SIMULATE LINK-STATE ROUTING ALGORITHM
PROJECT REPORT

FNU Ashutosh
Illinois Institute of Technology
aashutosh@hawk.iit.edu
A20377427

Contents

Abstract

In the network, the link-state protocol is performed by every switching node. The node are known as routers which forwards the packets. The basic idea behind the Link-state algorithm is that it constructs a graph. The graph displays all the nodes and how all the nodes are connected together in a network. Each node then independently calculates the next best logical path from it to every possible destination in the network. A routing table is created using the collection of the shortest path from one source or starting node to another node. The main objective of the algorithm is to find out the shortest path from one router to another router in the network topology using the routing table. This project has been built using Python language. The project is done in such a way that it handles all the cases where there is a possibility of failure or a crash.

Objective

The objective of the projects is to develop a simulator. The simulator implements Link-State Routing Protocol. The project consists of the following functions.

1. doCaseone – It is the process of generating the network topology by taking the input file in a text format.
2. doCasetwo – It uses Dijkstra’s algorithm to calculate the interface, optimal path and cost to all the routers in the topology from a source router.
3. doCaseThree – It calculates the optimal path and cost to the destination from Source router.
4. doCaseFour – It modifies the router by deleting a router from the topology.
5. doCaseFive – It finds the best broadcast router.
6. doCaseSix – It exits the program.

Introduction

Link state protocols are based upon the principle of a "distributed map". They use the Dijkstra's algorithm to find the shortest path between the routers. All the routers keep the information of its adjacent router. Each router calculates its own best paths to send the packets to reach destination. Link-state routing protocols are one of the two main classes of routing protocols. It is used for communicating the packets in the packet switching networks. Some of the examples of link-state routing protocols are Open Shortest Path First which is also known as OSPF and Intermediate System to Intermediate System which is also known as IS-IS.
The basic concept of link-state routing is that every node constructs a map of the connectivity to the network, in the form of a graph, showing which nodes are connected to which other nodes. Each node then independently calculates the next best logical path from it to every possible destination in the network. The collection of best paths will then form the node's routing table.

Dijkistras Algorithm
This algorithm is used to find the shortest path. Here in the initial step all the nodes value are set to be 999(infinite). The connection between the router to itself is set to be zero and if a connection does not exist with the other router then -1 is assigned.

Requirement

Hardware Requirements

- Memory: 4GB 1600 Mhz DDR3
- Processor: Intel Core i5 Processor 1.6 Ghz

Software Requirements

Operating System: macOS

Programing Language: Python

Runnable environment: Terminal

Implementation

Algorithm

The algorithm is used to find out the shortest paths from source to router in the given graph. Dijkstra's algorithm is used to find the shortest path. The steps of the algorithm are as follow.

- This algorithm maintains the list of unvisited nodes. There are two sets **key** and **value.** Key maintains the cost value and value maintains the nodes which are visited or unvisited.
- It creates a set **parent** which keeps the track of the immediate parent of all the vertices while visiting them.
- It selects a vertex as source and assigns a maximum possible cost (i.e. infinity) to every other vertex which can be used to calculate the best path.
- The cost to the source will always be 0
- It tries to minimize the cost for every vertex. The cost here is the time taken or distance covered to reach from one node to another. The minimization of cost takes several steps.
- For each unvisited neighbor of current vertex, calculate the new cost. New cost is calculated as sum of all distances from the source to that vertex.
- When all adjacent nodes of the current nodes are considered, marks the current node as visited node and update cost in key set.
- End the algorithm when all the nodes are visited.

Pseudo Code

```
Function Dijkstra(source)
Create vertex set Q #initialization
For each vertex v in Graph:
        Dist[v] ← INFINITY #Unknown distance from source to destiny
        Prev[v] ← Undefined #Previous is optimal; from source
        Add v to Q # All the nodes in Q visited
        Dist[source] ← 0
        While q is not None:
                U ← vertex in q with min dist[u] #Node with the least distance
                For each neighbor v of u: #where v is still in q
                        Alt ← dist[u] + length (u,v)
```

```
        If alt < dist[v]:  #shorter path found
            Dist[v] ← alt
            Prev[v] ←
Return dist[], prev[]
```

