Padrón de Cáceres

EDI 17-18

Estudiante:Mario Bermejo Sánchez
DNI:76063621-E
Grupo de prácticas:Grupo-01
Profesor de prácticas:Mariscal

Estudiante:Enrique Moreno Ávila
DNI:76047833-N
Grupo de prácticas:Grupo-01
Profesor de prácticas:Mariscal

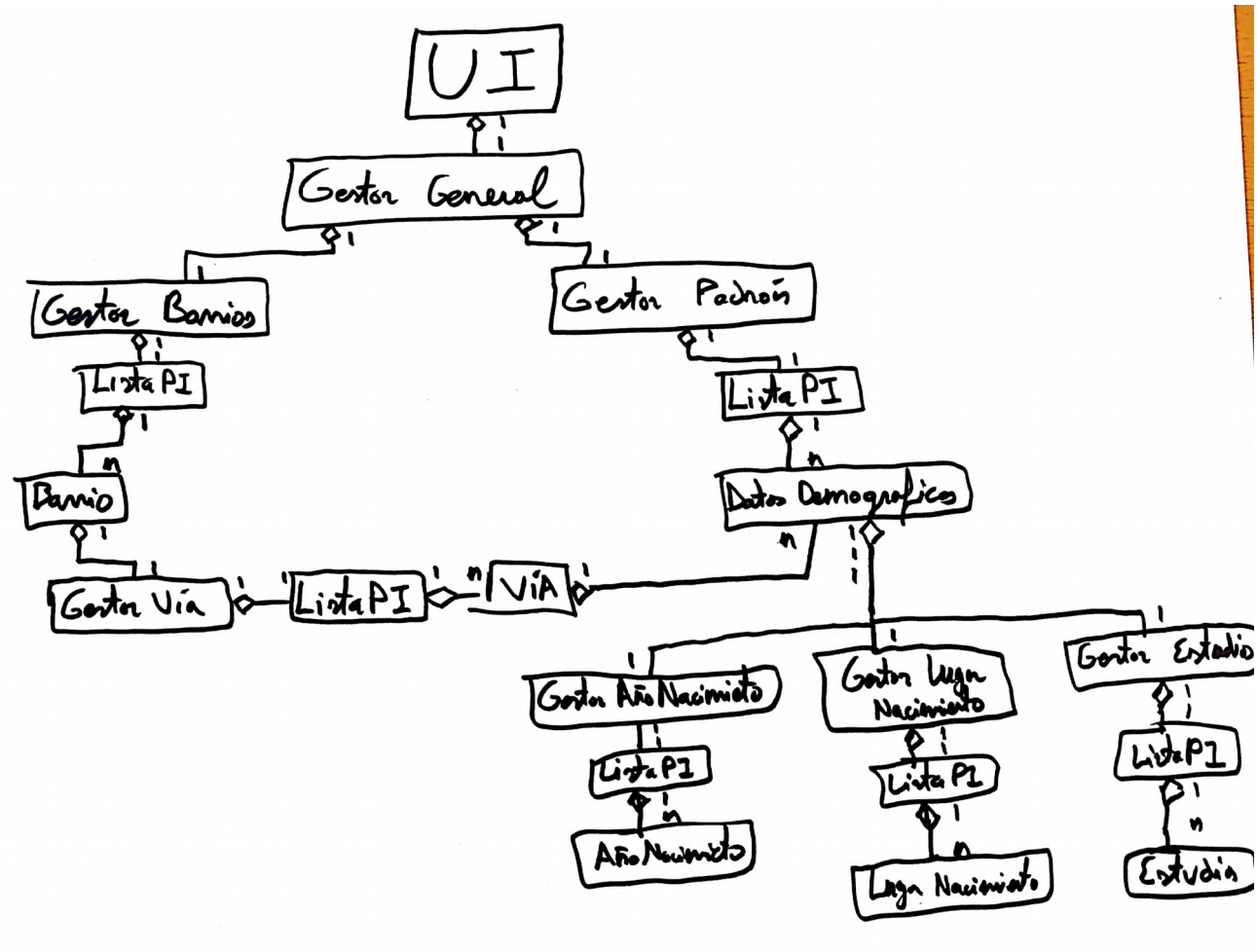
Fecha de entrega:20/05/2018
Convocatoria:Mayo-Junio

Contenido

Descripción del problema y la solución

El diseño y posterior implementación de la aplicación proporcionada surge debido al problema de tener que almacenar y tratar los distintos datos del ayuntamiento de Cáceres, dichos datos son los barrios, vías y datos del padrón de cada vía.

La solución fue diseñar dicha aplicación la cuál, almacena los datos en distintos tipos de estructuras como pueden ser listas y árboles binarios. Una vez almacenados, se procede a tratarlos haciendo operaciones sobre ellos, tales como mostrar las vías de un barrio, el barrio con mayor población de hombres y el barrio con mayor población de mujeres...
Esta aplicación en rasgos generales consta de 12 algoritmos básicos, uno de ellos siendo el primero, la carga de todos los datos en la estructura.



