

## روند اصلی الگوریتم از ابتدا تا پایان:

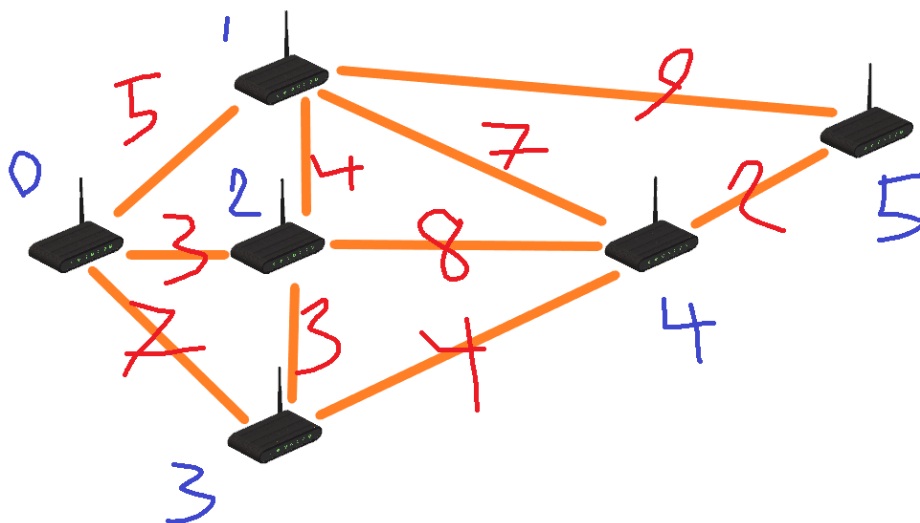
در ابتدا safe چاپ می شود سپس acknowledgement ها با همسایه ها برقرار شده و یک timeout کوچک برای سینک شدن در نظر گرفته شده است. سپس manager پیام ok را به همه ی روتر های موجود در شبکه ارسال می کند.

سپس packet ها از همسایه ها گرفته می شود و طبق الگوریتم مسیریابی دایجسترا کوتاه ترین مسیر ها بین روتر ها شناسایی شده و سپس forwarding table بدست می آید. در مرحله ی بعد روتر ها آماده ی ارسال بسته به یک دیگر می شوند و پس از ارسال هر data packet با توجه به forwarding table بدست آمده در مراحل قبل (مرحله ارتباط manager با روتر ها) به روتر مقصد رسیده و marked می شود.

در قسمت shell نیز مسیر هایی که بسته باید طی کند در داخل براکت ها مانند شکل زیر مشخص شده است:

```
C:\Users\arman\AppData\Local\Programs\Python\Python37\python.exe
FIN 127.0.0.7
FIN 5000
[['5', '1', '9', '5006', '127.0.0.5'], ['5', '4', '2', '5009', '127.0.0.6']]
safe
THIS 127.0.0.7 5010
Message from Server b'ACK FROM127.0.0.5:5006'
Message from Server b'ACK FROM127.0.0.6:5009'
HELL
HEEY
OK
[['127.0.0.2', '5005'], ['127.0.0.3', '5007'], ['127.0.0.4', '5008'], ['127.0.0.5', '5006'], ['127.0.0.6', '5009'],
['127.0.0.7', '5010']]
THIS 127.0.0.7 5010
LIST ['1', ['0', '5'], ['2', '4'], ['4', '7'], ['5', '9']]
THIS IS MESSAGE b"[['1', '0', '5', '5005', '127.0.0.2'], ['1', '2', '4', '5007', '127.0.0.3'], ['1', '4', '7', '5009',
'127.0.0.6'], ['1', '5', '9', '5010', '127.0.0.7']]"
LIST ['4', ['1', '7'], ['2', '8'], ['3', '4'], ['5', '2']]
THIS IS MESSAGE b"[['4', '1', '7', '5006', '127.0.0.5'], ['4', '2', '8', '5007', '127.0.0.3'], ['4', '3', '4', '5008',
'127.0.0.4'], ['4', '5', '2', '5010', '127.0.0.7']]"
Message from Server b'ACK FROM127.0.0.5:5006'
Message from Server b'ACK FROM127.0.0.6:5009'
HELL
HEEY
[['1', ['0', '5'], ['2', '4'], ['4', '7'], ['5', '9']], ['4', ['1', '7'], ['2', '8'], ['3', '4'], ['5', '2']], ['5',
['1', '9'], ['4', '2']]]
THIS 127.0.0.7 5010
```

شکل زیر ۷ روتر موجود در کد و هزینه های در نظر گرفته شده را نمایش می دهد:



الگوریتم دایجسترا:

```

18 def minDistance(self, dist, queue):
19     # Initialize min value and min_index as -1
20     minimum = float("Inf")
21     min_index = -1
22
23     # from the dist array, pick one which
24     # has min value and is till in queue
25     for i in range(len(dist)):
26         if dist[i] < minimum and i in queue:
27             minimum = dist[i]
28             min_index = i
29     return min_index
30
31     # Function to print shortest path
32     # from source to j
33     # using parent array
34     def printPath(self, parent, j):
35         global res
36
37         # print("DEBUG", parent[j])
38
39         # Base Case : If j is source
40         if parent[j] == -1:
41             print(j)

```

کد های موجود در قسمت dijkstra\_algorithm.py مطابق با الگوریتم های موجود در اینترنت فقط با این تفاوت که در کد ما یک تابع به نام printPath در نظر گرفته شده که به صورت بازگشتی مسیر را چاپ می کند.

