UNIVERSIDAD ICESI

Departamento de TIC
Algoritmos y Estructuras de Datos
2020-2 - Tarea Integradora 2

Objetivos

Unidad 3: Algoritmos y Estructuras Recursivas

OE3.1. Calcular la complejidad temporal de algoritmos recursivos.

OE3.3. Resolver ecuaciones de recurrencia que sean resultado del análisis de complejidad temporal de algoritmos recursivos.

OE3.5. Aplicar la técnica de diseño de algoritmos Dividir y Conquistar en la solución de problemas.

OE3.6. Utilizar estructuras recursivas de datos para representar la información del modelo de datos cuando sea conveniente.

OE3.7. Evaluar la utilidad del concepto de orden en un árbol binario de búsqueda para la solución de problemas.

OE3.8. Evaluar la utilidad del concepto de balanceo en un árbol binario de búsqueda para la solución de problemas.

OE3.9. Desarrollar estructuras de datos recursivas ABB.

Opcional: OE3.10. Desarrollar estructuras de datos recursivas R&N.

OE3.11. Desarrollar estructuras de datos recursivas AVL.

OE3.14. Desarrollar las pruebas unitarias de cada una de las estructuras de datos recursivas implementadas.

Gestión Eficiente de Base de Datos de Personas

Enunciado

Después del excelente rendimiento en sus trabajos académicos previos, sus profesores de algoritmos los han recomendado a diversos profesores de la Facultad de Ingeniería, para que trabajen en sus proyectos de investigación. Su equipo ha sido contratado para una monitoría en un proyecto de investigación interna de la Universidad, como parte del Equipo VIP de Simulación[1].

Generación de los Datos

El sub-proyecto que le ha sido asignado, consiste en el desarrollo de un prototipo de software que permita gestionar eficientemente las operaciones CRUD sobre una base de datos de personas de nuestro continente. La población del continente americano se estima, en 2020, en poco más de mil millones de personas[2], representando cerca del 13% del total mundial. Por tanto, usted debe **simular la creación de** (generar) un número similar de registros de personas, para este continente, con los siguientes datos: código (autogenerado), nombre, apellido, sexo, fecha de nacimiento, estatura, nacionalidad y fotografía.

Usted debe desarrollar un programa que lleve a cabo la generación de todos estos registros de personas de acuerdo con las condiciones que se detallan a continuación. Los datasets que el programa utilizará como entrada para generar los datos de las personas deben ser descargados a un directorio del proyecto (/data).

Para la generación de nombres completos, se deben tomar los nombres de este dataset de nombres de data.world. Los apellidos se deben tomar del archivo completo (el archivo pequeño -1000- debe ser ignorado) de este dataset de apellidos de data.world. La combinación de todos los nombres con todos los apellidos debe producir la cantidad (o similar) de personas que deseamos.

La fecha de nacimiento debe ser generada aleatoriamente, suponiendo una distribución de edad para toda América basada en esta distribución de edad de Estados Unidos. La distribución de la población en sexo indicada en el enlace anterior puede ser ignorada, y se puede asumir una cantidad igual de hombres y mujeres.

La estatura debe ser generada aleatoriamente en un intervalo que tenga sentido. La nacionalidad debe ser asignada a cada persona, generada de tal forma que se mantengan los porcentajes relativos de población de cada país respecto del continente de acuerdo con estos datos de población por países (se puede también asignar exactamente la misma población de cada país, y si hay diferencia en el total, dicha diferencia -positiva o negativa- puede quedar en el país con mayor población).

Usted puede filtrar el dataset para dejar en el archivo únicamente los registros de los países necesarios. Su programa debe basar los cálculos en el archivo y no en condicionales por país quemados en el código (hardcode).

---

[1] VIP Curse in Data Intensive Processing and Simulation

[2] https://www.worldometers.info/geography/7-continents/

Bonus: 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.

El programa debe tener una opción para empezar a generar los datos siguiendo las especificaciones anteriores. Debe tener una barra de progreso si el proceso tarda más de 1 segundo en terminar, y debe indicar cuánto tiempo se demoró la operación. La opción de generar debe tener un campo de texto en el cual se pueda digitar cuántos registros se desea generar. Por defecto, debe estar en el campo el máximo valor posible.

