Análisis y Diseño de Algoritmos

Introducción: El papel de los Algoritmos en Computación

DR. JOSÉ MARTÍNEZ CARRANZA

CIENCIAS COMPUTACIONALES
INAOE

Notas sobre derechos de autor

- Varias de las imágenes o videos utilizados en las presentaciones de este curso son tomadas de Internet, de sitios públicos como Wikipedia, Youtube, Google, etc. Los derechos, si existiesen, permanecen con su autor o autores.

- El uso de éstas imagenes o videos es meramente para fines ilustrativos sin propósito de lucro, publicidad o comercialización.

Temario

1. Introducción
2. Notación Asintótica
3. Recurrencias
4. Ordenamiento
5. Algoritmos Voraces
6. Divide y Vencerás
7. Programación Dinámica
8. Algoritmos Elementales para Grafos
9. Árboles
10. Teoría de la Complejidad

Evaluación

- 2 Exámenes, 2 x 20% = 40%
- 1 Examen Final = 30%
- Tareas = 30%

- Puntos Extras (Participacioń en clase) = 10%

- Página del curso:
  - http://ccc.inaoep.mx/~carranza/alg.html

- Contacto:
  - carranza@inaoep.mx
  - Asunto: **ALG2020**
  -
- Ayudante de Curso:
  - **Arturo Cocoma Ortega**
  - arturo.cocoma@gmail.com

Calendario

		Semana		Sesión	Tema
Agosto	Martes	1	18/08/2018	1	Introducción
	Jueves	1	20/08/2018	2	Notación Asintótica
	Martes	2	25/08/2018	3	Notación Asintótica
	Jueves	2	27/08/2018		Ejercicios
	Martes	3	01/09/2018	4	Recurrencias
Septiembre	Jueves	3	03/09/2018	5	Recurrencias
	Martes	4	08/09/2018		Ejercicios
	Jueves	4	10/09/2018	6	Ordenamiento
	Martes	5	15/09/2018	7	Ordenamiento
	Jueves	5	17/09/2018		Ejercicios
	Martes	6	22/09/2018	8	Algoritmos Voraces
	Jueves	6	24/09/2018	9	Algoritmos Voraces
Octubre	Martes	7	29/09/2018		Ejercicios
	Jueves	7	01/10/2018		Examen 1
	Martes	8	06/10/2018	10	Divide y Vencerás
	Jueves	8	08/10/2018	11	Divide y Vencerás
	Martes	9	13/10/2018		Ejercicios
	Jueves	9	15/10/2018	12	Programación Dinámica
	Martes	10	20/10/2018	13	Programación Dinámica
	Jueves	10	22/10/2018		Ejercicios
	Martes	11	27/10/2018	14	Grafos
	Jueves	11	29/10/2018	15	Grafos
Noviembre	Martes	12	03/11/2018		Ejercicios
	Jueves	12	05/11/2018	16	Árboles
	Martes	13	10/11/2018	17	Árboles
	Jueves	13	12/11/2018		Ejercicios
	Martes	14	17/11/2018		Examen 2
	Jueves	14	19/11/2018	18	Teoría de la Complejidad
	Martes	15	24/11/2018	19	Teoría de la Complejidad
	Jueves	15	26/11/2018		Ejercicios
Dic	Martes	16	01/12/2018		Examen Final

