ESTRUCTURA DE DATOS 2
Código ST0247

Taller en Sala 2
Fuerza Bruta

Objetivos: 1. Identificar un ejemplo práctico para cada una de las estrategias de diseño de algoritmos. **2.** Traducir problemas de la vida real a soluciones algorítmicas.

Consideraciones: Lean y verifiquen las consideraciones de entrega,

Trabajo en Parejas

Mañana, plazo de entrega

Docente entrega plantilla de código en GitHub

Sí .cpp, .py o .java

No .zip, .txt, html o .doc

Alumnos entregan código sin comprimir GitHub

En la carpeta Github del curso, hay un código iniciado y un código de pruebas (tests) que pueden explorar para solucionar los ejercicios

Estructura del documento: a) Datos de *vida real*, **b)** I*ntroducción* a un problema, **c)** Problema a resolver, **d)** Ayudas. Identifiquen esos elementos así:

a)

b)

c)

d)

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

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

Ejercicios a resolver

1 En el videojuego *Dark Souls 2*, para elegir los elementos que uno desea tener en su inventario, no siempre es tan trivial como encontrar un subgrupo de elementos cuya suma de pesos sea el peso máximo.

En muchas ocasiones se debe tener en cuenta que ciertas características del personaje, dan mejor desempeño a ciertas armas o que las armas se pueden vender a otros jugadores. Una solución es dejar que el usuario sea quien tome la decisión y que el sistema simplemente le muestre cuáles son los posibles subgrupos

Implementen un algoritmo recursivo que, dado un grupo de elementos, le muestre en pantalla todos los posibles subgrupos que se pueden formar con esos elementos.

Formalmente, implementen un programa que compute las combinaciones de longitud *k* con *0* <= *k* <= *n* de una cadena de letras de longitud *n.*

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

2 Durante el desarrollo de la *Operación Fénix*, por parte de las Fuerzas Militares de Colombia en marzo 1 de 2008, fueron encontrados varios computadores de Raúl Reyes que detallaban información histórica de las actividades de las FARC.

Los computadores fueron enviados a la INTERPOL para analizar la información que se encontró y la veracidad de la misma. La información se encontraba encriptada.

Una solución para desencriptar un archivo es realizar un ataque de fuerza bruta. Este ataque consiste en probar todas las permutaciones posibles hasta encontrar la contraseña que permite el acceso.

Implementen un algoritmo recursivo para desencriptar un archivo, probando todas las permutaciones posibles de los caracteres de una cadena de caracteres, teniendo en cuenta, por simplicidad, que las letras de la contraseña no se repiten y que una la contraseña es una permutación de la cadena de caracteres ingresada por el usuario.

Utilicen el algoritmo que genera las permutaciones para desencriptar el archivo *archivoEncriptado.txt* que se encuentra en GitHub. Utilicen la clase *AdvancedEncryptionStandard*. El *password* es una permutación de "abcd".

3 **[Ejercicio opcional]** En 1996, el computador *Deep Blue*, desarrollado por IBM, venció al campeón mundial de ajedrez Garry Kasparov. Posteriormente, en 2016, el programa AlphaGo, desarrollado por Google, venció a uno de los mejores jugadores de Go en el mundo.

Programas que sean capaz de jugar ajedrez o Go, son de mucho interés para las grandes multinacionales porque dan los cimientos para sistemas de computación cognitiva como Watson de IBM. Un primer paso para construir sistemas que jueguen ajedrez, es resolver el acertijo de las *n reinas*.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

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

4 **[Ejercicio opcional]** Para hacer el ataque por fuerza bruta más realista, **calculen las permutaciones de una cadena, como en el ejercicio anterior, pero permitiendo que las letras se repitan; por ejemplo, permutaciones nuevas que aparecerían para la cadena "abc" son aaa, aab, bcc, etc**.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Ayudas para resolver los Ejercicios

Ejercicio 1

Pista 1: 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.”