Una vez los datos se generan, debe haber una opción para guardarlos en la base de datos del programa, y así poderlos consultar posteriormente. Todos los datos del programa deben ser persistentes (es decir, si se cierra el programa, deben seguir allí una vez se inicie nuevamente).

Responda este par de preguntas: (1) ¿Es posible serializar los datos generados? (2) ¿Cuánto pesa el archivo serializado, cuando se persisten los datos, al generarse el número máximo posible de registros?.

Operaciones CRUD

En programación, se conoce como CRUD (Create, Read, Update y Delete) a las cuatro funciones básicas del almacenamiento persistente[3]. El programa desarrollado por su equipo debe tener la posibilidad de llevar a cabo cualquiera de estas funciones. Por tanto, la interfaz con el usuario deberá contar con un formulario para agregar a una nueva persona, otro para buscar a una persona por los criterios mencionados más adelante y uno más para actualizar los campos de una persona existente. Este último formulario debe también tener una opción para eliminar a una persona.

Crear, Actualizar y Eliminar

El formulario para agregar debe tener todos los campos requeridos (menos el código, que es autogenerado) para la información de una persona y la opción de Guardar.

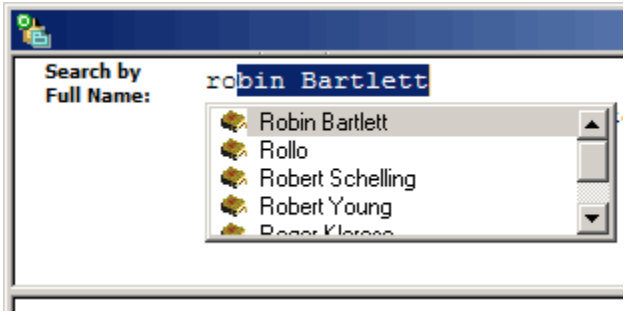
El formulario para actualizar una persona, debe tener todos los campos editables (menos el código, que no se puede actualizar) de información de una persona, cargados con su información actual, la opción de Actualizar (para guardar los cambios, si hubo) y la opción Eliminar (si se desea eliminar a esta persona).

Buscar (la estrella de la solución)

El formulario para buscar debe tener la posibilidad de realizar la búsqueda por cualquiera de los siguientes criterios, de forma excluyente (es decir, buscas por un criterio o por otro, pero no por varios al tiempo):

1. Nombre
2. Apellido
3. Nombre Completo (Nombre + " " + Apellido)
4. Código

El programa debe permitir que, en la medida en que se digite los caracteres de búsqueda en el campo (para los criterios 1 a 3), vayan apareciendo en una lista **emergente** debajo del campo, máximo 100 (parametrizable) nombres de la base de datos, que empiecen con los caracteres digitados hasta el momento.



Ejemplo de la búsqueda y la lista emergente

Para cada una de las cuatro búsquedas, usted debe mantener un árbol binario de búsqueda autobalanceado y persistente. El árbol debe ser implementado completamente por su equipo de desarrollo.

[3] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

Bonus: si uno de los árboles binarios de búsqueda autobalanceado es un Árbol Rojo y Negro.

Mientras se hace la 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 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 nos llevará al formulario con la posibilidad de modificar o eliminar ese registro.

El programa debe tener el diseño e implementación de pruebas unitarias automáticas de las estructuras de datos implementadas y de las operaciones principales de las clases del modelo.

1. El programa debe contar con una interfaz gráfica con el usuario, tener un componente menú en la parte superior de la ventana que permita cambiar todos los elementos de la ventana cada vez que se esté trabajando en opciones diferentes. Es decir, **no** se desea tener una ventana llena al mismo tiempo de elementos que se usarán en momentos diferentes.

1. Analisis.

1.1. Requerimientos Funcionales.

RF1. Generar datos de personas ficticias: Se debe generar una cantidad determinada previamente por el usuario de personas ficticias de tal modo que compla con una serie de parametros.

