IcesiTinder Enunciado V1.0

Un grupo de ingenieros de sistemas y telemática de la universidad Icesi se preocuparon por la falta de comunicación de su comunidad universitaria, ya que debido a la pandemia del Covid-19 las relaciones, reuniones y competencias de esta se tuvieron que eliminar en vista de que la presencialidad en todas estas actividades no está permitida hasta el momento. Este grupo de ingenieros caleños diseñaron un sistema de Software que permite que los individuos se conozcan por medio de alguien en común. Lo que la comunidad espera que la solución de software brinde las siguientes funcionalidades:

El programa funciona con cuentas personales que se crean al momento del registro de cada usuario, donde se solicita que ingrese información tal como sus nombres completos, número de teléfono, correo electrónico, género, género de interés y facultad a la que pertenece, además se solicita un nombre de usuario que será con el que podrá realizarse funciones tales como búsqueda dado que este es el que lo representa dentro del software y una contraseña para poder ingresar a la cuenta.

Se espera que el Software pueda permitir a los usuarios del programa conocerse con otros usuarios por medio de las relaciones de las personas que sigue, las relaciones pueden ser de conocidos, amigos o mejores amigos y de esto facilitará la manera en cómo se conozcan las personas. Los usuarios que el usuario vaya a conocer se espera que tengan relación de mejores amigos con las personas que este sigue, para que de esta manera sea más cómodo para ambos usuarios.

Además, se espera que cuando una persona crea su cuenta, el programa le muestre de manera aleatoria usuarios que podría seguir que tengan o no intereses en común.

Los usuarios pueden definir qué tipo de relación tienen siempre y cuando se sigan mutuamente, como previamente se mencionó pueden ser de tipo **conocido, amigos o mejores amigos.**

El programa debe realizar operaciones donde se pueda buscar, modificar y eliminar.

Buscar

Permite al usuario buscar a otro usuario que se encuentre registrado en el programa por medio del usuario y mostrar la información básica tal como nombres y género únicamente. Si desea, puede seguir el usuario buscado para ver más sobre este. Si el usuario buscado si lo sigue la persona que busca, se le mostrará toda la información de la cuenta de este usuario y podrá dejar de seguirlo.

Eliminar

Permite al usuario eliminar de manera definitiva su cuenta del programa, donde para poder hacerlo debe confirmar con su contraseña para que verifique su identidad, al eliminarse la cuenta no podrá recuperarla y perdería toda la información y relaciones que tenga en el programa.

Modificar

Permite al usuario cambiar su nombre de usuario por otro que se encuentre disponible o cambiar la contraseña de su cuenta por una nueva, la demás información de la cuenta no es posible cambiarla.

Para manejar el sistema de cuentas, por lo tanto, todos los datos del programa deben ser persistentes (es decir, si se cierra el programa, deben seguir allí una vez se inicie nuevamente).

Mostrar información

Se debe mostrar todas las personas que sigue la persona y los seguidores en una lista, donde se pueda ver el nombre de usuario de cada uno de estos.

El programa debe manejar una interfaz gráfica donde el usuario pueda iniciar sesión con su nombre de usuario único y contraseña, en cuento el usuario ingrese a su cuenta la interfaz debe contar con todas las funcionalidades anteriores para que de este modo el usuario pueda hacer uso de cualquiera de estas que desee. Si el usuario no tiene una cuenta aún, la interfaz debe contener una opción de registro donde se le solicite la información necesaria y la cuenta quede en el sistema.

Functional requirements

Name:	R. #1. Find a path that takes to a new person who can be met
Summary:	Find a path that takes to a new person who can be met and this path is the best one
Input:	
Results:	Path found
	Path not found

Name:	R. #2. Model the force that unites a person with another
Summary:	This refers to the fact that they can be Friends, best friends or acquaintances
Input:	Force that user chose
Results:	

Name:	R. #3. Show all the people that user follows
Summary:	Show the username of all the people that user follows
Input:	
Results:	User follows with username
	User doesn't follow anyone

Name:	R. #4. Show all the user followers
Summary:	Show the username of all the user followers
Input:	
Results:	User followers with username
	User doesn't have followers

Name:	R. #6. Remove an specific follow
Summary:	User can unfollow users
Input:	String username
Results:	

Name:	R. #7. Modify user information
Summary:	Modify user information (username or password) to change the password user need to enter actual password before change it
Input:	String newInformation
Results:	Information change correctly
	Information could not be change

