交通地理信息系统

Geographic information system for Transportation (GIS-T)

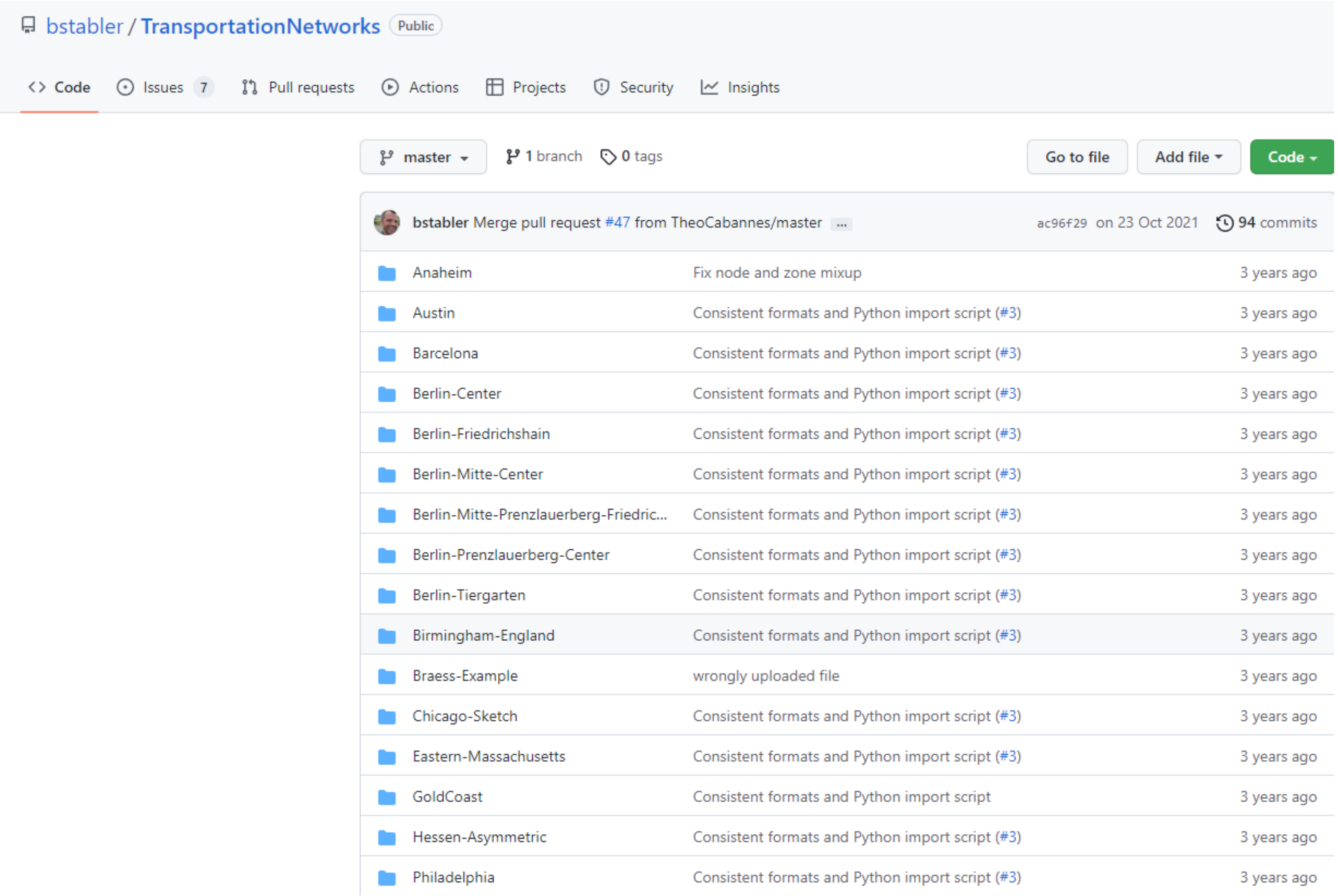
主讲：徐占东 讲师

zhandong.xu@swjtu.edu.cn

最短路：编程实现

程序部分：数据结构-Link

交通网络文件 https://github.com/bstabler/TransportationNetworks



bstabler Merge pull request #47 from TheoCabannes/master … ac96f29 on 23 Oct 2021 94 commits

Anaheim	Fix node and zone mixup	3 years ago
Austin	Consistent formats and Python import script (#3)	3 years ago
Barcelona	Consistent formats and Python import script (#3)	3 years ago
Berlin-Center	Consistent formats and Python import script (#3)	3 years ago
Berlin-Friedrichshain	Consistent formats and Python import script (#3)	3 years ago
Berlin-Mitte-Center	Consistent formats and Python import script (#3)	3 years ago
Berlin-Mitte-Prenzlauerberg-Friedric…	Consistent formats and Python import script (#3)	3 years ago
Berlin-Prenzlauerberg-Center	Consistent formats and Python import script (#3)	3 years ago
Berlin-Tiergarten	Consistent formats and Python import script (#3)	3 years ago
Birmingham-England	Consistent formats and Python import script (#3)	3 years ago
Braess-Example	wrongly uploaded file	3 years ago
Chicago-Sketch	Consistent formats and Python import script (#3)	3 years ago
Eastern-Massachusetts	Consistent formats and Python import script (#3)	3 years ago
GoldCoast	Consistent formats and Python import script	3 years ago
Hessen-Asymmetric	Consistent formats and Python import script (#3)	3 years ago
Philadelphia	Consistent formats and Python import script (#3)	3 years ago

