Examen parcial 2 de Análisis y Diseño de Algoritmos. Fecha: Viernes, 20 de Noviembre de 2015.
Nombre del alumno: Angel de Jesus Bañuelos Sahagun_____ Aciertos: _____

I.- Se desea armar la torre de bloques más alta posible, dado un conjunto de bloques con diferente peso y capacidad de carga que incluye a su propio peso. Considere el ejemplo. El bloque C sólo puede cargar 1 unidad de peso adicional.

Bloque	Peso	Capacidad
A	1	7
B	5	10
C	3	4

Torre válida:

C
A

Torre no válida:

B
C

Torre válida de máxima altura:

C
A
B

Restricciones:

1) No se esperan más de 30 bloques de entrada.
2) Todos los pesos son enteros positivos menores que 100.
3) La capacidad de un bloque es un entero positivo mayor a su peso y no mayor que 200.

I.- Implementa un algoritmo *voraz* que reciba los datos de los bloques y devuelva los datos de una torre válida con la máxima altura encontrada. No se espera la solución óptima. Como estrategia, el mejor candidato a incluirse en la solución será aquél con mayor capacidad disponible (excluyendo su peso). Para el ejemplo anterior y usando dicha estrategia, el algoritmo voraz deberá devolver la torre: [A, B].

Método a implementar: `ObtenerTorre`

Argumentos de entrada: arreglo de pesos, arreglo de capacidades. Todos son números enteros. Los arreglos recibidos son del mismo tamaño. La posición *k* de cada arreglo refiere a los datos del bloque *k*.

Salida: arreglo de índices de bloques que representa la mejor torre obtenida, y se llena del bloque inferior al superior. Para el ejemplo mostrado, el arreglo de salida debe ser: {0, 1}.

Valor del ejercicio: 25 puntos

```
static Integer[] ObtenerTorre(int[] weights, int[] capacities) {
	PriorityQueue<Container> queue = new PriorityQueue();
	ArrayList<Integer> maxWeight = new ArrayList();
	// no sabemos cual sea la maxima capacidad, Hay que buscarla.
	int maxCapacity = -1;
	// Cual es el bloque con la maxima capacidad para ser insertado al principio.
	int maxCapacityIndex = -1;
	// Aqui calculo el bloque con mas capacidad.
	for (int i = 0; i < weights.length; i++) {
		int currentCapacity = capacities[i] - weights[i];
		// agrego los calculos al container
		queue.add(new Container(currentCapacity, i, weights[i]));
		if (currentCapacity > maxCapacity) {
			maxCapacity = currentCapacity;
			maxCapacityIndex = i;
		}
	}
	// agrego el bloque con mayor capacidad.
	maxWeight.add(maxCapacityIndex);
	while (!queue.isEmpty() && maxCapacity != 0) {
		// tomo de mayor a menor con respecto la maxima capacidad.
		Container current = queue.poll();
		// ya no agrego el bloque con mayor capacidad lo omito.
		if (maxCapacityIndex == current.index) {
```

```
                        continue;
                    }
                    // si aun hay espacion agrego el bloque y le resto el paso, para tener la
nueva capacidad.
                    if (maxCapacity >= current.weight) {
                        maxCapacity -= current.weight;
                        // Ya no puedo agregar bloques con mayor peso a la capacidad maxima.
                        if (current.currentCapacity < maxCapacity) {
                            // la capacidad maxima sera la del ultimo bloque
                            maxCapacity = current.currentCapacity;
                        }
                        maxWeight.add(current.getIndex());
                    }
                }
                Integer[] maxWeightI = new Integer[maxWeight.size()];
                maxWeight.toArray(maxWeightI);
                return maxWeightI;// returno el arreglo con los bloques.
        }

        static class Container implements Comparable<Container> {

                int currentCapacity;
                int index;
                int weight;

                public Container(int currentCapacity, int index, int weight) {
                        this.currentCapacity = currentCapacity;
                        this.index = index;
                        this.weight = weight;
                }

                public int getIndex() {
                        return index;
                }

                @Override
                public int compareTo(Container o) {
                        if (this.currentCapacity < o.currentCapacity) {
                                return 1; // If it is lower than return -1
                        }
                        if (this.currentCapacity > o.currentCapacity) {
                                return -1; // If it is greter than return 1
                        }
                        return 0; // if are equals return 0
                }

        }
```

II.- Implementa un algoritmo *backtracking* que, dados los mismos datos que el ejercicio anterior, devuelva la mayor altura posible. Este algoritmo deberá explorar a lo largo de todas las soluciones potenciales. El resultado debe ser el óptimo.