Name:	R. #8. Register a new user
Summary:	Register a new user that doesn't have a account, user have to enter name and last name, gender, faculty, username and password
Input:	String name, String lastname, char gender, String faculty, String username, String password
Results:	User account was created correctly
	User account could not be created

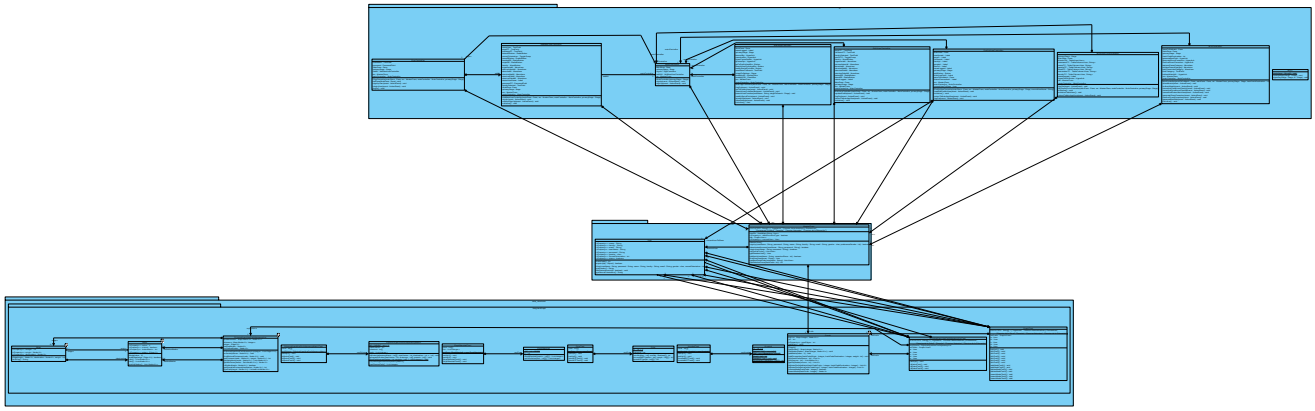
Name:	R. #9. Allow sign-in
Summary:	Allow sign-in in accounts already create if password and username match
Input:	String username, String password
Results:	Sign-in correctly
	Username or password doesn't match

Name:	R. #10. Delete account
Summary:	User can delete his account, this action Will remove all followers and won't be able to sign-in again. To delete account user have to enter password
Input:	String password

Results:	Account deleted correctly
	Password doesn’t matc, account couldn’t be deleted

Non-functional requirements

- Implement a graph (adjacent matrix and list)
- Implement graph methods like Dijkstra, BFS, DFS and others
- Must have a graphical interface



TAD Graph:

TAD:	<WeightedGraph>
Description: A Graph is a non-linear collection containing vertices and edges connecting vertices. This ADT does not specify if the edges are directed, leaving that to an implementation. The edges have non-negative weights.	
Invariantes:	• 1. Empty graph: number of vertices is 0; number of edges is 0. • 2. Self-loops are not allowed. • 3. Edge weights must be >= 1
Operaciones Primitivas:	**Graph**() **addEdge**(Vertex v1 , double w, Vertex v2) **addEdge**(Vertex v1 , Vertex v2) **addVertex**(Vertex v) **getEdgeWeight**(Vertex v1 , Vertex v2) **setEdgeWeight**(Vertex v1 , double newWeight, Vertex v2) **removeVertex**(Vertex v) **removeEdge**(Vertex v1 , Vertex v2) **getNeighbors**(Vertex v) **getNumberOfVertices**() **getNumberOfEdges**()

graph()
"Initialize the WeightedGraph."
Pre-condition: none **Responsibilities:** initializes the graph attributes. **Post-condition:** number of vertices is 0. number of edges is 0 (1.0).

addEdge(Vertex v1 , double w, Vertex v2)
"Add edge in the WeightedGraph."
Pre-condition: v1 and v2 are Vertices in this graph and aren't already connected by an edge; w is >= 0. **Responsibilities:** connect Vertices v1 to v2 with weight w; if this is an undirected graph, this edge also connects v2 to v1. **Post-condition**: an edge connecting v1 and v2 with weight w is added to this Graph. *number of edges* is incremented by 1 **Exception:** if v1 or v2 are not in the graph, are already connected by an edge, or w < 0. **Returns:** nothing.

