Introduction to Computer Networks

HW2 B07303024 李品樺

Problem1

After reading the input file, I save the 2-dim array of graph representation and number of routers in a Graph class. Then, call the method dijkstra(src) for n times, where n is the number of routers, and src is the starting router. Dijkstra method is shown in the following image.

- dist[i] is the shortest path from src to i, initialize to a large value except src
- unvisited[i] is initialize to True for all the routers except src
- hop[i] stores the next hop from the src to i of the shortest paths, initialize to -1 except src

Then, we find the node (min_node) that has not been visited with the minimum distance (min_dist). Set unvisited to False for min_node. Finally, using min_node as the key, relax the path and store the dist and hop if the path is shorter for a node v to go from min_node to src. We repeat this process for n times, and return dist and hop for final output.

```
def dijkstra(self, src):
    # initialize
    dist = [MAX_DISTANCE] * self.v
    unvisited = [True] * self.v
    hop = [-1] * self.v

    unvisited[src] = False
    dist[src] = 0
    hop[src] = src
    min_node = src

    for i in range(self.v):
        min_dist = MAX_DISTANCE
        # find the min_dist and min_node
        for v in range(self.v):
            if dist[v] < min_dist and dist[v] != -1 and unvisited[v] == True:
                min_dist = dist[v]
                min_node = v
        unvisited[min_node] = False
        # print("min_node", min_node)
        # relax vertex v using min_node
        for v in range(self.v):
            if( self.graph[min_node][v] > 0 and unvisited[v] == True and
               dist[v] > dist[min_node] + self.graph[min_node][v]):
                dist[v] = dist[min_node] + self.graph[min_node][v]
                if min_node == src:
                    hop[v] = v
                else:
                    hop[v] = hop[min_node]
        # print("dist", dist)
        # print("hop", hop)
    return dist, hop
```

Problem2

After reading the file, first store every path to the to_remove to -1, then call the dijkstra method described in Problem1 for n times. When writing output, we use the function writeFile() as shown below. If the result of hop[i][j] is -1, which means that this is the router to remove and there are no paths to this router, so we write the result as -1, -1

```
elif len(sys.argv) == 3: # problem 2
    filename = sys.argv[1]
    to_remove = int(sys.argv[2])
    output_filename = os.path.join('./testcase', filename[:-4]+'_RmRouter'+str(to_remove)+ '.txt')
    num, graph = readFile(filename)
    for i in range(num):
        graph[i][to_remove-1] = -1
# print(graph)
g  = Graph(vertices_num=num, graph=graph)
```

```
def writeFile(filename, dists, hops, to_remove):
    cnt = 1
    with open(filename, 'w') as f:
        for i in range(len(dists)):
            if cnt == to_remove:
                cnt += 1
                continue
            f.write('Routing table of router {}:\n'.format(cnt))
            for j in range(len(hops)):
                if(hops[i][j] == -1):
                    f.write('-1 -1\n')
                else:
                    f.write(str(dists[i][j]) + ' ' + str(hops[i][j]+1) + '\n')
            cnt += 1
```