程序部分：Sioux Falls

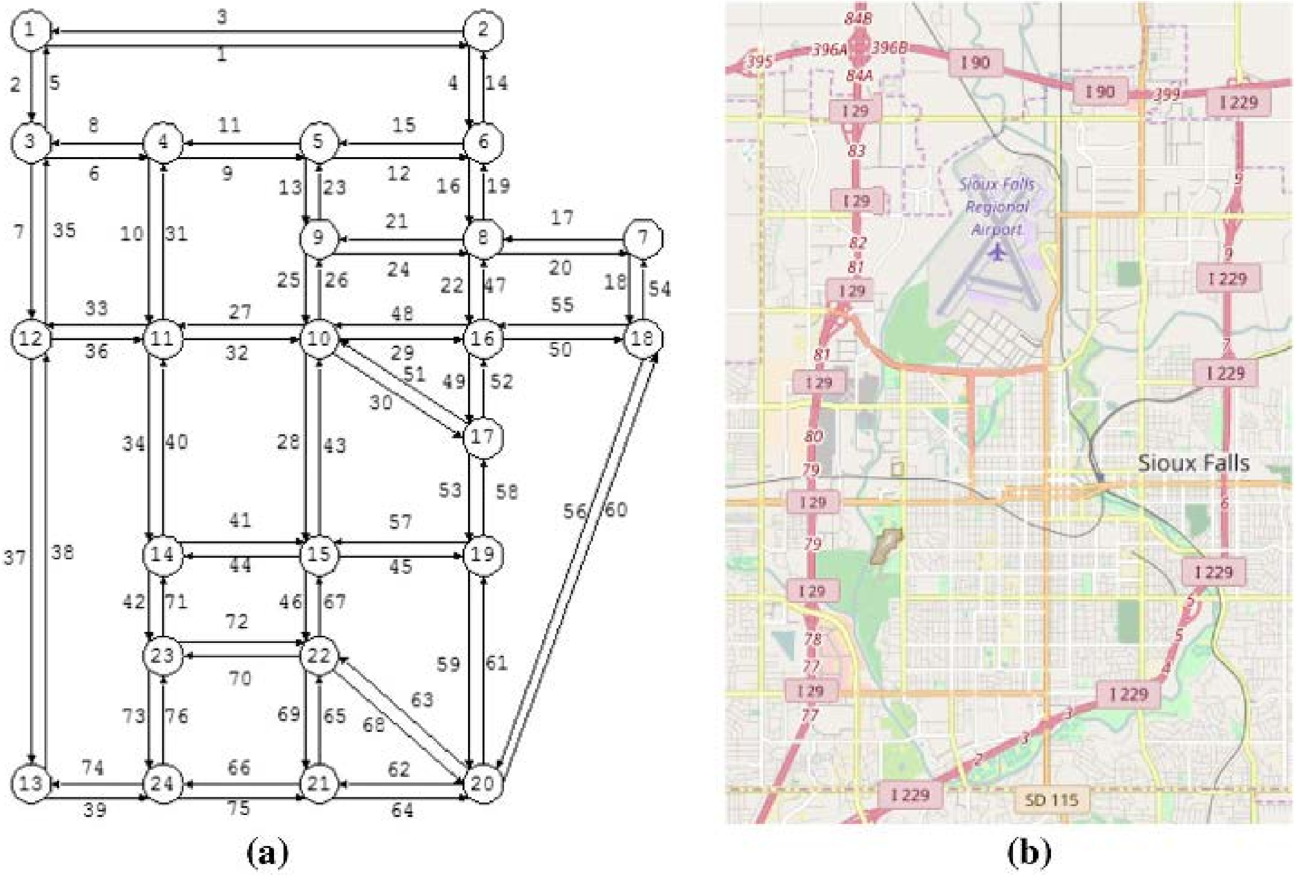
