SIMULAZIONE DI UN PROTOCOLLO DI ROUTING
"DISTANCE VECTOR ROUTING"

INTRODUZIONE:

Il progetto ha lo scopo di simulare un protocollo di routing basato sull'algoritmo "Distance Vector Routing" (DVR). L'obiettivo è mostrare come i nodi di una rete scambino informazioni per calcolare i percorsi più brevi verso gli altri nodi della rete. Questa simulazione rappresenta una rete semplificata, utile per studiare i concetti fondamentali di routing.

OBIETTIVI:

Implementare una classe Nodo che gestisca le tabelle di routing per ciascun dispositivo.
Implementare una classe Rete che consenta di creare una topologia di rete e propagare i messaggi di routing.
Mostrare come le tabelle di routing si aggiornano fino a raggiungere una convergenza.
Produrre un codice Python leggibile, ben commentato e facilmente espandibile.

SCELTE PROGETTUALI:

- STRUTTURA MODULARE: il codice è suddiviso in tre file: "nodo.py", "rete.py", "main.py" per mantenere separate la logica dei nodi, della rete e della simulazione.

- ALGORITMO DVR: il progetto utilizza un semplice meccanismo iterativo per simulare l'aggiornamento delle tabelle di routing, emulando il comportamento reale dei protocolli come RIP.

- ORIENTAMENTO DEGLI OGGETTI: l'uso di classi (Nodo e Rete) rende il codice più leggibile e modulare, permettendo una facile manutenzione ed estensione.

- PROPAGAZIONE ITERATIVA: implementata con un ciclo "while", simula lo scambio continuo dei messaggi di routing fino alla convergenza.

- USO DI DIZIONARI: i dizionari permettono un accesso rapido ai dati e sono intuitivi per rappresentare coppie.

ELEMENTI USATI NEL PROGETTO:

1. NODO:

un nodo rappresenta un dispositivo nella rete(ad esempio un router).
Il suo ruolo è quello di memorizzare la tabella di routing che viene aggiornata basandosi sulle informazioni ricevute dai vicini.

- Tabella di routing:
  è strutturata utilizzando un dizionario Python {destinazione: (distanza, nodo_successivo)}.
  Inizialmente un nodo conosce solo i vicini diretti.
- Metodo "aggiorna_tabella":

riceve una tabella di routing di un nodo vicino e per ogni destinazione nella tabella del vicino calcola il percorso stimato passando attraverso il vicino e aggiorna la sua tabella se trova un percorso più breve.

2. RETE:

Modella la topologia della rete, gestendo nodi e connessioni.
Permette di aggiungere nodi e collegamenti e gestisce la propagazione dei messaggi di routing.

- Connessioni:
  sono strutturate utilizzando un dizionario annidato {nodo1: {nodo2: costo}}.
  Ogni connessione è bidirezionale.
- Metodo "propaga_messaggi":
  cicla su tutti i nodi simulando lo scambio delle tabelle di routing.
  Continua finché non viene raggiunta una convergenza delle tabelle.

3. ALGORITMO DISTANCE VECTOR:

il Distance Vector Routing Protocol è un metodo per calcolare i percorsi più brevi in una rete distribuita come le reti IP; ogni nodo della rete mantiene una tabella di routing dove registra:

- Destinazioni: tutti i nodi raggiungibili;
- Distanze: il costo del percorso verso ogni nodo;
- Nodi successivi: il prossimo nodo lungo il percorso verso una destinazione.

L'implementazione della logica VDR viene distribuita tra i metodi "aggiorna_tabelle" (in "nodo.py") e "propaga_messaggi" (in "rete.py").

Ogni nodo calcola il costo totale di un percorso sommando la distanza al vicino con la distanza registrata nella tabella del vicino.
Il calcolo viene eseguito come:

$$\text{Distanza}_{\text{stima}} = \text{Distanza}_V + \text{Costo}_{\text{link}}(\text{nodo}, V)$$

Dove V è il nodo vicino.
Se questa stima è inferiore alla distanza attualmente registrata, il nodo aggiorna la propria tabella.

La propagazione continua in modo iterativo fino al raggiungimento di uno stato stabile dove le tabelle di tutti i nodi riflettono i percorsi più brevi all'interno della rete.

RISULTATI:

Tabelle di routing iniziali: ogni nodo, inizialmente, conosce solo le connessioni dirette.

```
Tabelle di routing iniziali:
Nodo A, Tabella di Routing: {'B': (1, 'B'), 'D': (4, 'D')}
Nodo B, Tabella di Routing: {'A': (1, 'A'), 'C': (2, 'C')}
Nodo C, Tabella di Routing: {'B': (2, 'B'), 'D': (1, 'D')}
Nodo D, Tabella di Routing: {'C': (1, 'C'), 'A': (4, 'A')}
```

Tabelle di routing finali: dopo la propagazione, ogni nodo conosce il percorso più breve verso tutti gli altri nodi

```
Tabelle di routing finali:
Nodo A, Tabella di Routing: {'B': (1, 'B'), 'D': (4, 'D'), 'C': (3, 'B')}
Nodo B, Tabella di Routing: {'A': (1, 'A'), 'C': (2, 'C'), 'D': (3, 'C')}
Nodo C, Tabella di Routing: {'B': (2, 'B'), 'D': (1, 'D'), 'A': (3, 'B')}
Nodo D, Tabella di Routing: {'C': (1, 'C'), 'A': (4, 'A'), 'B': (3, 'C')}
```

CONCLUSIONI:

Il progetto dimostra come un protocollo di routing, basato su algoritmi di vettori di distanza, calcola percorsi ottimali. La struttura modulare del codice rende semplice aggiungere funzionalità avanzate come la gestione di fallimenti o l'integrazione di metriche più complesse.

Tommaso Valeriani (0001081795)
Federico Panzavolta (0001077805)