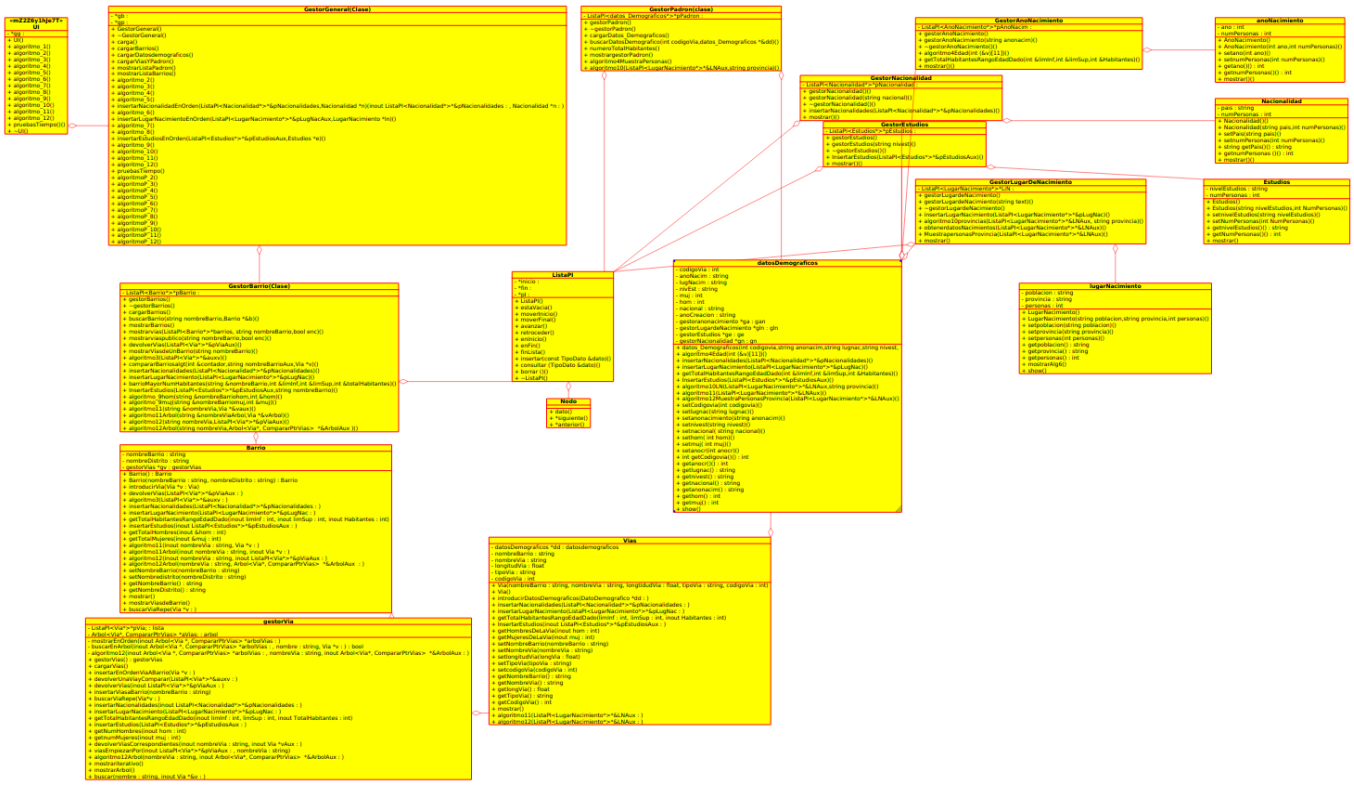
Diseño estructural

Describe los pasos que has dado para llegar al diseño del proyecto.
Incluye el diagrama de clases totalmente detallado. Recuerda que el diagrama de clases UML debe corresponderse con la implementación que hayas hecho del proyecto.

Para el diseño de esta aplicación, hemos comenzado ideando como sería la estructura desde un nivel más alto, hasta el más bajo de todos.
En el nivel más alto, se encuentra la interfaz de usuario proporcionada por los profesores, a partir de ahí, hemos enlazado esta con un gestor general donde se realizarán los algoritmos y la carga de datos en sus respectivas estructuras.
Este gestor general está compuesto por un gestor de barrios y un gestor de padrón. Por los cuales ambos se puede llegar al nivel más bajo de la estructura.
El gestor padrón estará compuesto por una lista la cual almacenará todos los datos demográficos de cada vía.

A su vez esta clase datos demográficos se divide en 3 subclases para poder dividir estos datos y poder tratarlos de forma correcta. La división está formada por los gestores año nacimiento, lugar nacimiento y estudios.
Cada gestor posee su lista almacenando todos estos datos en ella.
Por otro lado, por el camino del gestor barrio, este almacena una lista con todos los barrios de Cáceres, cada barrio, compuesto por un gestor de vías, el cual al igual que todos los demás, almacena una lista de vías, cada vía a su vez, sus datos demográficos.

Todos estos gestores se crean con todas las listas y datos vacíos, para llenarlos, habría que ejecutar el algoritmo uno de carga, el cual daría lugar a la ejecución de todos los submétodos de carga.
Estos métodos de carga leen los archivos proporcionados en formato csv y los almacena en sus correspondientes listas.
En esta carga también se realiza la unión de la vía con sus correspondientes datos demográficos según el código de vía.



Implementación del proyecto

Implementación del proyecto

Las estructuras de datos utilizadas son listas y árboles, hemos usado estas estructuras debido a la comodidad, sencillez y eficiencia de su uso. Estas estructuras son: Una lista de barrios, listas de vías incluidas en cada barrio, un puntero que direcciona a un objeto datos demográficos en cada vía, y por último, cada dato demográfico se ha dividido en tres listas almacenando los distintos datos de año de nacimiento, lugar de nacimiento y estudios.