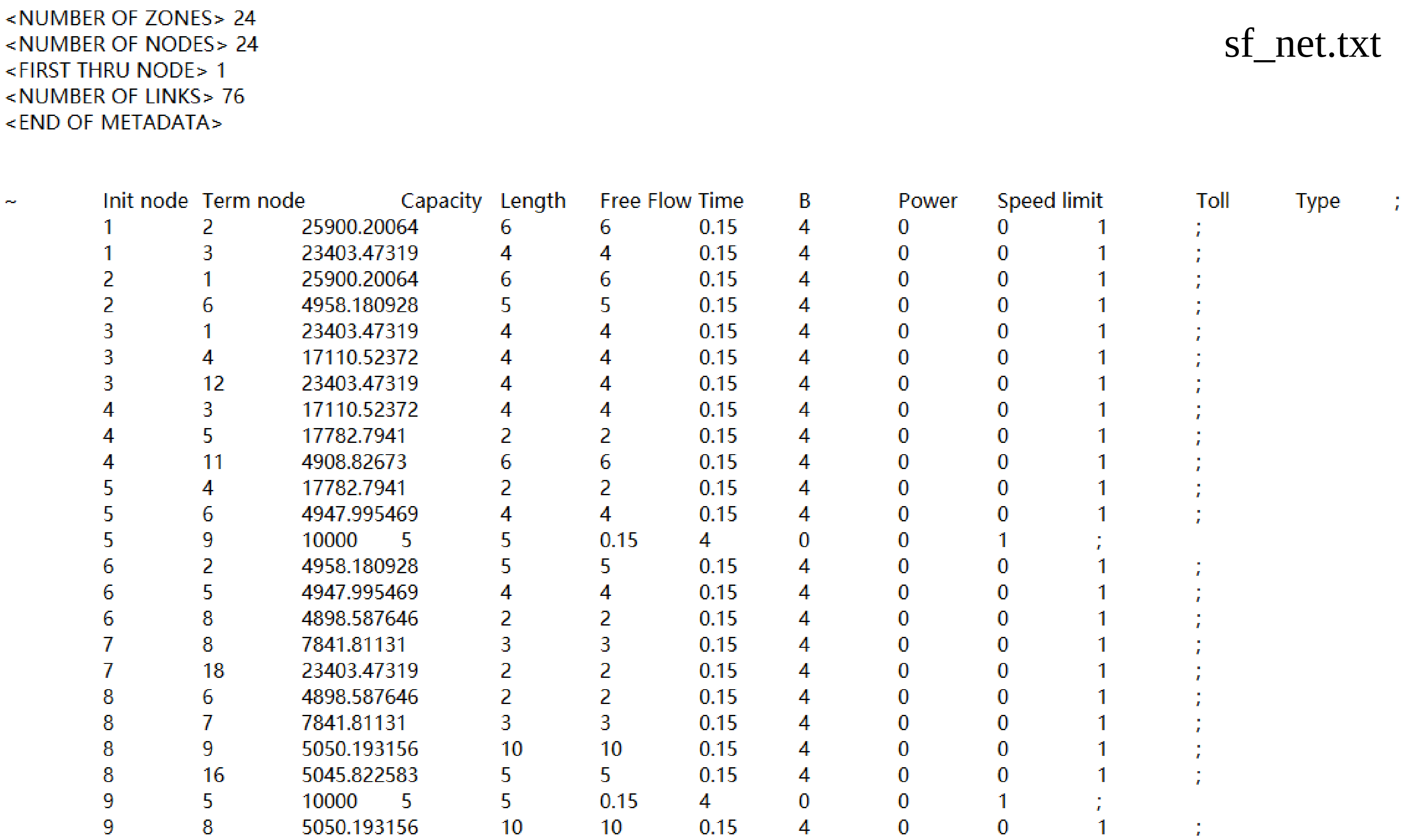


