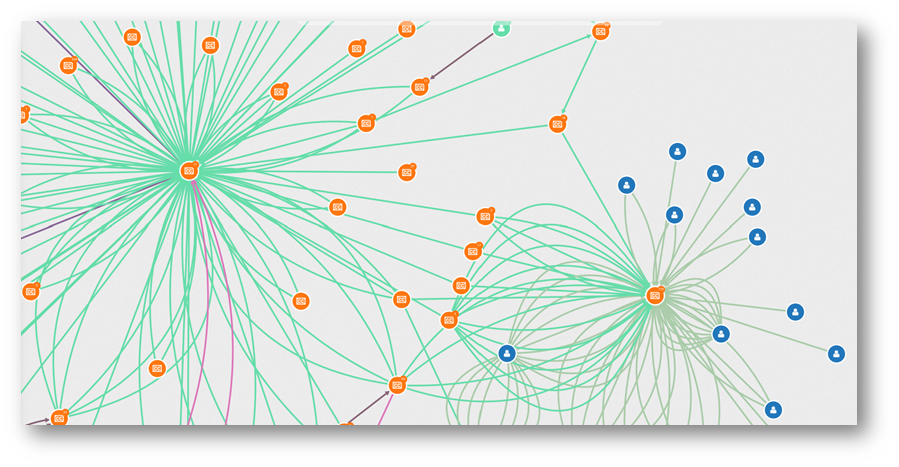


Informe estructura de datos



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.

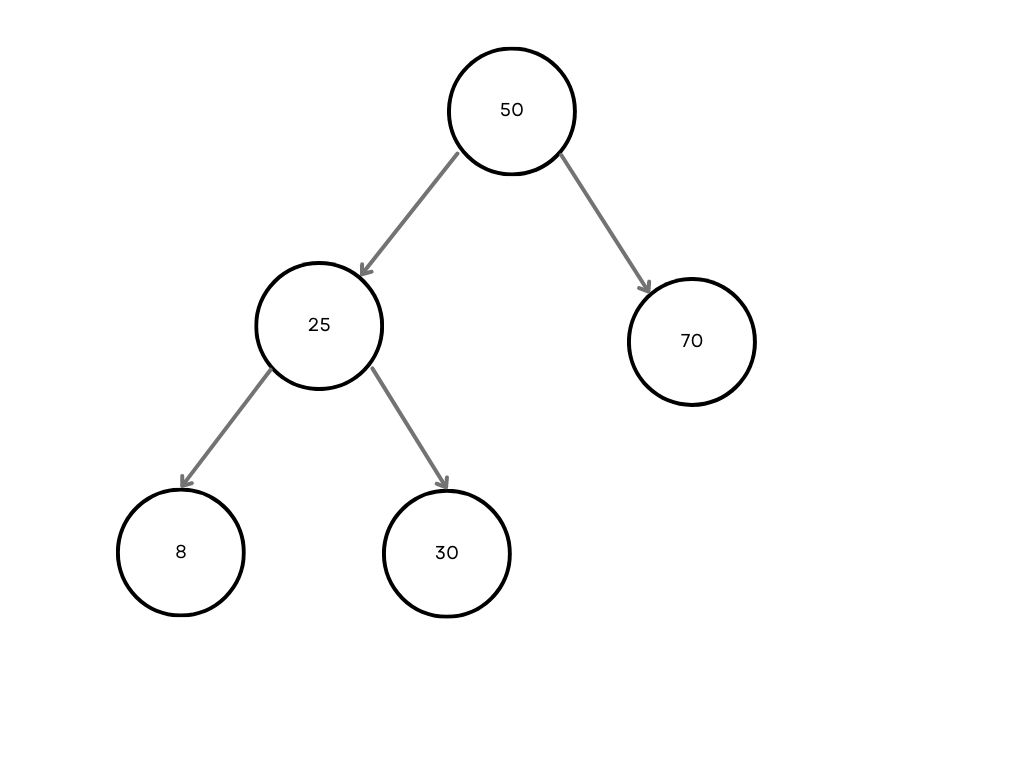
Los nodos representan puestos de la empresa, labores, funciones y nombres de los trabajadores que realizan y poseen dichos puestos.

