Solución problemas

Grupo de Algoritmia Avanzada y Programación Competitiva
Univesidad Nacional Autónoma de México
Facultad de Estudios Superiores Acatlán

October 2014

1. A - Avería

El problema nos pide hallar el primer momento en el que ambos limpiaparabrisas vuelven a su posicion original al mismo tiempo.

Entonces el problema nos pide calcular el **Minimo Común Múltiplo**.

Para el caso fácil, se puede ir aumentando por si misma una de las variables hasta que la otra la divida exactamente, esto es el primer momento en el que ambos se encuentran en su punto de inicio otra vez.

Para el caso difícil, las restricciones no nos permiten hacer el mismo cálculo debido al numero de casos de prueba.

La solución es ver que el Mínimo Común Multiplo de dos números se puede calcular de la siguiente forma:

$$MCM(a,b) = \frac{ab}{MCD(a,b)}$$

donde MCD es el máximo común divisor de los dos números.

Ahora, para calcular el Máximo Común Divisor de dos números eficientemente, podemos utilizar el algoritmo de Euclides.

La siguiente rutina basada en ese algoritmo nos devuelve el $MCD(a, b)$ y la calcula en $O(\log(n))$ donde n es el máximo de a y b.

```
int MCD(int a, int b)
{
        if (b == 0)
                return a;
    else
            return MCD( b, a \% b);
}
```

Con esto se puede resolver el caso difícil.

2. B - Bob y Peter

2.1. Problema

En éstos problemas te dábamos el número de casos y tenías que decir si una palabra es un palíndromo(caso fácil), o tiene una subsecuencia que sí es palíndromo.

2.2. Solución fácil

Aquí sólo tenías que revisar si 'T' palabras eran palíndromos o no, entonces el algoritmo sería más o menos así:

```
N = Numero de casos
for(i = 1; i <= N; i++)
        P_i = palabra i
    Espalindromo = 1
    for(pos=0; pos<tam(P_i); pos++)
        if(P_i[pos] != P_i[size(P_i) - 1 - i]
                Espalindromo = 0;
    if (Espalindromo == 1)
        Imprime(SI)
    else
        Imprime(NO)
```

2.3. Solución (difícil)

La solución más sencilla de programar de este algoritmo era revisando todos los tamaños, pero esto es muy tardado de calcular para una computadora, ya que para cada palabra se tienen máximo 1000 letras, y revisar que exista algún palíndromo de tamaño 2 hasta 1000 requiere $\approx 10^6$ cálculos (≈ 1000 comparaciones por cada tamaño de palíndromo), y eso multiplicado por las 2000 palabras, da un total de $\approx 2 * 10^9$ cálculos, lo cuál, por lo general, le toma más de 1 segundo hacerlos a una computadora común.

Pero si analizamos un poco los palíndromos, todos los palíndromos de tamaño ≥ 2 tienen en el centro de estos un subpalíndromo de tamaño 2 o 3, por lo que sólo buscaremos palíndromos de tamaño 2 o palíndromos de tamaño 3 (si no encontramos algún palíndromo de tamaño 2, menos encontraremos un palíndromo de tamaño 4, 6, 8...; y de manera similar para los palíndromos de tamaño impar)

```
T = Numero de casos
for(i = 1; i <= T; i++)
        P_i = palabra i
    TieneSubcad = 0
    for(pos = 1; pos < tam(P_i); pos++)
        if(P_i[pos] == P_i[pos-1])
                TieneSubcad = 1 //Palindromo de tamanio 2
    for(pos = 2; pos < tam(P_i); pos++)
        if(P_i[pos] == P_i[pos-2])
                TieneSubcad = 1 //Palindromo de tamanio 3
    if(TieneSubcad == 1)
        Imprime(SI)
    else
        Imprime(NO)
```

3. C - Conejos

El problema consistía en encontrar el n-ésimo fibonacci par. Sea $f_n = 4f_{n-1} + f_{n-2}$, con $f_1 = 2$ y $f_2 = 8$. Mostraremos que f_n nos da el n-ésimo fibonacci par. Dado que cada tercer elemento de la sucesión de fibonacci es par, basta con probar que $fib_n = 4fib_{n-3} + fib_{n-6}$.

$$\begin{aligned} fib(n) &= fib_{n-1} + fib_{n-2} \\ &= fib_{n-2} + 2fib_{n-3} + fib_{n-4} \\ &= 3fib_{n-3} + 2fib_{n-4} \\ &= 3fib_{n-3} + fib_{n-4} + fib_{n-5} + fib_{n-6} \\ &= 4fib_{n-3} + fib_{n-6} \end{aligned}$$

Con esto sólo necesitamos calcular f_n. Para el caso fácil era posible hacerlo en $O(n)$ con un **for**. Sin embargo para las restricciones difíciles ésto ya no era posible. Para resolver el caso difícil necesitamos observar que:

$$\begin{pmatrix} f_3 \\ f_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix}$$

Utilizando el mismo razonamiento tenemos:

$$\begin{aligned} \begin{pmatrix} f_4 \\ f_3 \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_3 \\ f_2 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} f_2 \\ f_1 \end{pmatrix} \end{aligned}$$

