Competitive Programming Workshop: Introducción

Giovanny Alfonso Chávez Ceniceros

Universidad Autónoma de Chihuahua
Facultad de Ingeniería

gchavezceniceros@acm.org

25 de marzo de 2019

Contenido

¿Qué es Programación Competitiva?

- Es un deporte mental en el cual los participantes resuelven un conjunto de problemas bien especificados a través de programas de computadora.
- Los estudiantes universitarios, principalmente de ingeniería y carreras asociadas a ciencias de la computación, participan en las competencias organizadas por ACM ICPC.

Todos los algoritmos implementados serán sujetos a las siguientes limitaciones:

Límite de tiempo

C	**C++**	**Java**	**Python**	**Haskell**	**Javascript**
1s	1s	1.5s	5s	2s	5s

Límite de memoria

512 Mb.

Comandos de compilación en C/C++

En Linux y MacOS, lo más probable es que tengas el compilador requerido. En Windows, puedes usar tu compilador favorito o instalar, por ejemplo, cygwin.

C (gcc 5.4.0). extensión: .c

```
gcc -pipe -O2 -std=c11 <filename> -lm
```

C++ (g++ 5.4.0). extensión: .cc, .cpp

```
g++ -pipe -O2 -std=c++14 <filename> -lm
```

Suma de dos números

Descripción

Calcular la suma de dos números de un sólo dígito.

Entrada

Dos enteros a y b en la misma línea (separados por un espacio).

Salida

La suma de a y b.

Restricciones

$0 \leq a, b \leq 9$.

Implementando el algoritmo en C++

```cpp
/**
 * @file    sumTwoDigits.cpp
 * @date    May 23, 2018
 * @brief   Code for the competitive programming workshop.
 */
#include <bits/stdc++.h>
/**
 * @brief   Main function.
 */
int main( void )
{
    int  a, b;
    std::cin >> a >> b;
    std::cout << a + b << std::endl;
    return 0;
}
```

Máximo producto de pares en una secuencia

	5	6	2	7	4
5		30	10	35	20
6	30		12	42	24
2	10	12		7	4
7	35	42	14		28
4	20	24	8	28	

Descripción

Dada una secuencia de enteros no negativos $a_1, ..., a_n$, calcular $\text{máx}\, a_i \cdot a_j$ donde $1 \leq i \neq j \leq n$.
Tenga en cuenta que i y j deben ser diferentes, aunque puede ser el caso de que $a_i = a_j$.

Máximo producto de pares en una secuencia

Entrada

La primer línea contiene un entero n. La siguiente línea contiene n enteros no negativos $a_1, ..., a_n$ (separados por espacios).

Salida

El producto por pares máximo.

Máximo producto de pares en una secuencia

Restricciones

$2 \leq n \leq 2 \cdot 10^5$
$1 \leq a_1, ..., a_n \leq 2 \cdot 10^5$.

Casos de prueba

Entrada	**Salida**
3	
1 2 3	6

Construyendo el código

```cpp
using vi = std::vector<int>;

int main( void )
{
    vi M; int n, a;
    std::cin >> n;
    M.reserve(n);

    while(n--) {
        std::cin  >> a;
        M.push_back(a);
    }

    int product = maxPairwiseSorting(M);
    std::cout <<  product  << std::endl;
    return 0;
}
```

Implementando un algoritmo ingenuo:

```cpp
int MaxPairwiseProduct( const vi &M )
{
    int product = 0, n = M.size( );

    for(int i = 0; i < M.size(); ++i)
        for(int j = i + 1; j < M.size(); ++j)
            product = std::max(product, M[i]*M[j]);
    return product;
}
```

Implementando un algoritmo optimizado:

```cpp
int maxPairwiseSorting( vi &M )
{
    unsigned int n = M.size( ) - 1;
    std::sort(M.begin( ), M.end( ));
    return M[n]*M[n - 1];
}
```

¿Como saber si un algoritmo es mejor que otro?

- El tiempo de ejecución de un algoritmo depende de cuánto tiempo le tome a una computadora ejecutar las líneas de código, y eso depende de la velocidad de la computadora, el lenguaje de programación y el compilador que traduce el programa, entre otros factores.
- Debemos enfocarnos en qué tan rápido crece un algoritmo respecto al tamaño de la entrada. A esto lo llamamos la tasa de crecimiento del tiempo de ejecución.

Análisis asintótico

Para analizar el tiempo de ejecución de un algoritmo se emplean las siguientes notaciones:

Notación Big-O

$$\exists k, n_0 > 0 : \forall n \geq n_0, T(n) \leq kf(n) \Longrightarrow T(n) = O(f(n))$$

Notación Ω

$$\exists k, n_0 > 0 : \forall n \geq n_0, T(n) \geq kf(n) \Longrightarrow T(n) = \Omega(f(n))$$

Notación Θ

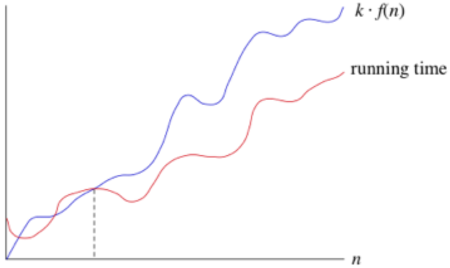
$$\exists k, n_0 > 0 : \forall n \geq n_0, kf(n) \leq T(n) \leq kf(n) \Longrightarrow T(n) = \Theta(f(n))$$

