Progetto Python: Simulazione di Protocollo di Routing

Nome: Nikolai
Cognome: Zanni
Matricola: 0001069041

Nome: Emily
Cognome: Pedini
Matricola: 0001081969

Nome: Benedetta
Cognome: Barone
Matricola: 0001081612

DESCRIZIONE:
Creare uno script Python che simuli un protocollo di routing semplice, come il Distance Vector Routing. Gli studenti implementeranno gli aggiornamenti di routing tra i nodi, con il calcolo delle rotte più brevi.

OBIETTIVI:
Implementare la logica di aggiornamento delle rotte, gestione delle tabelle di routing e calcolo delle distanze tra nodi.

CONSEGNE RICHIESTE:
Codice Python ben documentato, output delle tabelle di routing per ogni nodo e relazione finale che spieghi il funzionamento dello script.

INTRODUZIONE:
Questo progetto, scritto in Python, ha l'obiettivo di simulare un protocollo di Distance Vector Routing (DVR), ricreando un ambiente di rete in cui i nodi determinano autonomamente i percorsi più brevi tra loro. Nel Distance Vector Routing, ogni nodo utilizza una metodologia distribuita per aggiornare e mantenere le proprie tabelle di instradamento tramite lo scambio periodico di informazioni con i suoi vicini.
Ogni nodo è implementato come un'istanza della classe Nodo e mantiene una propria tabella di instradamento che include i costi e i percorsi per raggiungere tutti gli altri nodi.
La simulazione si evolve fino a raggiungere uno stato di convergenza, cioè quando tutte le tabelle di instradamento si stabilizzano. Questo significa che ogni nodo ha la conoscenza del percorso ottimale per raggiungere ogni altro nodo.
Abbiamo progettato questa simulazione in modo che ogni passaggio del processo sia chiaro e ben documentato, rendendo il progetto un utile strumento per capire concretamente il funzionamento del protocollo di routing distribuito.

FUNZIONAMENTO DEL DISTANCE VECTOR:

Il protocollo Distance Vector Routing, che abbiamo implementato, si basa sull'algoritmo di Bellman-Ford per determinare i percorsi più brevi all'interno di una rete distribuita.
Vediamo le fasi principali che caratterizzano il funzionamento di questo protocollo:

1) *Algoritmo Bellman-Ford*: ogni nodo aggiorna iterativamente la sua stima delle distanze a tutti gli altri nodi basandosi sulle informazioni ricevute dai suoi vicini. La tabella di instradamento viene mantenuta sotto forma di mappa in cui ogni chiave è una destinazione e ogni valore è una coppia contenente il costo e il prossimo passo. Ad ogni aggiornamento ricevuto da un vicino, il nodo verifica se il nuovo percorso rappresenta un miglioramento e aggiorna la sua tabella di conseguenza tramite il metodo *aggiorna_tabella.*
2) *Distribuzione locale*: ogni nodo invia periodicamente ai suoi vicini il proprio vettore delle distanze tramite il metodo *invia_vettore_distanze*. Questo permette di condividere le informazioni più aggiornate sui percorsi. Questo processo distribuito assicura che i nodi collaborino per trovare le rotte più efficienti.
3) *Convergenza graduale*: la rete raggiunge uno stato di convergenza dopo un certo numero di iterazioni. All'inizio ogni nodo conosce solo le distanze dei suoi vicini diretti. Con ogni iterazione, i nodi aggiornano le proprie tabelle di instradamento fino a quando non ci sono più cambiamenti. Questo stato stabile rappresenta la convergenza.

STRUTTURA DEL CODICE:

Il progetto è organizzato in modo modulare per facilitare la comprensione e l'espandibilità.
Di seguito vengono descritti i componenti principali del codice:

1) *Classe Nodo*:
   - *Inizializzazione*: ogni nodo è rappresentato da un'istanza della classe Nodo, contenente un nome, una tabella di instradamento inizialmente vuota, e una lista di vicini diretti con i relativi costi.
   - *Aggiornamento della tabella*: la funzione *aggiorna_tabella* consente al nodo di aggiornare i percorsi verso tutti gli altri nodi sulla base delle informazioni ricevute dai vicini.
   - *Invio vettore delle distanze*: ogni nodo, tramite il metodo *invia_vettore_distanze*, genera il proprio vettore delle distanze da condividere con i vicini.
2) *Funzioni Principali*:
   - *simula_instradamento*: questa funzione simula il protocollo DVR sull'intero grafo della rete. Viene eseguita un'interazione continua di aggiornamenti delle tabelle fino al raggiungimento della convergenza.
   - *costruisci_vettore_distanze*: funzione che permette di convertire la tabella di instradamento in un formato vettoriale per semplificare l'invio delle informazioni.
   - *log_tabella*: utilizzata per registrare e stampare lo stato corrente delle tabelle di instradamento per ciascun nodo, facilitando il debug e la comprensione del processo di instradamento.
3) *Gestione del Grafo*:
   - *nodi*: ogni nodo è una chiave.
   - *archi e pesi*: ogni coppia nodo-vicino ha un peso che rappresenta il costo del collegamento.

EVOLUZIONE DELLE TABELLE:

1) *Inizializzazione*:
    - all'inizio della simulazione, ogni nodo conosce solo le distanze verso se stesso (costo 0) e verso i suoi vicini diretti. Tutte le altre distanze sono impostate su infinito (DISTANZA_MASSIMA).
2) *Iterazione*:
    - *passo 1*: ogni nodo condivide il suo Distance Vector con i vicini.
    - *passo 2*: i nodi ricevono i vettori dei vicini e aggiornano le proprie tabelle, ricalcolando i percorsi più economici. Questo avviene tramite il metodo *aggiorna_tabella.*
    - *passo 3*: questo processo continua finché non ci sono più aggiornamenti nelle tabelle di instradamento, segnalando il raggiungimento della convergenza.
3) *Convergenza*:
    - la rete raggiunge la convergenza quando ogni nodo conosce i percorsi più brevi verso tutti gli altri nodi. A questo punto, le tabelle non subiscono più cambiamenti.

ANALISI:

1) Output iniziale:
- Mostra le tabelle iniziali con solo i vicini diretti conosciuti.

Esempio:
DV(A) = {'A': 0, 'B': 1, 'C': 6, 'D': inf, 'E': inf}

2) Iterazioni: ad ogni iterazione, vengono registrati:
    - Le informazioni ricevute da un vicino.
    - Lo stato della tabella prima e dopo l'aggiornamento.
    - La nuova tabella aggiornata.

Esempio:
A riceve DV(B): {'B': 0, 'A': 1, 'C': 2, 'D': 1, 'E': inf}
A Tabella prima: [A: 0,A; B: 1,B; C: 6,C; D: inf; E: inf]
A Tabella dopo: [A: 0,A; B: 1,B; C: 3,B; D: 2,B; E: inf]

A riceve DV(C): {'C': 0, 'A': 6, 'B': 2, 'D': 3, 'E': 7}
A Tabella prima: [A: 0,A; B: 1,B; C: 3,B; D: 2,B; E: inf]
A Tabella dopo: [A: 0,A; B: 1,B; C: 3,B; D: 2,B; E: 13,C]

3) Stato finale:
- Una volta che non ci sono più aggiornamenti, le tabelle mostrano i percorsi ottimizzati.

Esempio:
DV(A): {'A': 0, 'B': 1, 'C': 3, 'D': 2, 'E': 4}