addEdge(Vertex v1 , Vertex v2)
"Add edge in the WeightedGraph."
Pre-condition: v1 and v2 are Vertices in this graph and aren't already connected by an Edge. **Responsibilities:** connect Vertices v1 to v2; if this is an undirected graph, this edge also connects v2 to v1. **Post-condition:** an edge connecting v1 and v2 is added to this graph *number of edges* is incremented by 1. **Exception:** if v1 or v2 are not in the graph or are already connected by an edge **Returns:** nothing.

addVertex(Vertex v)
"Add the vertex in WeightedGraph."
Pre-condition: v is not already in the graph. **Responsibilities:** insert a Vertex into this graph. **Post-condition:** a Vertex is added to this graph *number of vertices* is incremented by 1. **Exception:** if Vertex v is already in this graph. **Returns:** nothing.

getEdgeWeight(Vertex v1 , Vertex v2)
"show the weight."
Pre-condition: v1 and v2 are Vertices in this graph and are connected by an Edge. **Responsibilities:** get the weight of the edge connecting Vertices v1 to v2. **Post-condition:** the graph is unchanged. **Exception:** if v1 or v2 are not in the graph or are not connected by an Edge. **Returns:** the weight of the edge connecting v1 to v2.

setEdgeWeight(Vertex v1 , double newWeight, Vertex v2)
"Change the weight."
Pre-condition: v1 and v2 are Vertices in this graph and are connected by an edge; newWeight is >= 0. **Responsibilities:** set the weight of the edge connecting Vertices v1 to v2 to newWeight. **Post-condition:** the graph is unchanged. **Exception:** if v1 or v2 are not in the graph, are not connected by an edge, or newWeight < 0. **Returns:** nothing.

removeVertex(Vertex v)
"remove the vertex of WeightedGraph. "
Pre-condition: v is a Vertex in this graph **Responsibilities:** remove Vertex v from this graph. **Post-condition:** Vertex v is removed from this graph, All edges incident on v are removed *number of vertices* is decremented by 1 *number of edges* is decremented by degree(v). **Exception:** if v is not in this graph. **Returns:** nothing.

removeEdge(Vertex v1 , Vertex v2)
"remove the edge of WeightedGraph."
Pre-condition: v1 and v2 are vertices in this graph and an edge exists from v1 to v2. **Responsibilities:** remove from this graph the edge connecting v1 to v2; if this is an undirected graph, there is no longer an edge from v2 to v1.

Post-condition: the edge connecting v1 and v2 is removed from this graph *number of edges* is decremented by 1.
Exception: if v1 or v2 are not in this graph, or if no edge from v1 to v2 exists.
Returns: nothing.

getNeighbors(Vertex v)
"return all terms adjacent to v."
Pre-condition: v is a Vertex in this graph. **Responsibilities:** get the neighbors of Vertex v from this graph. **Post-condition:** the graph is unchanged. **Exception:** if v is not in this graph. **Returns:** a collection containing the Vertices incident on v.

getNumberOfVertices()
"Allows get number of vertex."
Pre-condition: none. **Responsibilities:** get the number of vertices in this graph. **Post-condition:** the graph is unchanged. **Returns:** the number of vertices in this graph.

getNumberOfEdges()
"Allows get number of edges."
Pre-condition: none. **Responsibilities:** get the number of edges in this graph. **Post-condition:** the graph is unchanged. **Returns:** the number of edges in this graph.

DISEÑO PRUEBAS UNITARIAS

PRUEBAS CLASE KruskalTest

Nombre	Clase	Escenario
setUp1	KruskalTest	Un objeto Kruskal y una matriz de enteros con los valores {0,2,0,6,0,2,0,3,8,5,0,3,0,0,7,6,8,0,0,9,0,5,7,9,0}
setUp2	KruskalTest	Un objeto Kruskal y una matriz de enteros con los valores {0,3,20,3,0,3,0,0,0,0,20,0,0,2,3,3,0,2,0,0,0,0,3,0,0}

- **Objetivo de la prueba:** Determinar si el método ***kruskalMST*** encuentra el árbol de recubrimiento mínimo correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
KruskalTest	**kruskalMSTTest1**	setUp1		El método retorna correctamente el peso mínimo del árbol que se esperaba, en este caso 16

- **Objetivo de la prueba:** Determinar si el método ***kruskalMST*** encuentra el árbol de recubrimiento mínimo correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
KruskalTest	**kruskalMSTTest2**	setUp2		El método retorna correctamente el peso mínimo del árbol que se esperaba, en este caso 11

