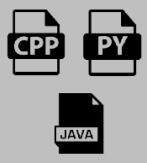
Taller en Sala Nro. 2
Fuerza Bruta

En seguridad informática, una estrategia de fuerza bruta para romper la seguridad de un sistema es probar todas las permutaciones válidas para una contraseña hasta encontrar la correcta.

Trabajo en Parejas

Hoy, plazo máximo de entrega

Docente entrega código suelto en GitHub



Sí .cpp, .py o .java

No .zip, .txt, html o .doc

Alumnos entregan código suelto por GitHub

Ejercicio a resolver

1. Pepito escribió un algoritmo que, dado un arreglo de enteros, decide si es posible escoger un subconjunto de estos, de tal forma que la suma de los elementos de ese subconjunto sea igual al parámetro *target*.

El parámetro *start* funciona como un contador y representa un índice en el arreglo de números *nums*. Por favor, ayúdenle a completarlo:

```
private static boolean sumaGrupo(int start, int[] nums, int
target) {

}
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

```
public static boolean sumaGrupo(int[] nums, int target) {
   return sumaGrupo(0, nums, target);
}
```

Como un ejemplo, si pregunta hay un subconjunto de {2, 4, 8} que sume 10, la respuesta es verdadero. **Como otro ejemplo,** si preguntamos si hay un subconjunto de {2, 4, 8} que sume 20, la respuesta es falso.

2. Escriban un programa que compute las combinaciones de longitud *k* con *0* <= *k* <= *n* de una cadena de letras de longitud *n*. Una combinación de longitud *k* es un subconjunto de *k* elementos de los *n* caracteres de la cadena. El enunciado se reduce a hallar todos los posibles subconjuntos de la cadena. Existen 2^n subconjuntos.

Como un ejemplo, para "abc", debe dar esta respuesta:

- " "
- "a"
- "b"
- "c"
- "ab"
- "ac"
- "bc"
- "abc"

3. Escriban un programa que calcule las n! permutaciones (sin repetición) de una cadena. **Como un ejemplo**, cuando uno le dicen "abc" debe dar la siguiente salida:

- "abc"
- "acb"
- "bac"
- "bca"
- "cab"
- "cba"

No importa el orden en que estén las n! permutaciones en el *ArrayList<String>* que retorna.

4. Escriban una implementación que resuelva el problema de las n-reinas contando todas las posibles formas de ubicar las reinas con fuerza bruta.

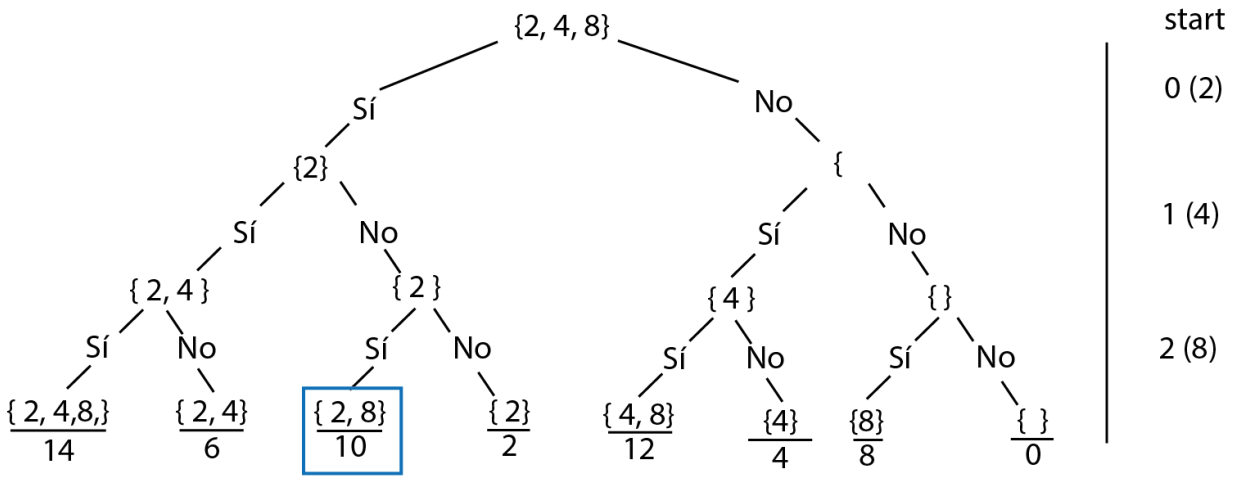
DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Ayudas para resolver los Ejercicios

Ayudas para resolver el Ejercicio 1

Pista 1: Generen todos los posibles subconjuntos del conjunto dado y pruebe con cada uno. Para esto se deben hacer dos llamados recursivos.

Como un ejemplo, a continuación, se muestra el árbol de ejecución para encontrar si un subconjunto de {2, 4, 8} suma 10 generando todos los posibles subconjuntos:



Error Común 1: Este algoritmo falla porque al llamarse recursivamente con el parámetro `start` se queda en una recursión infinita.