RF2. Agregar las personas ficticias a una base de datos: Las personas que fueron previamente creadas de manera random y datos genéricos serán agregadas a una base de datos eficiente y eficaz para posteriormente r ealizar operaciones con ellas.

RF3. Agregar una persona unitaria: Se debe agregar una persona unica con criterios no generados, dichos criterios son dados previamente por el u suario.

RF4. Buscar persona: Buscar una persona dada la seleccion de una serie de criterios de busqueda con despliege de un menu de sugerencias.

RF6. Desplegar listado de las personas según un criterio de busqueda: Al buscar una persona en la base de datos, ya sea por nombre, nombre y apellido ó número de identificación, se debe mostrar una lista con las personas que coincidan con el criterio de busqueda a partir del cual se está buscando y también se debe permitir la edición de los datos de la persona que sea seleccionada.

RF6. Eliminar una persona: Eliminar a una persona de la base de datos dada una seleccion en la lista de resultados de la busqueda.

RF7. Modificar datos de una persona: Modificar los datos especificos a una persona de la base de datos dada una seleccion en la lista de resultados de la busqueda.

1.2. Requerimientos no Funcionales.

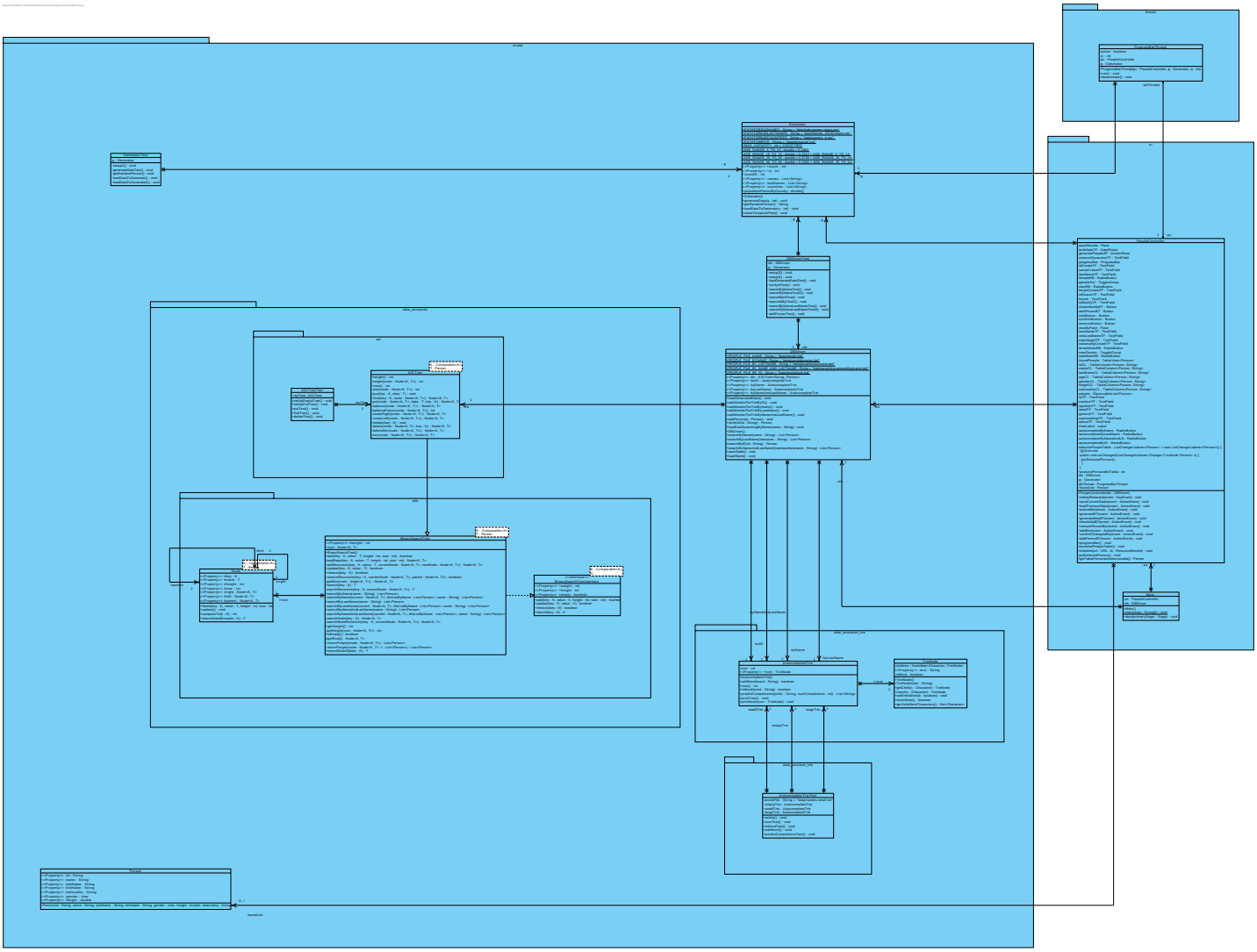
RNF1. Generar los nombres de las personas de manera aleatoria: Se deben tomar los nombres y apellidos de datasets y combinarse los nombres con los apellidos de manera aleatoria para de esta forma crear de manera aleatoria los nombres de las personas.

