Universidad José Antonio Paez.
Facultad de Ingeniería.
Escuela de Ingeniería en Computación.

Simulador de Red de Dispositivos (LAN) — CLI Estilo Router

En el proyecto anterior y sobre el mismo repositorio, programar e implementar los siguientes módulos

Módulo 1: Tabla de Rutas Balanceada con AVL *(5 ptos)*

En este módulo se implementará una **lista enlazada** donde cada nodo representa la tabla de enrutamiento de un dispositivo de la red. Dentro de cada nodo de la lista, la información de rutas se organizará usando un árbol AVL

Aquí la **tabla de enrutamiento** de cada router se mantiene en un **árbol AVL** balanceado por clave de *prefijo* (ej. 10.0.0.0/24) y, secundariamente, por *métrica*. Cada inserción o borrado dispara las rotaciones necesarias para mantener altura O(log n). El reenvío consulta el "**mejor prefijo**" (longest-prefix match simplificado por orden y comparación de máscaras) y obtiene el siguiente salto.

El reenvío del Módulo 2 usa lookup(dest_ip) sobre el AVL; si no hay coincidencia, se recurre a vecinos directos (BST ARP) o se descarta. El módulo de estadísticas cuenta rotaciones para fines didácticos.

Comandos.

ip route add <prefix> <mask> via <next-hop> [metric N]
ip route del <prefix> <mask>
show ip route
show route avl-stats # nodos, altura, rotaciones LL/LR/RL/RR, mostrar arbol

Ejemplo.

Router1# ip route add 10.0.0.0 255.255.255.0 via 192.168.1.2 metric 10
Router1# show ip route
10.0.0.0/24 via 192.168.1.2 metric 10
Default: none
Router1# show route avl-stats
nodes=5 height=3 rotations: LL=1 LR=0 RL=0 RR=2
Router1# show ip route-tree

```
          [172.16.0.0/16]
      /                  \
[10.0.0.0/24]          [192.168.1.0/24]
                    /            \
```

[192.168.0.0/24] [200.200.200.0/24]

Módulo 2: Índice Persistente de Configuraciones con B-tree *(5 ptos)*

Para potenciar la **persistencia** del Módulo 6, se incorpora un **B-tree** como índice en disco de *snapshots* de configuración y de logs. Cada clave puede ser un timestamp (2025-08-12T09:30) o un nombre ("lab-grupoA"), y el valor es el puntero al archivo de configuración/log. El B-tree brinda inserciones y búsquedas en O(log n) aún con gran volumen de snapshots.

Cada save running-config crea un snapshot y lo **indexa** en el B-tree. load config <key> primero resuelve <key> en el índice y luego carga el archivo apuntado.

Comandos.

```
save snapshot <key>          # guarda y indexa con <key>
load config <key>            # resuelve <key> en el B-tree y carga
show snapshots               # recorre el B-tree en-orden
btree stats                  # orden, altura, nodos, splits/merges
```

Ejemplo.

```
Router1# save snapshot lab-grupoA
[OK] snapshot lab-grupoA -> file: snap_00017.cfg (indexed)

Router1# show snapshots
2025-08-12T09:30 -> snap_00016.cfg
lab-grupoA       -> snap_00017.cfg

Router1# btree stats
order=4 height=2 nodes=7 splits=3 merges=0
```

Módulo 3: Árbol N-ario (Trie) para Prefijos IP y Políticas Jerárquicas *(5 ptos)*

Este módulo añade un **trie** (árbol n-ario) de prefijos donde cada nodo representa un bit (trie binario) o un octeto (trie base-256). El trie sirve para dos propósitos: (1) **resolución de prefijos** eficiente (longest-prefix match "real") y (2) **herencia de políticas** por segmentos (QoS, bloqueo, límites de TTL, etc.). Una política aplicada a un nodo se hereda por todos los subprefijos, salvo override explícito.

Antes del reenvío, el router consulta el trie para obtener ruta/política por *longest-prefix*. Si existe una política de "TTL mínimo" o "bloqueo", se aplica; de lo contrario continúa con la ruta del AVL. El trie también permite visualizar la estructura jerárquica de la red.

