**Análisis y justificación de la implementación**  
  
Para la solución del problema planteado se procedió en primer lugar a la creación y distribución de nuevas clases. Dado que el problema habla sobre un grafo no dirigido que represente a la red social, se creó la clase “TGrafoRedSocial”, la cual extiende de la clase “TGrafoNoDirigido” del tipo “TUsuario”. “TUsuario” corresponde a una nueva clase que extiende de “TVertice”, debido a que los vértices del grafo red social están representados por usuarios. Por lo tanto, el grafo se encuentra conformado por los vértices “TUsuario” y las aristas que representan la amistad entre los vértices.

La clase “TGrafoRedSocial” implementa algunos métodos simples de su propia interfaz “IGrafoRedSocial”, como el de agregar usuario, eliminar usuario, agregar amistad, etc.

Para realizar el método de mostrar las recomendaciones de amistad en base a grupos de usuarios con el mismo interés en común, se creó el método “amigosDeComunidad” que toma como parámetro al usuario al cual se le quiere recomendar amigos y devuelve una colección de usuarios (los posibles amigos). Este método llama a “componentesConexos”, el cual se encarga de buscar los distintos componentes conexos existentes en el grafo que comparten el mismo interés que el usuario en cuestión.

El método “componentesConexos” se implementa a nivel de usuario, iterando sobre todos los usuarios existentes en el grafo, verificando que si estos no fueron visitados y tienen el mismo interés que el usuario principal, se crea un nuevo grafo que se añade a la lista de grafos y se llama al método recursivo “componenteConexoAux”, el cual realiza una búsqueda en profundidad entre los adyacentes, verificando que si tienen el mismo interés, son agregados al subgrafo y marcados como visitados. Esta implementación se realizó así debido a que era necesario recorrer todos los usuarios conectados entre sí de cada componente, por lo que la idea de realizar una búsqueda en profundidad resultaba lo más óptimo.

Para finalizar el método “amigosDeComunidad”, este toma la lista de subgrafos de “componentesConexos” e itera entre los usuarios de cada grafo, agregándolos a la lista de usuarios a recomendar, retornándola como último paso. El tiempo de ejecución de este algoritmo se encuentra marcado principalmente por la iteración final, ya que esta se realiza en un tiempo O(n^2).

Por otro lado, para realizar la función de recomendar amigos de amigos indicando la cantidad de amigos en común, se implementó el método “amigosDeAmigos” que toma como parámetro la etiqueta del usuario y devuelve un diccionario con el usuario a recomendar y la cantidad de amigos que tienen en común. Este método realiza una búsqueda en amplitud a nivel de usuario para seleccionar los amigos de los amigos que no fueron ni visitados ni son el propio usuario. Se realizó una búsqueda en amplitud debido a que se necesitaba recorrer los usuarios conectados de cada amigo por separado, ya que la idea principal es que sean amigos de los amigos del usuario. Esta búsqueda no es recursiva debido a que solo necesita llegar al segundo nivel de profundidad.   
Este algoritmo tiene un tiempo de ejecución marcado por la búsqueda en amplitud ya que toma un tiempo de ejecución de O(n\*v) donde a es la cantidad de amigos del usuario y v la cantidad de amigos de los amigos O(n^2).

Para completar esta última implementación, se creó la clase “TClasificador” donde se encuentra el método de heapSort utilizado para ordenar la secuencia de amigos en común de la recomendación de amigos. Este método convierte el map en una lista con el par clave-valor, arma el heap inicial, ordena los elementos en el heap y finalmente crea el mapa ordenado. Se utilizó el método de ordenación heapsort debido a que su tiempo de ejecución es de O(nlogn) para todos los casos, resultando en el más eficiente.  
  
Para facilitar la carga del grafo desde archivos, se creó la clase UtilGrafosRed que contiene el método cargarGrafoRed. Este método lee dos archivos, uno para los vértices (usuarios) y otro para las aristas (amistades).  
El método comienza leyendo los archivos para luego crear listas de usuarios y aristas. Itera sobre las líneas del archivo de vértices para crear instancias de TUsuario y agregarlas a la lista de usuarios. Para las aristas, itera sobre las líneas del archivo correspondiente y crea instancias de TArista, agregándolas a la lista de aristas.

Finalmente, se crea y retorna una instancia de TGrafoRedSocial utilizando las listas de usuarios y aristas

Por último, paso se creó la clase “Programa” que sirve como punto de entrada para ejecutar la funcionalidad del grafo de la red social.

En el método main de la clase Programa, se comienza cargando el grafo utilizando el método cargarGrafoRed de la clase UtilGrafosRed. A continuación, se crea un nuevo usuario (TUsuario) con el interés "futbol" y se inserta en el grafo. Después, se utilizan los métodos del grafo para mostrar posibles amigos con el mismo interés, donde se invoca el método amigosDeComunidad. Se imprime la lista de usuarios que comparten el mismo interés que el nuevo usuario.

Luego, se buscan dos usuarios específicos en el grafo y se establecen amistades con el nuevo usuario. Esto permite demostrar la funcionalidad de agregar amistades en el grafo.

Seguidamente, se muestran los posibles amigos de amigos del nuevo usuario, junto con la cantidad de amigos en común. Este proceso se realiza utilizando el método amigosDeAmigos, y los resultados se imprimen en la consola.

Finalmente, se ordenan los posibles amigos de amigos utilizando el método heapSort de la clase TClasificador. El mapa ordenado se imprime para mostrar los amigos sugeridos de acuerdo a la cantidad de amigos en común, demostrando la funcionalidad completa del sistema de recomendaciones basado en el grafo de la red social.