Esta aplicación consiste en un gestor general, en el cuál se han realizado los distintos algoritmos que debíamos obtener junto con otro método de pruebas, también están otros distintos módulos para leer y cargar datos.
Este gestor se divide en dos gestores almacenadores de listas para subsanar los problemas de almacenamiento de los datos, el primero, el gestor de barrios, hecho con el fin de contener una lista que almacenara los barrios, esta lista, se carga con datos en el propio gestor con un módulo diseñado para leer el archivo de barrios e ir guardándolos.
Cada clase barrio, contenedora de los propios datos característicos de cada barrio y de sus métodos correspondientes de para tratarlos tales como get y set. También, posee un puntero a

un gestor vías, este gestor, diseñado como el gestor barrio, almacenara una lista de vías, en este gestor a diferencia del gestor barrios no se realiza la carga de las vías, esta, se realiza en el gestor general, debido a que cada vía tiene un puntero a sus datos demográficos, por lo tanto, en el gestor general se realizara la lectura del fichero de vías y se leerá el código de la vía para así poder comprobar el mismo código en el gestor padrón, y poder relacionar a cada vía con su dato demográfico.
La clase vía, al igual que la clase barrio, posee sus datos propios de la vía ,sus métodos modificadores y métodos varios de paso de los algoritmos, junto al puntero a datos demográficos mencionados anteriormente.

Por otra parte del gestor general sale el ya mencionado gestor padrón, que simplemente es un lista de datos demográficos, la carga de esta lista se realiza en el gestor padrón, y además este gestor contendrá los diferentes métodos de trabajo sobre dicha lista.
Los datos demográficos, es la clase usada para, al igual que barrio y vía, almacenar sus datos característicos y sus métodos operacionales como los set y get.

Para que el uso de esta clase resultara más sencillo, la hemos dividido en otras tres. Para así tener los datos que nos pedían en los algoritmos más a mano, estas tres clases en un principio no las habíamos incluido, por lo que al no poder disponer de los datos en su tipo correspondiente y de forma correcta almacenada, los algoritmos se nos complicaban mucho.
Una vez corregimos este problema y metimos estas tres clase, las cuales simplemente son 3 listas con sus datos correspondientes almacenados para cada vía ,los problemas fueron más fáciles de resolver.

Las últimas tres clases, contenidas en estos tres gestores mencionados, al igual que barrio, vía y datos demográficos dichos anteriormente, cada una contiene sus datos propios característicos, los métodos modificadores, y algunos métodos usados en los algoritmos.

Batería de pruebas

Para las pruebas hemos creado un módulo en el gestor general el cual ejecuta todos los algoritmos de seguido con unos parámetros indicados por defecto. Con esta ejecución mostramos por cada uno el tiempo que tarda en realizarse junto con los datos conseguidos con los parámetros introducidos.
Por otra parte, hay un tad “PRUEBASclases” en el que hemos realizado una serie de pruebas cuando hemos ido creando cada una de las clases con sus modulos para comprobar su eficiencia, y evitar así, cualquier error inesperado durante el proyecto. Cabe decir, que gracias a estas pruebas nos ayudaron a detectar múltiples errores, que probablemente si hubiesemos hecho los módulos y algoritmos sin dichas pruebas, no hubiesemos detectado los errores.
Así, llegamos a la conclusión, de que las pruebas son eficientes, ya que nos ayudan a crear módulos o algoritmos eficientes y correctos, y además, ofrecer una garantía al usuario de que dicho programa funciona correctamente.

Evaluación del diseño

Eficiencia del algoritmo X

Algoritmo 1:
En este algoritmo,para realizar la carga de datos, una vez se ejecuta lo que sucede es que se crea un objeto gestor general, este objeto, en su constructor, tiene, la creacion de los objetos gestor de barrio y gestor de padron, ademas, una llamada a un modulo llamado carga, este modulo,realiza una llamada a los distintos modulos de carga de datos que leen los ficheros y posteriormente va introduciendo los objetos en sus respectivas listas. De este modo toda la estructura queda completamente cargada con todos los datos, a su vez relacionados según el diseno de la aplicación.

Algoritmo 2:
En este algoritmo, se pide un barrio por consola, una vez introducido, se procede a, en el gestor barrios, buscar en la lista el barrio deseado y una vez se obtiene, se muestra por consola su gestor de vias, es decir, la lista de vias perteneciente al barrio.

Tabla Algoritmo 2: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	1 dato a introducir para su funcionionamiento.	----------
Ordenación	Requiere de ordenacion por parten de las vías, deben mostrarse por orden alfabético.	----------
Complejidad	O(1)	----------
Mejor caso	Que el barrio se encuentre al inicio de la lista.	----------
Caso promedio	Que el barrio se encuentre en la mitad de la lista.	----------
Pero caso	Que el barrio sea el último de la lista.	----------
Excepciones	Introducir un barrio que no sea válido	----------

Algoritmo 3:
La solucion propuesta a este algoritmo es, mediante una lista auxiliar de vias, llenarla de vias sin repetir, que no esten en varios barrios, para ello se van sacando vias y comprobando si el nombre de la via ya estaba introducida en la lista auxiliar,si no lo esta, se inserta.
Una vez obtenida esta lista auxiliar, lo que se hace es ir recorriendo la lista auxiliar sacando vias una a una. Con cada via, se pasa al gestor barrio,en el que se realiza una comprobacion barrio a barrio, si no se encuentra ninguna via igual a la pasada en el barrio, no hace nada, si no, si se encuentra la primera vez simplemente se almacena el nombre del barrio en el que se ha encontrado la via igual, y se sigue buscando en los barrios posteriores, si se vuelve a encontrar, ya muestra por consola la via que es, junto con el primer barrio encontrado, y el segundo en el que se ha vuelto a encontrar la misma via, como la busqueda sigue hasta el final de la lista de barrios, simplemente es ir comprobando si se vuelve a repetir esa via en mas barrios para asi sacar el nombre del barrio y mostrarlo por consola.

