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

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 =>

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 de elementos
    Permutator p = new Permutator (elementos.length);
    long maximo = 0;
    while (p.hasMore ()) {
        indices = p.getNext ();// obtenemos el orden que da lugar a la permutacion
        long n = ToLong(elementos, indices);
        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

```
```

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 =>

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

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.

- 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 cycle 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 indices 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, veriamos 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 linea 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 limitacion de que solo se pueden representar 2^63-1 numeros por encima del 0 (numeros de 19 cifras). Esto quiere decir que no podemos poner 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
- PYTHON
- C

Información adicional

Para todos los lenguages de programación escogidos, a la hora de ejecutar el programa todos tienen el factor común de que admiten los siguientes parámetros:

- -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.