# Especificación de requerimientos funcionales

Gen\_1: El programa debe ser capaz de generar registros de personas al iniciar el programa. La cantidad de registros la puede indicar el usuario, en caso de no hacerlo se debe generar un numero cercano a mil millones de personas (1.000.000.000) o el numero mas alto que el programa permita. Los datos de los registros abarcaran, un código (autogenerado por el programa), nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía.

Cre\_1: El programa debe permitir al usuario crear nuevos registros. Este debe digitar todos los campos (datos) que se mencionaron en el requerimiento (Gen\_1) menos el código, puesto que este se autogenerará. Debe dar la opción de guardar.

Act\_1: El programa debe permitir al usuario actualizar la información de los registros de personas. Todos los datos son editables excepto por el código. Debe dar la opción de actualizar los datos (En caso de haber cambios).

Elim\_1: El programa debe permitir al usuario eliminar registros. Se borrarán todos los datos asociados al registro de la persona del sistema y del archivo que serializa la información (Ser\_1). La opción para eliminar debe estar en la interfaz destinada a actualizar (Act\_1).

Ser\_1: El programa debe permitir la opción de guardar los registros. El guardado de datos debe implementar persistencia, almacenando los registros o los datos de estos en un archivo serializable.

# Método de la ingeniería

## Identificación del problema

Un proyecto de investigación de la universidad ICESI requiere el desarrollo de un prototipo de software que permita gestionar eficientemente las operaciones CRUD sobre una base de datos de personas de nuestro continente. Se debe simular la creación o generación de registros de personas con los siguientes datos: nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad, fotografía y un código que autogenerará el programa. Adicionalmente, se deben implementar las funciones CRUD (Create, Read, Update y Delete), de modo que sea posible:

* Crear registros, ingresando la información de estos (Exceptuando el código autogenerado).
* Actualizar registros, siendo capaz de editar algunos de sus datos (Exceptuando el código).
* Buscar registros mediante los siguientes campos:
  + Nombre
  + Apellido
  + Nombre y Apellido
  + Código.
* Eliminar registros

## Recopilación de información.

* En la sección se búsqueda para los criterios Nombre y Nombre completo debe aparecer una lista emergente de personas de la base de datos cuyos nombres coincidan con los caracteres digitados hasta el momento (Máximo 100 personas)
* En la sección de búsqueda, para cada una de las cuatro búsquedas, se debe mantener un árbol binario de búsqueda auto balanceado o un árbol rojo y negro, que sean persistentes.
* En la sección de búsqueda debe mostrarse al lado del campo donde se digita la cadena, un número que indica la cantidad total de elementos que hasta el momento coinciden con el prefijo digitado.
* En la sección de búsqueda, en el momento en que haya 20 (parametrizable) o menos elementos que coinciden con la búsqueda, debe desplegarse un listado con los registros que coinciden y un botón al lado de cada registro con la opción de Editar, que llevará al formulario con la posibilidad de modificar o eliminar ese registro.
* Se deben implementar pruebas unitarias de las estructuras de datos implementadas y de las operaciones principales de las clases del modelo.
* El programa debe contar con interfaz gráfica.
* Al iniciar se deben preguntar la cantidad de registros que se desean generar, por defecto tiene que ser el mayor valor posible.
* Al generar los registros, el proceso debe contar con una barra de progreso y mostrar cuanto demoro la generación de registros.
* Los datos generados deben poder guardarse en la base de datos del programa mediante una opción.
* Todos los datos del programa deben ser persistentes (es decir, si se cierra el programa, deben seguir allí una vez se inicie nuevamente).
* El programa debe estar en disposición de trabajar con al menos un millón de registros (1.000.000).
* Los datasets para generar los registros se deben guardar en un directorio del proyecto (/data); a continuación, aparecen los datasets junto con la entradas o datos que proporcionan:
* <https://data.world/alexandra/baby-names> - Nombres
* <https://data.world/uscensusbureau/frequently-occurring-surnames-from-the-census-2010> - Apellidos
* <https://www.indexmundi.com/es/estados_unidos/distribucion_por_edad.html> - Distribución de edades
* <https://www.kaggle.com/tanuprabhu/population-by-country-2020> - Población por país de América
* <https://data.world/alexandra/baby-names> - Nombres
* En la generación de registros y en cuanto a la entrada sexo, se puede asumir la misma cantidad de hombres como de mujeres.
* En la generación de registros, la estatura puede escogerse aleatoriamente en un intervalo que tenga sentido.
* En la generación de registros, la fotografía debe ser generada aleatoriamente de este sitio: <https://thispersondoesnotexist.com/> sin importar que su imagen no se corresponda exactamente con su edad, sexo u otros.

Después haber detallado la información del problema, es necesario detallar la información de ciertas estructuras de datos que nos serán útiles para resolver el problema:

### Árbol Binario:

Es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre "binario"). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno.

Recuperado de: <https://es.wikipedia.org/wiki/%C3%81rbol_binario>

### Árbol Binario de búsqueda:

es un árbol binario con la propiedad de que todos los elementos almacenados en el subárbol izquierdo de cualquier nodo x son menores que el elemento almacenado en x, y todos los elementos almacenados en el subárbol derecho de x son mayores que el elemento almacenado en x.

Recuperado de: <https://ccia.ugr.es/~jfv/ed1/c++/cdrom4/paginaWeb/abb.htm>

Véase también: <https://es.wikipedia.org/wiki/%C3%81rbol_binario_de_b%C3%BAsqueda>

### Árbol Binario de búsqueda AVL:

Los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio (o balanceo), la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O(log n). El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

Para conseguir esta propiedad de equilibrio, la inserción y el borrado de los nodos se ha de realizar de una forma especial. Si al realizar una operación de inserción o borrado se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos.

Recuperado de: <https://es.wikipedia.org/wiki/%C3%81rbol_AVL>

### Árbol rojo y negro:

Un árbol rojo-negro es un árbol binario de búsqueda en el que cada nodo tiene un atributo de color cuyo valor es rojo o negro. En adelante, se dice que un nodo es rojo o negro haciendo referencia a dicho atributo.

Además de los requisitos impuestos a los árboles binarios de búsqueda convencionales, se deben satisfacer las siguientes reglas para tener un árbol rojo-negro válido:

1. Todo nodo es o bien rojo o negro.
2. La raíz es negra.
3. Todas las hojas (NULL) son negras.
4. Todo nodo rojo debe tener dos nodos hijos negros.
5. Cada camino desde un nodo dado a sus hojas descendientes contiene el mismo número de nodos negros.

Recuperado de: <https://es.wikipedia.org/wiki/%C3%81rbol_rojo-negro>

## Búsqueda de soluciones creativas:

## Transición de la formulación de ideas a diseños preliminares:

## Evaluación y selección de la mejor solución:

## Preparación de informes y especificaciones:

## Implementación del diseño:

# Análisis de complejidad

## Espacial:

## Temporal:

# Diseño de TAD:

# Casos de prueba: