 The 3n plus 1 problem accepted 0.39 10M

```
#include <iostream>

#include <stack>

#include <map>

#include <queue>

#include <string.h>

using namespace std;

long long int n,m,a,b,i,cont;

queue <long long int> q ;

long long int mem2[1000001];

long long int cycle(long long int n){

        long long int cont;

        long long int cont2;

        if(mem2[n]!=0) return mem2[n];

        cont = 1;

        while (!q.empty()) q.pop();

        while(n!=1){

                q.push(n);

                if(n%2==0) n = n/2;

                else n = 3*n+1;

                cont++;

        }

        cont2 = cont;

        while( !q.empty() ){

                if(q.front() <= 1000000)

                  mem2[q.front()] = cont2;

                cont2--;

                q.pop();

        }

        return cont;

}
```

Outra tentativa de otimização que não foi muito boa.

```
#include <iostream>

#include <queue>

using namespace std;

long long int n,m,a,b,i,cont;

queue <long long int> q ;

long long int mem2[1000001];

long long int cycle(long long int n){

	long long int cont;

	long long int cont2;

	if(mem2[n]!=0) return mem2[n];

	cont = 1;

	while (!q.empty()) q.pop();

	while(n!=1){

		q.push(n);

		if(n%2==0) n = n/2;

		else n = 3*n+1;

		if(n <= 1000000)

		 if(mem2[n]!=0)

		   return mem2[n]+cont;

		cont++;

	}

	cont2 = cont;

	while( !q.empty() ){

		if(q.front() <= 1000000)

		  mem2[q.front()] = cont2;

		cont2--;

		q.pop();

	}

	return cont;

}
```

Acredito que exista um modo de pré-calcular todos os valores de ciclo de maneira mais inteligente para evitar recálculos. Essa idéia abaixo precisa ser melhor ser trabalhada.

```
memset(mem2,sizeof(long long int),0);

cont = 1;

for(i=1;i<=1000000;i=i*2){

       mem2[i] = cont;

       if((i-1)%3==0){

         mem2[(i-1)/3] = cont+1;

       }

       cont++;

}
```