Análisis Quinto Punto

Para solucionar el quinto punto inicialmente se utiliza el algoritmo de ordenamiento MergeSort para obtener un arreglo ordenado de las fechas de finalización de cada rodal.

Posteriormente con cada elemento del arreglo (que contiene cada rodal) verifico si las fechas no se solapan con los elementos siguientes del arreglo (fecha final comparada con la fecha inicial del siguiente rodal), y también se utiliza una función llamada reverse, con lo cual se verifica que las fechas del rodal seleccionado no se solapen con cada uno de los elementos anteriores (fecha de inicio con la fecha final del elemento anterior) y se genera una lista con todos los elementos compatibles almacenada en el vector de tipo Rodal “MayorNumerodeArbolesTemporales”.

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal “MayorNumerodeArbolesTemporales” se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles, además seo obtiene el total de personas que serán empleadas y se almacena en la variable tmpNumeroEmpleados, luego se valida que no supere el umbral de arboles que se pueden talar y la cantidad de empleados este por encima del umbral mínimo, si se cumplen estas restricciones entonces se compara con la variable “MayorNumerodeArbolesTemporal” (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que “tmpNumerodeArboles” sea mayor a “MayorNumerodeArbolesTemporal”, en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal “MayornumerodeArboles”, de lo contrario se queda con la ya almacenada anteriormente.

Dado que se puede dar el caso que las mejores soluciones están en los extremos y hasta el punto anterior estas no son tomadas dentro de las opciones(esto debido a que el total de beneficios ya esta próximo a alcanzar el umbral o ya llego a el) es necesario crear una copia del vector original que contiene los rodales y utilizar de nuevo el merge sort para realizar un ordenamiento esta vez no por la fecha de finalización, sino por el numero de arboles que contiene cada rodal(orden descendente), con lo anterior utilizamos de nuevo los algoritmos MaximoNumeroRodales y MaximoNumeroRodalesReverse y con cada elemento del arreglo (que contiene cada rodal) verifico si las fechas no se solapan con los elementos siguientes del arreglo (fecha final comparada con la fecha inicial del siguiente rodal), y con la función llamada reverse se verifica que las fechas del rodal seleccionado no se solapen con cada uno de los elementos anteriores (fecha de inicio con la fecha final del elemento anterior) y se genera una lista con todos los elementos compatibles almacenada en el vector de tipo Rodal “MayorNumerodeArbolesTemporales”.

Con cada una de las listas de rodales compatibles almacenadas en el vector de tipo Rodal “MayorNumerodeArbolesTemporales” se obtiene el beneficio total que se obtendrá al talarlos(la suma de numero de arboles que contiene cada rodal) y se almacena en una variable tmpNumerodeArboles, además seo obtiene el total de personas que serán empleadas y se almacena en la variable tmpNumeroEmpleados, luego se valida que no supere el umbral de arboles que se pueden talar y la cantidad de empleados este por encima del umbral mínimo, si se cumplen estas restricciones entonces se compara con la variable “MayorNumerodeArbolesTemporal” (la cual contiene el mejor beneficio que se ha obtenido hasta el momento) mientras que “tmpNumerodeArboles” sea mayor a “MayorNumerodeArbolesTemporal”, en caso de ser así esta lista pasaría a ser la mejor opción y se reemplazaría en el vector de tipo Rodal “MayornumerodeArboles”, de lo contrario se queda con la ya almacenada anteriormente.

Por ultimo al haber obtenido la mejor solución se pasa a guardar en un archivo plano la lista que este en el vector de tipo Rodal “MayornumerodeArboles” (teniendo en cuenta que el objeto contiene el numero del rodal, numero de arboles, de empleados y las fechas).