# 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 más 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 más de un rol, más 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.

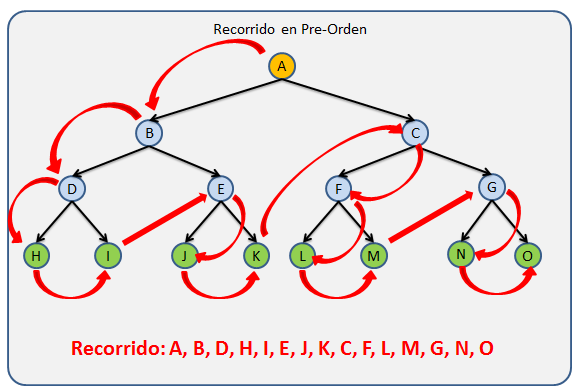
# Problema 2:

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.



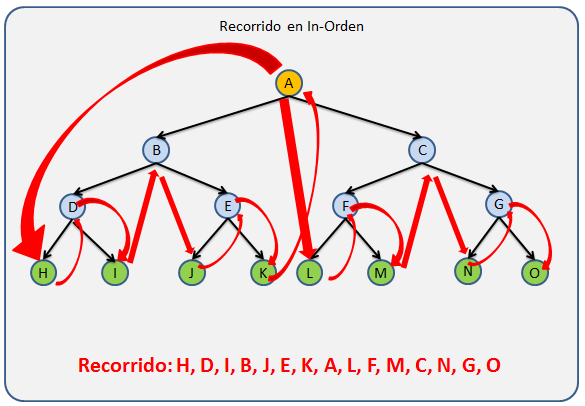
Recorrido preorden:

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.



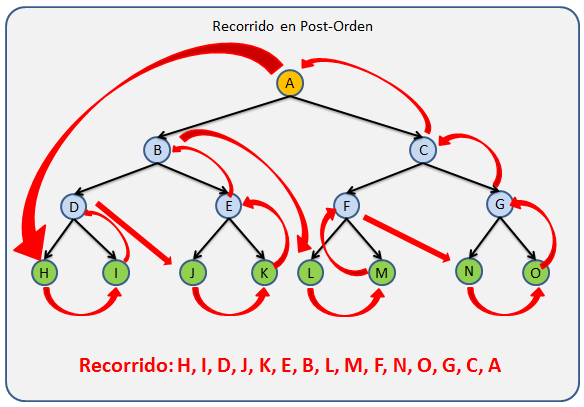
Recorrido in-orden:

Este tipo de recorrido consiste en visitar primero el subárbol izquierdo, luego la raíz y finalmente el subárbol derecho.



Recorrido post-orden:

En este recorrido se procede a visitar inicialmente el subárbol izquierdo, seguido por el subárbol derecho y, por último, se visita la raíz del árbol.



# Estructura de datos

# Como puede apreciarse, el nodo raíz de este árbol es el número 50. Sus nodos hijos son el 25 y el 70, lo cual los clasifica como nodos hermanos. A su vez, el nodo 25 tiene como hijos los nodos 8 y 30, los cuales también son considerados nodos hermanos. Es importante destacar que estos nodos 8 y 30 son, a su vez, nodos hojas. En cuanto a las características del árbol, se trata de un árbol binario no lleno con un peso de 5 y una altura de 3.