Tabla Algoritmo 3: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	4 datos para su minimo funcionamiento, no hay que introducir nada, devuelve todas las vias.	----------
Ordenación	No requiere de ordenación	----------
Complejidad	O(n).	
Mejor caso	Que las vias se encuentren al inicio de la lista	----------
Caso promedio	Que las vias se encuentren en la mitad de la lista.	----------
Pero caso	Que las vias se encuentren las últimas en la lista.	----------
Excepciones	Que las listas no hayan sido cargadas, y por tanto no se puedan buscar las vías.	----------

Algoritmo 4:
En este algoritmo, se ha ideado un vector el cual esta dividido en que cada posicion, sea un rango de edad de 10 anos. Mediante el gestor padron, se va leyendo la lista de datos demograficos. Para cada dato demografico obtenido de la lista, se accede a su gestor de anos de nacimiento, una vez ahi, se van obteniendo las diferentes fechas de nacimiento, y se realizan diversas operaciones para obtener la edad hoy en dia, y para obtener la posicion del vector en la que sera colocada. Todo esto se repite, sumandose el numero de personas que hay de cada edad en cada posicion del vector y se muestra por consola.

Tabla Algoritmo 4: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	2 datos para su minimo funcionamiento, ninguno a introducir, devuelve todas las edades en un rango.	----------
Ordenación	No requiere de ordenación.	----------
Complejidad	O(1)	----------
Mejor caso	Que las edades se encontrasen todas ordenadas.	----------
Caso promedio	Que las edades no estuviesen demasiado desordenadas.	----------
Pero caso	Que las edades estuviesen del todo desordenadas	----------.
Excepciones	Que las listas no hayan sido cargadas, y por tanto no se puedan buscar las edades.	----------

Algoritmo 5:
Aqui hemos usado dos listas auxiliares de nacionalidad, en la primera, se obtienen todas las nacionalidades sin ningun tipo de orden de cada via, se almacenan y se devuelve la lista.
Esta lista posteriormente se va recorriendo creando objetos nuevos pero con los mismos atributos de cada nacionalidad y se van insertando en orden en la otra lista, de este modo conseguimos que al sumar el numero de habitantes de una misma nacionalidad, al sumarse no se modifique el objeto nacionalidad original. Una vez hecho esto, se muestra esta segunda lista por consola.

Tabla Algoritmo 5: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	4 datos a utilizar para su mínimo funcionamiento, ninguno a introducir, y resultados por consola	----------
Ordenación	Requiere de mostrar las nacuinalidades ordenadas de forma descendente por número de habitantes	----------
Complejidad	O(n)	----------
Mejor caso	Que las nacionalidades de Cáceres se encuentren al inicio de la lista.	----------

Caso promedio	Que las nacionalidades de Cáceres se encuentren en el medio de la lista.	----------
Pero caso	Que las nacionalidades de Cáceres se encuentren de las últimas en la lista.	----------
Excepciones	Que las listas no hayan sido cargadas, y por tanto no se puedan buscar las nacionalidades.	----------

Algoritmo 6:
Este algoritmo es muy similar al anterior,teniendo dos listas, primero se obtienen todos los lugares de nacimiento de cada via, una vez hecho esto se van sacado de esta lista para introducirlos en la otra de forma que se cree un nuevo objeto para no modificar el objeto original en la suma de datos de los habitantes, y ademas se insertan tambien en el orden deseado. Posteriormente se muestra esta segunda lista por consola.

Tabla Algoritmo X: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	4 datos para su minimo funciionamiento, ninguno introducido por consola, se devuelve por consola.	----------
Ordenación	Requiere de habitantes ordenados alfabéticamente por provincia	----------
Complejidad	O(n)	----------
Mejor caso	Que las provincias se encuentren al principio de la lista.	----------
Caso promedio	Que las provincias se encuentren en la mistad de la lista.	----------
Pero caso	Que las provincias se encuentren de las últimas en la lista.	----------
Excepciones	Que las listas no hayan sido cargadas, y por tanto no se puedan buscar las provincias.	----------

Algoritmo-7.

