

# PROGRAMACIÓN III

## INFORME DE LA PRÁCTICA 2 (FUERZA BRUTA/GREEDY)

---

### 1. Nombre de los participantes:

- Marcos Jesús Santana Pérez
- Chetani Mesa Guzmán
- Ían Marrero Martín

### 2. Descripción del problema a resolver:

Por medio de combinaciones sin repetición, donde si importa el orden y los elementos son cogidos de  $n$  en  $n$ , queremos obtener el mayor valor generado de entre todas las permutaciones.

### 3. Lenguajes de programación escogidos:

- JAVA v1.8.0\_191
- PYTHON v2.7
- C

# Estrategia de Fuerza Bruta

**IMPORTANTE: PARA SABER COMO COMPILAR Y EJECUTAR EN CADA LENGUAJE, VER APARTADO DE INFORMACIÓN ADICIONAL AL FINAL DEL DOCUMENTO.**

## 4. Nombre de los ficheros que contienen el algoritmo de fuerza bruta implementado

- JAVA => /Pr3/Java/src/Max/Main.java
- PYTHON => /Pr3/Python/main.py
- C => /Pr3/C/main.c

## 5. Copia de pantalla del fragmento de código que implementa el algoritmo de fuerza bruta (en los 3 lenguajes).

- JAVA

```
public long BruteForce (String [] elementos){
    int[] indices; // orden a mostrar elementos[]
    Permutator p = new Permutator (elementos.length);
    long maximo = 0; // numero mas grande generado
    long n = 0;
    while (p.hasMore ()) {
        indices = p.getNext (); // obtenemos el orden que da lugar a la nueva
// permutacion
        n = ToLong(elementos, indices); // permutacion actual
        if(n > maximo){ // comprobamos si el nuevo n es mayor que maximo
            maximo = n;
        }
    }
    return maximo;
}
```

- PYTHON

```
def BruteForce (linea):
    lst = etc.lstStringToInt(linea)
    # maximo contiene el valor mas grande encontrado hasta el momento
    maximo = 0
    for permutacion in itertools.permutations(lst):
        # Construimos el numero a partir de la lista permutada
        n = etc.lstToInt(permutacion)
        # Comprobamos si el nuevo n es mayor que maximo
        if n > maximo:
            maximo = n
    return maximo
```

- C

```
long long BruteForce (char **elementos,int tamElementos,int * nDigElementos) {
    int *indices = malloc(tamElementos*sizeof(int));
    Permutator(tamElementos);
    long long maximo = 0;
    long long valor = 0;
    while(hasMore()) {
        indices = getNext();
        valor = ToLong(elementos, indices, tamElementos, nDigElementos);
        if (valor > maximo) {
            maximo = valor;
        }
    }
    return maximo;
}
```

## 6. Nombre de los ficheros que contienen el iterador (sólo en los lenguajes en los que hayan tenido que implementar el iterador)

- JAVA =>/Pr3/Java/src/Itertools/Permutator.java
- C =>/Pr3/C/permutator.c

## 7. Explicación del iterador de permutaciones o combinaciones utilizado en cada lenguaje (en caso de haber utilizado un iterador disponible en el lenguaje, referenciar la página Web de documentación del iterador utilizado; en caso de haberlo programado, referenciar la página Web que describe el algoritmo implementado, o explicar el algoritmo implementado). - JAVA

El código del iterador fue desarrollado en base al código que se encuentra en la siguiente página web:

[http://people.rennes.inria.fr/Arnaud.Gotlieb/resources/java\\_exp/min/perm.htm](http://people.rennes.inria.fr/Arnaud.Gotlieb/resources/java_exp/min/perm.htm)

En nuestra adaptación nos deshicimos de los métodos que nunca se llegaban a utilizar, cambiamos los tipos "BigInteger" por "long", y pusimos comentarios que muestran que se pretende hacer en cada momento con mayor claridad.