PRUEBAS CLASE GraphTest

Nombre	Clase	Escenario
setUp1	GraphTest	Un objeto Graph y 4 objetos User con los siguientes valores: User a: “andrea” User b: “danna” User c: “escobar” User d: “cordoba” User e: “reyes” donde : a tiene conexión con d a tiene conexión con c d tiene conexión con c c tiene conexión con e a tiene conexión con b

- **Objetivo de la prueba:** Determinar si el método ***dfs*** me devuelve el camino en profundidad desde un nodo de inicio hasta recorrer todo el grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
GraphTest	**dfsTest1**	setUp1	String expected= "andreacordobaescobarreyesdanna" Nodo inicial= andrea	El método retorna correctamente el camino en profundidad desde el nodo andrea hasta llegar a danna, recorriendo todos los nodos del grafo.

- **Objetivo de la prueba:** Determinar si el método ***dfs*** me devuelve el camino en profundidad desde un nodo de inicio hasta recorrer todo el grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
GraphTest	**dfsTest2**	setUp1	String expected= "dannaandreacordobaescobarreyes" Nodo inicial= danna	El método retorna correctamente el camino en profundidad desde el nodo danna hasta llegar a reyes, recorriendo todos los nodos del grafo.

- **Objetivo de la prueba:** Determinar si el método ***dfs*** me devuelve el camino en profundidad desde un nodo de inicio hasta recorrer todo el grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
GraphTest	**dfsTest3**	setUp1	String expected= "escobarandreacordobadannareyes" Nodo inicial= escobar	El método retorna correctamente el camino en profundidad desde el nodo escobar hasta llegar a reyes, recorriendo todos los nodos del grafo.

- **Objetivo de la prueba:** Determinar si el método ***bfs*** me devuelve el camino en amplitud desde un nodo de inicio hasta recorrer todo el grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
GraphTest	**bfsTest1**	setUp1	String expected= "andreadannacordobaescobarreyes" Nodo inicial= andrea	El método retorna correctamente el camino en amplitud desde el nodo andrea hasta llegar a reyes, recorriendo todos los nodos del grafo.

- **Objetivo de la prueba:** Determinar si el método ***bfs*** me devuelve el camino en amplitud desde un nodo de inicio hasta recorrer todo el grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
GraphTest	**bfsTest2**	setUp1	String expected= "dannaandreacordobaescobarreyes" Nodo inicial= danna	El método retorna correctamente el camino en profundidad desde el nodo danna hasta llegar a reyes, recorriendo todos los nodos del grafo.

- **Objetivo de la prueba:** Determinar si el método ***bfs*** me devuelve el camino en amplitud desde un nodo de inicio hasta recorrer todo el grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
GraphTest	**bfsTest3**	setUp1	String expected= "reyesescobarandreadannacordoba" Nodo inicial= reyes	El método retorna correctamente el camino en profundidad desde el nodo reyes hasta llegar a cordoba, recorriendo todos los nodos del grafo.

PRUEBAS CLASE DijkstraAlgorithmForAdjacencyListTest

Nombre	Clase	Escenario
setUp1	DijkstraAlgorithmForAdjacencyListTest	Un objeto DijsktraAlgorithmForAdjacencyList. Un objeto Graph y 4 objetos User con los siguientes valores: User a: “andrea” User b: “danna” User c: “escobar” User d: “cordoba” User e: “reyes” donde : a tiene conexión con d a tiene conexión con c d tiene conexión con c c tiene conexión con e a tiene conexión con b

- **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino.

Clase	Método	Escenario	Valores de Entrada	Resultado
DijkstraAlgortihm ForAdjacencyListTest	**dijkstraTest**	setUp1	String expected= "andreacordobaescobarreyes" Nodo origen= andrea Nodo destino= reyes	El método retorna correctamente el camino más corto entre los nodos andrea y reyes.

- **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino.

Clase	Método	Escenario	Valores de Entrada	Resultado
DijkstraAlgortihm ForAdjacencyListTest	**dijkstraTest2**	setUp1	String expected= "dannaandreacordobaescobar" Nodo origen= danna Nodo destino= escobar	El método retorna correctamente el camino más corto entre los nodos danna y escobar.

- **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino.

Clase	Método	Escenario	Valores de Entrada	Resultado
DijkstraAlgortihm ForAdjacencyListTest	**dijkstraTest3**	setUp1	String expected= "reyesescobarcordobaandrea" Nodo origen= reyes Nodo destino= andrea	El método retorna correctamente el camino más corto entre los nodos reyes y andrea.

