**1. Scopo dell'Algoritmo**

L'algoritmo **DVR** permette ai nodi di una rete di scambiarsi informazioni di routing per calcolare i percorsi più brevi verso altre destinazioni nella rete. Ogni nodo aggiorna la propria tabella di routing in base alle informazioni ricevute dai vicini, fino a raggiungere uno stato che non richiede più aggiornamenti.

**2. Componenti del Codice**

## Classi e Metodi

- **Node (Classe)**

Rappresenta un nodo della rete e contiene:

- **name:** il nome del nodo.
- **neighbors:** un insieme di nodi vicini diretti.
- **routing_table:** una tabella di routing che associa ogni destinazione a una coppia (distanza, gateway).

Metodi principali:

- **add_neighbor(neighbor, distance):** Aggiunge un nodo vicino con la distanza associata e inizializza la tabella di routing per tale vicino.
- **update_routing_table():** Aggiorna la tabella di routing del nodo sulla base delle informazioni ricevute dai vicini. Se si trova un percorso più corto, la tabella viene aggiornata e il metodo restituisce True per indicare che è stato fatto un aggiornamento.

- **print_routing_tables(nodes)**

Utilizza la libreria PrettyTable per stampare in modo leggibile le tabelle di routing di tutti i nodi della rete.

- **distance_vector_routing()**

La funzione principale che simula l'algoritmo DVR. Effettua iterazioni in cui ogni nodo aggiorna la propria tabella di routing finché non viene raggiunto uno stato stabile, cioè quando nessun nodo aggiorna più la sua tabella.

**3. Funzionamento dell'Algoritmo**

1. **Inizializzazione**

   Vengono creati i nodi A, B, C e D, definite le connessioni tra i nodi (vicinati) e le relative distanze (in questo caso una rete ad anello):

- A è connesso a B con distanza 1 e a D con distanza 4.
- B è connesso a A con distanza 1 e a C con distanza 2.
- C è connesso a B con distanza 2 e a D con distanza 3.
- D è connesso a A con distanza 4 e a C con distanza 3.

2. **Stampa delle Tabelle Iniziali**

Ogni nodo inizia con informazioni limitate sui vicini diretti, indicando per ogni vicino:

- La distanza verso il vicino.
- Il vicino stesso come gateway.

3. **Iterazioni**

Ogni nodo verifica se esistono percorsi più brevi verso altre destinazioni, utilizzando le informazioni fornite dai vicini. Questo processo segue la formula fondamentale del DVR:

Nuova Distanza=Distanza al vicino+Distanza dal vicino alla destinazione

Se si trova un percorso più breve, la tabella viene aggiornata. L'algoritmo continua finché nessun nodo trova più percorsi più corti.

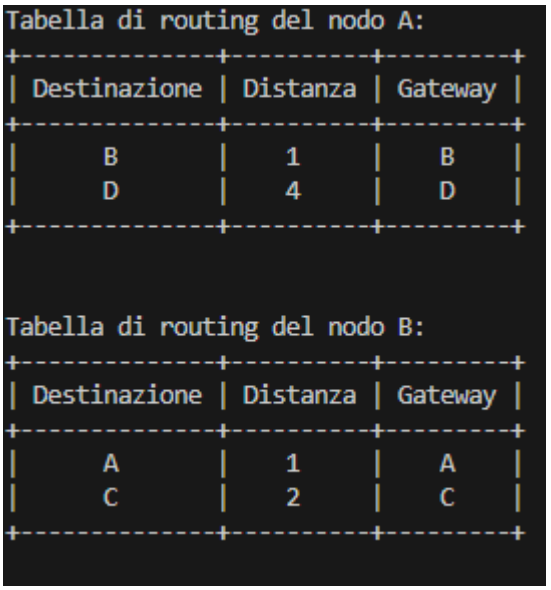
4. **Convergenza**

Dopo un certo numero di iterazioni, le tabelle di routing si stabilizzano. A questo punto, ogni nodo conosce i percorsi più brevi verso tutti gli altri nodi della rete, incluso il gateway da utilizzare per raggiungere ogni destinazione.

**4. Esempio di Output**

Durante l'esecuzione, l'output del programma mostra:

- Le tabelle di routing iniziali per ogni nodo.
- Gli aggiornamenti delle tabelle a ogni iterazione.
- Lo stato finale delle tabelle di routing, che rappresenta la convergenza del protocollo.



## 5. Tipi di Reti

Nel progetto i nodi sono stati scelti per simulare una rete ad anello, ma è possibile utilizzare reti di tipo diverso, di seguito alcuni esempi:

- **RETE LINEARE:**

```
A.add_neighbor(B, 2)

B.add_neighbor(A, 2)
B.add_neighbor(C, 3)

C.add_neighbor(B, 3)
C.add_neighbor(D, 1)

D.add_neighbor(C, 1)
```

- **RETE COMPLESSA: (più collegamenti per ogni nodo)**

```
A.add_neighbor(B, 1)
A.add_neighbor(C, 5)
A.add_neighbor(D, 3)

B.add_neighbor(A, 1)
B.add_neighbor(D, 1)
B.add_neighbor(E, 2)


C.add_neighbor(A, 5)
C.add_neighbor(E, 2)

D.add_neighbor(A, 3)
D.add_neighbor(B, 1)
D.add_neighbor(E, 1)

E.add_neighbor(B, 2)
E.add_neighbor(C, 2)
E.add_neighbor(D, 1)
```

- **RETE ASIMMETRICA:**

```
A.add_neighbor(B, 2)

B.add_neighbor(A, 2)
B.add_neighbor(C, 3)

C.add_neighbor(B, 3)
C.add_neighbor(D, 4)

D.add_neighbor(C, 1)
D.add_neighbor(E, 5)

E.add_neighbor(D, 1)
```