Universidad Peruana de Ciencias Aplicadas

Escuela de Ingeniería de Software

Complejidad Algorítmica

Razonamiento Cuantitativo

Profesor
Canaval Sanchez, Luis Martín

Alumno

- Jair Huamán Bellido - u201413797

Lima - 2019

1. **Interpretación**

   El problema identificado a resolver es el de realizar la cantidad de viajes posibles para colocar de manera óptima una cantidad “n” de elementos en un container. Es por ello que se implementará un algoritmo para resolver dicho problema.

2. **Representación**

   Para el siguiente proyecto se ha tomado en cuenta las siguientes variables:

   wContainer: Ancho del contenedor (wContainer > 0)

   hContainer: Altura del contenedor (hContainer > 0)

   dContainer: Profundidad del contenedor (zContainer > 0)

   nCategoriasCubos: Cantidad de tipos de cubos (nCategoriasCubos > 0)

   nCubo: Cantidad de cubos de tipo “N” (n > 0)

   wCubo: Ancho del cubo (wCubo > 0)

   hCubo: Altura del cubo (hCubo > 0)

   dCubo: Profundidad del cubo (dCubo > 0)

   Las siguientes variables son valores enteros:
   nCategoriasCubos y nCubo

   Las siguientes variables son valores reales:
   wContainer, hContainer, dContainer, wCubo, hCubo y dCubo