Método a implementar: `MaximaAltura`

Argumentos de entrada: arreglo de pesos, arreglo de capacidades; tal y como se definieron en la sección I.

Salida: número entero que denota la máxima altura posible. Para el ejemplo mostrado, devuelve 3, que corresponde a la altura de la torre: [B, A, C] = {1, 0, 2}.

Valor del ejercicio: 25 puntos

```java
/**
 * Haciendo uso de Backtraking buscamos la mejor combinacion para tener la
 * torre mas alta.
 *
 * @param weights
 * @param capacities
 * @return
 */
static int MaximaAltura(int[] weights, int[] capacities) {
	ArrayList<Integer> solutionTemp = new ArrayList();
	ArrayList<Integer> solution = new ArrayList();
	int best = 0;
	for (int i = 0, counter = 1; i < weights.length; i++) {
		solutionTemp.add(i);
		int tempCapacity = capacities[i] - weights[i];
		for (int j = 0; j < weights.length; j++) {
			if (i == j) {// Ya calculamos capacida antiormente
				continue;
			}
			if (tempCapacity < 0) {// No tiene caso seguir buscando ya nos
                                                // terminamos la capacidad
				break;
			}
			if (tempCapacity >= weights[j]) {
				tempCapacity -= weights[j];
				solutionTemp.add(j);
				counter++;
			}
		}
		if (counter > best) { // tengo una solucion buena solucion
			best = counter;
			solution = new ArrayList(solutionTemp); // clonamos la solucion
		}
		solutionTemp.clear();
		counter = 1;
	}
	System.out.println(solution);
	return solution.size();
}
```

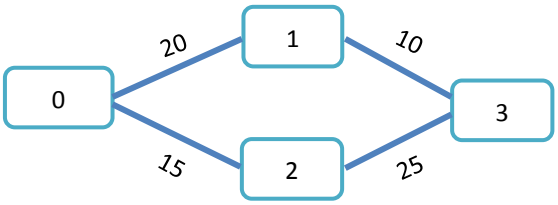
III.- Implementa un algoritmo que resuelva el siguiente problema.

Un guía de turistas tiene la misión de llevar a 12 turistas desde un pueblo de partida a uno destino, en una región del país donde existen muchos pueblos pintorescos comunicados mediante carreteras de dos carriles. Como el guía no tiene vehículo, tiene que subirlos a un camión rural. El problema es que los camiones sólo van de un pueblo a otro, y tienen capacidad limitada. Como no existen camiones disponibles desde un pueblo a todos los demás, y cada camión puede tener diferente capacidad, el guía de turistas tiene que buscar una secuencia de camiones que lo lleve del pueblo inicial al destino, tal que el camión más pequeño en la ruta pueda llevar a los N turistas . Implementa el siguiente método que determine si es posible encontrar dicha ruta. Por simplicidad, los pueblos se identifican con índices.

Método a implementar: `ExistsRoute`

Argumentos de entrada: mapa de carreteras como una matriz de adyacencia, número de turistas que puede llevar el camión más pequeño N, número de turistas a pasear T, ciudad inicial, ciudad destino.

Salida: *verdadero*, si es posible ir de la ciudad inicial a la ciudad destino; *falso*, en otro caso.



Para el mapa de la figura, con N = 4, T = 12, partida = 0, destino = 3, el guía sólo puede elegir la ruta 0–2–3. La ruta 0–1–3 no se puede elegir porque el camión que va de 1 a 3 tiene capacidad para 10 pasajeros y existen 12 turistas. El mapa de entrada se representa mediante una matriz de adyacencia de un grafo no dirigido tal que, si no existe un camión entre las ciudades *j*, *k*, *map*[*j*][*k*] = 0; en caso contrario, *map*[*j*][*k*] almacena la capacidad del camión.