Fig. 2 Sioux-Falls city network. **a** Test network and **b** real network

Zhang X, Waller S T. Implications of link-based equity objectives on transportation network design problem[J]. Transportation, 2019, 46(5): 1559-1589.

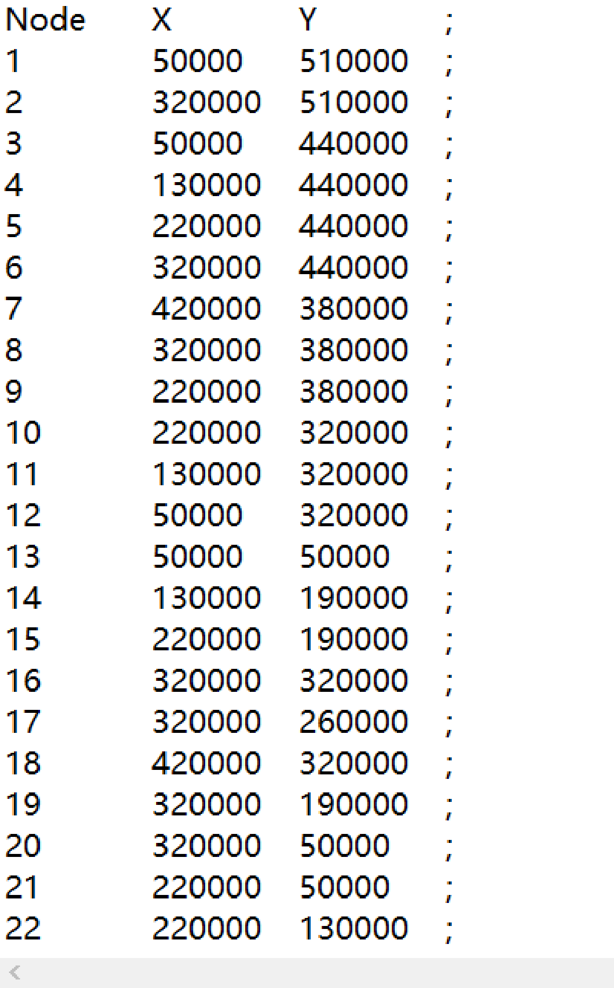
程序部分：网络文件格式

sf_net.txt



```
<NUMBER OF ZONES> 24
<NUMBER OF NODES> 24
<FIRST THRU NODE> 1
<NUMBER OF LINKS> 76
<END OF METADATA>


~           Init node   Term node      Capacity Length      Free Flow Time          B           Power       Speed limit             Toll        Type        ;
            1           2           25900.20064 6           6           0.15        4           0           0           1           ;
            1           3           23403.47319 4           4           0.15        4           0           0           1           ;
            2           1           25900.20064 6           6           0.15        4           0           0           1           ;
            2           6           4958.180928 5           5           0.15        4           0           0           1           ;
            3           1           23403.47319 4           4           0.15        4           0           0           1           ;
            3           4           17110.52372 4           4           0.15        4           0           0           1           ;
            3           12          23403.47319 4           4           0.15        4           0           0           1           ;
            4           3           17110.52372 4           4           0.15        4           0           0           1           ;
            4           5           17782.7941  2           2           0.15        4           0           0           1           ;
            4           11          4908.82673  6           6           0.15        4           0           0           1           ;
            5           4           17782.7941  2           2           0.15        4           0           0           1           ;
            5           6           4947.995469 4           4           0.15        4           0           0           1           ;
            5           9           10000  5    5           0.15        4           0           0           1           ;
            6           2           4958.180928 5           5           0.15        4           0           0           1           ;
            6           5           4947.995469 4           4           0.15        4           0           0           1           ;
            6           8           4898.587646 2           2           0.15        4           0           0           1           ;
            7           8           7841.81131  3           3           0.15        4           0           0           1           ;
            7           18          23403.47319 2           2           0.15        4           0           0           1           ;
            8           6           4898.587646 2           2           0.15        4           0           0           1           ;
            8           7           7841.81131  3           3           0.15        4           0           0           1           ;
            8           9           5050.193156 10          10          0.15        4           0           0           1           ;
            8           16          5045.822583 5           5           0.15        4           0           0           1           ;
            9           5           10000  5    5           0.15        4           0           0           1           ;
            9           8           5050.193156 10          10          0.15        4           0           0           1           ;
```

程序部分：网络文件格式

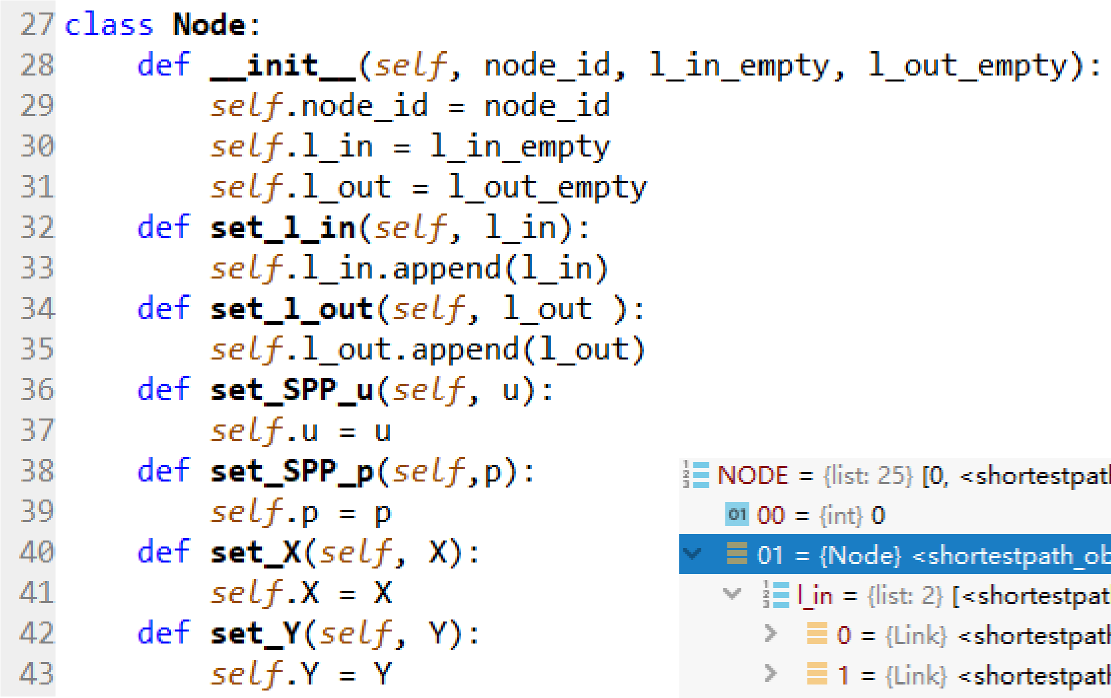


```
Node    X       Y       ;
1       50000   510000  ;
2       320000  510000  ;
3       50000   440000  ;
4       130000  440000  ;
5       220000  440000  ;
6       320000  440000  ;
7       420000  380000  ;
8       320000  380000  ;
9       220000  380000  ;
10      220000  320000  ;
11      130000  320000  ;
12      50000   320000  ;
13      50000   50000   ;
14      130000  190000  ;
15      220000  190000  ;
16      320000  320000  ;
17      320000  260000  ;
18      420000  320000  ;
19      320000  190000  ;
20      320000  50000   ;
21      220000  50000   ;
22      220000  130000  ;
```