-Ahora, vamos a hablar sobre el algoritmo 7. Así en este algoritmo se trataba de aportarle al usuario el barrio con el mayor numero de personas indicado según el rango de edad introducido por el usuario y mostrarlo por consola.
Así lo primero, fue declarar aquellas variables con las que trabajar ese rango de edad(un límite superior y otro inferior), y por supuesto aquella variable que guardase el total de todas esas personas. Más tarde, se le pide al usuario que meta el límite inferior, y poco después el superior. Una vez introducidos, se crean varios métodos que nos permiten

descender desde la lista de barrios, que va obteniendo el nombre del barrio del que se está haciendo cálculos, y poco después, nos procedemos a descender, pasando por la clase barrio, después por la clase que contiene la lista de vias, luego por la clase vía, accedemos a su dato demográfico, y por último, llegamos a la clase que contiene la lista de años de nacimiento, y allí, habiendo pasado los parametros de límite superior e inferior y una variable de habitantes auxiliar por entrada salida en todos los métodos empleados, nos procedemos a calcular la edad de todos los datos de la lista, y si cumple el limite establecido añadimos ese número de personas a la variable habitantes auxiliar creada previamente en la lista del barrio. Y una vez que terminamos de calcular todas esas personas volvemos a la lista de barrios y allí, sumamos estas personas al total del numero de personas que tenemos que devolver, y volvemos a consultar de nuevo otro barrio y procedemos a hacer lo mismo de nuevo, sin embargo, comparamos con los cálculos realizados del anterior barrio para ver si supera al numero de personas y hubiese que devolver este. Al final del todo, se devuelve el nombre del barrio que más número de personas tienen con ese rango de edad.

Cabe destacar, que solo hemos utilizado las estructuras de listas, habiéndolas cargado previamente, para que nos devuelvan la información correctamente.

Tabla Algoritmo 7: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	Se necesita el límite superior e inferior de edad, el nombre barrio a devolver, las variables de habitantes auxiliar y no auxiliar(total), con lo cuál 5 variables mínimo salvo aquellas empleadas en los demás métodos para su cálculo. Solo se devuelve un dato.	----------
Ordenación	No requiere de ordenación.	----------
Complejidad		----------
Mejor caso	Que sea el primer barrio el que tiene el mayor número de personas, según rango.	----------
Caso promedio	Que el barrio se encuentre en la mitad de la lista, el que tiene el mayor número de personas, según rango.	----------
Pero caso	Que sea el último barrio de la lista, el que tiene el mayor número de personas, según rango.	----------
Excepciones	Que el rango de edad no sea válido.	----------

Algoritmo-8.

En este algoritmo se nos pide que creemos un fichero con el nivel de estudios de todos los ciudadanos de un barrio y el número de habitantes para cada uno de los niveles de estudio, y además ordenados por el número de habitantes de cada uno de esos niveles.
Lo primero que hacemos es declarar todas aquellas variables que nos van a hacer falta, el nombre del barrio, el estudio, el nivel de estudio, el número de personas con ese nivel de estudios, las listas auxiliares para coger aquellos niveles de estudios que nos interesen y

ordenarlos. Lo segundo, se trata de pedirle al usuario el barrio del que quiera que creemos el fichero, y procedamos a acceder a la lista de barrio para ir buscando ese barrio, y una vez obtenido, y que sea el barrio que buscamos, descender hasta la lista de los estudios con una lista de estudios auxiliar, y proceder a meter en la lista auxiliar las personas con cada nivel de estudios, sumando aquellas personas que tengan el mismo nivel de estudios. Una vez hecho eso, volvemos a nuestro algoritmo, y procedemos a ordenar esa lista auxiliar según el número de personas de cada nivel de estudio, empleando otra lista auxiliar, cabe destacar que este algoritmo podría ser mejorable, creando solo una lista auxiliar y ordenándola allí mismo, pero por falta de tiempo y a su vez por no extender demasiado el código, decidimos emplearlo de esta manera que aunque gaste un poco más de memoría, es eficiente. Una vez ordenada, creamos el archivo dónde vamos a almacenar toda la información, poniéndole el nombre del correspondiente barrio y pasaremos todos los datos obtenidos de la lista a este archivo .

Cabe destacar, que solo hemos utilizado las estructuras de listas, habiéndolas cargado previamente, para que nos devuelvan la información correctamente, las estructuras auxiliares empleadas han sido listas, por su fácil uso.

Aquí tenemos una tabla para resumir un poco el algoritmo:

Tabla Algoritmo 8: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	14 como mínimo para su correcto funcionamiento. 1 dato introducido, el nombre del barrio, y devuelve un archivo.	----------
Ordenación	Se requiere ordenar el archivo de níveles de estudios según el número de habitantes de mayor a menor.	----------
Complejidad		
Mejor caso	Que el barrio que se busca se busca se encuentre en primer lugar en la lista.	----------
Caso promedio	Que el barrio que se busca se encuentre en la mitad de la lista.	----------
Pero caso	Que el barrio que se busca se encuentre en el último lugar de la lista.	----------
Excepciones	Nombre de barrio introducido inválido	----------

Algoritmo-9.

En este algoritmo se debe de mostrar por consola el barrio con mayor número de hombres y el barrio con mayor número de mujeres.
Lo primero que debemos hacer es declararnos aquellas variables que nos hagan falta para el funcionamiento del algoritmo, dos parametros tipo int para devolver el número de hombres y mujeres, y dos parametros tipo string para devolver el nombre del barrio de los hombres y de las mujeres. Lo segundo, será acceder a la lista de barrios y obtener el primer barrio y calcular el número de hombre y de mujeres, para ello descenderemos por la lista de vias, por todas las vias que haya dentro de la lista y accediendo a los datos demográficos, con parametros de entrada-salida para guardar los datos, de cada una

obtenemos el número de hombres y mujeres, y los sumamos, a continuación actualizamos aquellas variables que vamos a devolver en el algoritmo, de tal manera que obtenemos todos los barrios de la lista y si hubiese alguno con mayor número de hombres o mujeres, se actualizarían cada una de las variables, tanto de número como de nombre.

