DOCUMENTAZIONE SCRIPT PER LA SIMULAZIONE DEL DISTANCE VECTOR ROUTING IN PYTHON

Nicolò Morini
nicolo.morini2@studio.unibo.it
Matr. 0001071241
Elaborato Reti di Telecomunicazioni - A.A. 24/25

Link al repository GitHub:

https://github.com/N1c0zz/Simulation-DistanceVectorRouting.git

INTRODUZIONE

Questo script riproduce il funzionamento del protocollo del **Distance Vector Routing**, partendo da una data rete presa in input come un grafo pesato, nella quale dopo un numero "n" di iterazioni (dipendenti dalla grandezza e topologia della rete) si arriva per ogni nodo ad avere un **Distance Vector** che mostra la distanza con tutti gli altri nodi della rete.

Il funzionamento di base del Distance Vector Routing è il seguente:

- Ogni router conosce solo le distanze verso i suoi vicini immediati, cioè i nodi adiacenti
- Ogni nodo comunica periodicamente con i suoi vicini per aggiornare le informazioni sulle distanze verso le altre destinazioni. Questo scambio di informazioni avviene utilizzando tabelle di routing che vengono aggiornate attraverso l'algoritmo.
- Quando un nodo riceve informazioni dai suoi vicini, aggiorna la propria tabella di routing. Se trova una nuova via più breve per una destinazione, la tabella viene aggiornata di conseguenza.
- L'algoritmo si ripete iterativamente fino a quando non ci sono più modifiche nelle tabelle, segnando così la convergenza.

SPIEGAZIONE DEL CODICE

All'avvio del programma viene chiesto all'utente di scegliere se eseguire l'algoritmo su una rete di default o se inserire manualmente la propria rete (nel caso in cui venga inserito un input errato, il sistema utilizza automaticamente la rete di default).
Questo passaggio viene gestito dalla funzione **choose_mode()**:

```python
def choose_mode():

    print("Vuoi utilizzare una rete predefinita o inserire la tua rete?")
    choice = input("Scrivi 'predefinita' per usare la rete di default o 'personalizzata' per inserirne una tu: ").strip().lower()

    if choice == 'predefinita':
        print("\nUtilizzando la rete predefinita:")
        print_network(default_network)  # Stampa la rete predefinita
        return default_network
    elif choice == 'personalizzata':
        return get_network_from_input()
    else:
        print("Scelta non valida, usando la rete predefinita.")
        print_network(default_network)
        return default_network
```

Nel caso in cui scelga di utilizzare la rete di default, quest'ultima è preimpostata nello script e viene stampata a video utilizzando la funzione **print_network(network)** che prende in input la rete scelta:

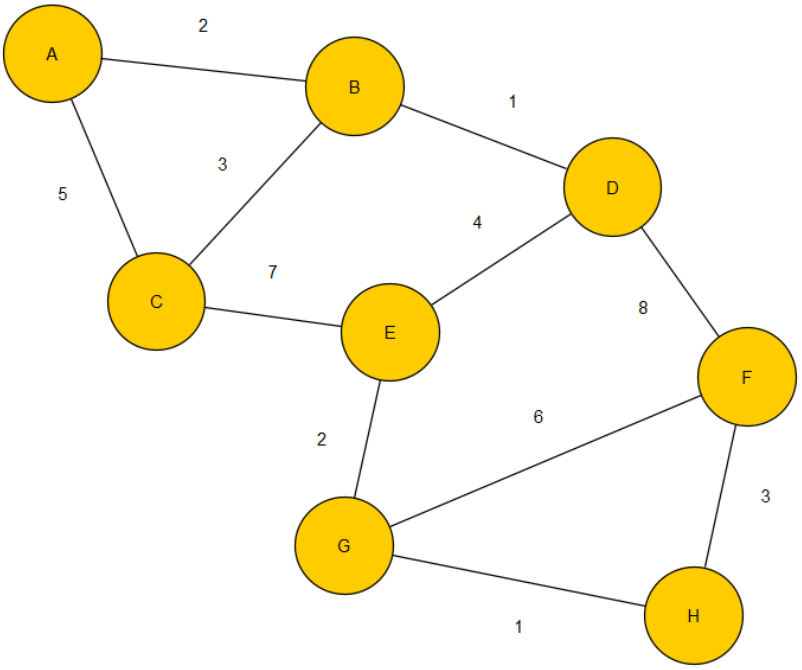
```python
def print_network(network):
    print("Rete attuale:")
    for node, connections in network.items():
        print(f"{node} -> {connections}")
```