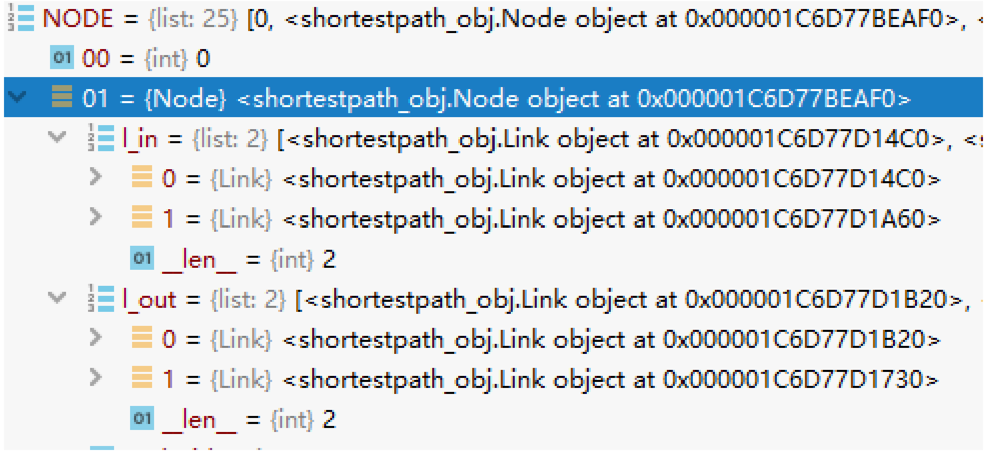
sf_nod.txt

程序部分：数据结构-Node

class Node（节点类）：



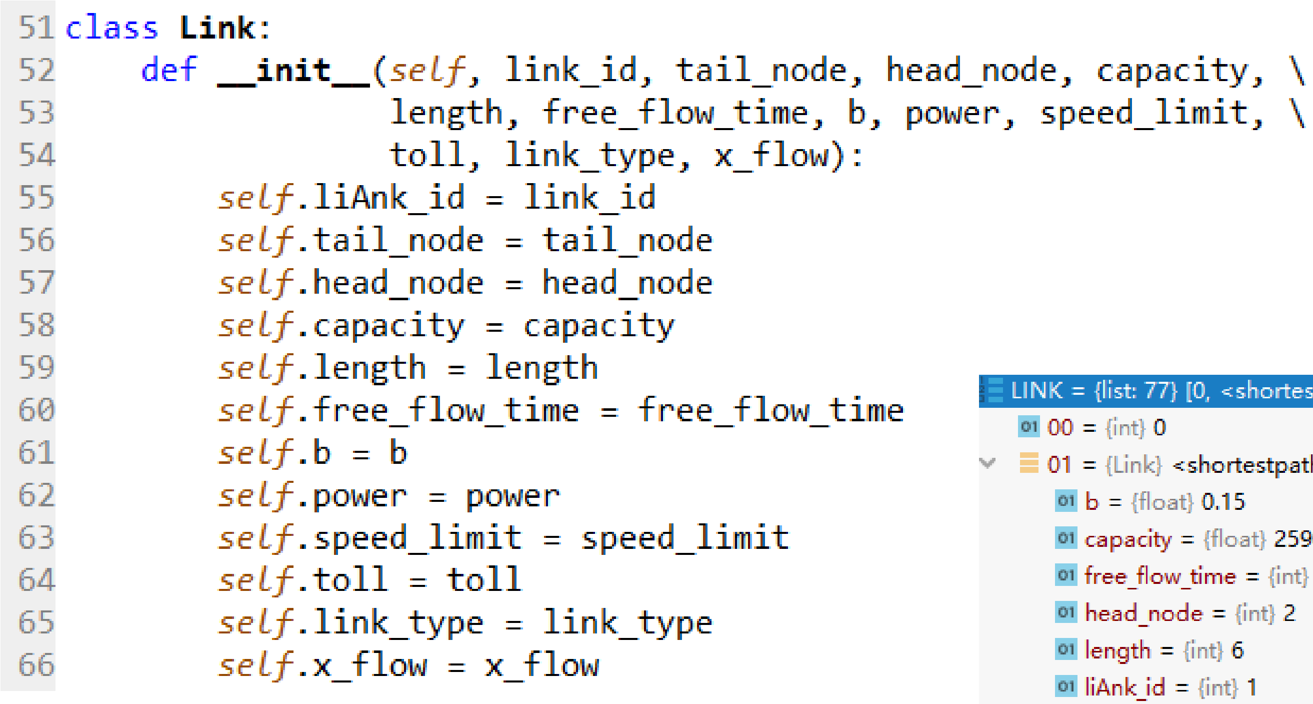
```python
class Node:
    def __init__(self, node_id, l_in_empty, l_out_empty):
        self.node_id = node_id
        self.l_in = l_in_empty
        self.l_out = l_out_empty
    def set_l_in(self, l_in):
        self.l_in.append(l_in)
    def set_l_out(self, l_out ):
        self.l_out.append(l_out)
    def set_SPP_u(self, u):
        self.u = u
    def set_SPP_p(self,p):
        self.p = p
    def set_X(self, X):
        self.X = X
    def set_Y(self, Y):
        self.Y = Y
```