RNF2. Generar la fecha de nacimiento de las personas de manera aleatoria: La fecha de nacimiento debe ser generada aleatoriamente, suponiendo una distribución de edad para toda América basada en la distribución de edad de Estados Unidos.

RNF3. Generar el genero de las personas de manera aleatoria: Se debe generar el genero de las personas de manera aleatoria y se puede asumir igual cantidad de hombres y mujeres.

RNF4. La búsqueda de personas se debe autocompletar de manera automatica: El programa debe permitir que a medida que se digiten los caracteres de la búsqueda vayan apareciendo en una lista emergente máximo 100 nombres de la base de datos que empiecen con los caracteres digitados hasta el momento

RNF5. Desplegarse un lista con los registros que coinciden y un bóton con la opción de editar: Se debe mostrar un listado con los registros que coincidan y un botón al lado de cada registro con la opción de editar, que nos lleve a un formulario con la posibilidad de modificar o eliminar el registro, siempre y cuando haya 20 o menos elementos que coincidan con esta búsqueda.



2. Diseño.

AutocompleteTrie

Nombre	Clase	Escenario
setUp	AutocompleteTrieTest	Tres objetos de la clase AutocompleteTrieTest, el primero está vacío, es decir, no tiene palabras para autocompletar; otro objeto que tiene ocho palabras y por último otro objeto con 4440 palabras.

- Objetivo de la prueba: Determinar si el método ***size*** devuelve la cantidad correcta de palabras que están guardadas en la estructura.

Clase	Método	Escenario	Valores de Entrada	Resultado
AutocompleteTrieTest	**sizeTest**	setUp		Las palabras que encuentra en el trie vacío, en el trie pequeño y en el trie grande son 0, 8 y 4440 respectivamente.

- Objetivo de la prueba: Verificar si el método ***isWord*** determina correctamente si una palabra ya ha sido agregada al trie.

Clase	Método	Escenario	Valores de Entrada	Resultado
AutocompleteTrieTest	**isWordTest**	setUp	“Andrea” “animal” “animalww” “”	Las palabras "Andrea” y “animal” se encuentran en el trie pequeño efectivamente, mientras que estas mismas palabras no se encuentran en el trie vacío como era de esperarse.

- Objetivo de la prueba: Verificar si el método ***addWord*** agrega correctamente una palabra al trie.

Clase	Método	Escenario	Valores de Entrada	Resultado
AutocompleteTrieTest	**addWordTest**	setUp	"camillow"	La palabra "camillow" ha sido agregada correctamente a los tres tries.

- Objetivo de la prueba: Verificar si el método ***predictCompletitionTest*** predice correctamente las palabras a autocompletar según lo que haya escrito el usuario.