Comandos.

policy set <prefix> <mask> ttl-min <N>
policy set <prefix> <mask> block
policy unset <prefix> <mask>
show ip prefix-tree # imprime el trie (prefijos y políticas)

Ejemplo.

Router1# policy set 10.0.0.0 255.255.0.0 ttl-min 3
Router1# policy set 10.0.2.0 255.255.255.0 block
Router1# show ip prefix-tree
10.0.0.0/16 {ttl-min=3}
└── 10.0.1.0/24 {}
└── 10.0.2.0/24 {block}

Módulo 4: Registro de Errores (3 ptos)

En este módulo se implementará un **sistema de registro de errores** que almacenará, en tiempo real, cualquier fallo que ocurra dentro del simulador: intentos de conexión a dispositivos inexistentes, envío de paquetes a direcciones inválidas, ejecución de comandos no reconocidos, fallos de sintaxis, o cualquier otra condición excepcional que impida completar una acción.

Para garantizar que el log se mantenga **en orden cronológico** y sea fácil de consultar, se utilizará **una lista enlazada implementada como cola**:

- Cada nuevo error se **inserta al final** (enqueue).
- La lectura o extracción de errores se hace desde el principio (dequeue), asegurando que los mensajes se muestren en el orden en que ocurrieron.

Cada nodo de la lista contendrá:

- **Marca de tiempo** (fecha y hora exacta del error)
- **Tipo de error** (por ejemplo: `SyntaxError`, `ConnectionError`, `CommandDisabled`)
- **Mensaje descriptivo** (explicación detallada del problema)
- **Comando que lo provocó** (si aplica)

El comando principal para consultar el log será:

```
show error-log
```

Este mostrará todos los errores almacenados en la cola, pudiendo opcionalmente limitar la cantidad de registros mostrados:

```
show error-log [n]
```

Donde `[n]` es la cantidad máxima de entradas recientes a visualizar.

Cuando un paquete llega a un router, el flujo queda así: primero se hace *lookup* de **políticas** en el **trie n-ario**; si el prefijo está bloqueado o viola una política (p. ej., ttl-min), el paquete se descarta y se contabiliza. Si pasa, el router consulta la **tabla AVL** para elegir el siguiente salto “mejor prefijo”. Para vecinos directos, valida o aprende la interfaz mediante el **BST de ARP**. Periódicamente, save snapshot registra el estado y el **B-tree** indexa estos archivos para búsquedas rápidas por clave o tiempo.

Todo árbol debe implementarse **desde cero**: nodos, inserción, búsqueda y borrado (donde aplique), más las invariantes de cada estructura (balance en AVL, orden y *splits/merges* en B-tree, prefijos y herencia en trie). Se deberán exponer **métricas internas** (show … stats) para que el corrector verifique altura, número de nodos y operaciones de balanceo. La CLI debe mostrar **recorridos** del árbol (show … imprime en-orden o en forma de diagrama ASCII), y se esperan **tests funcionales** que prueben casos extremos: borrado del nodo raíz, rotaciones dobles, *split* en raíz del B-tree, colisiones de claves, y prefijos anidados en el trie.

Pautas de Evaluación.

1. La evaluación es individual o en equipo de un máximo de tres personas.
2. Usar paradigma de programación orientada a objetos
3. Asistencia obligatoria
4. Utilizar repositorio tiene un valor de 2 ptos.
5. Los códigos iguales tendrán una penalización de puntos menos.
6. La entrega y defensa se realizará de forma presencial en hora de clases.
7. Realizar validaciones de datos introducidos por el usuario en los comandos y datos cargados.
8. Los datos deben ser guardados como archivos.json y cargados al momento de iniciar el programa.
9. El código deberá estar comentado.
10. Tener datos por defectos para tomarlos como prueba.
11. En cada módulo se evaluará los siguiente:
    - Funcionamiento del módulo(errores, resultados correctos, independencia).
    - Legibilidad del código(nombres de variables, código comentado correctamente)

- El alumno aplicó elementos conceptuales en la programación del módulo
- Módulo entregado puntualmente.