Calificacion

	Puntos
Examen 1	2
Examen 2	2
Examen Final	3
Tareas	3

TOTAL

Puntos Extra

Participación	1

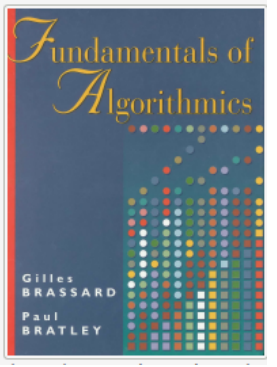
Reglas

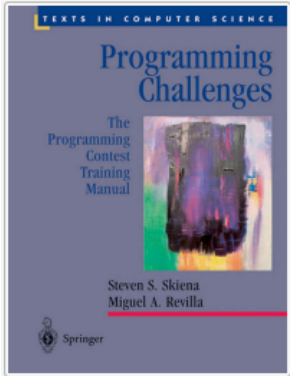
Respetar la participación de otros

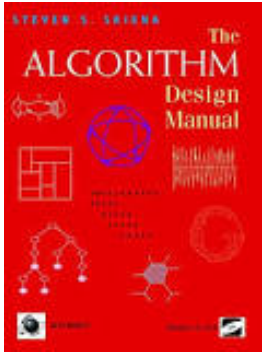
Usar el chat para hacer preguntas

Libros de Apoyo

- Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1988). Estructuras de datos y algoritmos (Vol. 1). Addison-Wesley Iberoamericana.
- Gilles Brassard, Paul Bratley, "Fundamentals of Algorithmics", Prentice-Hall, Inc.,1996, ISBN 0-13-335068-1.
- Steven S. Skiena, "The Algorithm Design Manual", Springer-Verlag, 2008, ISBN 978-1848000698.
- Skiena, S. S., & Revilla, M. A. (2003). Programming challenges: The programming contest training manual. ACM SIGACT News, 34(3), 68-74.







Algoritmos

- Informalmente
  - Cualquier procedimiento computacional bien definido
    - Valores de entrada
    - Valores de salida
  - Secuencia computacional de pasos que transforman una entrada en una salida
  - Herramienta para resolver un problema computacional bien especificado

Problema de Ordenamiento

- Entrada: secuencia de números $(a_1, a_2, \ldots, a_n)$
- Salida: permutación (reordenamiento) $(a'_1, a'_2, \ldots, a'_n)$ de la secuencia de entrada tal que $a'_1 <= a'_2 <= \ldots <= a'_n$.
- Para (31, 41, 59, 26, 41, 58), la salida es
  - (26, 31, 41, 41, 58, 59)
- A la secuencia de entrada se le llama Instancia
- Una instancia del problema es una entrada
  - Satisface las restricciones del problema
  - Necesaria para calcular la solución al problema

Problema de Ordenamiento

- Ordenamiento es una operación fundamental en computación
  - Utilizado como paso intermedio
  - Hay muchos algoritmos de ordenamiento buenos
  - ¿Cuál es el mejor para una aplicación dada?
    - Depende de:
      - ¿Número de elementos a ordenar?
      - ¿Algunos elementos ya están ordenados?
      - ¿Restricciones en los valores de los elementos?
      - ¿Qué tipo de almacenamiento usamos?
        - » Memoria principal, disco, cintas
- Vamos a trabajar con varios algoritmos de ordenamiento

Algoritmo Correcto

- Se dice que un algoritmo es correcto si para toda instancia de entrada, se detiene al obtener la salida correcta
- Decimos que un algoritmo correcto resuelve el problema computacional dado
- Un algoritmo no-correcto puede no detenerse para algunas instancias de entrada o detenerse con una respuesta incorrecta
  - Los algoritmos incorrectos pueden ser útiles, si podemos controlar su tasa de error
  - Sin embargo, nos concentraremos sólo en algoritmos correctos

Especificación de un Algoritmo

- Inglés, español
- Como programa computacional
- Pseudocódigo
- Sólo una restricción
  - La especificación debe proveer una descripción precisa del procedimiento computacional a seguir

¿Qué Clases de Problemas Resuelven los Algoritmos?

- Muchos más que el ordenamiento
  - El proyecto del Genoma Humano, identificar 100,000 genes en el ADN humano, la secuencia de 3 billones de pares de bases químicas que componen el ADN…
  - Algoritmos de búsqueda para internet, con grandes volúmenes de datos
  - Comercio electrónico (criptografía, firmas digitales)
  - Industrias

Características Comunes de Algoritmos

- Muchas soluciones candidatas, la mayoría no nos interesan, encontrar la que queremos puede ser difícil

- Tienen aplicaciones prácticas, ruta más corta

Estructuras de Datos

- Una estructura de datos es una manera de almacenar y organizar datos para facilitar su acceso y modificación

- Una estructura de datos no trabaja bien para todo propósito, tienen ventajas y limitaciones