Clase	Método	Escenario	Valores de Entrada	Resultado
AutocompleteTrieTest	**predictAutocompletitionTest**	setUp	"" "" "ann"	La cadena vacía "" da como resultado 0 predicciones en el trie vacío y 8 en el trie pequeño. Al buscar si dentro de esas 8 predicciones de "" en el trie pequeño se encuentra el prefijo "ann" deberían aparecer 4 ya que hay 4 palabras en ese trie que empiezan por "ann" y efectivamente el test lo corrobora.

DBDriverTest

Nombre	Clase	Escenario
setUp2	DBDriverTest	Un objeto de la clase DBDriver y un objeto de la clase Generator con 1000 personas generadas

setUp3	DBDriverTest	Un objeto de la clase DBDriver y un objeto de la clase Generator

Objetivo de la prueba: Determina si el método loadGenerateData genera de manera correcta las personas

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**loadGenerateDataTest**	setup2		**El método genera correctamente las 1000 personas solicitadas**

Objetivo de la prueba: Determina si el método verifyIdTest verifica de manera correcta el ID

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**verifyIdTest**	setup2		**El método no verifica el ID dado que no hay personas en la base de datos**

Objetivo de la prueba: Determina si el método searchByName busca por el nombre de manera correcta la persona solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**searchByNameTest**	setup3		**El método no encuentra el nombre solicitado dado que no existe ninguna persona con ese nombre**

Objetivo de la prueba: Determina si el método searchByName busca por el nombre de manera correcta la persona solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**searchByNameTest2**	setup3		**El método encuentra el**

				nombre solicitado dado que existen personas con ese nombre en la base de datos

Objetivo de la prueba: Determinar si el método search busca correctamente a un cliente

Clase	Método	Escenario	Valores de Entrada	Resultado
CashierModuleTest	**searchTest**	signUp2		**El cliente se busca de manera correcta y retorna el cliente que se buscaba**

Objetivo de la prueba: Determina si el método searchById busca por el id de manera correcta la persona solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**searchById2**	setup3		**El método encuentra la persona buscada por el ID solicitado**

Objetivo de la prueba: Determina si el método searchByName busca por el nombre de manera correcta la persona solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**searchById**	setup3		**El método no encuentra el id solicitado dado que no existe ninguna persona en la base de datos**

Objetivo de la prueba: Determina si el método searchByNameLastName busca por el nombre y el apellido de manera correcta la persona solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**searchByNameLastNameTest**	setup3		**El método no encuentra el nombre y apellido solicitado dado que no existe ninguna persona en la base de datos**

Objetivo de la prueba: Determina si el método searchByNameLastName busca por el nombre de manera correcta la persona solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**searchByNameLastNameTest**	setup3		**El método encuentra el nombre y apellido solicitado dado que si existen personas con estos nombres y apellidos**

Objetivo de la prueba: Determina si el método addPerson agrega de manera correcta la persona solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
DBDriver	**addPersonTest**	setup3		**El método agrega de manera correcta la persona que se le solicita**

GeneratorTest

Nombre	Clase	Escenario
setUp1	GeneratorTest	Un objeto de la clase Generator

Objetivo de la prueba: Determina si el método addPerson agrega de manera correcta la persona solicitada

Clase	Método	Escenario	Valores de Entrada	Resultado
GeneratorTest	**generateDataTest**	setup1		**El método genera de manera correcta las 1000 personas solicitadas**

Objetivo de la prueba: Determina si el método getRandomPerson retorna de manera correcta una persona de manera aleatoria

Clase	Método	Escenario	Valores de Entrada	Resultado
GeneratorTest	**getRandomPersonTest**	setup1		**El método retorna de manera correcta la persona obtenida de manera aleatoria**

Objetivo de la prueba: Determina si el método loadDataToGenerate genera los archivos necesarios para la generacion de las personas de manera correcta

Clase	Método	Escenario	Valores de Entrada	Resultado
GeneratorTest	**loadDataToGenerateTest**	setup1		**El método genera de manera correcta los archivos**

				necesarios para la generación de personas

Objetivo de la prueba: Determina si el método loadDataToGenerate genera los archivos necesarios para la generacion de las personas de manera correcta

Clase	Método	Escenario	Valores de Entrada	Resultado
GeneratorTest	**loadDataToGenerateTest2**	setup1		**El método genera de manera correcta los archivos necesarios para la generación de personas**

AVLTest.