```
public static boolean sumaGrupo(int start, int[] nums, int
target) {
   if (start >= nums.length) return target == 0;
   return sumaGrupo(start+1, nums, target - nums[start])
      || sumaGrupo(start, nums, target );
}
```

DOCENTE MAURICIO TORO BERMÚDEZ
Teléfono: (+57) (4) 261 95 00 Ext. 9473. Oficina: 19 - 627
Correo: mtorobe@eafit.edu.co

Error Común 2: Este algoritmo falla porque al llamarse recursivamente con el parámetro `start-1` se sale del arreglo cuando `start = 0`.

```
public static boolean sumaGrupo(int start, int[] nums, int
target) {
 if (start >= nums.length) return target == 0;
 return sumaGrupo(start+1, nums, target - nums[start])
    || sumaGrupo(start-1, nums, target );
}
```

Error Común 3: Este algoritmo falla porque al llamarse recursivamente con el parámetro `start` se sale del arreglo cuando `start = 0`.

```
public static boolean sumaGrupo(int start, int[] nums, int
target) {
 if (start >= nums.length) return target == 0;
 return sumaGrupo(start+1, nums, target - nums[start])
    || sumaGrupo(start+1, nums, target - nums[start - 1] );
}
```

Ayudas para resolver el Ejercicio 2

Pista 0: Consideren el siguiente código:

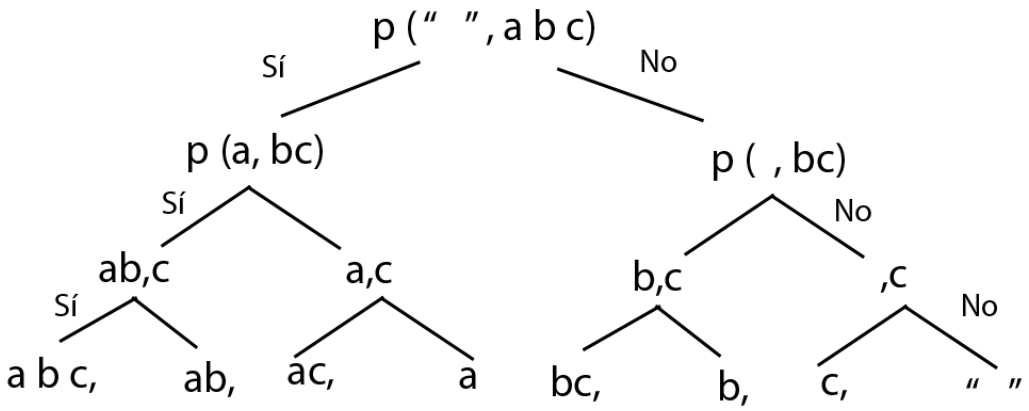
```
public static ArrayList<String> combinations(String s) {

}
```

Pista 1: Si tienen dudas sobre cuáles son los subconjuntos de un conjunto, vean este video https://www.youtube.com/watch?v=Bxv2KvlItxs

Pista 2: Creen una función recursiva que le permita llenar el arreglo con las combinaciones de forma más fácil. Algo como esto:

```
private static void combinations(String pre, String pos,
ArrayList<String> list) {

}
```

Pista 3: La función recursiva debe generar el siguiente árbol de ejecución para generar los subconjuntos de la cadena “abc”



Nótese que el árbol de la recursión (y por tanto la solución al problema) es prácticamente igual (sólo que con letras) al del ejemplo del punto 1.

Error Común 1: Un error común es calcular los prefijos en lugar de los subconjuntos, como se muestra en el siguiente ejemplo. La solución es hacer 2 llamados recursivos y no uno solo.