# Solución:

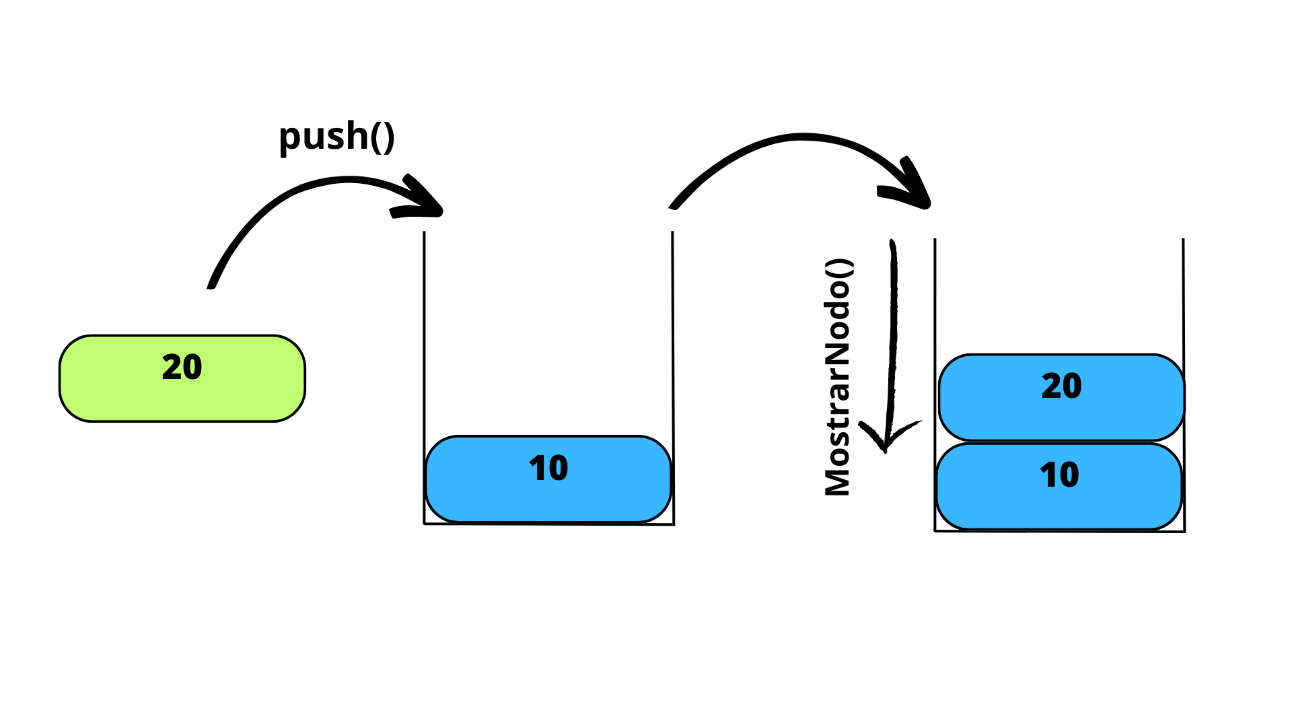
Para abordar la resolución de esta problemática, hemos desarrollado una función denominada "insertar". Dicha función solicita el número asignado al nodo a insertar. En caso de que no exista un nodo raíz previamente establecido, el primer nodo a insertar será considerado como la raíz del árbol. En caso contrario, se buscará la posición adecuada para la inserción del nuevo nodo dentro de la estructura. Además, hemos implementado funciones adicionales con el propósito de visualizar el árbol en los distintos recorridos: preorden, inorden y postorden.