Cabe decir que el algoritmo empleado para generar las permutaciones, fue descrito por Kenneth H. Rosen en su libro *"Discrete Mathematics and Its Applications, 2nd edition (NY: McGraw-Hill, 1991), pp. 282-284"*, y como sabemos que puede ser algo tedioso encontrar dicho libro, ofrecemos capturas de pantalla de las páginas del libro donde se habla del algoritmo.

set of employees is to generate all sets of 12 of these employees and check whether they have the desired skills. These examples show that it is often necessary to generate permutations and combinations to solve problems.

## Generating Permutations



Any set with  $n$  elements can be placed in one-to-one correspondence with the set  $\{1, 2, 3, \dots, n\}$ . We can list the permutations of any set of  $n$  elements by generating the permutations of the  $n$  smallest positive integers and then replacing these integers with the corresponding elements. Many different algorithms have been developed to generate the  $n!$  permutations of this set. We will describe one of these that is based on the **lexicographic** (or **dictionary**) **ordering** of the set of permutations of  $\{1, 2, 3, \dots, n\}$ . In this ordering, the permutation  $a_1 a_2 \dots a_n$  precedes the permutation  $b_1 b_2 \dots b_n$ , if for some  $k$ , with  $1 \leq k \leq n$ ,  $a_1 = b_1, a_2 = b_2, \dots, a_{k-1} = b_{k-1}$ , and  $a_k < b_k$ . In other words, a permutation of the set of the  $n$  smallest positive integers precedes (in lexicographic order) a second permutation if the number in this permutation in the first position where the two permutations disagree is smaller than the number in that position in the second permutation.

**EXAMPLE 1** The permutation 23415 of the set  $\{1, 2, 3, 4, 5\}$  precedes the permutation 23514, because these permutations agree in the first two positions, but the number in the third position in the first permutation, 4, is smaller than the number in the third position in the second permutation, 5. Similarly, the permutation 41532 precedes 52143. ◀

An algorithm for generating the permutations of  $\{1, 2, \dots, n\}$  can be based on a procedure that constructs the next permutation in lexicographic order following a given permutation  $a_1 a_2 \dots a_n$ . We will show how this can be done. First, suppose that  $a_{n-1} < a_n$ . Interchange  $a_{n-1}$  and  $a_n$  to obtain a larger permutation. No other permutation is both larger than the original permutation and smaller than the permutation obtained by interchanging  $a_{n-1}$  and  $a_n$ . For instance, the next larger permutation after 234156 is 234165. On the other hand, if  $a_{n-1} > a_n$ , then a larger permutation cannot be obtained by interchanging these last two terms in the permutation. Look at the last three integers in the permutation. If  $a_{n-2} < a_{n-1}$ , then the last three integers in the permutation can be rearranged to obtain the next largest permutation. Put the smaller of the two integers  $a_{n-1}$  and  $a_n$  that is greater than  $a_{n-2}$  in position  $n - 2$ . Then, place the remaining integer and  $a_{n-2}$  into the last two positions in increasing order. For instance, the next larger permutation after 234165 is 234516.



On the other hand, if  $a_{n-2} > a_{n-1}$  (and  $a_{n-1} > a_n$ ), then a larger permutation cannot be obtained by permuting the last three terms in the permutation. Based on these observations, a general method can be described for producing the next larger permutation in increasing order following a given permutation  $a_1 a_2 \dots a_n$ . First, find the integers  $a_j$  and  $a_{j+1}$  with  $a_j < a_{j+1}$  and

$$a_{j+1} > a_{j+2} > \dots > a_n,$$

that is, the last pair of adjacent integers in the permutation where the first integer in the pair is smaller than the second. Then, the next larger permutation in lexicographic order is obtained by putting in the  $j$ th position the least integer among  $a_{j+1}, a_{j+2}, \dots$ , and  $a_n$  that is greater than  $a_j$  and listing in increasing order the rest of the integers  $a_j, a_{j+1}, \dots, a_n$  in positions  $j + 1$  to  $n$ . It is easy to see that there is no other permutation larger than the permutation  $a_1 a_2 \dots a_n$  but smaller than the new permutation produced. (The verification of this fact is left as an exercise for the reader.)