Técnica

- Puede que el algoritmo que necesitamos no esté publicado

- Necesitamos una técnica para diseñar y analizar algoritmos para que podamos desarrollar nuevos
  - Demostrar que producen la respuesta correcta
  - Entender su eficiencia

Problemas Duros

- Hay algunos problemas para los que no se conoce una solución eficiente
  - Algoritmos NP-completos
  - No se ha encontrado un algoritmo eficiente para un problema NP-completo
    - Tampoco se sabe si existen (o no) algoritmos eficientes para problemas NP-completos
  - Propiedad de algoritmos NP-completos
    - Si existe un algoritmo eficiente para alguno de ellos, entonces existen algoritmos eficientes para todos ellos
  - Varios problemas NP-completos son similares (no idénticos) a problemas para los que sabemos que hay algoritmos eficientes
    - Cambio pequeño en la definición del problema cause un gran cambio en la eficiencia del mejor algoritmo conocido

Problemas NP-Completos

- Es bueno saber sobre problemas NP-completos
  - Aparecen frecuentemente en aplicaciones reales
  - Podemos invertir mucho tiempo en ellos
  - Si demostramos que el algoritmo es NP-completo
    - Invertir tiempo en desarrollar un algoritmo que obtenga una buena respuesta (no la mejor posible)

Algoritmos como una Tecnología

- ¿Qué pasaría si las computadoras fueran infinitamente rápidas y la memoria fuera gratis?
- Pero como no son infinitamente rápidas y la memoria no es gratuita
  - El tiempo de cómputo y el espacio en memoria son recursos limitados
    - Hay que utilizarlos inteligentemente
    - Algoritmos eficientes en tiempo y espacio

Eficiencia

- Algoritmos que resuelven el mismo problema frecuentemente tienen una eficiencia diferente
  - Más significativas que las diferencias por HW o SW
- Algoritmos de ordenamiento
  - **InsertionSort** $c_1 n^2$ para ordenar n elementos, c_1 es constante no dependiente de n
  - **MergeSort** $c_2 n \lg n$, donde $\lg n$ es $\log_2 n$ y c_2 es otra constante que no depende de n
  - $c_1 < c_2$
    - factores constantes no tan importantes como n
    - n mucho mayor a $\lg n$
    - InsertionSort más rápido que MergeSort para entradas pequeñas
    - Hay un punto (tamaño de n) en el que MergeSort ya es más rápido que InsertionSort

Ejemplo de Eficiencia

- Computadora A más rápida que Computadora B
  - A, 100 millones de inst. x seg. → 10^8 ins/sec
  - B, 1 millón de inst. x seg. → 10^6 ins/sec
- Implementamos InsertionSort en A
  - $2n^2$ ($c_1 = 2$)
- Implementamos MergeSort en B
  - $50n\lg n$ ($c_2 = 50$)
- Ordenar 1,000,000 de números (n = 10^6)

Ejemplo de Eficiencia

$$A \rightarrow \frac{2(10^6)^2\, inst}{10^8\, inst/\text{sec}} = 20{,}000 \text{ seconds} \approx 5.56 \text{ hours},$$

$$B \rightarrow \frac{50 \cdot 10^6 \lg 10^6\, inst}{10^6\, inst/\text{sec}} \approx 1{,}000 \text{ seconds} \approx 16.67 \text{ minutes}.$$

- Con 10,000,000 de números:
  - B (20 min) es 20 veces más rápida que A (2.3 días) para ordenar los números
    - Ventaja de la eficiencia de MergeSort

Análisis de Algoritmos

- Predecir los recursos que requiere el algoritmo
  - Memoria, tiempo de cómputo
- Modelo de la tecnología de implementación
  - 1 procesador
  - Modelo RAM
    - Una instrucción tras otra, no operaciones concurrentes
- Requerimos herramientas matemáticas

Ejemplos

3.8.3 Common Permutation

PC/UVa IDs: 110303/10252, **Popularity:** A, **Success rate:** average **Level:** 1

Given two strings a and b, print the longest string x of letters such that there is a permutation of x that is a subsequence of a and there is a permutation of x that is a subsequence of b.