Code and Results

```
#CS542-04 - Project

#Link State Routing Algorithm


#Importing the libraries
import os
import sys
import os
import os.path
import heapq


#Printing the choices available

print("Select one option from below: \n")
print("1. Press 1 for creating the network topology")
print("2. Press 2 for building the connection table")
print("3. Press 3 for finding the shortest path to destination")
print("4. Press 4 to modify the topology")
print("5. Press 5 for finding the best router for broadcast ")
print("6. Press 6 to exit the program")


node_edge = list()
dictionary_values= dict()
graphical_representaion = list()
nodestot = 0
a = 0
nodes_dict = dict()
notVisited = dict()

def __init__(self):
   self.grah = list()
```

```
    self.cst_dict = {}
    self.vetex = []
    self.lininterface = []
    self.pth = list()
    self.unSen = {} #doubtful
    self.prvNode = {}
    self.senNode = {}
    self.qwdqw = {}


def errhandler():

    print("The input is not a valid input, Enter an integer value between 1-6.\n")
    command = input("Please Enter the command: \n")
    takeaction.get(command, errhandler)()

def err1handler():
    takeaction = {"1" : doCaseOne, "6" : doCaseSix}
    print("The input is not a valid input, Enter 1 for taking the input file, or enter 6 to exit\n")
    command = input("Please Enter the command: \n")
    takeaction.get(command, err1handler)()

def erronehandler():
    takeaction = {"2" : doCasetwo, "5" : doCaseFivepointone,"6" : doCaseSix}
    print("The input is not a valid input, Enter 2 for the source router, or enter 5 to see the best router or enter 6 to exit\n")
    command = input("Please Enter the command: \n")
    takeaction.get(command, erronehandler)()

def errtwohandler():
    takeaction = {"3" : doCaseThree, "5" : doCaseFivepointone, "6" : doCaseSix}
    command = input("Please Enter the command, Enter 3, to find shortest path to destination or 5, to see the best router or 6 to exit the program: \n")
    takeaction.get(command, errtwohandler)()

def errS3handler():
```

```
    takeaction = {"4" : doCaseFour, "5" : doCaseFivepointone, "6" : doCaseSix}
    print("The input is not a valid input, Enter an integer value between 4-6.")
    command = input("Please Enter the command: \n")
    takeaction.get(command, errS3handler)()

def err5handler():
    takeaction = {"5" : doCaseFivepointtwo, "6" : doCaseSix}
    print("Press 5 to see the best router, with modifying the updates on router or
press 6 to exit")
    command = input("Please Enter the command: \n")
    takeaction.get(command, err5handler)()

def errexithandler():
    takeaction = {"5": doCaseFivepointone, "6" : doCaseSix}
    command = input("Not many choices, press 6 and exit: \n")
    takeaction.get(command,errexithandler())

def errexhandler():
    takeaction = {"6" : doCaseSix, "7" : "Please press 6 and exit"}
    command = input("Not n*n matrix, press 6 and exit and import a correct n*n file:
\n")
    takeaction.get(command,errexhandler())

algorithm to find the shortest path ~~~~~~~~~~~~~~~#



def dijkstra(start):
    global dictionary_values
    global notVisited
    global nodeBefore
    global visitedNode
    global nodes_dict
    #creates a new dictionary with key as the routers and values as infinite.
    notVisited = {node: float('inf') for node in node_edge}
    visitedNode = dict()
    nodeBefore = dict()
```

```
    newnod = list()
    startingNode = start
    initialDist = 0
    #creates a new dict with all the routers as key and an empty list as the value
    nodes_dict = {node: [] for node in node_edge}
    notVisited[startingNode] = initialDist
    while len(notVisited) > 0:
        for key, value in dictionary_values[startingNode].items():
            if key not in notVisited:
                continue
            newDistance = initialDist + value
            if newDistance < notVisited.get(key, float('inf')):
                notVisited[key] = newDistance
                nodeBefore[key] = startingNode
                if not nodes_dict[startingNode]:
                    nodes_dict[key] = [key]
                else:
                    nodes_dict[key] = list(nodes_dict[startingNode])
        visitedNode[startingNode] = initialDist
        del notVisited[startingNode]
        if not notVisited:
            break
        currentStatus = [node for node in notVisited.items() if node[1]]
        #returns the distance from the current node or the starting node to all other
nodes in the topology
        startingNode, initialDist = sorted(currentStatus, key=lambda x: x[1])[0]


def doCaseOne():
    global q
    global graphical_representaion
    #takes the input file in .txt format only.
    print("Enter the input file in a text format - .txt \n")
    file = input()
    #file should not be empty and it should exist in the folder.
    if file.endswith('.txt') and os.path.isfile(file) and os.stat(file).st_size != 0:
        print("The Network topology is : \n")