## بخش manager:

```
16
17 # FILE_NAME = sys.argv[1]
18 class Manager:
19     socket_tcp = None
20     TCP_PORT = 5000
21     TCP_IP = "127.0.0.1"
22     TCP_IP_INIT = 2
23
24     def read_init_file(self):
25         f = open(FILE_NAME, "r")
26         f.seek(0)
27         return f.read()
28
29     def make_table(self):
30         inner_lines = [{"From", "To", "Cost"}]
31         with open(FILE_NAME, "r") as f:
32             lines = [line.rstrip() for line in f]
33             for num in lines:
34                 org, dest, cost = num.split()
35                 if org != dest:
36                     line = [org, dest, cost]
37                     inner_lines.append(line)
38                     # print(line)
39                     line = [dest, org, cost]
```

ارتباط manager با هر روتر TCP در نظر گرفته شده است. آیی و پورت ثابتی برای manager در نظر گرفته شده و در ابتدا از فایل config توپولوژی شبکه و مشخصات روتر های موجود (۷ روتر در نظر گرفته شده در شکل صفحه قبل) را دریافت می کند. سپس برای هر روتر یک process در نظر گرفته می شود و پورت و آیی منحصر به فردی به روتر اختصاص داده می شود.

```
58     with open(FILE_NAME, "r") as f:
59         lines = [line.rstrip() for line in f]
60         for num in lines:
61             org, dest, _ = num.split()
62             if org not in router_names:
63                 router_names.append(org)
64                 router_udps[org] = None
65                 router_tcp_addresses[org] = "127.0.0." + str(self.TCP_IP_INIT)
66                 self.TCP_IP_INIT += 1
67             elif dest not in router_names:
68                 router_names.append(dest)
69                 router_udps[dest] = None
70                 router_tcp_addresses[dest] = "127.0.0." + str(self.TCP_IP_INIT)
71                 self.TCP_IP_INIT += 1
```

## بخش routing: (نمایش distance و مسیر به صورت مرحله به مرحله)

```
HEEY3
[['1', ['0', '5'], ['2', '4'], ['4', '7'], ['5', '9']]
[['2', ['0', '3'], ['1', '4'], ['3', '3'], ['4', '8']]
[['3', ['0', '7'], ['2', '3'], ['4', '4']]
[['4', ['1', '7'], ['2', '8'], ['3', '4'], ['5', '2']]
[['5', ['1', '9'], ['4', '2']]
[['0', ['1', '5'], ['2', '3'], ['3', '7']]
OP....
Vertex          Distance from Source  Path
4 --> 0          10
4
3
2
0
Next Hop 3
4 --> 1           7
4
1
Next Hop 1
4 --> 2           7
4
3
2
Next Hop 3
4 --> 3           4
4
3
Next Hop 3
4 --> 5           2
4
5
Next Hop 5
[['4', '0', '3'], ['4', '1', '1'], ['4', '2', '3'], ['4', '3', '3'], ['4', '5', '5']]
#####
THIS 127.0.0.6 5009
```

## بخش router:

```
socket_tcp = None

UDP_PORT = 5005
UDP_IP = "127.0.0.1"

TCP_PORT = 5000
TCP_IP = "127.0.0.1"

adjacent_info = list()
lsp_info = list()

all_routers = list()
lsp_full_info = list()

def __init__(self, name):
    super().__init__()
    self.name = name
    self.socket_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.UDP_PORT += name

    self.socket_tcp = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket_tcp.connect((self.TCP_IP, self.TCP_PORT))
    self.socket_tcp.send(bytes(str(self.UDP_PORT), 'utf-8'))
```

برای اجرای موازی روتر ها ، از thread استفاده شده است. در ابتدا به هر روتر یک پورت UDP اختصاص داده می شود سپس به manager متصل شده و پورت UDP خود را اطلاع می دهد. سپس به کمک تابع adjacent\_info لیستی از همسایگان خود را دریافت کند. سپس لیست همه ی روتر ها و LSP ها را نیز دریافت می کند.

### بخش ارسال data packet:

Manager از طریق کانکشن TCP بسته ی شروع کننده را به روتر ها می فرستد. روتر ها نیز زمانی که بسته شروع کننده خود را گرفت ، می تواند بسته ای را به روتر دیگر با UDP بفرستد. در ابتدا باید در forwarding table مسیر های موجود و روتر های همسایه را شناسایی کند.