Los pasos que utilice para resolverlos

* Ordenamiento con MergeSort. ? **O(nlgn)**
* For(recorrer arreglo de elementos) Compljida total **O(n2)**
  + funcionalidades.MaximoNumeroRodales  **O(n)**
  + funcionalidades.MaximoNumeroRodalesReverse  **O(n)**
* comparación para saber si es el mayor elemento.
* Comparación()  **O(n)**

De esta forma determino que la complejidad del algoritmo es de **O(n2)**

La complejidad de esta solución

Es determinado por la complejidad del algoritmo de ordenamiento MergeSort la cual es de **O(nlgn).** Sumado a la complejidad de recorrer los elementos previamente organizados para su selección que obtenemos una complejidad de **O(n)** encada selección pero al realizarlo en un ciclo **for** se convierte en **O(n2)** elemento de esta forma determina que nuestra solución tiene una complejidad total de **O(n2).**

Uso de aplicativo

Es un proyecto de Netbeans para su ejecución solo es necesario construir el ejecutable e iniciar la ejecución. **”Triángulo de reproducción”**



Al ejecutarse despliega una pantalla para la selección del archivo que se va a tomar como entrada, la estructura del archivo de entrada cumple con las especificaciones del documento del proyecto.

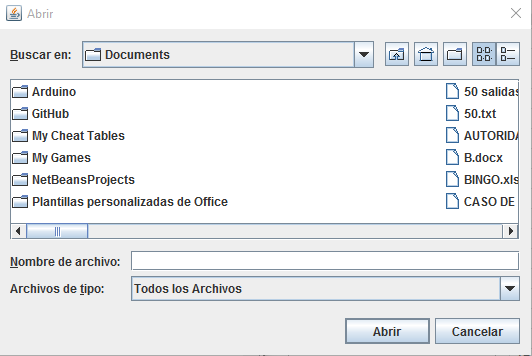


Ilustración 1

Posteriormente pide que se le indique un directorio para el almacenamiento de los resultados

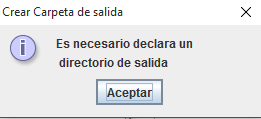


Ilustración 2

Se selecciona la ruta donde se va a realizar el almacenamiento de los resultados.

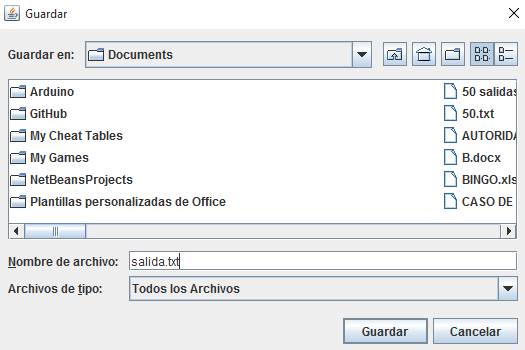


Ilustración 3

Al finalizar todos los procedimientos Retorna un mensaje de finalización.

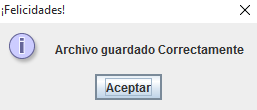


Ilustración 4

Análisis de complejidad.

Se realizaron pruebas de rendimientos con múltiples entradas que fueron obtenidas con la construcción de un algoritmo de generación de entradas de forma aleatoria, Este aplicativo está disponible con el nombre **“Generador de prueba de entrada “**como proyecto de netBeans los archivos de prueba se encuentra en la carpeta” **PruebasInput”.**

Tabla de rendimiento

|  |  |  |
| --- | --- | --- |
| **Nombre de Archivo** | **Numero de registro de entradas** | **Tiempo Milisegundos** |
| 50 registros | 50 | 532 |
| 150 registros | 150 | 672 |
| 500 registros | 500 | 5485 |
| 1600 registros | 1600 | 14891 |
| 2999 registros | 2999 | 33579 |
| 5200 registros | 5200 | 33157 |

Ilustración 5

Entorno de pruebas

Procesador: Amd E1-2500 1.40Ghz.

Memoria: 4GB Ddr3.

Disco duro: 500 Gb 5400 rpm.

Sistema Operativo: Windows 10 Pro 64bit.