Input

The input file contains several cases, each case consisting of two consecutive lines. This means that lines 1 and 2 are a test case, lines 3 and 4 are another test case, and so on. Each line contains one string of lowercase characters, with first line of a pair denoting a and the second denoting b. Each string consists of at most 1,000 characters.

Output

For each set of input, output a line containing x. If several x satisfy the criteria above, choose the first one in alphabetical order.

Sample Input

```
pretty
women
walking
down
the
street
```

Sample Output

```
e
nw
et
```

3.8.7 Doublets

PC/UVa IDs: 110307/10150, **Popularity:** C, **Success rate:** average **Level:** 3

A *doublet* is a pair of words that differ in exactly one letter; for example, "booster" and "rooster" or "rooster" and "roaster" or "roaster" and "roasted".

You are given a dictionary of up to 25,143 lowercase words, not exceeding 16 letters each. You are then given a number of pairs of words. For each pair of words, find the shortest sequence of words that begins with the first word and ends with the second, such that each pair of adjacent words is a doublet. For example, if you were given the input pair "booster" and "roasted", a possible solution would be ("booster," "rooster," "roaster," "roasted"), provided that these words are all in the dictionary.

Input

The input file contains the dictionary followed by a number of word pairs. The dictionary consists of a number of words, one per line, and is terminated by an empty line. The pairs of input words follow; each pair of words occurs on a line separated by a space.

Output

For each input pair, print a set of lines starting with the first word and ending with the last. Each pair of adjacent lines must be a doublet.

If there are several minimal solutions, any one will do. If there is no solution, print a line: "`No solution.`" Leave a blank line between cases.

Sample Input

```
booster
rooster
roaster
coasted
roasted
coastal
postal

booster roasted
coastal postal
```

Sample Output

```
booster
rooster
roaster
roasted

No solution.
```

5.9.2 *Reverse and Add*

PC/UVa IDs: 110502/10018, **Popularity:** A, **Success rate:** low **Level:** 1

The *reverse and add* function starts with a number, reverses its digits, and adds the reverse to the original. If the sum is not a palindrome (meaning it does not give the same number read from left to right and right to left), we repeat this procedure until it does.

For example, if we start with 195 as the initial number, we get 9,339 as the resulting palindrome after the fourth addition:

$$\begin{array}{rrrr} 195 & 786 & 1{,}473 & 5{,}214 \\ 591 & 687 & 3{,}741 & 4{,}125 \\ +\text{---} & +\text{---} & +\text{---} & +\text{---} \\ 786 & 1{,}473 & 5{,}214 & 9{,}339 \end{array}$$

This method leads to palindromes in a few steps for almost all of the integers. But there are interesting exceptions. 196 is the first number for which no palindrome has been found. It has never been proven, however, that no such palindrome exists.

You must write a program that takes a given number and gives the resulting palindrome (if one exists) and the number of iterations/additions it took to find it.

You may assume that all the numbers used as test data will terminate in an answer with less than 1,000 iterations (additions), and yield a palindrome that is not greater than 4,294,967,295.

Input

The first line will contain an integer N ($0 < N \leq 100$), giving the number of test cases, while the next N lines each contain a single integer P whose palindrome you are to compute.

Output

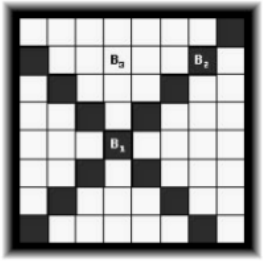
For each of the N integers, print a line giving the minimum number of iterations to find the palindrome, a single space, and then the resulting palindrome itself.

Sample Input	*Sample Output*
3 195 265 750	4 9339 5 45254 3 6666

8.6.1 *Little Bishops*

PC/UVa IDs: 110801/861, **Popularity:** C, **Success rate:** high **Level:** 2

A bishop is a piece used in the game of chess which can only move diagonally from its current position. Two bishops attack each other if one is on the path of the other. In the figure below, the dark squares represent the reachable locations for bishop B_1 from its current position. Bishops B_1 and B_2 are in attacking position, while B_1 and B_3 are not. Bishops B_2 and B_3 are also in non-attacking position.



Given two numbers n and k, determine the number of ways one can put k bishops on an $n \times n$ chessboard so that no two of them are in attacking positions.

