LINK STATE ROUTING

COMP4322 Internetworking

01

Link State Routing

Finding Shortest Path

Introductions

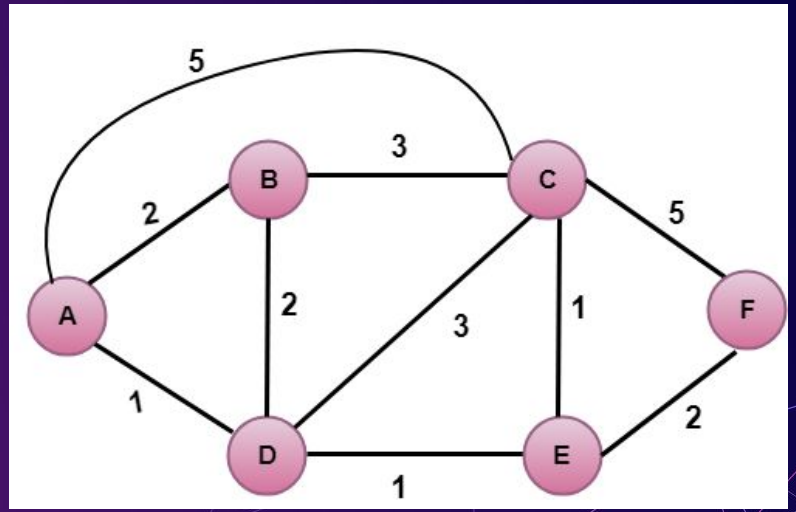
- The Goal is to have a full topography mapping of **optimal routes between each router**
- **Link State Database (LSDB) :** stores network topology information
- **Flooding** : router sends link information to all nodes except its neighbors
- **Information Sharing** : router sends information to every other router when information changes
- **Link State Advertisement (LSA)** : Contains router information, Connected links, State of Links

OSPF
Open Shortest Path First

IS-IS
Intermediate System to Intermediate System

Pseudocode

Initialization
N = {A} // A is a root node.
for all nodes v
if v adjacent to A
then D(v) = c(A,v)
else D(v) = infinity
loop
find w not in N such that D(w) is a minimum.
Add w to N
Update D(v) for all v adjacent to w and not in N:
D(v) = min(D(v) , D(w) + c(w,v))
Until all nodes in N



Advantages / Disadvantages

- Fast network convergence
- Routers individually determine shortest path.
- Event driven routing updates
- Handle Loops better

- High memory requirements - database and SPF tree
- Requires more processing power than distance vector protocol
- Initial flooding degrades performance of network

Time Complexity

- Worst Case for Dijkstra's algorithm is O(V^2) because it allows for directed cycles. V is the number of vertices.

- With min-prioirty queue, worst case drops to (V + ELogV), wherein V is the number of vertices and E is the number of Edges.

