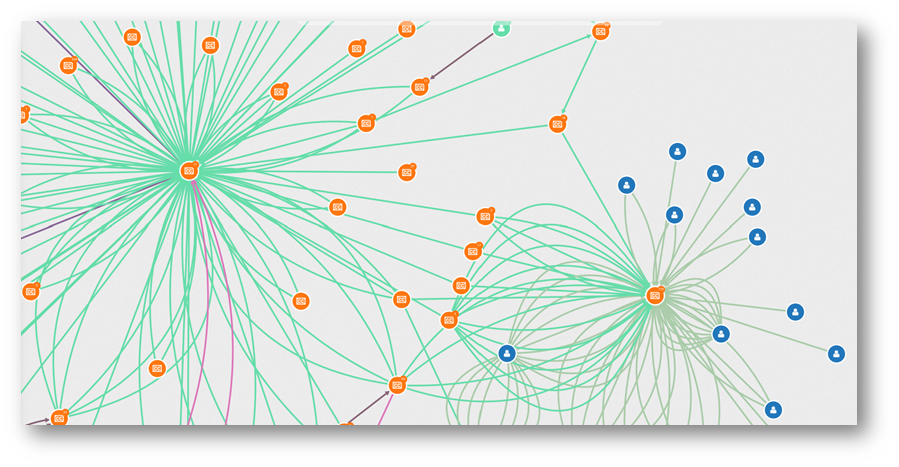


Informe estructura de datos



Integrantes:

* Vicente Córdova
* Hian Lart

# Problema:

Están en una empresa que desea almacenar sus cargos roles, permisos y personas en un programa en C, por lo que les solicita crear a través de “Linked List” una infraestructura que soporte los datos de la empresa. Esta tiene por finalidad poder consultar en cualquier dirección por lo que si deseo preguntar por una persona el programa debe arrojarme los permisos de ella o roles o su cargo dependiendo de la consulta; o en el caso de que pregunte por un cargo me de todos sus roles o sus permisos o las personas en ese cargo.

Diagrama

Descripción generada automáticamente

Con este se busca que la empresa pueda consultar por cualquier elemento de las listas y obtener los datos que están ligadas a ella por ejemplo si pregunto por el permiso de “Edición de archivos gerenciales” debería ser capaz de responderme por Raúl, Analista y gerente.

Los métodos de construcción no fueron especificados por la empresa además de las 4 Linked List y el uso de algoritmos de búsquedas presentados en la clase por lo que Uds. Pueden construirlo de la manera que quieran, pero en C.

# Estructura de datos:

El grafo que observamos anteriormente presenta una estructura de nodos los cuales se encuentra entrelazados, la estructura consta de 4 listas, las cuales presentan un nodo inicial y un nodo final, los nodos de las listas se encuentran entrelazados, es decir el primer nodo de la primera lista, presenta una conexión con el ultimo nodo de la última lista.