```

```
fo = open(file)
#extracting the data from the input file and creating a list of list.
graphical_representaion = [list(map(int, x.split())) for x in fo]
#prints the data in the form of a matrix
for line in graphical_representaion:
   for item in line:
      print(item, end='    ')
   print()
#checks the length of the matrix or the rows
rows = len(graphical_representaion)
count = 0
#checks the columns of the matrix
for row in graphical_representaion:
   column_len = len(row)
#The matrix should be of the form n*n matrix.
if column_len == rows:
   print("The input file is a n*n matrix\n ")
   nodestot=len(graphical_representaion)
   #The number of rows or the number of routers in the network.
   print("\nTotal number of nodes present in the given topology: ",
len(graphical_representaion))
   #It creats a dictionary of the graph, the dictionary is of the form {1,{2,3}}.
   #this will be later used in calculating the path using the path finding function.
   for i in range(len(graphical_representaion)):
      mat = dict()
      raj = list()
      for j in range(len(graphical_representaion)):
         if i != j and graphical_representaion[i][j] != -1:
            mat[j + 1] = graphical_representaion[i][j]
      dictionary_values[i + 1] = mat
      node_edge.append(i + 1)
      q = len(dictionary_values)
#The input text file is not of the form n*m.
#It will Exit the sytem by displaying the message.
else:
   print("It is not a n*n matrix\n")
   print("Import a n*n matrix file agin by running the program again\n")
```

```
        while True:
          raise SystemExit



    #input file is not a text file or an empty file or dosenot exist.
    else:
      print("Enter the input file again, as the file entered is not a valid one.")
      takeaction = {"1" : doCaseOne, "6" : doCaseSix}
      command = input("Please, Enter the command, press 1 or 6 \n")
      takeaction.get(command, err1handler)()
    #Once the caseOne is done, the choice available will be 2,5,6
    takeaction = {"2" : doCasetwo,  "5" : doCaseFivepointone, "6" : doCaseSix}
    command = input("Please, Enter the command, options are 2 or 5 or 6: \n")
    takeaction.get(command,erronehandler)()


def doCasetwo():
    global a
    global q
    #Just a print statement
    print("\nPlease, select a source router. It should be an integer value")
    #Checks if the input is valid or not.
    #Input should always be an integer value
    while True:
      try:
        start = int(input("Please Enter an integer value, Do not enter any negative
value or an invalid input"))
      except ValueError:
        print("Not an integer! Please enter a valid router ID")
        continue
      else:
        break
    q = len(dictionary_values)
    #The input for source router should be a valid router.
    #By valid, it means the router should be present in the network topology.
    #no negative integer accepted.
```

```
  if start > q or start < 1:
    print("Not a valid router!")
    takeaction = {"2" : doCasetwo,  "5" : doCaseFivepointone, "6" : doCaseSix}
    command = input("Please Enter the command 2 or 5 or 6: \n")
    takeaction.get(command,erronehandler)()
  a = start
  #calling path finding function with the parameter as the input taken.
  dijkstra(start)
  print("\nThe connection table for router is")
  print("\n\tDestination \tInterface")
  print("=================================")
  for key in nodes_dict:
    print("\t",key, "\t\t", nodes_dict[key])

  #once casetwo is finished, the available options are 3, 5, 6
  takeaction = {"3" : doCaseThree, "5" : doCaseFivepointone, "6" : doCaseSix}
  command = input("Please Enter the command, options are 3,5,6: \n")
  takeaction.get(command, errtwohandler)()

def doCaseThree():
  global a
  global q
  global destination

  print("\nSelect the destination router to find the shortes path from Source to
Destnation:\n")
  #it checks the input, it should be an integer
  while True:
      try:
        destination = int(input("Please enter a valid router and an integer value
\n"))
      except ValueError:
        print("Not an integer! Please enter a valid router ID")
        continue
      else:
        break
  #the input should be an existing router.
```

```
    q = len(dictionary_values)
    if destination < 1 or destination > q:
        print("Enter a valid router again by pressing the command 3")
        takeaction = {"3" : doCaseThree,  "5" : doCaseFivepointone, "6" : doCaseSix}
        command = input("Please Enter the command, choices -> 3 or 5 or 6: \n")
        takeaction.get(command,errtwohandler)()
    #Finding the minimum cost to the destination.
    destination1 = destination
    b = destination
    c = visitedNode[destination]
    print("\nThe minimum cost from the source %s to the detination %s is equal to
%s" % (a,b,c))
    #Finding the shortest path to the destination.
    path = list()
    while 1:
        path.append(destination)