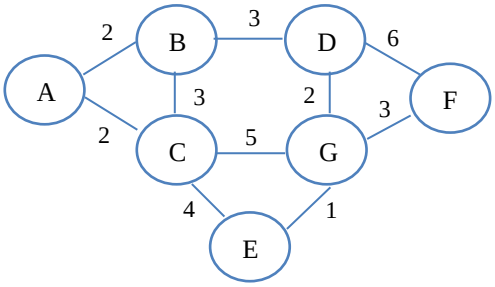
Valor del ejercicio: 25 puntos

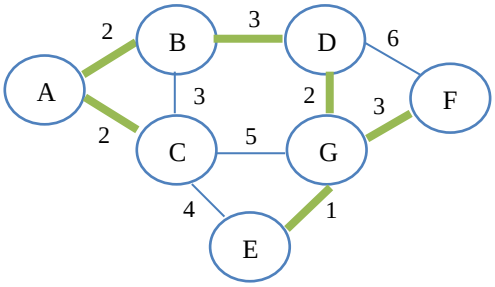
```
/**
 * Usando el algoritmo de Dijkstra se resuelve el problema
 *
 * @param graph
 * @param N
 * @param T
 * @param initial
 * @param destination
 */
static void ExistsRoute(int[][] graph, int N, int T, int initial,
            int destination) {
      Stack<Integer> nodeStack = new Stack<Integer>();
      Stack<Integer> distStack = new Stack<Integer>();
      boolean[] visited = new boolean[graph.length];
      int[] minDistances = new int[graph.length];
      for (int i = 0; i < minDistances.length; i++) {
            minDistances[i] = 0;
      }
      minDistances[initial] = 0;
      nodeStack.push(initial);
      distStack.push(0);
      while (!nodeStack.isEmpty()) {
            int currentNode = nodeStack.pop();
            int currentDist = distStack.pop();
            if (!visited[currentNode]) {
                  visited[currentNode] = true;
                  minDistances[currentNode] = currentDist;
                  for (int i = 0; i < graph.length; i++) {
```

```
                        // validacion para que tome solo rutas con la capacidad
                        // correcta.
                        if (graph[currentNode][i] != 0
                                    && graph[currentNode][i] >= T) {
                              nodeStack.push(i);
                              distStack.push(currentDist + graph[currentNode][i]);
                        }
                  }
            }
      }
      System.out.println(Arrays.toString(minDistances));
      if (minDistances[destination] != 0) {
            System.out.println("verdadero");
      } else {
            System.out.println("falso");
      }
}
```

IV.- Dibuje el *árbol de recubrimiento mínimo* del grafo que se obtiene con el algoritmo de *Prim*, comenzando con A.

Valor del ejercicio: 10 puntos.