PRUEBAS CLASE DijkstraAlgorithmForAdjacencyMatrixTest

Nombre	Clase	Escenario
setUp1	DijkstraAlgorithmForAdjacencyMatrixTest	Un objeto DijsktraAlgorithmForAdjacencyMatrix y una matriz de enteros de costos con los valores: {0, 3, 20, 3, 0}, {3, 0, 0, 0, 0}, {20, 0, 0, 2, 3}, {3, 0, 2, 0, 0}, {0, 0, 3, 0, 0},

- **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino a partir de la matriz de costos del grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
DijkstraAlgortihm ForAdjacencyMatrixTest	**dijkstraTest**	setUp1	String expected= "0324" Nodo origen=0 Nodo destino=4	El método retorna correctamente el camino más corto entre los nodos andrea(0) y reyes(4).

- **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino a partir de la matriz de costos del grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
DijkstraAlgortihm ForAdjacencyMatrixTest	**dijkstraTest2**	setUp1	String expected= "103" Nodo origen=1 Nodo destino=3	El método retorna correctamente el camino más corto entre los nodos danna(0) y escobar(3).

- **Objetivo de la prueba:** Determinar si el método dijkstra me devuelve el camino más corto entre un nodo origen y un nodo destino a partir de la matriz de costos del grafo.

Clase	Método	Escenario	Valores de Entrada	Resultado
DijkstraAlgortihm ForAdjacencyMatrixTest	**dijkstraTest3**	setUp1	String expected= "42301" Nodo origen=4 Nodo destino=1	El método retorna correctamente el camino más corto entre los nodos reyes(4) y danna(1).

PRUEBAS CLASE PrimTest

Nombre	Clase	Escenario
setUp1	PrimTest	Un objeto Prim y una matriz de enteros con los valores { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 }, { 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 }, { 0, 5, 7, 9, 0 },
setUp2	PrimTest	Un objeto Prim y una matriz de enteros con los valores {0, 3, 20, 3, 0}, {3, 0, 0, 0, 0}, {20, 0, 0, 2, 3}, {3, 0, 2, 0, 0}, {0, 0, 3, 0, 0},

- **Objetivo de la prueba:** Determinar si el método ***primMST*** encuentra el árbol de recubrimiento mínimo correctamente

Clase	Método	Escenario	Valores de Entrada	Resultado
PrimTest	**PrimMSTTest1**	setUp1	16	El método retorna correctamente el peso mínimo del árbol que se esperaba, en este caso 16

Clase	Método	Escenario	Valores de Entrada	Resultado
PrimTest	**PrimMSTTest2**	setUp2	27	El método retorna correctamente el peso mínimo del árbol que se esperaba, en este caso 27

PRUEBAS CLASE FloydWarshallTest

Nombre	Clase	Escenario
setUp1	FloyWarshallTest	Un objeto FloydWarshall y una matriz de enteros con los valores { 0, 2, 0, 6, 0 }, { 2, 0, 3, 8, 5 }, { 0, 3, 0, 0, 7 }, { 6, 8, 0, 0, 9 }, { 0, 5, 7, 9, 0 },
setUp2	FloydWarshallTest	Un objeto FloydWarshall y una matriz de enteros con los valores {0, 3, 20, 3, 0}, {3, 0, 0, 0, 0}, {20, 0, 0, 2, 3}, {3, 0, 2, 0, 0}, {0, 0, 3, 0, 0},

- **Objetivo de la prueba:** Determinar si el método ***floydWarshall*** encuentra los caminos minimos entre todos los vértices que conforman al grafo

Clase	Método	Escenario	Valores de Entrada	Resultado
FloydWarshallTest	**floydWarshallTest1**	setUp1	un lista de enteros con los valores {6, 3, 5, 3, 8, 3, 6, 8, 6, 11, 5, 8, 4, 2, 3, 3, 6, 2, 4, 5, 8, 11, 3, 5, 6} que representa los caminos minimos que hay entre cada par de vertices.	El método retorna correctamente los caminos mínimos entre cada para de vértices que están en el grafo

Clase	Método	Escenario	Valores de Entrada	Resultado
FloydWarshallTest	**floydWarshallTest1**	setUp2	un lista de enteros con los valores {4, 2, 5, 6, 7, 2, 4, 3, 8, 5, 5, 3, 6, 11, 7, 6, 8, 11, 12, 9, 7, 5, 7, 9, 10} que representa los caminos minimos que hay entre cada par de vertices.	El método retorna correctamente los caminos mínimos entre cada para de vértices que están en el grafo