(La scelta della rete di default è stata fatta in modo da avere un certo numero di iterazioni da visualizzare prima di arrivare alla convergenza, in modo da osservare al meglio il funzionamento del protocollo).

Rete di default:

```python
default_network = {
    'A': {'B': 2, 'C': 5},
    'B': {'A': 2, 'C': 3, 'D': 1},
    'C': {'A': 5, 'B': 3, 'E': 7},
    'D': {'B': 1, 'E': 4, 'F': 8},
    'E': {'C': 7, 'D': 4, 'G': 2},
    'F': {'D': 8, 'G': 6, 'H': 3},
    'G': {'E': 2, 'F': 6, 'H': 1},
    'H': {'F': 3, 'G': 1}
}
```

Grafo della rete di default:



Nel caso invece in cui l'utente scelga di inserire la propria rete, viene chiamata la funzione **get_network_from_input()**:

```python
# Funzione per acquisire la rete da input dell'utente.
# Permette all'utente di configurare manualmente una rete
# definendo nodi, connessioni e relativi costi.
def get_network_from_input():
    network = {}
    print("Configurazione manuale della rete...")

    # Numero di nodi della rete che si vuole inserire
    num_nodes = int(input("Inserire il numero di nodi nella rete:"))

    # Configurazione dei nodi e dei loro collegamenti
    for i in range(num_nodes):
        node = input(f"Inserisci il nome del nodo {i+1}: ")
        network[node] = {}
        print(f"Inserisci i collegamenti per il nodo {node}:")

        # Aggiunta dei collegamenti (destinazione e costo)
        while True:
            connection = input(f"Collegamenti per {node} (forma: Nodo destinazione,Costo) oppure 'fine' per terminare: ")
            if connection == "fine":
                break

            try:
                # Utilizzo la funzione split() sull'input per separare il nodo di destinazione
                # e il costo del collegamento nelle rispettive variabili
                dest, cost = connection.split(',')
                cost = int(cost)
                network[node][dest] = cost
                # Aggiungo anche il collegamento inverso (grafo non orientato)
                if dest not in network:
                    network[dest] = {}
                network[dest][node] = cost
            except ValueError:
                print("Errore nel formato del collegamento, riprova (es. 'B,1').")
    # Ritorno la rete completa da utilizzare
    return network
```

Dopo aver deciso la rete da utilizzare, lo script prosegue inizializzando le tabelle di routing per ogni nodo, attraverso la funzione **initialize_routing_tables(network)** :

```python
# Inizializzazione delle tabelle di routing con la seguente
# logica:
# - Distanza verso sé stesso: 0
# - Distanza verso i vicini: costo del collegamento
# - Distanza verso tutti gli altri nodi: infinito
def initialize_routing_tables(network):

    routing_tables = {}
    for node in network:
        # Inizializza tutte le distanze a infinito
        routing_tables[node] = {n: float('inf') for n in network}
        routing_tables[node][node] = 0  # La distanza verso sé stesso è 0
        for neighbor, cost in network[node].items():
            routing_tables[node][neighbor] = cost  # Distanza verso i vicini
    return routing_tables
```