Cabe destacar, que solo hemos utilizado las estructuras de listas, habiéndolas cargado previamente, para que nos devuelvan la información correctamente.

Aquí tenemos una tabla para resumir un poco el algoritmo:

Tabla Algoritmo 9: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	Se utilizan como mínimo 4 datos para su funcionamiento, y se devuelven esos 4 datos por consola.	----------
Ordenación	No requiere ordenación.	----------
Complejidad		----------
Mejor caso	Que el barrio con mayor número de hombres y mujeres, se encuentren al principio de la lista.	----------
Caso promedio	Que el barrio con mayor número de hombres y mujeres, se encuentren en la mitad de la lista.	----------
Pero caso	Que el barrio con mayor número de hombres y mujeres, se encuentren de los últimos en la lista.	----------
Excepciones	Que no se calcule bien dichas personas, por no haber cargado las listas previamente.	----------

Algoritmo-10.

Aquí, tenemos que crear un algoritmo que cree un archivo y contenga el nombre de la población y el número de personas que proceden de una provincia dada por el usuario por consola.
Lo primero que haremos será declararnos nuestras variables para el funcionamiento mínimo del algoritmo, una lista auxiliar de lugares de nacimientos, un objeto lugar de nacimiento, y un string provincia. Una vez que el usuario nos escribe una provincia procederemos a entrar en la lista de todos los datos demográficos, dónde descenderemos hasta la lista de lugares de nacimiento,con parametros de entrada-salida lista auxiliar lugar de nacimiento y la provincia. Así, meteremos en esta lista auxliar aquellos lugares de nacimiento que tengan la misma provincia, y ya de paso, una vez vayamos a meter el lugar de nacimiento allí dónde haya el mismo nombre de la población se sumen las personas, y no repitamos la misma población una y otra vez. Luego, una vez acabado volveremos a nuetro algoritmo, dónde crearemos el archivo con el nombre de la provincia introducida, y pasaremos todos los datos obtenidos de la lista a este archivo.

Cabe destacar, que solo hemos utilizado las estructuras de listas, habiéndolas cargado previamente, para que nos devuelvan la información correctamente, las estructuras auxiliares empleadas han sido listas, por su fácil uso.

Aquí tenemos una tabla para resumir un poco el algoritmo:

Tabla Algoritmo 10: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	4 datos como minimo para su funcionamiento, se devuelve 1 dato.	----------
Ordenación	No requiere ninguna ordenacion.	----------
Complejidad		----------
Mejor caso	Que la provincia se encuentre al principio de la lista.	----------
Caso promedio	Que la provincia se encuentre en el medio de la lista.	----------
Pero caso	Que la provincia se encuentre sea la última de la lista.	----------
Excepciones	Introducir una provincia inválida.	----------

Algoritmo-11.

En este algoritmo se trata de crear un archivo con los lugares de nacimiento de las personas de una via, introducida por un usuario.
Lo primero que haremos será declararnos las variables que nos ayuden al funcionamiento mínimo del algoritmo,dos objetos via auxiliar uno para la lista y otro para el arbol, dos strings un nombrevia para la lista y otro para el arbol, una lista auxiliar de lugares de nacimiento y un objeto de lugar de nacimiento.
Lo segundo será entrar en la lista de barrio e ir buscando en las listas de vias de cada barrio para encontrar la via, pasando como entrada salida el nombre de la via y el objeto via, ya que una vez encontrada esa via la devolvamos con toda su información. Poco después accederemos a los datos demográficos de esa vía para devolver los lugares de nacimiento de esa vía, incluyéndolos en la lista auxiliar de lugares de nacimiento que hemos pasado como parametro de entrada salida. Para terminar, crearemos el archivo con el nombre de la vía que hemos introducido y todo el contenido de la lista auxiliar se cargará en el archivo.
Cabe destacar que este era un algoritmo que se debía hacer con listas y árboles, y tenemos los siguientes comentarios…
Tenemos que para ambas se tiene una complejidad O(n), para la vía “San Juan” encontrar esa via le ha supuesto 0.000510454 segundos y para el árbol 0.00014773, con lo cuál la diferencia de búsqueda es considerable, tan solo probado en la búsqueda para observar que la diferencia es notable.
Cabe destacar, que solo hemos utilizado las estructuras de listas, habiéndolas cargado previamente, para que nos devuelvan la información correctamente, las estructuras auxiliares empleadas han sido listas, por su fácil uso.

Aquí tenemos una tabla para resumir un poco el algoritmo:

Tabla Algoritmo 11: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles

Nº de datos	6 datos para el funcionamiento mínimo, y devuelve un archivo.	6 datos para el funcionamiento mínimo, y devuelve un archivo.
Ordenación	No requiere ordenación.	No requiere ordenamiento.
Complejidad	O(n)	O(n)
Mejor caso	Que se encuentre la vía al principio de la lista.	Que se encuentre la via en la raíz del árbol.
Caso promedio	Que se encuentre la vía en la mitad de la lista.	Que se encuentre la vía en la mitad del árbol.
Pero caso	Que se encuentre la vía la última de la lista.	Que se encuentre la vía la última del árbol.
Excepciones	Nombre de via mal introducido.	Nombre de via mal introducido.

Algoritmo-12.