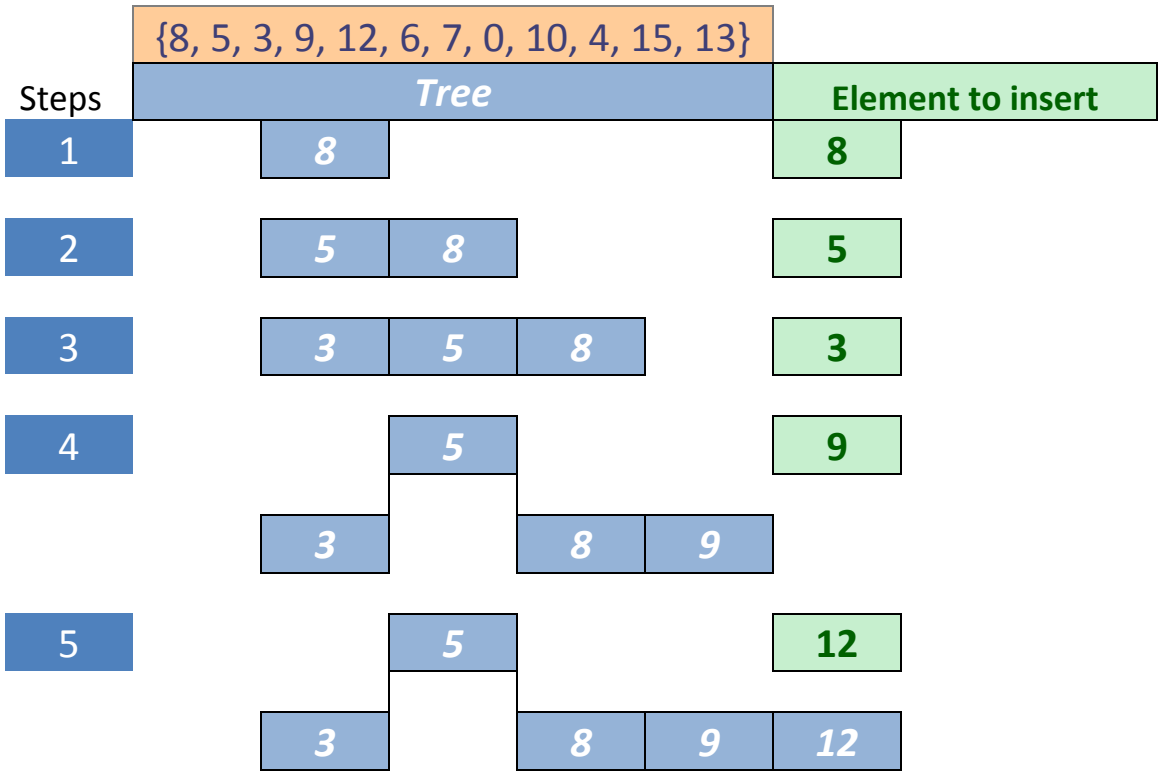


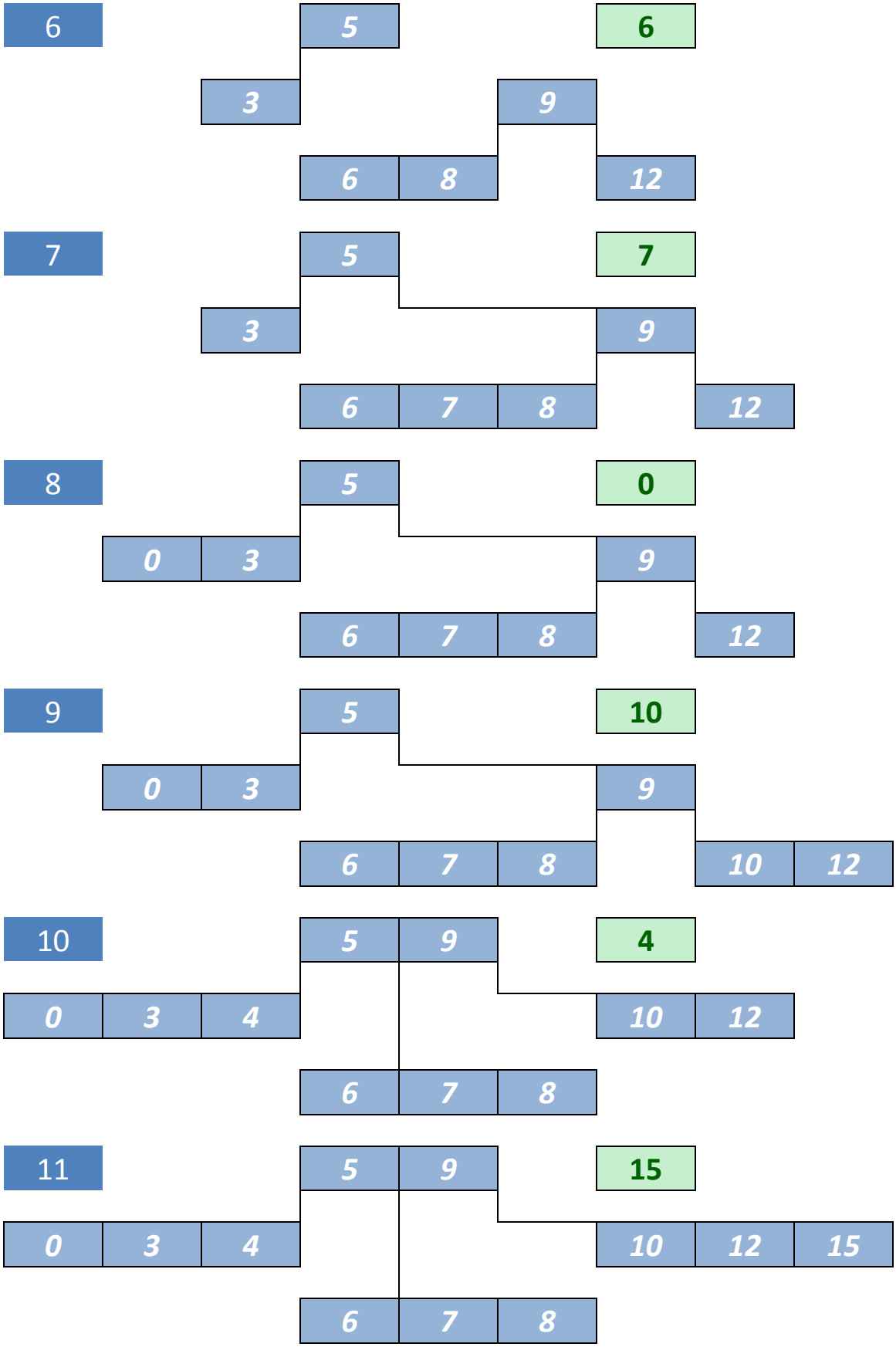


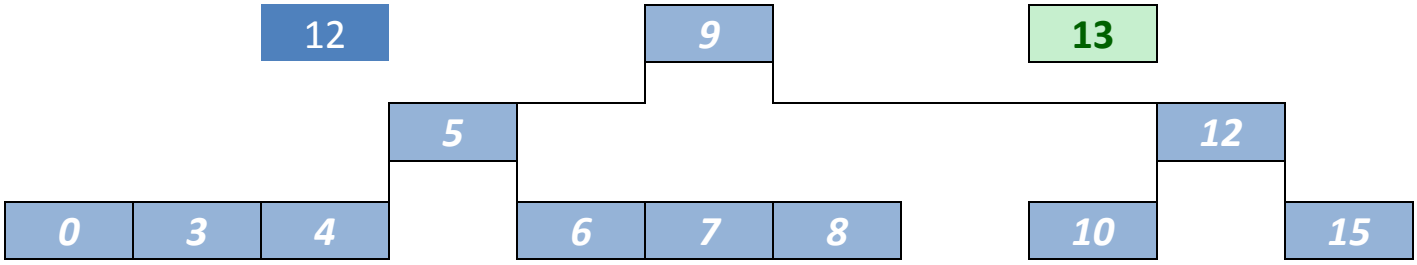
AB,AC,BD,DG,GE,GF $\sum$ deg(AB,AC,BD,DG,GE,GF) = 13

V.- Construya el árbol 2-3-4 para la lista {8, 5, 3, 9, 12, 6, 7, 0, 10, 4, 15, 13}. Incluya en la respuesta cómo va quedando el árbol después de cada cambio en la estructura del árbol.

Valor del ejercicio: 10 puntos.




6
5
6
3
9
6
8
12
7
5
7
3
9
6
7
8
12
8
5
0
0
3
9
6
7
8
12
9
5
10
0
3
9
6
7
8
10
12
10
5
9
4
0
3
4
10
12
6
7
8
11
5
9
15
0
3
4
10
12
15
6
7
8



VI.- Se desean realizar búsquedas en tiempo constante sobre un catálogo de servicios médicos de un hospital privado, identificados por una clave formada por una combinación de cinco símbolos del siguiente conjunto de nueve elementos: {6, #, A, ☺, f, @, 3, ©, δ}. ¿Cuál sería el código *hash* asignado a la clave [#f6@δ] tomando en cuenta que no se esperan más de 500 servicios en el catálogo y el caracter de la extrema derecha se considera como el dato menos significativo? Incluya el desarrollo.