**EXAMPLE 2** What is the next permutation in lexicographic order after 362541?

**Solution:** The last pair of integers  $a_j$  and  $a_{j+1}$  where  $a_j < a_{j+1}$  is  $a_3 = 2$  and  $a_4 = 5$ . The least integer to the right of 2 that is greater than 2 in the permutation is  $a_5 = 4$ . Hence, 4 is placed in the third position. Then the integers 2, 5, and 1 are placed in order in the last three positions, giving 125 as the last three positions of the permutation. Hence, the next permutation is 364125. ◀

To produce the  $n!$  permutations of the integers  $1, 2, 3, \dots, n$ , begin with the smallest permutation in lexicographic order, namely,  $123 \dots n$ , and successively apply the procedure described for producing the next larger permutation of  $n! - 1$  times. This yields all the permutations of the  $n$  smallest integers in lexicographic order.

**EXAMPLE 3** Generate the permutations of the integers 1, 2, 3 in lexicographic order.

**Solution:** Begin with 123. The next permutation is obtained by interchanging 3 and 2 to obtain 132. Next, because  $3 > 2$  and  $1 < 3$ , permute the three integers in 132. Put the smaller of 3 and 2 in the first position, and then put 1 and 3 in increasing order in positions 2 and 3 to obtain 213. This is followed by 231, obtained by interchanging 1 and 3, because  $1 < 3$ . The next larger permutation has 3 in the first position, followed by 1 and 2 in increasing order, namely, 312. Finally, interchange 1 and 2 to obtain the last permutation, 321. We have generated the permutations of 1, 2, 3 in lexicographic order. They are 123, 132, 213, 231, 312, and 321. ◀

Algorithm 1 displays the procedure for finding the next permutation in lexicographic order after a permutation that is not  $n \ n - 1 \ n - 2 \ \dots \ 2 \ 1$ , which is the largest permutation.

**ALGORITHM 1** Generating the Next Permutation in Lexicographic Order.

```

procedure next permutation( $a_1 a_2 \dots a_n$ : permutation of
     $\{1, 2, \dots, n\}$  not equal to  $n \ n - 1 \ \dots \ 2 \ 1$ )
     $j := n - 1$ 
    while  $a_j > a_{j+1}$ 
         $j := j - 1$ 
    { $j$  is the largest subscript with  $a_j < a_{j+1}$ }
     $k := n$ 
    while  $a_j > a_k$ 
         $k := k - 1$ 
    { $a_k$  is the smallest integer greater than  $a_j$  to the right of  $a_j$ }
    interchange  $a_j$  and  $a_k$ 
     $r := n$ 
     $s := j + 1$ 
    while  $r > s$ 
        interchange  $a_r$  and  $a_s$ 
         $r := r - 1$ 
         $s := s + 1$ 
    {this puts the tail end of the permutation after the  $j$ th position in increasing order}
    { $a_1 a_2 \dots a_n$  is now the next permutation}

```

- PYTHON

El itertools es muy parecido al que se emplea en Java y C.

Se basa en una lista que contiene los índices que referencian al array pertinente, la idea es ir moviendo los **índices** de derecha a izquierda, además de ayudarse de una lista llamada **cycles** que cuenta las veces que se ha devuelto el elemento en esa posición. A continuación se verá mejor:

```
pool = tuple(iterable)
n = len(pool)
r = n if r is None else r
if r > n:
    return
indices = range(n)
cycles = range(n, n-r, -1)
```

La primera parte del algoritmo se encarga de preparar el terreno, ya que hace una copia de la lista pasada, coge la longitud de dicha lista, y en caso de necesitar **r**, la crea y guarda. Una vez hecho esto se crea **índices**, que es un rango que va desde [0, n], ambos inclusive. También se crea **cycles**, que va a contener el rango que va de **n** a **r**.

La forma de devolver una permutación es la siguiente:

```
yield tuple(pool[i] for i in indices[:r])
```

**yield** hace lo mismo que un **return** pero con la capacidad de que si se llama otra vez a la función, esta continua su ejecución desde la instrucción siguiente al **yield**.