Nombre	Clase	Escenario
setUpFullTree	AVLTreeTest	Un arbol AVL con tres nodos de tipo Person agregados: • P1 = "1", "Andrea", "Nunez", "19/12/2000", 'f', 1.64, "Colombia" • P2 = "2", "Danna", "Garcia", "19/12/2000", 'f', 1.64, "Colombia" • P3 = "3", "Camilo", "Cordoba", "19/12/2000", 'f', 1.64, "Colombia"
setUpEmptyTree	AVLTreeTest	Un árbol AVL vacío de Person (con cero nodos).

Objetivo de la prueba: Determinar si el método find busca correctamente a un nodo.

Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTreeTest	**findTest**	setUpFullTree	p = new Person"1010", "Cristian", "Garcia", "19/12/2000", 'f', 1.64, "Colombia"	**El nodo p es ahora la raíz del árbol auto balanceado.**

Objetivo de la prueba: Determinar si el método put agrega correctamente a un cliente

Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTreeTest	**putTest**	setUpEmptyTree sign	p = new Person"1010", "Cristian", "Garcia", "19/12/2000", 'f', 1.64, "Colombia"	**El nodo p es ahora la raíz del árbol auto balanceado.**
AVLTreeTest	**putTest**	setUpFullTree	p = new Person"1010", "Cristian", "Garcia", "19/12/2000", 'f', 1.64, "Colombia"	**El nodo p es ahora la raíz del árbol auto balanceado.**

Objetivo de la prueba: Determinar si el método delete remueve correctamente a un cliente

Clase	Método	Escenario	Valores de Entrada	Resultado
AVLTreeTest	**deleteTest**	setUpFullTree	p = ! new Person"1010", "Cristian", "Garcia", "19/12/2000", 'f', 1.64, "Colombia"	**El método delete al no encontrar al nodo p en el árbol retorna null.**

ABBTest.

Nombre	Clase	Escenario

setup2	BinarySearchTreeTest	Un Abb con un nodo de tipo Person agregados: • P1 = "2", "Andrea", "Nunez", "19/12/2000", 'f', 1.64, "Colombia"
setup1	BinarySearchTreeTest	Un árbol AVL vacío de Person (con cero nodos).

Objetivo de la prueba: Determinar si el método add busca correctamente a un nodo.

Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTreeTest	**addTest1**	setup1	• P1 = "2", "Andre", "Nunez", "19/12/2000", 'f', 1.64, "Colombia"	**El nodo p1 es la raíz del árbol ahora.**
BinarySearchTreeTest	**addTest2**	setup2	• P2 = "1", "Danna", "Garcia", "19/12/2000", 'f', 1.64, "Colombia" • P3 = "3", "Camilo", "Cordoba", "19/12/2000", 'f', 1.64, "Colombia".	**El nodo P3 ahora es el hijo derecho de la raíz del árbol y P2 es el izquierdo de la raíz del árbol.**

Objetivo de la prueba: Determinar si el método remove agrega correctamente a un cliente

Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTreeTest	**removeTest1**	setup1	**Key = "1"**	**El nodo a remover no ha sido encontrado.**
BinarySearchTreeTest	**removeTest2**	setup2	**Key = "2"**	**El nodo con key = "2" ha sido removido del arbol.**

Objetivo de la prueba: Determinar si el método search remueve correctamente a un cliente

Clase	Método	Escenario	Valores de Entrada	Resultado
BinarySearchTreeTest	**searchTest1**	setup1	Key = “1”	**El nodo con key = “1” no se encuentra en el arbol.**
BinarySearchTreeTest	**searchTest2**	setup2	• P3 = "3", "Camilo", "Cordoba", "19/12/2000", 'f', 1.64, "Colombia".	**El nodo con key = “3” se ha encontrado en el arbol.**