La logica di aggiornamento delle tabelle basata sulle informazioni che i vicini si scambiano viene gestita dalla funzione **update_routing_tables(network, routing_tables)** :

```python
# Funzione per aggiornare le tabelle di routing.
# Aggiorna le tabelle di routing basandosi sulle
# informazioni ricevute dai vicini.
def update_routing_tables(network, routing_tables):
    updated = False  # Flag per verificare se ci sono stati cambiamenti
    new_tables = copy.deepcopy(routing_tables)  # Copia delle tabelle attuali

    for node in network:  # Per ogni nodo
        for neighbor, cost in network[node].items():  # Per ogni vicino
            for dest, distance in routing_tables[neighbor].items():
                # Calcola la distanza passando dal vicino
                if routing_tables[node][neighbor] != float('inf'):
                    new_distance = routing_tables[node][neighbor] + distance
                    # Se la nuova distanza è minore, aggiorna
                    if new_distance < new_tables[node][dest]:
                        new_tables[node][dest] = new_distance
                        updated = True  # Cambiamento rilevato

    return new_tables, updated
```

Infine, la funzione **simulate_distance_vector_routing(network, max_iterations=100)** simula il protocollo utilizzando le funzioni precedenti, andando ad ogni iterazione ad aggiornare le tabelle di routing fino allo stato di convergenza, nel quale non ci sono più modifiche da effettuare nelle varie tabelle.
Tra un'iterazione e l'altra c'è un sleep time di 1 secondo, inoltre viene passato alla funzione il parametro "max_iterations" che limita il massimo di iterazioni a 100, in modo da evitare situazioni di errore.

```python
# Simulazione del Distance Vector Routing.
# Simula il Distance Vector Routing aggiornando iterativamente
# le tabelle di routing fino alla convergenza o al raggiungimento
# del numero massimo di iterazioni.
def simulate_distance_vector_routing(network, max_iterations=100):
    # Inizializza le tabelle di routing
    routing_tables = initialize_routing_tables(network)
    print("\nInizializzazione delle tabelle di routing:")
    print_routing_tables(routing_tables)

    # Iterazioni per l'aggiornamento delle tabelle
    for iteration in range(max_iterations):
        print(f"\nIterazione {iteration + 1}:")
        new_routing_tables, updated = update_routing_tables(network, routing_tables)
        print_routing_tables(new_routing_tables)

        if not updated:  # Se non ci sono cambiamenti, la convergenza è raggiunta
            print("\nConvergenza raggiunta")
            print("\nTabelle finali dopo la convergenza:")
            print_routing_tables(new_routing_tables)
            break

        routing_tables = new_routing_tables  # Aggiorna le tabelle per la prossima iterazione

        time.sleep(1)  # Sleep di 1 secondo tra le iterazioni
    else:
        print("\nMassimo numero di iterazioni raggiunto senza convergenza.")

    return routing_tables
```

SOLUZIONE DEI PUNTI CRITICI NELLA REALIZZAZIONE DELLO SCRIPT

Punti critici incontrati durante la realizzazione dello script:

- **Rilevamento e gestione della convergenza**
  Una delle difficoltà più rilevanti è stata implementare un meccanismo per definire quando le tabelle di routing avessero raggiunto la convergenza. Ho utilizzato un *flag* booleano "updated" per tracciare i cambiamenti durante un'iterazione. Questo ha evitato aggiornamenti non necessari e migliorato l'efficienza dell'algoritmo, interrompendo il processo non appena la rete raggiungeva uno stato stabile.
- **Gestione dei cicli di aggiornamento**
  Durante il calcolo delle nuove distanze, potevano verificarsi situazioni in cui i valori oscillavano a causa di cicli nella rete. Per risolvere questo problema, ho applicato una logica che verifica se la distanza calcolata da un nodo verso un'altra destinazione attraverso un vicino è inferiore al valore esistente, aggiornandolo solo in questo caso. Questo ha eliminato comportamenti anomali nei risultati.
- **Test e debug**
  Per verificare l'accuratezza dell'algoritmo, ho testato il programma su reti predefinite di complessità variabile e simulato scenari estremi, come nodi isolati o costi molto elevati. Questi test mi hanno permesso di individuare e correggere errori, ad esempio nel calcolo delle distanze verso nodi non direttamente connessi.

ISTRUZIONI PER L'AVVIO DELLO SCRIPT

All'interno del folder dello script eseguire il seguente comando per l'esecuzione (necessita dell'interprete python installato):

- python main.py

Seguire le istruzioni stampate in output per scegliere l'esecuzione di default o l'inserimento di una rete personalizzata.