Notación Big-O

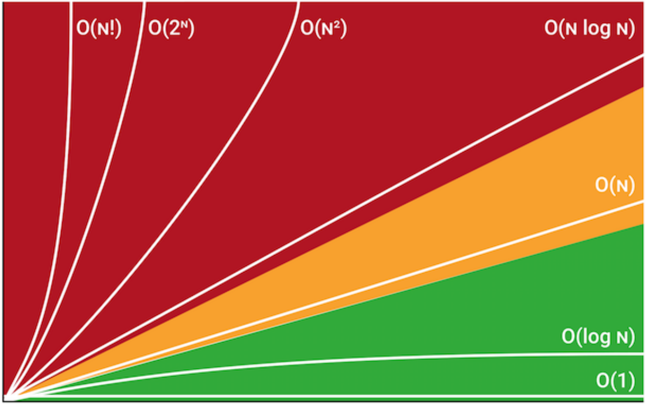
- Usamos estas notaciones para acotar de manera asintótica el crecimiento de un tiempo de ejecución que esté dentro de factores constantes por arriba y por abajo.
- A veces queremos acotar solo por arriba, por ello usamos la notación Big-O justo para estas ocasiones.

Notación Big-O

Si un tiempo de ejecución $T(n)$ es $O(f(n))$, entonces para n suficientemente grande, $T(n)$ es a lo más $kf(n)$ para alguna constante k.



Crecimiento asintótico



Crecimiento asintótico

Big-O	**Nombre**	**Ejemplo**
$O(1)$	constante	un número es par o impar
$O(lgn)$	logarítmico	búsqueda binaria
$O(n)$	lineal	máximo de un arreglo desordenado
$O(nlgn)$	*linearítmico*	ordenamiento con *mergesort/quicksort*
$O(n^2)$	cuadrático	operaciones con matrices
$O(n^k)$	polinomial	multiplicación de matrices
$O(2^n)$	exponencial	subconjuntos de un conjunto
$O(!n)$	factorial	permutaciones

Máximo Común Divisor

$$
\begin{aligned}
&\mathrm{GCD}(1344, 217) \\
=&\mathrm{GCD}(217, 42) \\
=&\mathrm{GCD}(42, 7) \\
=&\mathrm{GCD}(7, 0) \\
=&7
\end{aligned}
$$

Descripción

El máximo común divisor $GCD(a, b)$ de dos enteros no negativos a y b es el mayor entero d que divide a y b. El objetivo es implementar el algoritmo euclidiano para el cálculo del máximo común divisor.

Máximo Común Divisor

Entrada

Dos enteros a y b en la misma línea (separados por un espacio).

Salida

El $GCD(a, b)$.

Máximo Común Divisor

Restricciones

$1 \leq a, b \leq 2 \cdot 10^9$.

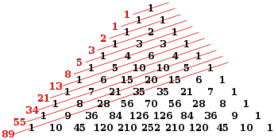
Casos de prueba

Entrada	**Salida**
18 35	1
28851538 1183019	17657

Máximo Común Divisor

```
/**
 * @brief  Euclidean Algorithm.
 */
unsigned int GCD( unsigned long a, unsigned long b )
{
    return (b == 0) ? a : GCD(b, a%b);
}
/**
 * @brief  Main function.
 */
int main( void )
{
    unsigned long a, b;
    std::cin >> a >> b;
    std::cout << GCD(a,b) << std::endl;
    return 0;
}
```

Números de Fibonacci



En matemáticas, la sucesión o serie de Fibonacci hace referencia a la secuencia ordenada de números descrita por Leonardo de Pisa, matemático italiano del siglo XIII. La sucesión se define como $F_0 = 0$, $F_1 = 1$, y $F_i = F_{i-1} + F_{i-2}$ para $i \geq 2$. El objetivo es implementar un algoritmo eficiente para el cálculo de números de Fibonacci.

Números de Fibonacci

Podríamos pensar en una implementación recursiva para calcular los números de Fibonacci:

```
unsigned int fibo( unsigned long n )
{
    if(n == 0 || n == 1)
        return n;
    else
        return fibo(n - 1) + fibo(n - 2);
}
```

Pero pronto veríamos que el cálculo de, por ejemplo, F_{40} ya lleva un tiempo considerable.

Fibonacci simple

Descripción

Dado un entero n, calcular el n-ésimo número de Fibonacci F_n.

Entrada

Un entero n.

Salida

El n-ésimo número de Fibonacci F_n.

Fibonacci simple

Restricciones

$0 \leq n \leq 45$.

Casos de prueba

Entrada	**Salida**
10	55
45	1134903170

Algoritmo de Fibonacci optimizado

```
/**
 * @brief  Botton-up approach for Fibonacci numbers.
 */
unsigned int memoFibo( unsigned long n )
{
    unsigned long memo[n + 1];

    memo[0] = 0;
    memo[1] = 1;

    for(int i = 2; i <= n; ++i)
        memo[i] = (memo[i - 1] + memo[i - 2]);

    return memo[n];
}
```

Último dígito del n-ésimo número de Fibonacci

Descripción

Dado un entero n, encuentre el último dígito del n-ésimo número de Fibonacci F_n (es decir, $F_n \ mod \ 10$).

Entrada

Un entero n.

Salida

$F_n \ mod \ 10$.

Restricciones

$0 \leq n \leq 10^7$.

Último dígito del n-ésimo número de Fibonacci

Casos de prueba

Entrada	**Salida**
200	5

$$F_{200} = 280571172992510140037611932413038677189525.$$

Entrada	**Salida**
331	9

$$F_{331} = 668996615388005031531000081241745415306766517246774551964595292186469.$$

Referencias

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009)
Introduction to algorithms

Skiena, S. S. (2003)
Programing Challenges. The Programming Contest Training Manual

Dynamic Programming (Fibonacci)
https://www.cs.usfca.edu/~galles/visualization/DPFib.html