El **while** que sigue funciona como un bucle infinito que se encarga de generar el resto de permutaciones:

```
while n:
    for i in reversed(range(r)):
        cycles[i] -= 1
        if cycles[i] == 0:
            indices[i:] = indices[i+1:] + indices[i:i+1]
            cycles[i] = n - i
        else:
            j = cycles[i]
            indices[i], indices[-j] = indices[-j], indices[i]
            yield tuple(pool[i] for i in indices[:r])
            break
    else:
        return
```

Dentro del **while** se recorre de derecha a izquierda la lista **cycles** que va a "contar" las veces que se han usado los índices para verlo con claridad vamos a separar el **if** y el **else** del for:

Nota: Tener en cuenta que antes que nada se decrementa el contador que contiene **cycles** en la posición **i**:

```
cycles[i] -= 1
```

Ahora veamos ese **if** de cerca:

```
if cycles[i] == 0:
    indices[i:] = indices[i+1:] + indices[i:i+1]
    cycles[i] = n - i
```

Lo que ocurrirá cuando **`cycles[i] == 0`**, es que lo que había en la posición **`i`** se irá al final de la lista y lo que había de **`i`** a la derecha, pues se moverá una posición a la izquierda. Ej: si tuviéramos la lista [4, 5, 6, 7] y quisiéramos hacer esta operación de desplazamiento masivo en la posición `i = 1`, veríamos lo siguiente:

[4, 5, 6, 7] >> [4, 6, 7, 5]

Además de esta migración de elementos también se reinicia el valor inicial de **`cycles[i]`**.

Bien ahora vendría lo que ocurrirá cuando **`cycles[i]`** tenga un valor distinto de 0, es decir que es una que no hemos hecho:

```
else:
    j = cycles[i]
    indices[i], indices[-j] = indices[-j], indices[i]
    yield tuple(pool[i] for i in indices[:r])
    break
```

Aquí se hace un swap entre la posición **`i`** y **`-j`**, este último no es apunte a alguna posición negativa, sino que python cuenta las posiciones negativas como si leyese la lista de derecha a izquierda, es decir, es la forma que tiene de leer las posiciones invertidas.

El ultimo **`else`** pertenece al **`for`** e indica que cuando se ejecute el **`for`** sin ejecutar ningún **`break`**, se hará el **`return`**.

- C

Mismo algoritmo implementado en java, pero adaptado a C.

## 8. Formato del fichero de entrada del programa

El fichero contendrá cada elemento de un conjunto separado por una coma. Cada conjunto de elementos se pondrá en una línea nueva.

Formato:

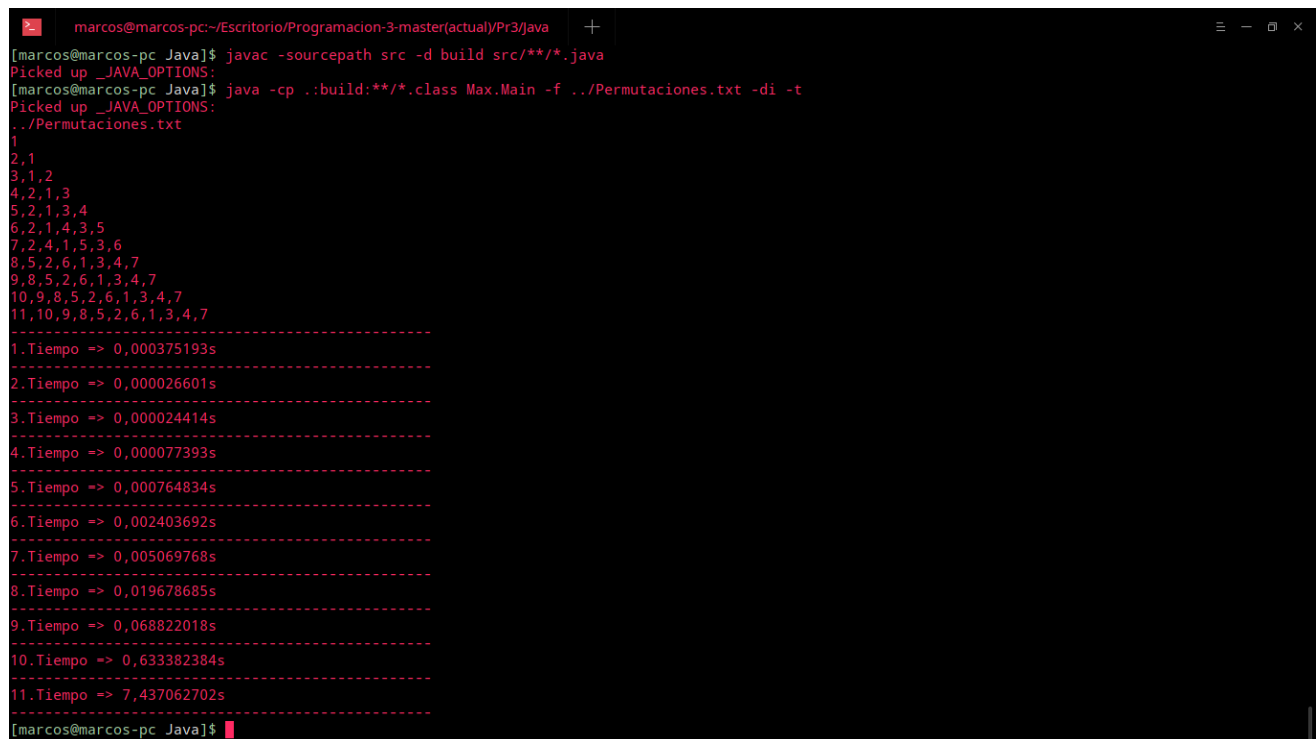
```
[elemento1],[elemento2]
[elemento1],[elemento2],[elemento3],[elemento4]
```