Ejemplo 1: Para “abc”, debe dar esta respuesta:

“ ”	“ab”
“a”	“ac”
“b”	“bc”
“c”	“abc”

Pista 2: Consideren el siguiente código:

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

}
```

Pista 3: Si tienen dudas sobre cuáles son los subconjuntos de un conjunto, vean este video https://bit.ly/2N3jBgI

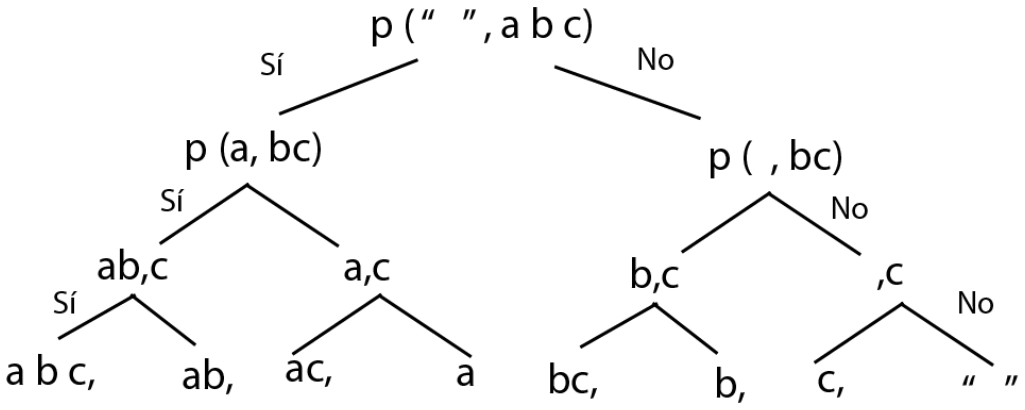
Pista 4: Creen una función recursiva que le permita llenar el arreglo con las combinaciones de forma más fácil. Algo como esto:

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

}
```

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Pista 5: 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: 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: 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.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Ejercicio 2

Pista 1: Escriban un programa que calcule las n! permutaciones (sin repetición) de una cadena.

Ejemplo 1: Cuando uno le dicen “abc” debe dar la siguiente salida:

“abc”	“bca”
“acb”	“cab”
“bac”	“cba”

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

Pista 2: Consideren el siguiente código:

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

}
```

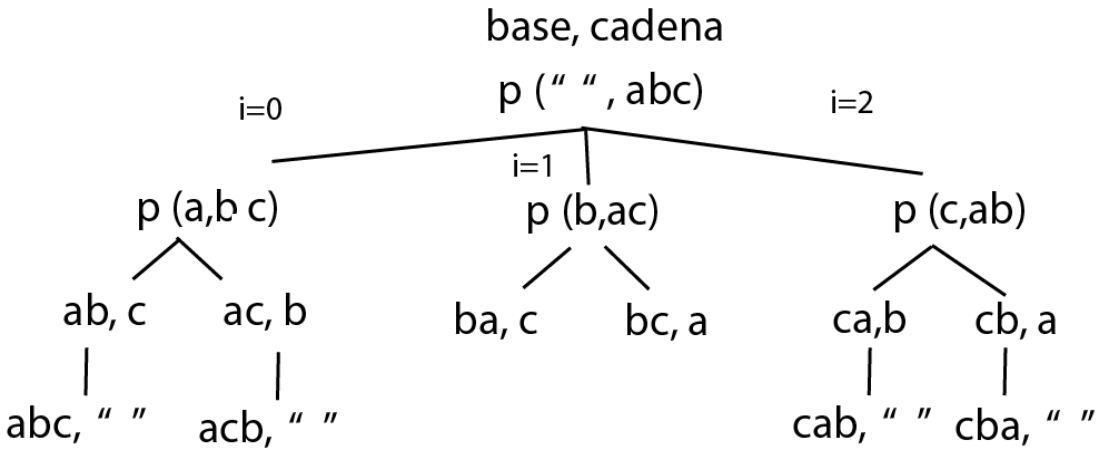
PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Pista 3: 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,
LinkedList<String> list) {

}
```

Pista 4: 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: Este código calcula algunos de los subconjuntos de una cadena en lugar de todas las permutaciones. ¿Por qué está pasando esto? ¿qué falta?

