**Cálculo de Combinaciones posibles**

Profesor:

Aschettino, Giovanna

Cuadrado Estrebou, Maria Fernanda

Grupo:

Rios, Mateo – LU: 1120860

Cassi, Daniel Ignacio – LU: 1115907

Von Elm, Lucas– LU: 1133639

Buenos Aires, XX de Junio de 2023.-

**Tabla de Contenidos**

[Introducción 3](#_Toc135681731)

[Descripción del Problema 3](#_Toc135681732)

[Estrategia de Resolución 3](#_Toc135681733)

[Pseudocódigo del Algoritmo de Resolución del Problema 3](#_Toc135681734)

[Análisis de Complejidad Temporal 3](#_Toc135681735)

[Conclusiones 3](#_Toc135681736)

[Bibliografía 4](#_Toc135681737)

# Introducción

El objetivo del proyecto es realizar un algoritmo, que dado un listado de operadores y numeros a usar, obtener todas las combinaciones posibles que den un numero “X” dado. Los operadores deben ser todos utilizados y del listado de números se deben solamente utilizar “n” números a la vez.

# Descripción del Problema

## Estrategia de Resolución

Estrategia: Para resolver el problema de las combinaciones se utiliza un algoritmo de Backtracking, con el cual iteramos por todas las combinaciones de orden posibles de los operadores, y todas las combinaciones de numeros para cada uno de ellas. Al obtener una combinacion posible, se realiza el calculo, y si el resultado de este da el valor deseado, se registra esa combinación.

*<<Descripción de la estrategia de resolución utilizada y técnica aplicada >>*

## Pseudocódigo del Algoritmo de Resolución del Problema

*<<Escribir el pseudocódigo correspondiente para la resolución del problema de acuerdo a la estrategia utilizada >>*

**Algoritmo buscarCombinaciones**

**Entrada:** (Entero n, Double x, List<String> listaOperadores, List<Entero> listaOperadoresUsados, List<String> ordenOperadores, List<Double> listaNumeros, List<Entero> listaNumerosUsados, List<Double> ordenNumeros, etapaOperadores, etapaNumeros)

**Salida:** (List<Combinacion> listadoCombinaciones)

Si etapaOperadores = listaOperadores.size

Si etapaNumeros = n

Si (calcularCombinacion(x, ordenOperadores, ordenNumeros))

Combinacion comb 🡨 Crear (ordenOperadores, ordenNumeros)

listadoCombinaciones 🡨 Agregar (comb)

Sino

Por cada j = 0 hasta listaNumeros.tamaño

Si puedeUsarse(j, listaNumerosUsados)

marcarUsadoNumero(j, listaNumerosUsados, ordenNumeros)

buscarCombinaciones (n, x, listadoCombinaciones, listaOperadores, listaOperadoresUsados, ordenOperadores, listaNumeros, listaNumerosUsados, ordenNumeros, etapaOperadores, etapaNumeros + 1)

desmarcarUsadoNumero(j, listaNumerosUsados, ordenNumeros)

Fin Si

Fin Por

Fin Sino

Sino

Por cada i = 0 hasta listaOperadores.tamaño

Si puedeUsarse(i, listaOperadoresUsados)

marcarUsadoOperador(i, listaOperadoresUsados)

buscarCombinaciones(n, x, listadoCombinaciones, listaOperadores, listaOperadoresUsados, ordenOperadores, listaNumeros, listaNumerosUsados, ordenNumeros, etapaOperadores + 1, etapaNumeros)

desmarcarUsadoOperador(i, listaOperadoresUsados)

Fin Si

Fin Por

Fin Sino

Devolver listadoCombinaciones

**Algoritmo puedeUsarse**

**Entrada**: (Entero x, List<> listaUsados)

**Salida**: Boolean

Devolver listaUsados.obtener(x) = 0

**Algoritmo marcarUsadoNumero**

**Entrada**: (Entero x, List<Double> listaNumeros, List<Entero> listaNumerosUsados, List<Double> ordenNumeros)

numeroUsado 🡨 listaNumeros.obtener(x)

listaNumerosUsados 🡨 Reemplazar (x, 1)

ordenNumeros 🡨 Agregar(numeroUsado)

**Algoritmo desmarcarUsadoNumero**

**Entrada**: (Entero x, List<Double> listaNumeros, List<Entero> listaNumerosUsados, List<Double> ordenNumeros)

indiceNumero 🡨 ordenNumeros.tamaño - 1

listaNumerosUsados 🡨 Reemplazar (x, 0)

ordenNumeros 🡨 Quitar(indiceNumero)