```
public static void prefijos(String base, String s) {
 if (s.length() == 0) {
    System.out.println(base);
 } else {
    prefijos(base + s.charAt(0), s.substring(1));
    System.out.println(base);
 }
```

```
}
```

Error Común 2:

Ayudas para resolver el Ejercicio 3

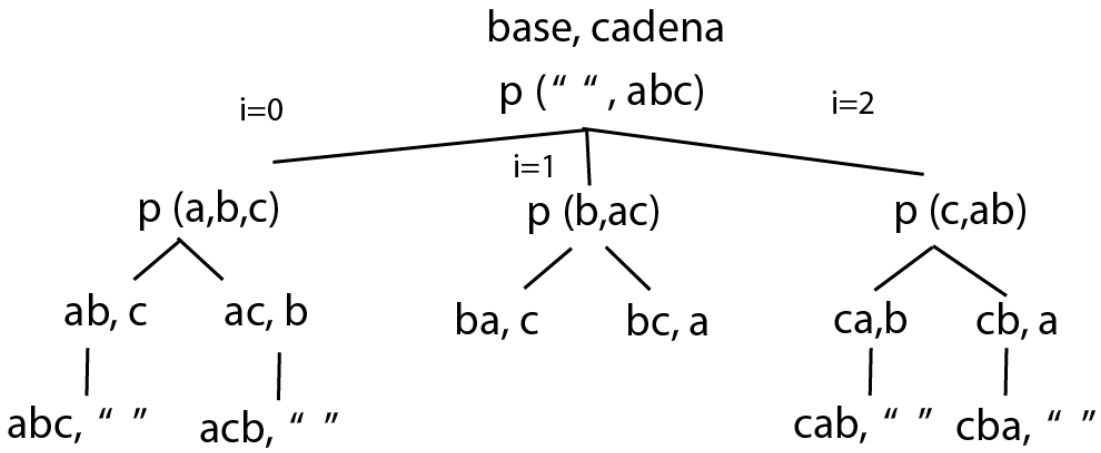
Pista 0: Consideren el siguiente código:

```
public static ArrayList<String> permutations(String s) {

}
```

Pista 1: Creen una función recursiva que les permita llenar el arreglo con las combinaciones de forma más fácil. Algo como esto:

```
private static void permutations(String pre, String pos,
ArrayList<String> list) {

}
```

Pista 2: La función recursiva debe generar el siguiente árbol de ejecución para generar las n! permutaciones de longitud 3 de la cadena “abc”:



Error Común 1: Un error común es calcular los prefijos en lugar de los subconjuntos, como se muestra en el siguiente ejemplo. La solución es hacer 2 llamados recursivos y no uno solo.

Ayudas para resolver el Ejercicio 4

Guía para la implementación:

1. Realicen un método para saber si dadas las posiciones de las reinas en un tablero estas se atacan o no. Esto para determinar si un tablero es válido o no.

```
public static boolean esValido(int[] tablero) {

}
```

Pista: http://mnemstudio.org/ai/ga/images/nqueens1.gif

2. Realicen un método que calcule recursivamente todas las posibles permutaciones de las posiciones de las reinas en el tablero y pruebe para cada uno si este es válido o no, y cuente la cantidad de tableros válidos.

```
public static int queens(int n) {

}
```

Pista 1: Utilicen la representación comprimida de tableros para las n-reinas: un arreglo en donde cada posición representa una fila del tablero, y el valor contenido en esta posición representa la columna en esa fila en donde se encuentra la reina.

Por ejemplo, para el siguiente tablero

#	#	Q	#
Q	#	#	#
#	#	#	Q
#	Q	#	#

La representación propuesta es un arreglo que luce así:

2	0	3	1

Se provee un método llamado `imprimirTablero(int[] tablero)` que imprime tableros representados de esta manera.

Esta representación además de consumir menos memoria permite generar todas las posibles permutaciones de las posiciones de las reinas en el tablero de forma más fácil (similar al punto 3).

Pista 2: Pueden generar todas las permutaciones de las reinas en el tablero utilizando tanto permutaciones con repetición (n^n permutaciones) como sin repetición (n! permutaciones). Si bien generar las permutaciones con repetición es computacionalmente más costoso, es bastante más fácil de implementar.

¿Alguna inquietud?

CONTACTO

Docente Mauricio Toro Bermúdez
Teléfono: (+57) (4) 261 95 00 **Ext.** 9473
Correo: mtorobe@eafit.edu.co
Oficina: 19- 627

Agende una cita con él a través de **http://bit.ly/2gzVg10** , en la pestaña *Semana. Si no da clic en esta pestaña, parecerá que toda la agenda estará ocupada.*