        if destination == a:
            break
        destination = nodeBefore[destination]
    path.reverse()

    destination = destination1
    print("\nThe shortest Path from the source %s to the detination %s is equal to
%s"%(a,b,path))

    #Once case3 is done, the available options are 4,5,6
    takeaction = {"4" : doCaseFour, "5" : doCaseFivepointone, "6" : doCaseSix}
    command = input("Please Enter the command: \n")
    takeaction.get(command, errS3handler)()

def doCaseFour():
    global notVisited
    global a
    global destination
    global q
    global newlis
```

```
print("\nPlease, select a Router to be Removed.")
#the input should be an integer
while True:
      try:
         down_router = int(input("Please enter a valid router - an integer
value\n"))
      except ValueError:
         print("Not an integer! Please enter a valid router ID")
         continue
      else:
         break
#The router should be a valid and existing router.
q = len(dictionary_values)
if down_router < 1 or down_router > q:
   print("Enter valid Router")
   takeaction = {"4" : doCaseFour,  "5" : doCaseFivepointone, "6" : doCaseSix}
   command = input("Please Enter the command, choice -> 4 or 5 or 6: \n")
   takeaction.get(command,errS3handler)()

#downs the router
downminusone = down_router - 1
for i in range(len(graphical_representaion)):
   matrix = {}
   for j in range(len(graphical_representaion)):
      if i != j != downminusone and i != j and graphical_representaion[i][j] != -1:
         matrix[j + 1] = graphical_representaion[i][j]

   dictionary_values[i + 1] = matrix
del dictionary_values[down_router]
del node_edge[downminusone]
#a case if deleting router was the start node. We will perform doSteptwo again.
if down_router == a:
   while True:
      try:
         a = int(input("Enter new start node again as removed node was the start
node \n"))
      except ValueError:
```

```
            print("Not an integer! Enter a valid router ID")

          else:
            break

    dijkstra(a)

    print("\nRouter %s Connection Table:" % a)
    print("\n\tDestination \tInterface")
    print("==================================")
    for key in nodes_dict:
      print(key, "\t\t", nodes_dict[key])
    #if the deleted node is the destination node.
    path = []
    destination2 = destination
    if down_router == destination:


      print("\nSelect the destination router:")

      while True:
          try:
            destination = int(input("Enter a new destination node again as removed 
node was the destination node \n"))
            destination2 = destination
          except ValueError:
            print("Not an integer! Enter a valid router ID")
          else:
            break
    while 1:
      path.append(destination)
      if destination == a:
        break
      destination = nodeBefore[destination]
    path.reverse()
    destination = destination2
```

```
    print("The new shortest distance to destination is ",visitedNode[destination])
    print("The new shortest path is ",path)
    newlis = list()
    #available options
    for key,value in dictionary_values.items():
        newlis.append(key)
    takeaction = {"5" : doCaseFivepointtwo, "6" : doCaseSix}
    command = input("Please Enter the command: \n")
    takeaction.get(command, err5handler)()

def doCaseFivepointone():
    global graphical_representaion
    global newlis

    lishtwa = list()
    for a in range(len(graphical_representaion)):
        dijkstra(a + 1)
        count = 0
        for key, val in visitedNode.items():
            count = count + val
        print("\t",a + 1,"\t\t", count)

        lishtwa.append(count)

    u = min(lishtwa)
    v = lishtwa.index(u)
    vplusone = v + 1
    r = u
    print("\nBest Router for broadcsting before modifying the topology is %s with
lowest cost of %s" % (vplusone, min(lishtwa)))
     takeaction = { "2" : doCasetwo, "5" : doCaseFivepointone, "6" : doCaseSix}
    command = input("Please Enter the command: \n")
    takeaction.get(command, erronehandler)()
```

```
def doCaseFivepointtwo():
    global graphical_representaion
    global newlis

    lisht = list()
    lest = list()
    for a in newlis:
        dijkstra(a)
        count = 0
        for key,val in visitedNode.items():
            count = count + val
        print("\t",a,"\t\t",count)
        lisht.append(count)
        lest.append(a)


    u = min(lisht)
    v = lisht.index(u)
    t = lest[v]

    print("\nBest Router for broadcasting after modifying the topology is %s with lowest cost of %s"%(t,u))

    takeaction = {"5": doCaseFivepointtwo, "6" : doCaseSix}
    command = input("Please Enter the command: \n")
    takeaction.get(command, errexithandler)()

def doCaseSix():
    print("\nExiting...")
    exit()

def main():
    takeaction = {"1" : doCaseOne, "6" : doCaseSix}
    command = input("Please Enter the command: \n")
    takeaction.get(command, err1handler)()

if __name__ == '__main__':
```