En este algoritmo se trata de que enseñe todas las personas nacidas en una provincia, dada por el usuario, de todas las vias que comiencen por cualquier palabra, introducida también por el usuario.
Lo primero es declarar las variables como vienen siendo el string del nombrevia, de la provincia, las listas auxiliares y los árboles empleados…
Lo segundo es acceder a la lista de barrios y en cada barrio buscar en cada lista de vias el trozo de nombre de via que haya introducido el usuario para encontrar dichas vias, y una vez encontradas introducirlas en la lista auxiliar pasada como entrada y salida.
Después se trata de introducir los lugares de nacimiento de las vias introducidas en dicha lista auxiliar, en una lista auxiliar de lugares de nacimiento. Al final, se se muestra por consola el número de personas junto con la provincia.
En este algoritmo también son necesarios los árboles y tenemos los siguientes comentarios…
Ambos tienen una complejidad O(n), y cada uno en buscar las vias que empiezan por “ Virgen“ y provincia ”Caceres” tarda en la lista 0.00056076 y en el árbol 0.000537872, con lo cual podemos apreciar que es una diferencia muy pequeña de búsqueda.

Cabe destacar, que solo hemos utilizado las estructuras de listas, habiéndolas cargado previamente, para que nos devuelvan la información correctamente, las estructuras auxiliares empleadas han sido listas y árboles, para comprobar su diferencia.

Aquí tenemos una tabla para resumir un poco el algoritmo:

Tabla Algoritmo 12: Descripción del contenido de la tabla

	Tiempos de ejecución en EDL	Tiempos de ejecución en árboles
Nº de datos	6 datos para el funcionamiento mínimo, y devuelve cosas por consola.	6 datos para el funcionamiento mínimo, y devuelve cosas por consola.
Ordenación	No requiere ordenación.	No requiere ordenamiento.
Complejidad	O(n)	O(n)
Mejor caso	Que se encuentren las vías al principio de la lista.	Que se encuentren las vias en la raíz del árbol.
Caso promedio	Que se encuentren las vías en	Que se encuentren las vías en la mitad

	la mitad de la lista.	del árbol.
Pero caso	Que se encuentren las vías las últimas de la lista.	Que se encuentren las vías las últimas del árbol.
Excepciones	Nombre de via y de la provincia mal introducido.	Nombre de via y de la provincia mal introducido.

Conclusiones

Conclusiones acerca del proyecto

Bueno por lo primero que vamos a empezar va a ser evaluando los tiempos de ejecución con la complejidad y la estructura de datos utilizada en cada algoritmo...

En el algoritmo 1, se trata de una complejidad O(1), ya que solo se trata de llamadas o bien a los métodos hechos en cada una de las listas o en el gestor general, el tiempo que le lleva en cargar todas las listas es de 0.135802, con lo cual no le lleva demasiado tiempo en cargarlas, así como que las estructuras que se han cargado han sido en su mayoría listas y en minoría árboles.

En el algoritmo 2, se trata de una complejidad O(1), ya que se trata de ir realizando llamadas a otros métodos creados en las diferentes listas y objetos, se emplea un tiempo de 39,5733 para mostrar las vías de Cánovas, y las estructuras utilizadas son listas.

En el algoritmo 3, se trata de una complejidad O(n), ya que se trata de descender a un método en el que hay que buscar n elementos, el tiempo empleado es de 0.194872, y las estructuras empleadas son listas.

En el algoritmo 4, se trata de una complejidad O(1), ya que solo se trata de ir realizando llamadas a otros métodos que hay en los objetos y listas, el tiempo empleado es de 0,0366235, y las estructuras empleadas son listas y vectores.

En el algoritmo 5, se trata de una complejidad O(n), ya que se trata de buscar en n elementos de la lista hasta encontrarlos, el tiempo empleado es de 0.0165687, las estructuras empleadas son las listas.

En el algoritmo 6, se trata de una complejidad O(n), ya que se trata de manipular elementos que para encontrarlos hay que pasar por n elementos, el tiempo empleado es de 0,0219789, las estructuras empleadas son listas.

En el algoritmo 7, se trata de una complejidad $O(n^2)$, porque requiere más de buscar n elementos, al igual que requiere manipular n elementos con lo cual sale esa complejidad, el tiempo empleado es de 3,07522, las estructuras empleadas son listas.

En el algoritmo 8, se trata de una complejidad O(1), porque solo requiere de llamadas a métodos de clases inferiores y listas, el tiempo empleado es de 3,73291, las estructuras empleadas son listas.

En el algoritmo 9, se trata de una complejidad O(1), porque solo requiere de llamadas a métodos de clases inferiores y listas, el tiempo empleado es de 0,000435114, las estructuras empleadas son listas.

En el algoritmo 10, se trata de una complejidad O(n), debido a que requiere de una búsqueda y hay que pasar por n elementos para encontrar dichos elementos, el tiempo empleado es de 3,37763 para la provincia Salamanca, las estructuras empleadas son listas.

En el algoritmo 11, se trata de una complejidad O(n), debido a que requiere de una búsqueda y hay que pasar por n elementos para encontrar dichos elementos, el tiempo empleado para búsqueda en la lista es de 0.000221968 y para el árbol $5{,}07832*10^{-5}$, y el tiempo empleado para todo es de 9,5269, las estructuras empleadas han sido listas y árboles.