Nota:

Tanto en Java (que usa el tipo long) como en C (que usa tipo long long) para representar el valor que queremos obtener, tenemos la limitación de que solo se pueden representar  $2^{63}-1$  números, por encima del 0 (números de 19 cifras). Dicho esto, se recomienda no usar elementos cuya suma de cifras sea mayor a 18.

## 9. Copia de pantalla que muestre el uso del programa desde consola activando la opción que muestra el tiempo consumido en la ejecución del programa (en los tres lenguajes)

- JAVA



```
marcos@marcos-pc:~/Escritorio/Programacion-3-master(actual)/Pr3/Java$ javac -sourcepath src -d build src/**/*.java
Picked up _JAVA_OPTIONS:
marcos@marcos-pc:~/Escritorio/Programacion-3-master(actual)/Pr3/Java$ java -cp .:build/**/*.class Max.Main -f ../Permutaciones.txt -di -t
Picked up _JAVA_OPTIONS:
../Permutaciones.txt
1
2,1
3,1,2
4,2,1,3
5,2,1,3,4
6,2,1,4,3,5
7,2,4,1,5,3,6
8,5,2,6,1,3,4,7
9,8,5,2,6,1,3,4,7
10,9,8,5,2,6,1,3,4,7
11,10,9,8,5,2,6,1,3,4,7
-----
1.Tiempo => 0,000375193s
-----
2.Tiempo => 0,000026601s
-----
3.Tiempo => 0,000024414s
-----
4.Tiempo => 0,000077393s
-----
5.Tiempo => 0,000764834s
-----
6.Tiempo => 0,002403692s
-----
7.Tiempo => 0,005069768s
-----
8.Tiempo => 0,019678685s
-----
9.Tiempo => 0,068822018s
-----
10.Tiempo => 0,633382384s
-----
11.Tiempo => 7,437062702s
-----
marcos@marcos-pc:~/Escritorio/Programacion-3-master(actual)/Pr3/Java$
```



- PYTHON

```
marcos@marcos-pc:~/Escritorio/Programacion-3-master(actual)/Pr3/Python$ python2 main.py -di -f ../Permutaciones.txt -t
Nombre del fichero => ../Permutaciones.txt
-----
1
2,1
3,1,2
4,2,1,3
5,2,1,3,4
6,2,1,4,3,5
7,2,4,1,5,3,6
8,5,2,6,1,3,4,7
9,8,5,2,6,1,3,4,7
10,9,8,5,2,6,1,3,4,7
11,10,9,8,5,2,6,1,3,4,7
-----
Tiempo => 0.125155 s
-----
Tiempo => 0.125172 s
-----
Tiempo => 0.12519 s
-----
Tiempo => 0.125242 s
-----
Tiempo => 0.125471 s
-----
Tiempo => 0.126793 s
-----
Tiempo => 0.136384 s
-----
Tiempo => 0.229886 s
-----
Tiempo => 1.102968 s
-----
Tiempo => 10.633277 s
-----
Tiempo => 121.023162 s
-----
[marcos@marcos-pc Python]$
```