main()

Results

The input file is of 5*5 matrix

First master command 1 is given.

```
Select one option from below:

1. Press 1 for creating the network topology
2. Press 2 for building the connection table
3. Press 3 for finding the shortest path to destination
4. Press 4 to modify the topology
5. Press 5 for finding the best router for broadcast
6. Press 6 to exit the program
Please Enter the command:
1
Enter the input file in a text format - .txt

in.txt
The Network topology is :

0       2       5       1       -1
2       0       8       7       9
5       8       0       -1       4
1       7       -1       0       2
-1       9       4       2       0
The input file is a n*n matrix


Total number of nodes present in the given topology:   5

Please, Enter the command, options are 2 or 5 or 6:
|
```

After this master command 2 is given and source is selected as 1.

```
Please, Enter the command, options are 2 or 5 or 6:
2

Please, select a source router. It should be an integer value
Please Enter an integer value, Do not enter any negative value or an invalid input1

The connection table for router is

        Destination     Interface
===================================
          1               []
          2               [2]
          3               [3]
          4               [4]
          5               [4]

Please Enter the command, options are 3,5,6:
```

After this master command 3 is given, and destination router is asked.

```
Please Enter the command, options are 3,5,6:
to scroll output; double click to hide

Select the destination router to find the shortes path from Source to Destnation:

Please enter a valid router and an integer value
5

The minimum cost from the source 1 to the detination 5 is equal to 3

The shortest Path from the source 1 to the detination 5 is equal to [1, 4, 5]

Please Enter the command:
```

Master command 4, and router selected is 2

```
The shortest Path from the source 1 to the detination 5 is equal to [1, 4, 5]
Please Enter the command:
4

Please, select a Router to be Removed.
Please enter a valid router - an integer value
2

Router 1 Connection Table:

        Destination     Interface
==================================
1                []
3                [3]
4                [4]
5                [4]
The new shortest distance to destination is  3
The new shortest path is  [1, 4, 5]

Please Enter the command:
```

In the last, program is given command 5 and 6.

```
Please Enter the command:
5
        1           9
        3           15
        4           9
        5           9

Best Router for broadcasting after modifying the topology is 1 with lowest cost of 9
Please Enter the command:
6

Exiting...
```

Test Cases

1 If the input is not a n*n matrix, it will ask for a n*n matrix file and it will exit the program.
2. If the input text file is not a text file or an empty file.
3. If the input is not an integer, it will raise an error and ask again.
4. If wrong master command is selected, it will ask again for the command.
5. If deleted router is a source router.
6. If the removed router is destination router.
7. New broadcasting table after the router is deleted.
8. If the text file does not exist it will raise an error.

Conclusion

Implemented the Link State routing protocol using Dijkstra's algorithm in an effective and efficient way to find out the optimal path to route the packet from one node to another node with the lower cost. The matrix represents the capacity of links on those routes. Each router is synchronized in such a way that it uses the latest information and produces best routing decisions. Therefore, it is very rare to occur routing loops. The careful network design can be implemented to reduce the link state database sizes. This leads to smaller Dijkstra calculations and faster convergence.

References

https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

https://brilliant.org/wiki/dijkstras-short-path-finder/