Valor del ejercicio: 10 puntos.

```
public class HashMapping {

	static class Alphabet {

		char Character;
		int value;

		public Alphabet(char Character, int value) {
			this.Character = Character;
			this.value = value;
		}

	}

	final static int MAX_SERVICES = 500;
	final static int ELEMENTS = 9;

	static int getValue(char c) {
		for (int i = 0; i < ELEMENTS; i++) {
			if (alphabet[i].Character == c) {
				return alphabet[i].value;
			}
		}
		return -1;
	}

	static int hashCode(String clave) {
		int hashCode = getValue(clave.charAt(0));
		for (int i = 1; i < clave.length(); i++) {
			hashCode = ((hashCode * ELEMENTS) + getValue(clave.charAt(i)));
		}
		return hashCode % MAX_SERVICES;
	}

	public static void main(String[] args) {
		String clave = "#f6@δ";
		System.out.println("HashCode clave '" + clave + "' = "
				+ hashCode(clave));
	}

	static Alphabet[] alphabet = new Alphabet[ELEMENTS];
```

```
	static {
		alphabet[0] = new Alphabet('6', 0);
		alphabet[1] = new Alphabet('#', 1);
		alphabet[2] = new Alphabet('A', 2);
		alphabet[3] = new Alphabet('☺', 3);
		alphabet[4] = new Alphabet('f', 4);
		alphabet[5] = new Alphabet('@', 5);
		alphabet[6] = new Alphabet('3', 6);
		alphabet[7] = new Alphabet('©', 7);
		alphabet[8] = new Alphabet('δ', 8);
	}
}
```

VII.- Consider the following 3d implicit formula, called the *super formula*:

$$F(x,y,z) = \begin{cases} 1 + F(x-2,y-1,z-1) + F(x-1,y-2,z-1) + F(x-1,y-1,z-2), & x,y,z > 0 \\ 0, & otherwise \end{cases}$$

The result is actually the number of times the *super formula* was called with all the parameters greater than zero.

If $x = 3$, $y = 2$, $z = 2$,
$F(2, 2, 3) = 1 + F(0, 1, 2) + F(1, 0, 2) + (1, 1, 1) =$
1 + 0 + 0 + 1 + F(-1, 0, 0) + F(0, -1, 0) + F(0, 0, -1)
1 + 0 + 0 + 1 + 0 + 0 + 0
2

Implement a method that calculates the result of the *super formula* given three integer values *x*, *y*, *z*. Since this number can be very high, output the result modulo 57413.

Run the method with the following examples to prove the correctness and efficiency of the algorithm implemented:

1) F (5, 6, 7) = 32
2) F (10, 9, 8) = 360
3) F (30, 29, 28) = 36574
4) F (95, 96, 97) = 43602
5) F (100, 100, 100) = 11862

Valor del ejercicio: 25 puntos

Para el caso 3 me da un diferente valor. Pero cuando estuve validando con la versión recursiva si me da el valor que con Programación Dinámica. ***29413***

```
public class SuperFormula {

	final static long MODULE = 57413;

	public static void main(String[] args) {
		System.out.println("1)    F(5, 6, 7) = " + superFormulaPD(5, 6, 7));
		System.out.println("2)    F(10,9,8) = " + superFormulaPD(10, 9, 8));
		System.out.println("3)    F(30,29,28) = " + superFormulaPD(30, 29, 28));
		System.out.println("4)    F(95,96,97) = " + superFormulaPD(95, 96, 97));
		System.out.println("5)    F(100,100,100) = "
				+ superFormulaPD(100, 100, 100));
	}

	static long superFormulaPD(int x, int y, int z) {
		long[][][] matrix = new long[x + 1][y + 1][z + 1];
		for (int i = 0; i <= x; i++) {
			for (int j = 0; j <= y; j++) {
				for (int k = 0; k <= z; k++) {
					if ((i * j * k) <= 0) {
						matrix[i][j][k] = 0;
						continue;
					}
					if (i < 2 || j < 2 || k < 2) {
						matrix[i][j][k] = 1;
					} else {
						matrix[i][j][k] = (1 + matrix[i - 2][j - 1][k - 1]
								+ matrix[i - 1][j - 2][k - 1] + matrix[i -
1][j - 1][k - 2])
								% MODULE;
```

```
                                }
                        }
                }
        }
        return matrix[x][y][z];
    }

    static long rF(int x, int y, int z) {
        if ((x * y * z) <= 0) {
            return 0;
        }
        return (1 + rF(x - 2, y - 1, z - 1) + rF(x - 1, y - 2, z - 1) + rF(
                x - 1, y - 1, z - 2)) % MODULE;
    }


}
```