**Algoritmo marcarUsadoOperador**

**Entrada**: (Entero x, List<String> listaOperadores, List<Entero> listaOperadoresUsados, List<String> ordenOperadores)

operadorUsado 🡨 listaOperadores.obtener(x)

listaOperadoresUsados 🡨 Reemplazar (x, 1)

ordenOperadores 🡨 Agregar(operadorUsado)

**Algoritmo desmarcarUsadoOperador**

**Entrada**: (Entero x, List<String> listaOperadores, List<Entero> listaOperadoresUsados, List<String> ordenOperadores)

indiceOperador 🡨 ordenOperadores.tamaño - 1

listaOperadoresUsados 🡨 Reemplazar (x, 0)

ordenOperadores 🡨 Quitar(indiceOperador)

**Algoritmo calcularCombinacion**

**Entrada**: (Entero x, List<> operadores, List<> numeros)

**Salida**: Boolean

List<> operadoresAux 🡨 Copiar(operadores)

List<> numerosAux 🡨 Copiar(numeros)

Double calculoAux 🡨 0

Entero j 🡨 0

Por cada i = 0 hasta 1

Mientras j < operadoresAux.tamaño

Si i = 0 //Hago primero las multiplicaciones y divisiones

Si operadorAux[j] = “\*” O operadorAux[j] = “/”

calculoAux 🡨 calculoSimple (operadoresAux[j], numerosAux[j], numerosAux[j+1])

operadoresAux 🡨 Quitar(j)

numerosAux 🡨 Quitar(j+1)

numerosAux🡨 Reemplazar(j, calculoAux)

Sino

j 🡨 j+1 //Busca el siguiente “\*” O “/”

Sino

calculoAux 🡨 calculoSimple (operadoresAux[j], numerosAux[j], numerosAux[j+1])

operadoresAux 🡨 Quitar(j)

numerosAux 🡨 Quitar(j+1)

numerosAux🡨 Reemplazar (j, calculoAux)

Fin Sino

Fin por

Fin por

Devolver numerosAux[0] = x

**Algoritmo calculoSimple**

**Entrada**: (String operador, Double numero1, Double numero2)

**Salida**: Double

Si operador = “\*”

Devolver numero1 \* numero2

Sino Si operador = “/”

Devolver numero1 / numero2

Sino si operador = “+”

Devolver numero1 + numero2

Sino

Devolver numero1 – numero2

## Análisis de Complejidad Temporal

## Análisis de Complejidad Teórica:

**Algoritmo buscarCombinaciones**

Costo: P! \* V n r

Siendo P = la cantidad de Operadores disponibles y P! referiendose a todas las permutaciones de orden de estas.

Siendo V n r = El calculo de la variación sin repetición de N numeros en R puestos.

N = La cantidad de números disponibles

R = La cantidad de números a usar.

Tal costo es obtenido debido a que el algoritmo diseñado calcula todas las permutaciones posibles de orden de los operadores y cada variacion de los numeros a usar para cada una de tales permutaciones.

**Algoritmo puedeUsarse**

Costo: 0(1) - Constante

**Algoritmo marcarUsadoNumero**

Costo: 0(n) en promedio – Dado por la función Agregar.

Costo Constante en caso de que la función agregar funcione en complejidad O(1)

**Algoritmo desmarcarUsadoNumero**

Costo: 0(n)– Dado por la función Quitar.

Costo Constante en caso de que la función Quitar funcione en complejidad O(1)

**Algoritmo marcarUsadoOperador**

Costo: 0(n) en promedio – Dado por la función Agregar.

Costo Constante en caso de que la función agregar funcione en complejidad O(1)

**Algoritmo desmarcarUsadoOperador**

Costo: 0(n)– Dado por la función Quitar.

Costo Constante en caso de que la función Quitar funcione en complejidad O(1)

**Algoritmo calcularCombinacion**

Costo: O(2J) Siendo J la cantidad de operadores.

**Algoritmo calculoSimple**

Costo: Constante

## Análisis de Complejidad Práctica:

*<<Todavía por realizar>>*

*Nota: En el Anexo de este documento se definen los costos para las operaciones incluidas en las implementaciones provistas en la librería del TPO. >>*

# Conclusiones

*<<Descripción resumida del problema, estrategia utilizada y justificaciones y análisis de los resultados>>*

# Bibliografía

* <*Titulo*> - <Autor 1, Autor2,..>- Editorial – Edición
* <*Titulo*> - <Autor 1, Autor2,..>- Editorial – Edición