Input

The input file may contain multiple test cases. Each test case occupies a single line in the input file and contains two integers $n(1 \leq n \leq 8)$ and $k(0 \leq k \leq n^2)$.

A test case containing two zeros terminates the input.

Output

For each test case, print a line containing the total number of ways one can put the given number of bishops on a chessboard of the given size so that no two of them lie in attacking positions. You may safely assume that this number will be less than 10^{15}.

Sample Input	*Sample Output*
8 6	5599888
4 4	260
0 0	

Dado un algoritmo se quiere saber:

- ¿Complejidad computacional?
- ¿Es óptimo? (¿hay otro algoritmo más efectivo?)
- ¿Siempre devuelve una respuesta?
- ¿Mejor y peor caso de prueba?
- ¿Que estructuras de datos se requieren?
- ¿Se puede clasificar en alguno de los tipos mencionados en el curso?

Eficiencia, Eficacia, Efectividad

- Eficiencia: “Capacidad para lograr un fin empleando los mejores medios posibles”.
- Eficacia: “Capacidad de lograr el efecto que se desea o se espera”.
- Efectividad: “Cuantificación del logro de la meta” o “Capacidad de lograr el efecto que se desea”.

¿Preguntas?

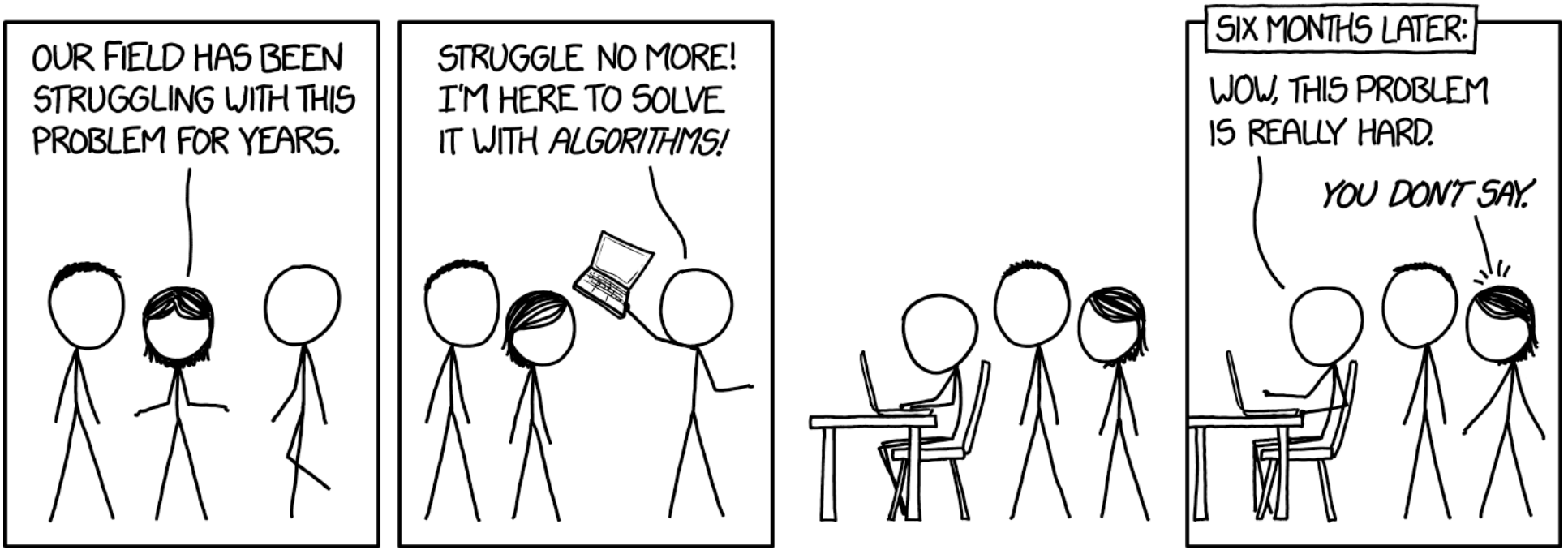


Imagen tomada de Internet