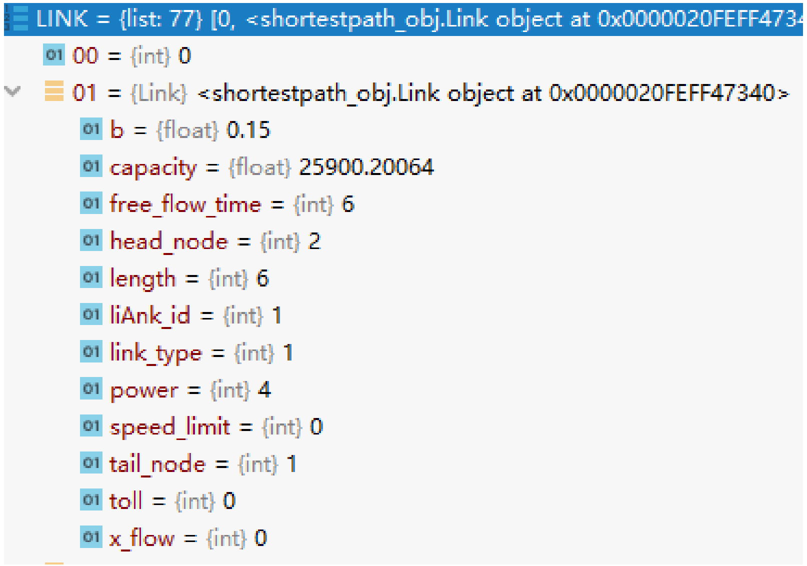
```
NODE = {list: 25} [0, <shortestpath_obj.Node object at 0x000001C6D77BEAF0>,
    00 = {int} 0
  ˅ 01 = {Node} <shortestpath_obj.Node object at 0x000001C6D77BEAF0>
      ˅ l_in = {list: 2} [<shortestpath_obj.Link object at 0x000001C6D77D14C0>, <
          > 0 = {Link} <shortestpath_obj.Link object at 0x000001C6D77D14C0>
          > 1 = {Link} <shortestpath_obj.Link object at 0x000001C6D77D1A60>
            __len__ = {int} 2
      ˅ l_out = {list: 2} [<shortestpath_obj.Link object at 0x000001C6D77D1B20>,
          > 0 = {Link} <shortestpath_obj.Link object at 0x000001C6D77D1B20>
          > 1 = {Link} <shortestpath_obj.Link object at 0x000001C6D77D1730>
            __len__ = {int} 2
```

程序部分：数据结构-Link

class Link（弧类）：

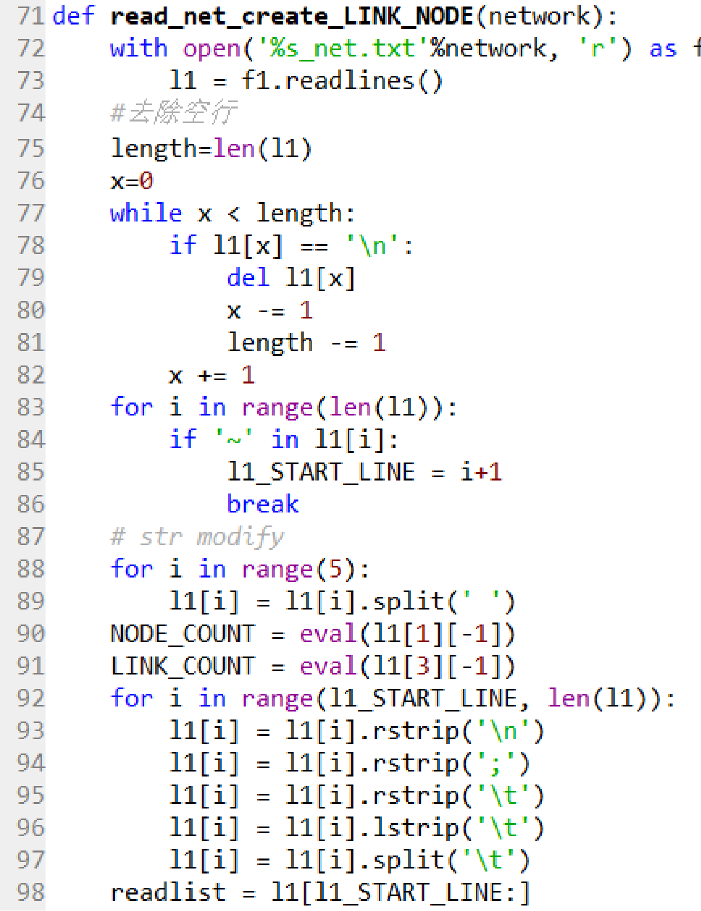


```python
class Link:
    def __init__(self, link_id, tail_node, head_node, capacity, \
                 length, free_flow_time, b, power, speed_limit, \
                 toll, link_type, x_flow):
        self.liAnk_id = link_id
        self.tail_node = tail_node
        self.head_node = head_node
        self.capacity = capacity
        self.length = length
        self.free_flow_time = free_flow_time
        self.b = b
        self.power = power
        self.speed_limit = speed_limit
        self.toll = toll
        self.link_type = link_type
        self.x_flow = x_flow
```

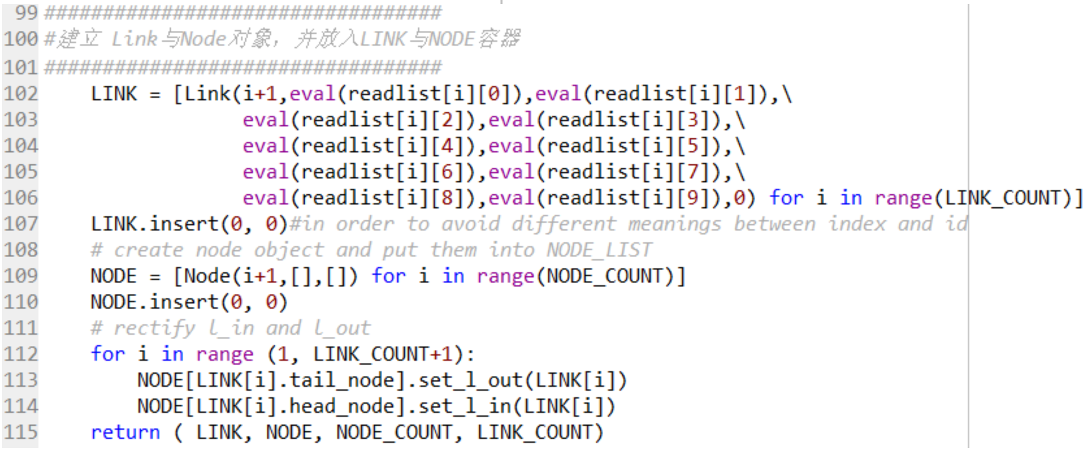


```
LINK = {list: 77} [0, <shortestpath_obj.Link object at 0x0000020FEFF4734
    00 = {int} 0
    01 = {Link} <shortestpath_obj.Link object at 0x0000020FEFF47340>
        b = {float} 0.15
        capacity = {float} 25900.20064
        free_flow_time = {int} 6
        head_node = {int} 2
        length = {int} 6
        liAnk_id = {int} 1
        link_type = {int} 1
        power = {int} 4
        speed_limit = {int} 0
        tail_node = {int} 1
        toll = {int} 0
        x_flow = {int} 0
```