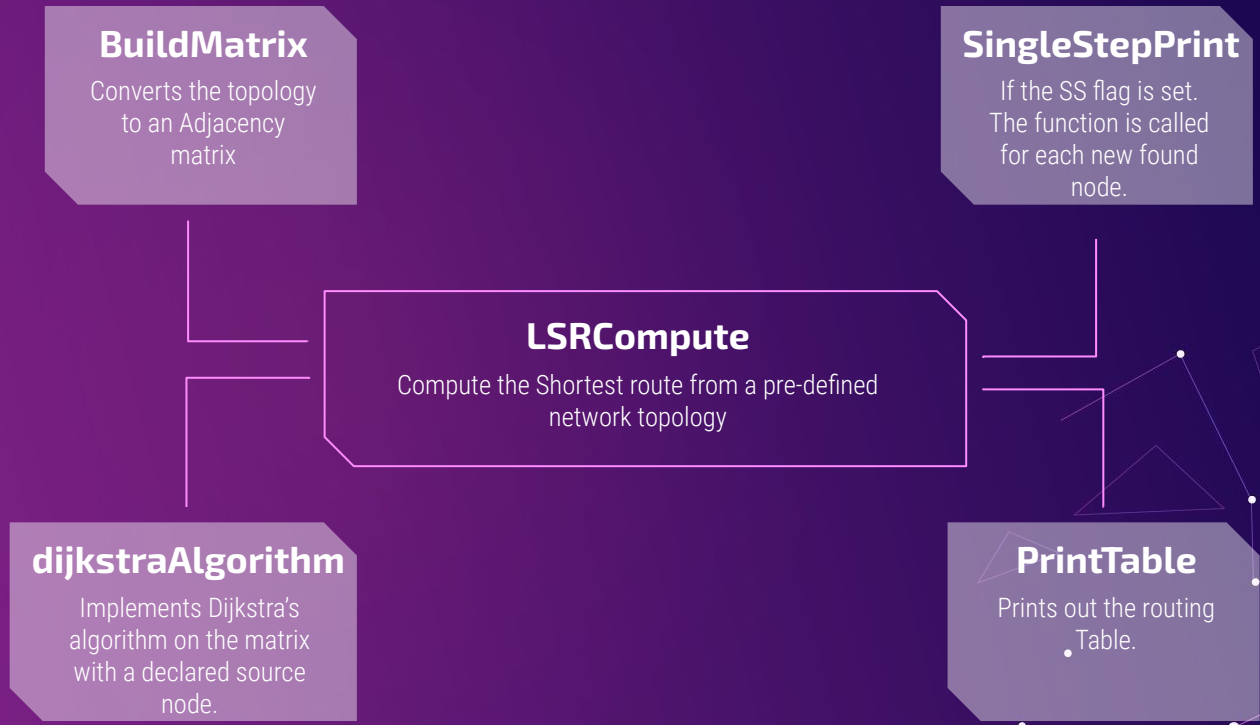
02

Implementation

Data Types and Variables

```
static int table[][]; // stores the adjacency matrix
static char nodes[]; // for mapping nodes back to respective characters
static boolean SSmode = false; // Single step flag
static int shortest[]; // Distance of the shortest patch for each node from the source
static int previous[]; // Previous node in the shortest path
static boolean visited[]; // Check if a node has been visited
static int source; // Source node
static int closest; // The new discovered node
static Stack<Integer> path; // Trace of the shortest path
```

Functionality



03

Demo!

User Guide

- The code can be cloned using :
  git clone https://github.com/harisch1/Link-State-Routing.git
- Use **cd ./src** to change directory to source code folder.
- The java class is compiled using jdk-18. However the source code can be recompiled using:
  javac LSRCompute.java
- And the program can be executed using:
  java LSRCompute [file_name] [source] [Execution mode]

Test for Single Step Mode

Input file:

A: B:2 C:7
B: A:2 C:4 D:4
C: A:7 D:1 F:5
D: B:6 C:1 E:4
E: D:4 F:1 G:1
F: C:5 E:1 G:3
G: E:1 F:3

```
PS C:\Users\haris\Downloads\PolyU\Internetworking\Project\Link State Routing\src> java LSRCompute routes.lsa E SS
Found F: E>F Cost = 1           [Press Enter key to continue...]

Found G: E>G Cost = 1           [Press Enter key to continue...]

Found D: E>D Cost = 4           [Press Enter key to continue...]

Found C: E>D>C Cost = 5         [Press Enter key to continue...]

Found B: E>D>B Cost = 10                [Press Enter key to continue...]

Found A: E>D>C>A Cost = 12              [Press Enter key to continue...]


Source: E
A: Path = E>D>C>A Cost = 12
B: Path = E>D>B Cost = 10
C: Path = E>D>C Cost = 5
D: Path = E>D Cost = 4
F: Path = E>F Cost = 1
G: Path = E>G Cost = 1
```

Test for Calculate All Mode

Input file:

A: B:2 C:7
B: A:2 C:4 D:4
C: A:7 D:1 F:5
D: B:6 C:1 E:4
E: D:4 F:1 G:1
F: C:5 E:1 G:3
G: E:1 F:3

```
PS C:\Users\haris\Downloads\PolyU\Internetworking\Project\Link State Routing\src> java LSRCompute routes.lsa E SS
Found F: E>F Cost = 1           [Press Enter key to continue...]

Found G: E>G Cost = 1           [Press Enter key to continue...]

Found D: E>D Cost = 4           [Press Enter key to continue...]

Found C: E>D>C Cost = 5         [Press Enter key to continue...]

Found B: E>D>B Cost = 10                [Press Enter key to continue...]

Found A: E>D>C>A Cost = 12              [Press Enter key to continue...]


Source: E
A: Path = E>D>C>A Cost = 12
B: Path = E>D>B Cost = 10
C: Path = E>D>C Cost = 5
D: Path = E>D Cost = 4
F: Path = E>F Cost = 1
G: Path = E>G Cost = 1
```

Thank You!