در شکل زیر ادامه ی توابع مورد نیاز برای روتر ها برای برقراری ارتباط با manager و سایر روتر ها موجود است که به جز تابع receive\_from\_routers توضیحات آنها مطابق با موارد ذکر شده است:

```
algorithm.py x router_new.py x manager.py x router.py x manager_resultLog x 0.log x 1.log x 2.log x 3.log
self.socket_tcp.close()

def UDP_send(self, message):
    self.socket_udp.sendto(message, (self.UDP_IP, 5006))

def UDP_recieve(self, UDP_PORT):
    self.socket_udp.bind((self.UDP_IP, UDP_PORT))

    while True:
        data, addr = self.socket_udp.recvfrom(1024)
        print("received message: %s" % data)

def print_name(self):
    print(self.name, self.UDP_IP, self.UDP_PORT)

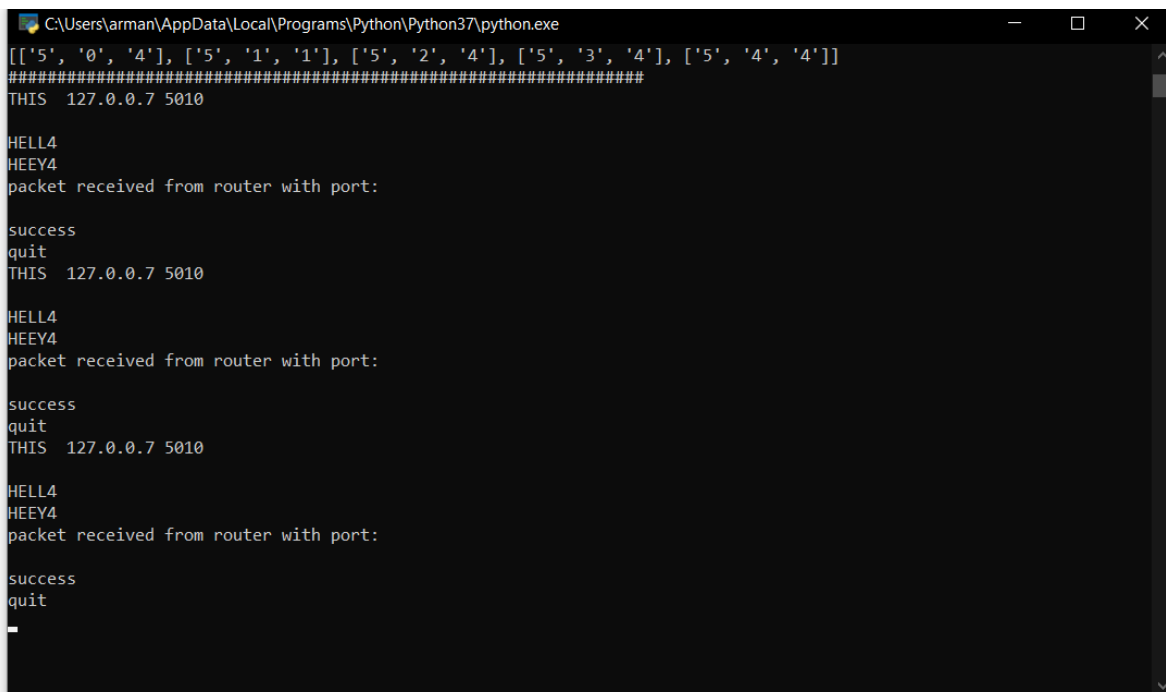
def receive_from_routers(self):
    lock = allocate_lock()
    global neig
    global bcList
    global graph
    global nodePort
    global dieList
    receiver = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
```

## تابع receive\_from\_routers:

در صورت فراخوانی این تابع ، thread روتر مورد نظر lock شده و بسته های broadcast شده را بررسی می کند. در صورتی که مقصد بسته ی ارسالی همان روتر باشد (از طریق پورت udp تشخیص می دهد) بسته را دریافت کرده و از شبکه خارج می کند. در غیر این صورت Lock روتر را آزاد کرده و دوباره برای یک روتر دیگر این جست و جو را انجام می دهد.

## بخش پایانی یا quit:

در پایان ، زمانی که manager دستور quit را به یک روتر ارسال می کند ، آن روتر متوقف شده و از لیست روتر های موجود در شبکه خارج می شود. البته در کد برای این بخش یک sleep طولانی مدت در نظر گرفته شده که shell بسته نشود و در صورتی که sleep با کد exit زبان پایتون جایگزین شود ، روتر واقعا متوقف شده و shell آن نیز بسته می شود.



```
C:\Users\arman\AppData\Local\Programs\Python\Python37\python.exe
[[['5', '0', '4'], ['5', '1', '1'], ['5', '2', '4'], ['5', '3', '4'], ['5', '4', '4']]]
#####
THIS 127.0.0.7 5010

HELL4
HEEY4
packet received from router with port:

success
quit
THIS 127.0.0.7 5010

HELL4
HEEY4
packet received from router with port:

success
quit
THIS 127.0.0.7 5010

HELL4
HEEY4
packet received from router with port:

success
quit
-
```