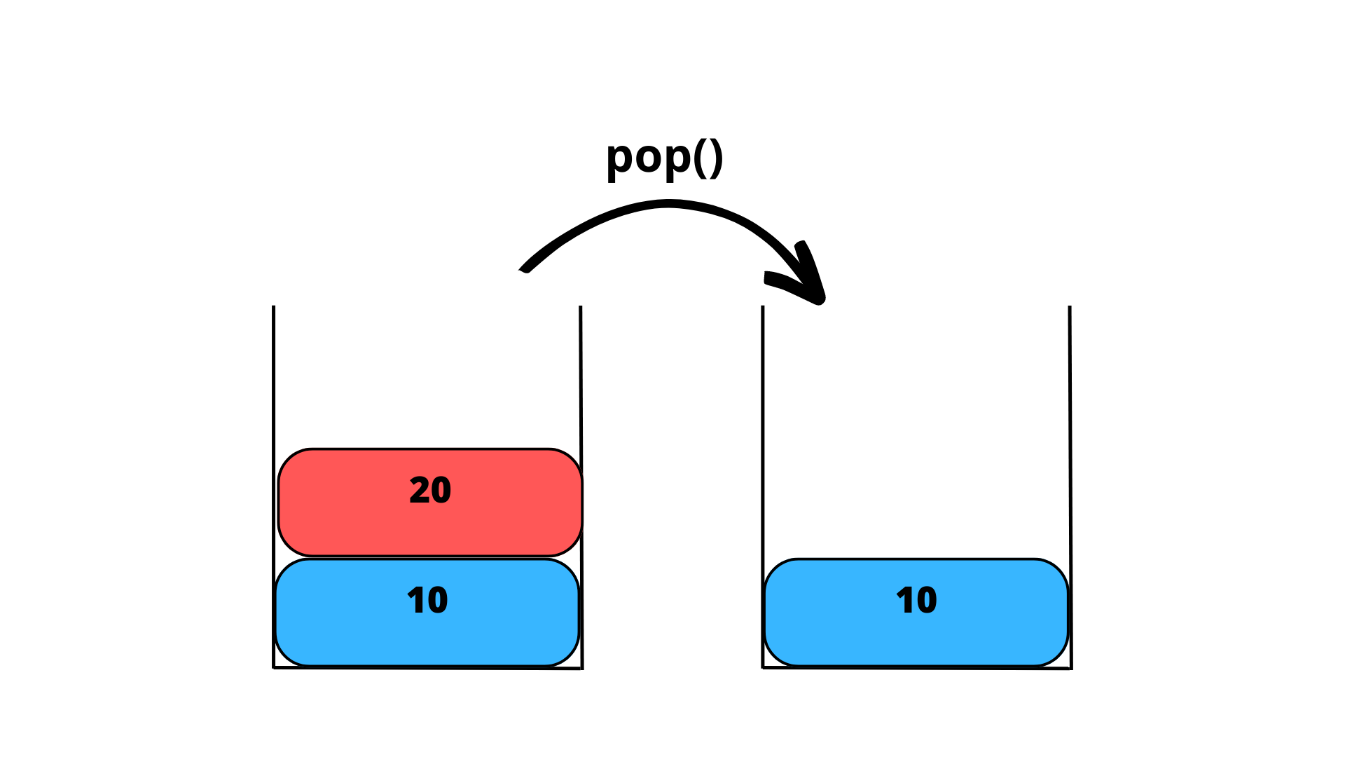
# Problema 3:

Se necesita implementar una estructura de datos conocida como "pila" que almacenará exclusivamente valores enteros. La pila debe contar con una función denominada "pop" que permita extraer un nodo de la pila según la solicitud del cliente.

Restricciones y requisitos:

1. La estructura de datos utilizada será una pila.
2. Los elementos almacenados en la pila serán exclusivamente números enteros.
3. La función "pop" deberá estar disponible para que el cliente pueda extraer un nodo de la pila.
4. La pila deberá ser implementada utilizando las operaciones y métodos adecuados para su correcto funcionamiento y manipulación.
5. Se requiere que la función "pop" extraiga el nodo de la pila de manera adecuada, respetando el principio "LIFO" (Last In, First Out) de las pilas.
6. En caso de que la pila esté vacía al momento de solicitar la operación "pop", se deberá manejar adecuadamente esta situación y proporcionar una respuesta o acción apropiada.





# Estructura de datos

En el contexto de esta problemática, se emplea una estructura de datos conocida como pila. La pila se asemeja en ciertos aspectos a una cola, pero se distingue por su característica LIFO (Last-In, First-Out), lo cual implica que el último elemento en ser insertado en la pila será el primero en ser extraído. En otras palabras, al realizar la operación "pop", se eliminará el nodo ubicado en la cima de la pila. Cabe destacar que cada componente de la pila tiene un único sucesor y predecesor, excepto por el primer y último elemento.

# Solución

Implementamos una función push() el cual va a guardar memoria para el nodo, definimos sus valores, otra función pop() para borrar el ultimo nodo de la pila y por ultimo la ultima función para MostrarPila() que muestra desde el ultimo nodo hasta el primero.

# Problema 4

Assdadsa

# Estructura de datos