Generalizando ésta idea se obtiene:

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix}$$

Ahora sólo necesitamos obtener $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ de forma eficiente. Basándonos en el hecho de que

$$x^n = \begin{cases} (x^{\frac{n}{2}})(x^{\frac{n}{2}}) & \text{si } n \text{ es par} \\ x(x^{\frac{n}{2}})(x^{\frac{n}{2}}) & \text{si } n \text{ es impar} \end{cases}$$

podemos utilizar el siguiente procedimiento para calcularlo en $O(\log(n))$

```
matriz matexp(matriz X, long long n)
{
	if (n == 1)
		return X;

	matriz ans = matexp(X, n / 2);
	if (n es impar)
		return matmult( matmult(ans, ans), X );

	return matmult(ans, ans);
}
```

Donde $X = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$

Ahora sí ya podemos resolver el caso difícil en el tiempo requerido.

4. D - Desafío

El planteamiento pide la suma de los números menores a N tales que sean divisibles por 3 y 5. Así la solución queda como sigue:

Para el caso fácil podemos correr 2 ciclos que verifiquen qué números son divisibles por 3 y por 5 respectivamente, sumarlos, incrementar `i` en 3, 5, y en alguno de los 2 ciclos no contar a aquellos números que son múltiplos de ambos, ya que estaríamos contándolos 2 veces. Esta idea corre en $O(n)$. Lo cual es bueno si n no es "grande".

Para el caso díficl, la idea anterior no funcionaría ya que las restricciones obligarían a usar un ciclo demasiado grande, entonces usamos una idea mejor:

Pensemos en la suma de los primeros n múltiplos de 3 como:

$$3 + 6 + 9 + \ldots + 3n$$
$$3(1 + 2 + 3 + \ldots + n)$$

Nos percatamos que el segundo factor de la expresión anterior es la suma de los primeros n naturales, entonces queda como:

$$3\frac{n(n+1)}{2}$$

De manera similar para el caso de 5:

$$5\frac{n(n+1)}{2}$$

Entonces la suma de las expresiones anteriores es casi lo que queremos, esto porque en cada una de ellas se está contemplando los múltiplos de 5 y 3 al mismo tiempo, así que solo necesitamos eliminar una vez la suma de éstos:

$$15\frac{n(n+1)}{2}$$

Ahora ¿cómo encontramos los primeros n múltiplos menores a N?, fácil, basta con hacer una división entera de $N-1$ entre el divisor deseado. De esta manera nuestro programa se vería de la siguiente forma:

```
long long sn(int n)
{
	return n(n+1)/2;
}

long long suma (int N)
{
	return 3*sn((N-1)/3) + 5*sn((N-1)/5) - 15*sn((N-1)/15);
}
```

La rutina anterior al ser una suma de 3 términos los cuáles son operaciones sencillas, se asume que corre en $O(1)$, que es la notación para constantes.

5. E - Embriagándose con Gragas

Empezaré por explicar un poco más el problema, tomemos como referencia el caso de entrada ejemplo:

```
5 3
1 2 100
2 5 100
3 4 100
```

Dice que tenemos 5 barriles inicialmente vacíos:

```
0 0 0 0 0
```

Después de la primera operación que consta de añadir 100 litros de cerveza a los barriles que van desde 1 hasta 2 inclusive, tenemos:

```
100 100 0 0 0
```

Para la segunda operación:

```
100 200 100 100 100
```

Por último la tercera operación

```
100 200 200 200 100
```

Por lo que tenemos un total de 800 litros litros litros de cerveza distribuidos en 5 barriles, lo que nos da un promedio de $\frac{800}{5} = 160$.

Dicho lo anterior la primer solución que se nos puede ocurrir es simular las operaciones conservando un vector de tamaño n donde iremos almacenando las sumas sobre cada barril y al final obtenemos el promedio de todo lo dado.

El problema con la solución anterior es que su peor caso se da cuando para cada operación (m en total) nos piden llenar los barriles con índice desde 1 hasta n, lo cual tiene un complejidad de $O(nm)$ que es lo suficientemente buena para el caso fácil, pero que jamás pasará en tiempo el caso difícil.

Para el caso difícil basta ver que en cada operación el total de cerveza en todos los barriles aumentará la cantidad en $(b - a + 1) * k$ litros haciendo el cálculo de cada operación constante en tiempo, esto es por que se aumentarán k litros por cada barril en el rango de a hasta b, por lo tanto, el tiempo total en el peor caso será $O(m)$ (más que suficiente para pasar ambos casos) y solo quedará por ver que el resultado puede ser muy grande y habrá que almacearlo en una variable entera de 64 bits (`long long`).

El siguiente código muesta una función que calcula cada caso el resultado incluyendo la lectura y salida de datos.

```
void solve()
{
	long long n, m;

	cin >> n >> m;

	long long total = 0, a, b, k;

	for(int i=0; i<m; i++)
	{
		cin >> a >> b >> k;
		total += (b-a+1)*k;
	}

	cout << total / n << "\n";
}
```