程序部分：读入net文件

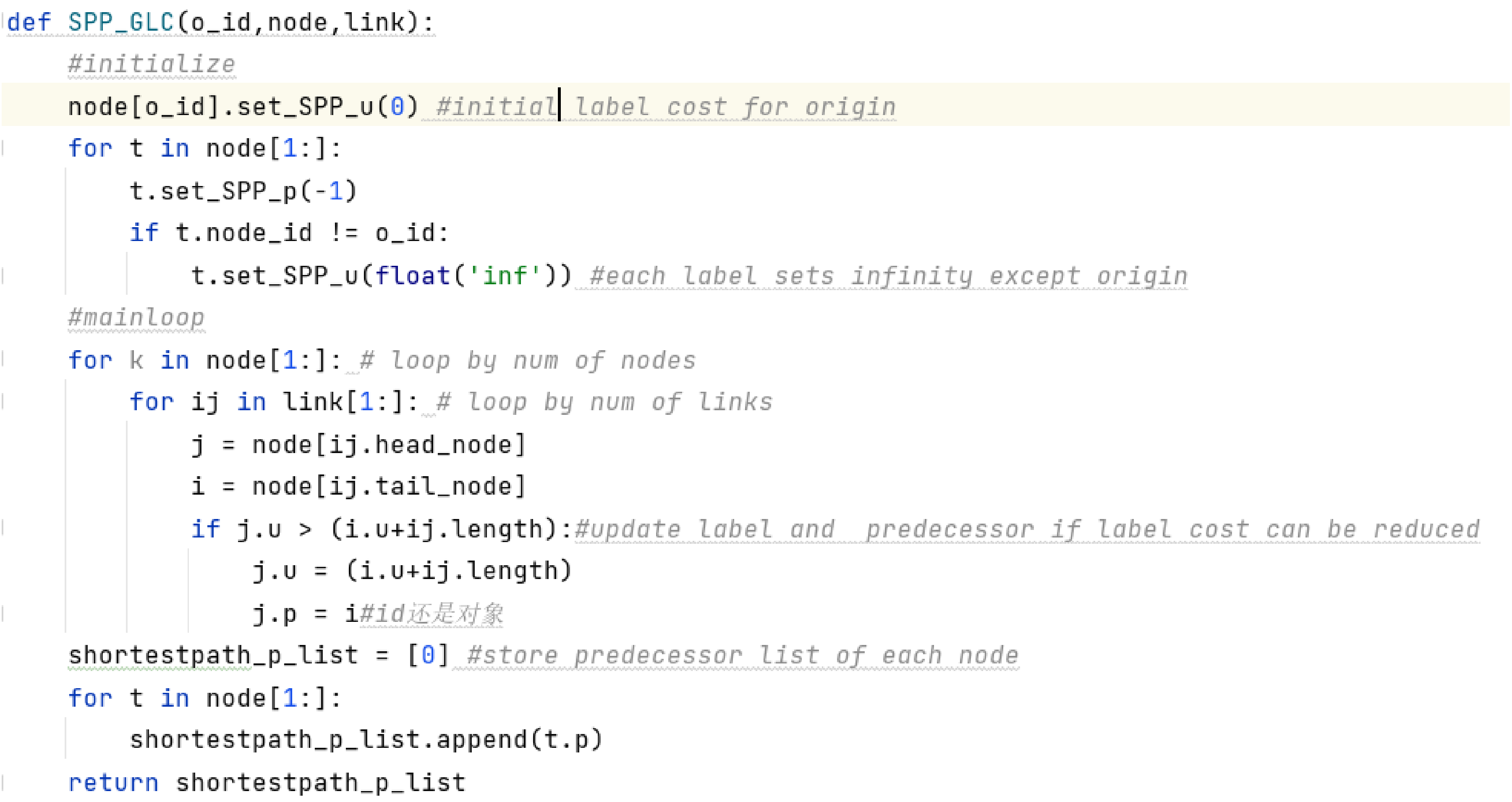


```python
def read_net_create_LINK_NODE(network):
    with open('%s_net.txt'%network, 'r') as f1:
        l1 = f1.readlines()
    #去除空行
    length=len(l1)
    x=0
    while x < length:
        if l1[x] == '\n':
            del l1[x]
            x -= 1
            length -= 1
        x += 1
    for i in range(len(l1)):
        if '~' in l1[i]:
            l1_START_LINE = i+1
            break
    # str modify
    for i in range(5):
        l1[i] = l1[i].split(' ')
    NODE_COUNT = eval(l1[1][-1])
    LINK_COUNT = eval(l1[3][-1])
    for i in range(l1_START_LINE, len(l1)):
        l1[i] = l1[i].rstrip('\n')
        l1[i] = l1[i].rstrip(';')
        l1[i] = l1[i].rstrip('\t')
        l1[i] = l1[i].lstrip('\t')
        l1[i] = l1[i].split('\t')
    readlist = l1[l1_START_LINE:]
```