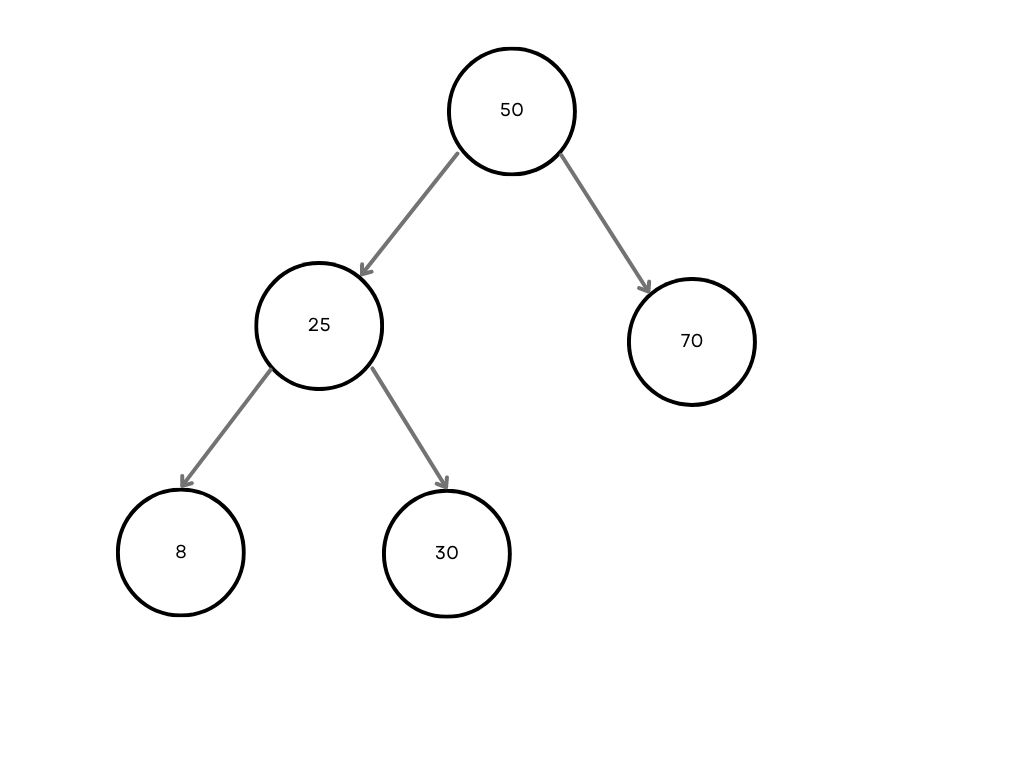
Los nodos representan puestos de la empresa, labores, funciones y nombres de los trabajadores que realizan y poseen dichos puestos.

# Solución:

* La solución a nuestro problema hemos decidido inicialmente plantearla en C, posteriormente la hemos implementado en Python, para el desarrollo de nuestra solución debemos comprender la lógica que presenta el grafo presentado.
* Comenzamos creando una lista la cual puede presentar un nodo inicial o final de una máximo de 20 caracteres que estará apuntando a un nodo inicial y a un nodo final.
* Asignaremos un valor en la memoria. Podemos observar que los nodos iniciales solo presentan un nodo hacia la derecha, esto en el código está representado por “next”, es decir, el recorrido de la lista avanzará hacia la derecha, por lo que siempre que nos posicionemos en el nodo inicial el único movimiento posible será “next”. En los nodos interiores, es decir, ni el inicial ni el final, presentan mas opciones, algunos nodos presentan conexiones con la lista inferior, es decir los nodos presentan 2 ramas inferiores, por lo que un nodo interior puede avanzar hacia adelante, atrás o hacia abajo en 2 opciones.
* La función que crea una lista enlazada se le entregará cual es el nodo inicial y cuáles son sus nodos adyacentes, en la primera lista de cargos, el cargo de “Gerente” presenta dos roles, “Analista” y “Administrador”, estos 2 roles cada uno presentan permisos independientes por lo que están conectados, pero a la vez separados. Para comprender esto de mejor manera podemos decir que un cargo puede presentar mas de un rol, mas de un permiso, pero un permiso y un rol solo se pueden direccionar hacia un cargo individualmente.
* Un nodo inicial en la primera lista de cargos vendría siendo “Gerente” y un nodo final en esta misma lista vendría siendo “Programador”, estos nodos solo podrán avanzar y bajar o retroceder y bajar, respectivamente.
* Al implementar la solución en C y Python, la búsqueda de los nodos y sus datos adyacentes, están construidos de distinta forma ya que en C tenemos un recorrido que avanza hacia la izquierda y posteriormente a la derecha, es decir, pasa sobre los mismos datos 2 veces, antes de encontrar el nodo en búsqueda, además, el muestreo se realiza de manera incorrecta ya que repite y/o omite datos. Mientras que en Python, la búsqueda ingresamos a nuestra primera lista, y comienza a realizar la búsqueda de manera más eficiente ya que recorremos cada lista hasta encontrar el nodo solicitado, además, el muestreo de datos se presenta de manera correcta y sin repetición ni omisión de datos.