En el algoritmo 12, se trata de una complejidad O(n), debido a que requiere de una búsqueda y hay que pasar por n elementos para encontrar dichos elementos, el tiempo empleado para búsqueda en la lista es de 0.000549793 y para el árbol 0.00071763 , y el tiempo empleado para todo es de 9,50624, las estructuras empleadas han sido listas y árboles.

Cabe destacar que si en algún algoritmo tarda demasiado tiempo es porque no sabíamos que nombre introducir y hemos consultado el archivo csv, y hasta que lo hemos introducido.

Hasta aquí la comparación entre los algoritmos, ahora las conclusiones que sacamos de todo esto…
La verdad que la implementación y diseño de este proyecto permite el funcionamiento correcto de casi todos los algoritmos, debido a que en alguno puede haber un pequeño uso más de memoría pero porque quizás hemos considerado esa opción para una mejor lectura del código. Pero por los demás puntos cabe destacar que el diseño no deja nada que desear, ya que tiene un procesamiento bastante rápido, observados en los algoritmos anteriormente comentados, y aunque hemos utilizado listas, que quizás con árboles podríamos hacer un pequeño incremento, podemos observar que en el algoritmo 12 no es la estructura más adecuada de búsqueda al tardar más que la lista, con lo cual podemos decir que los árboles para algunas cuestiones son más rapidos.

¿El por qué del uso de listas? Bueno aunque podría emplear varías listas comentando los pro y los contra respecto a otras estructuras, las listas nos proporcionaban mayor facilidad tanto a la hora de lectura de código, como en su utilización, cosa que por ejemplo en los árboles utilizados en el algoritmo 11 y 12, pues no eran tan fáciles de ver.
En cuánto al rendimiento podríamos decir que quizás alomejor si hubiese un patrón para quedar ordenadas las listas serían algo más rápidas, sin embargo muestran un rendimiento que nos dejaron fascinados al terminar todos los algoritmos.

Y esta sería la conclusión acerca de todos estos temas.

Opinión personal y conclusiones

Cuéntanos cómo has ido desarrollando la aplicación, por dónde empezaste, cómo tomaste las decisiones, qué has tenido que rehacer, las tareas que has realizado, el tiempo que has dedicado al proyecto, etc.
Cuéntanos qué te ha parecido el tema y el planteamiento del proyecto, ayúdanos a mejorar dando tu opinión para que podamos cambiar aquello que está mal en próximos cursos.

Pues empezando por los primeros puntos…
Lo primero que pensamos fue, vamos a ver que es lo que hemos aprendido y reunido en todas las clases y sesiones de laboratorio, a ver si nos dan una idea por dónde empezar, una vez que juntamos todo, decidimos por decantarnos por las listas por su fácil uso, y porque al dibujarlas en el cuadernos nos aclaraban bastantes dudas. Nos topamos con diversos problemas, y es que no sabíamos cuántas listas meter, muchos días estancados hasta que profesores nos aclararon dicha duda, y al final obtuvimos la estructura. Después, nos dispusimos a realizar cada uno de los algoritmos, que sin duda los que más problemas nos han dado han sido el 3 y cuando teníamos que implementar los árboles, pero logramos superar esos problemas, ya que eran problemas de comprensión de qué debía hacer la estructura. Cabe destacar que mientras haciamos esto, ibamos documentando los objetos y listas.
Pues sin ir más lejos habremos empleado más de 50 horas en este proyecto, debido a los problemas que se nos han ido interponiendo por el camino, y muchas decisiones las hemos tenido que tomar por el poco tiempo que nos quedaba o porque el código se alargaba demasiado y su lectura era una amargura. Quizás el tiempo que hayamos empleado menos ha sido en documentar, pero porque la documentación no nos ha supuesto demasiados problemas.

Cabe destacar que una vez hechos algunos algoritmos daban algún problema menor y los hemos tenido que rehacer de nuevo, debido a que incluso el debug no sabía decir que problema daba(algoritmo 3).

Ahora toca la segunda parte, vamos a ser sinceros, la verdad es que el tema no llamaba demasiado la atención, debido a que se parecía mucho al del año pasado, e incluso la estructura era tan alargada que muchas veces hemos tenido que hacer muchos descansos, porque nos liabamos con algunas partes, y sí, se trata de una asignatura de estructuras de datos y de información, pero se podría prestar algo más de motivación por un proyecto "más agradable", por ejemplo, en introducción a programación, hicimos un juego, que podría ser también difícil para ese nivel, pero era motivador porque estabas haciendo un juego y para alguien que empieza a programar pues le llena.
Un punto bastante importante y gracias al que hemos podido sacar el proyecto adelante, han sido las tutorías y las clases, sin embargo, algo muy importante es, o bien, mostrarnos un proyecto resuelto, que funcione correctamente, para ver que resultados deberíamos mostrar, ya que, sí ,hemos conseguido resolver el proyecto, pero aunque hayamos hecho pruebas, realmente no sabemos a ciencia cierta, si deberían dar esos resultados o no.
Otra cuestión, es que se nos hubiese enseñado algo básico sobre umbrello, ya que muchas de las opciones no las sabíamos contemplar, cabe destacar que tampoco guardaba archivos y tuvimos que descargar archivos adicionales para ello.

Quizás eso sería un resumen de nuestras opiniones, no es un mal tema claro está, pero algo que hay que destacar es que la estructura era larga y cansaba mucho.

Y esta sería la opinión.