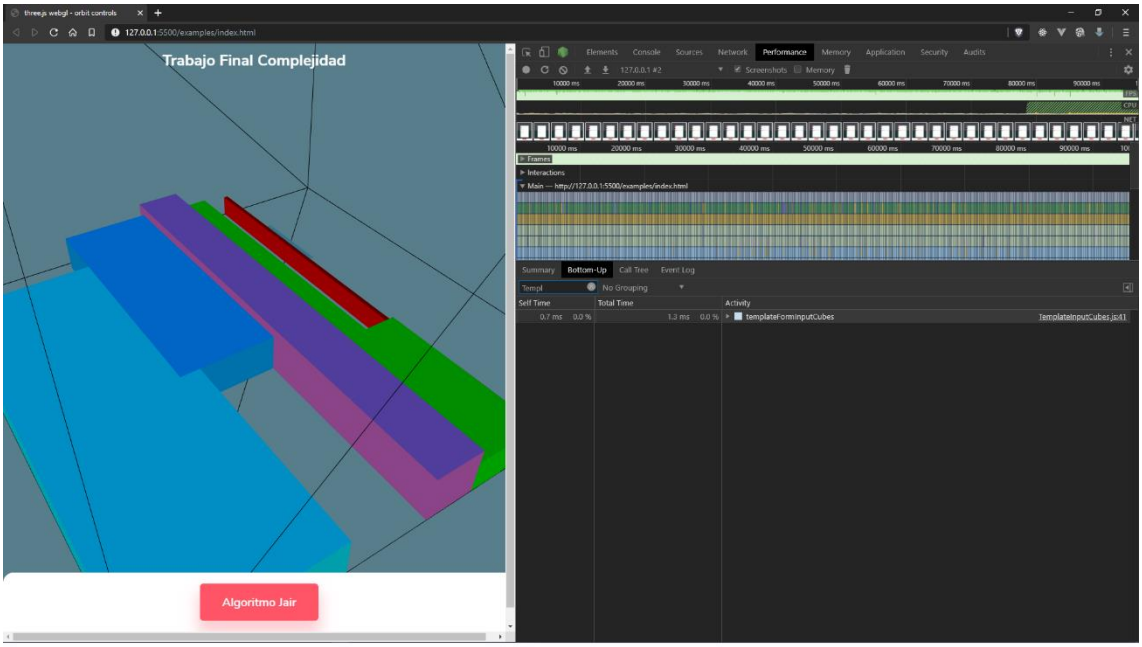
3. Cálculo

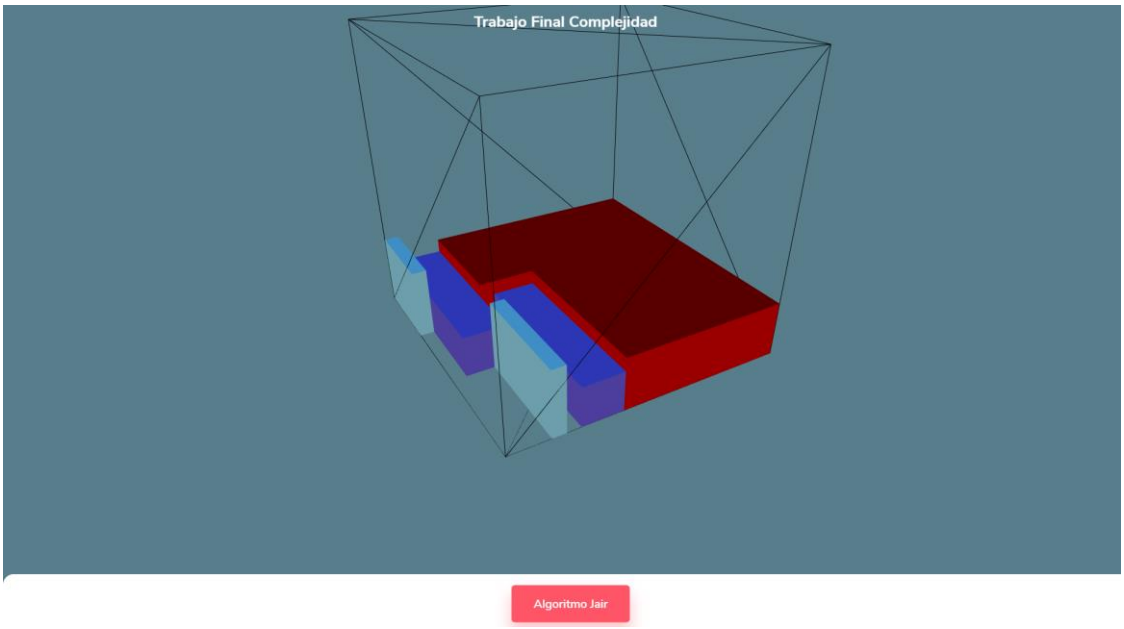
```
Empezar

        wContainer =  null

        hContainer <- null

        dContainer <-  null

        cubosArray [] = nul

        Proceso IngresarDatos()

                wContainer <- input
                hContainer <- input
                dContainer <- input

                Si wContainer < 1 ||  hContainer < 1 || dContainer < 1
                        print "Las dimensiones tiene que ser mayor a 0"
                Si no:
                        nTiposCubos <- input

                        Si nTiposCubos < 1
                                print "Tienes que elegir al menos un tipo de
cubo"
                        Si no
                                nCubosTDeTipo <- input

                                Si nCubosTDeTipo < 1

                                        print "Tienes que insertar al menos
un cubo de tipo n"

                                Si no
                                        wCube <- input
                                        hCube <- input
                                        dCube <- input

                                        Si wCube < 0 || hCube || dCube
                                                print "Los cubos deben
tener dimensiones mayor a 1"
                                        Si no
                                                nuevoCubo.w <- wCube
                                                nuevoCubo.h <- hCube
                                                nuevoCubo.d <- dCube


        cubosArray.añadir(nuevoCubo)

        Funcion  OrdernarCubosPorAnchoPorProfundidad()

                cubosArray.sort()

        Funcion jumpLineZ(arrCubes, lastCubeTracked)
                found = lastCubeTracked.d / 2 + lastCubeTracked.z
```

```
                i = 0

                Mientras i<arrCubes.size()
                        Si lastCubeTracked.z < arrCubes[i].z
                                found = arrCubes[i].z + arrCubes[i].d / 2
                                Terminar

                retornar found


        Funcion findSpaceUpLevel(arrCubes,lastCubeTracked)

                newHeight = lastCubeTracked.y + (lastCubeTracked.h  / 2)
                i = 0
                Mientras i < arrCubes.size()
                        Si newHeight < arrCubes[i].y + lastCubeTracked.h / 2
                                newHeight = arrCubes[i].y + arrCubes[i].h /
2

        Funcion PosicionarCubos()
                Tracker_x = -wContainer / 2
                Tracker_z = -dContainer / 2
                Tracker_y = 0

                i = 0

                Mientras i < cubosArray.size()



                        Si Tracker_x +  cubosArray[i].w > wContainer / 2
                                Tracker_x = -wContainer / 2
                                Tracker_z =
jumpLineZ(cubosArray,cubosArray[i])

                        Si Tracker_z >= dContainer / 2
                                Tracker_y =
findSpaceUpLevel(cubosArray,cubosArray[i])
                                Tracker_y = Tracker_y + 0.1333
                                Tracker_x = -wContainer / 2
                                Tracker_z = -dContainer / 2

                        cubosArray[i].setPosition(Tracker_x +
cubosArray[i].w/2, Tracker_y + cubosArray[i].h / 2, Tracker_d +
cubosArray[i].d / 2)

                        Tracker_x = Tracker_x + cubosArray[i].w




        Funcion Algoritmo()

                OrdernarCubosPorZ()

                PosicionarCubos()
Terminar
```

4. **Análisis**

   El resultado del pseudocódigo aplicado a un conjunto de cubos y un contenedor es el ordenamiento de manera óptima de todos ellos dentro del mismo contenedor





5. Comunicación / Argumentación

Como resultado del algoritmo, los cubos se ordenan en dirección Z, puesto que al iniciar el algoritmo el conjunto de cubo se ordena por profundidad x ancho. Para lograr dicho resultado mostrado en las figuras anteriores se ha tomado en cuenta varias validaciones tales como, que el siguiente cubo por insertar no se le asigne una posición fuera del contenedor. La técnica de búsqueda usada en este proyecto es Fuerza Bruta, la cual tiene mayor consumo de recursos de CPU ya que calcula todas las posibles soluciones.