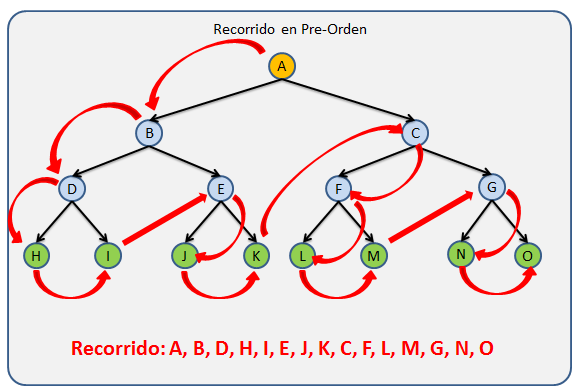
# Problema 2:

Se desea desarrollar un programa para la administración de un árbol ordenado con información de tipo int. Recorrer el árbol en pre, entre y post orden.



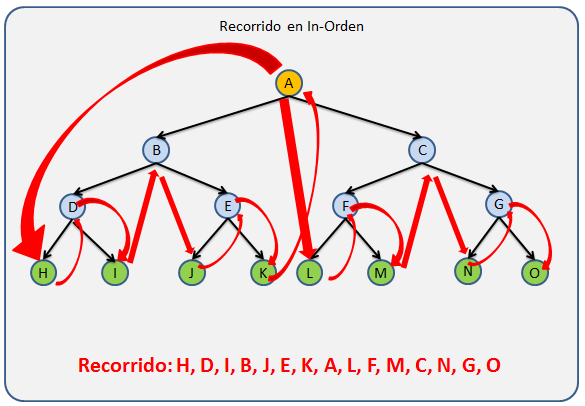
Recorrido preorden:

Se recorre desde el nodo raiz y luego va por los subarboles de la izquierda hasta la hoja del sub-árbol izquierdo, luego va por el sub arbol del nivel más alto no recorrido del lado derecho, asi sucesivamente hasta recorrer la hoja del ultimo sub-árbol derecho.



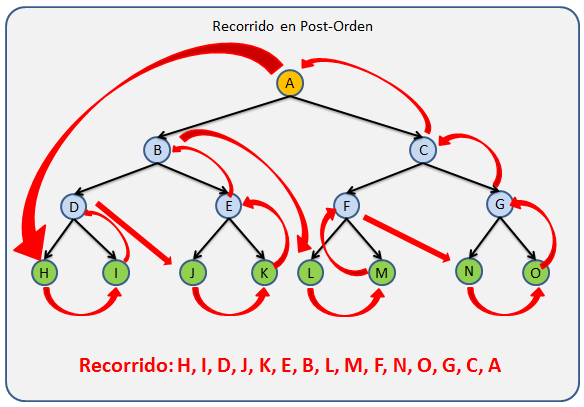
Recorrido in-orden:

Este recorrido lo que hace lo que se hace es que se visita el sub-árbol izquierdo, luego se visita la raíz y después el sub-árbol derecho.



Recorrido post-orden:

El recorrido que se hace es visitar el sub-árbol izquierdo, luego el sub-árbol derecho y por ultimo la raíz.



# Estructura de datos

Como podemos observar el 50 es nodo raíz, los cuales sus hijos son los nodos 25 y 70, por lo cual estos son nodos hermanos, y los hijos del nodo 25 son los nodos 8 y 30, estos nodos son hermanos, y a la vez también son nodos hojas. Este árbol binario no lleno tiene un peso de 5, altura de 3.

# Solución:

Para la solución de este problema que realizamos fue crear una función insertar el cual pide el número que va a tener el nodo, si no hay un nodo raíz, el primer nodo que se va a insertar va a ser la raíz, si no se va a buscar la posición correcta para insertar el nuevo nodo. Luego creamos funciones para poder mostrar el árbol de manera pre, in y post orden.