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

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Ejercicio 3

Pista 0: Este video explica el problema muy bien:
https://www.youtube.com/watch?v=WOZ4wDt-iYA

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

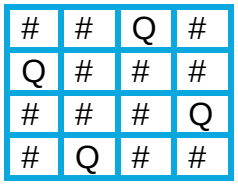
2. Realicen un método que calcule recursivamente todas las posibles permutaciones de las posiciones de las reinas en el tablero y pruebe para cada uno si este es válido o no, y cuente la cantidad de tableros válidos. **Esta función retorna un entero que representa el número de soluciones que existen para las n-reinas. Por ejemplo, para n=4, retorna 2 y para n=8 retorna 92.**

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

}
```

Pista 2: 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.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473



Ejemplo 1: Para el siguiente tablero

La representación propuesta es un arreglo que luce así:

1	3	0	2

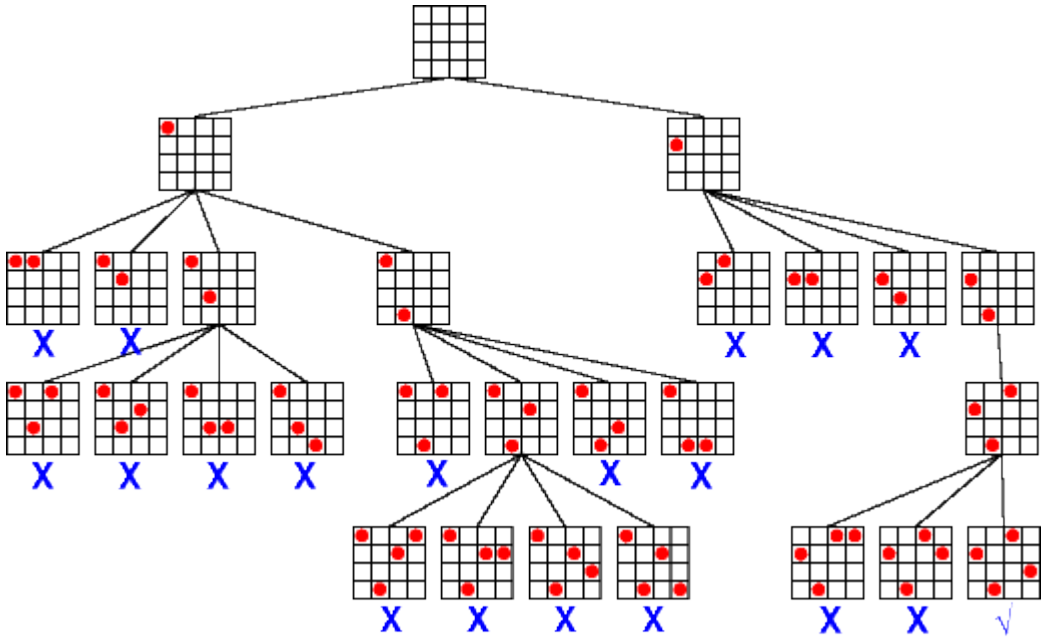
Se provee un método llamado `imprimirTablero(int[] tablero)` que imprime tableros representados de esta manera.

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 3: 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.

PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

Pista 4: Consideren el siguiente árbol de ejecución para encontrar una solución a las n-reinas con n=4.



PhD. Mauricio Toro Bermúdez
Docente | Escuela de Ingeniería | Informática y Sistemas
Correo: mtorobe@eafit.edu.co | Oficina: Bloque 19 – 627
Tel: (+57) (4) 261 95 00 Ext. 9473

¿Alguna inquietud?

CONTACTO

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

Agenden una cita dando clic en la pestaña -*Semana*- de http://bit.ly/2gzVg10