- C

```
marcos@marcos-pc:~/Escritorio/Programacion-3-master(actual)/Pr3/C$ gcc *.c
[marcos@marcos-pc C]$ ./a.out -f ../Permutaciones.txt -t --di
Ruta del fichero => ../Permutaciones.txt
1
2,1
3,1,2
4,2,1,3
5,2,1,3,4
6,2,1,4,3,5
7,2,4,1,5,3,6
8,5,2,6,1,3,4,7
9,8,5,2,6,1,3,4,7
10,9,8,5,2,6,1,3,4,7
11,10,9,8,5,2,6,1,3,4,7
-----
Tiempo => 0.000009s
-----
Tiempo => 0.000006s
-----
Tiempo => 0.000007s
-----
Tiempo => 0.000017s
-----
Tiempo => 0.000074s
-----
Tiempo => 0.000408s
-----
Tiempo => 0.003279s
-----
Tiempo => 0.014765s
-----
Tiempo => 0.069473s
-----
Tiempo => 0.658806s
-----
Tiempo => 7.869793s
-----
[marcos@marcos-pc C]$
```

## Información adicional

- JAVA

Instrucciones de compilación y ejecución con Java:

1. COMPILACIÓN:

**javac -sourcepath src -d build src/\*\*/\*.\*.java**

[ **-sourcepath src** ] selecciona src como carpeta donde se encuentra el código fuente.

[ **-d build src/\*\*/\*.\*.java** ] selecciona build como directorio donde se guardaran los archivos compilados y **src/\*\*/\*.\*.java** indica los archivos que se quieren compilar.

2. EJECUCIÓN: **java -cp .:build:\*\*/\*.\*.class Max.Main** con la instrucción anterior podemos ejecutar todos los archivos de la carpeta build, e indicarle que Max.Main es donde se encuentra el main de nuestro programa. A la derecha del Max.Main podemos indicarle diversos parámetros al programa.

3. PARÁMETROS ADMITIDOS:

- **-di:** (Debug Input) Permite mostrar el contenido del fichero pasado por parámetro.
- **-f file:** Indica al programa el input del que se van a obtener los conjuntos de elementos.
- **-do:** (Debug Output) Permite mostrar el valor máximo obtenido combinando los elementos del conjunto.
- **-t:** Permite mostrar el tiempo que se ha tardado en calcular el valor máximo.

- PYTHON

Instrucciones de ejecución con python:

1. INTERPRETACIÓN:

**python main.py** ó **python2 main.py** A la derecha del main.py van los parámetros del programa.

2. PARÁMETROS ADMITIDOS:

- **-di:** (Debug Input) Permite mostrar el contenido del fichero pasado por parámetro.
- **-f file:** Indica al programa el input del que se van a obtener los conjuntos de elementos.
- **-do:** (Debug Output) Permite mostrar el valor máximo obtenido combinando los elementos del conjunto.
- **-t:** Permite mostrar el tiempo que se ha tardado en calcular el valor máximo.
- **-h:** Despliega un menú de ayuda, que ofrece información acerca de los parámetros que se pueden utilizar.

- C

Instrucciones de compilación y ejecución con C:

1. COMPILACIÓN:

**gcc \*.c -o nombre\_del\_programa**

2. EJECUCIÓN: **./nombre\_del\_programa** A la derecha del main.c van los parámetros del programa.

3. PARÁMETROS ADMITIDOS:

- **--di:** (Debug Input) Permite mostrar el contenido del fichero pasado por parámetro.
- **-f file:** Indica al programa el input del que se van a obtener los conjuntos de elementos.
- **--do:** (Debug Output) Permite mostrar el valor máximo obtenido combinando los elementos del conjunto.
- **-t:** Permite mostrar el tiempo que se ha tardado en calcular el valor máximo.