

Nombre y apellido: _____

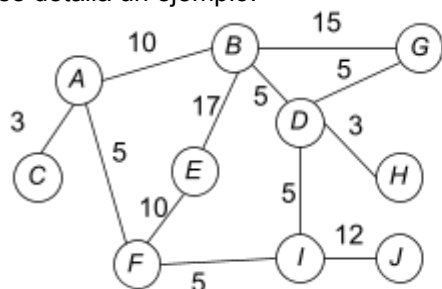
Ejercicios (puntaje)			Nota
1 (4 pts)	2 (3 pts)	3 (3 pts)	

CONSIDERACIONES: RESOLVER CADA EJERCICIO EN UNA HOJA DIFERENTE. LOS EJERCICIOS QUE NO ESTÉN CORRECTOS EN UN 50% NO SUMARÁN PUNTOS PARA LA NOTA FINAL. PARA CADA EJERCICIO DEBE **DEFINIR EL TIPO DE DATO** DE CADA TDA UTILIZADO. **EN TODOS LOS CASOS QUE UTILICE ESTRUCTURAS QUE NO SEAN TDAs "ESTÁNDAR" DEBE DEFINIR LOS STRUCTS.**

Ejercicio 1: Dada una red social modelada por un grafo ponderado, **desarrollar una función que devuelva una lista dinámica simplemente enlazada con los contactos sugeridos para un determinado perfil (contact)**. El vínculo entre dos perfiles es ponderado en función de las interacciones por mes que existen entre ellos y se van a considerar como contactos sugeridos aquellos que sean amigos de un contacto amigo con valor mayor a **Na** interacciones por mes y a su vez la interacción con el amigo en común tiene que ser mayor a **Naa** interacciones por mes.

Encabezado función principal: `fs_list* friendship_suggestion (graph* social_networking, t_graph_elem contact, int Na, int Naa );`

A continuación se detalla un ejemplo:



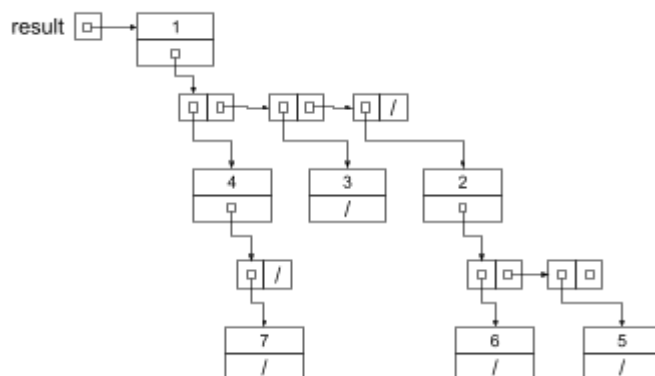
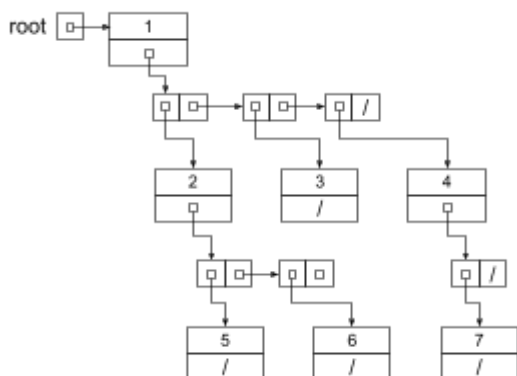
Para D con $Na = 5$ y $Naa = 10$ el vector de sugerencias de amistad devuelto es:



Ejercicio 2: Desarrollar una función y todas las necesarias para, dado un vector (TDA Vector) de punteros a personas (con DNI, NOMBRE, APELLIDO) ordenado por apellido, **crear un árbol binario de búsqueda de personas ordenado por apellido**. Resolver lo más eficientemente posible, de modo que el árbol quede balanceado.

Encabezado función principal: `btn* sbt_create_from_vector (vector* v);`

Ejercicio 3: Desarrollar una función y todas las necesarias para que, dado un árbol n-ario implementado con listas dinámicas enlazadas, se **crea un árbol nuevo clonando el árbol en espejo**; es decir, el primer hijo de cada nodo debe ser el último y viceversa. Ejemplo:



Encabezado función principal: `ntn* ntn_create_mirror (ntn* root);`