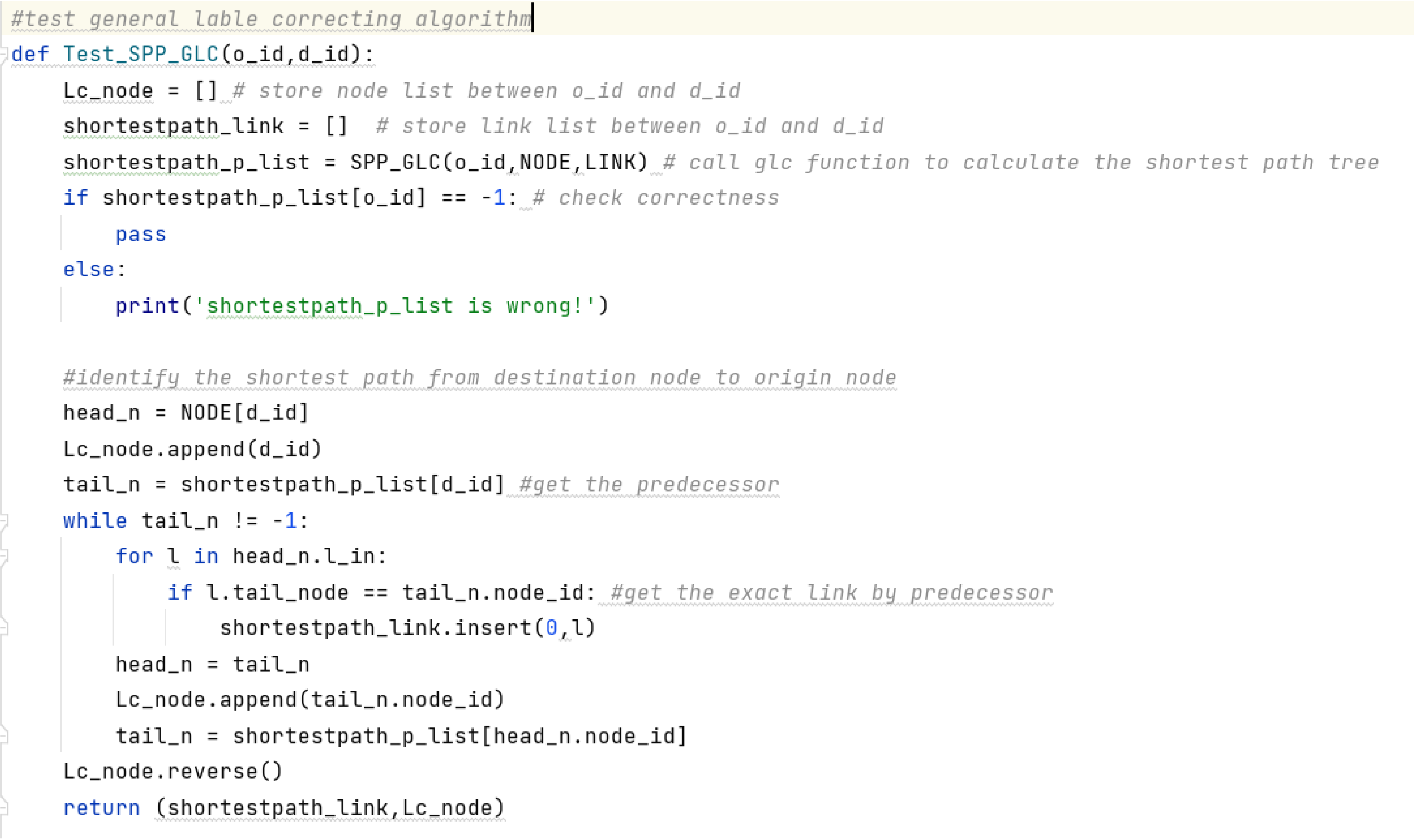
```python
####################################
#建立 Link与Node对象，并放入LINK与NODE容器
####################################
    LINK = [Link(i+1,eval(readlist[i][0]),eval(readlist[i][1]),\
                 eval(readlist[i][2]),eval(readlist[i][3]),\
                 eval(readlist[i][4]),eval(readlist[i][5]),\
                 eval(readlist[i][6]),eval(readlist[i][7]),\
                 eval(readlist[i][8]),eval(readlist[i][9]),0) for i in range(LINK_COUNT)]
    LINK.insert(0, 0)#in order to avoid different meanings between index and id
    # create node object and put them into NODE_LIST
    NODE = [Node(i+1,[],[]) for i in range(NODE_COUNT)]
    NODE.insert(0, 0)
    # rectify l_in and l_out
    for i in range (1, LINK_COUNT+1):
        NODE[LINK[i].tail_node].set_l_out(LINK[i])
        NODE[LINK[i].head_node].set_l_in(LINK[i])
    return ( LINK, NODE, NODE_COUNT, LINK_COUNT)
```

程序部分：General label correcting算法



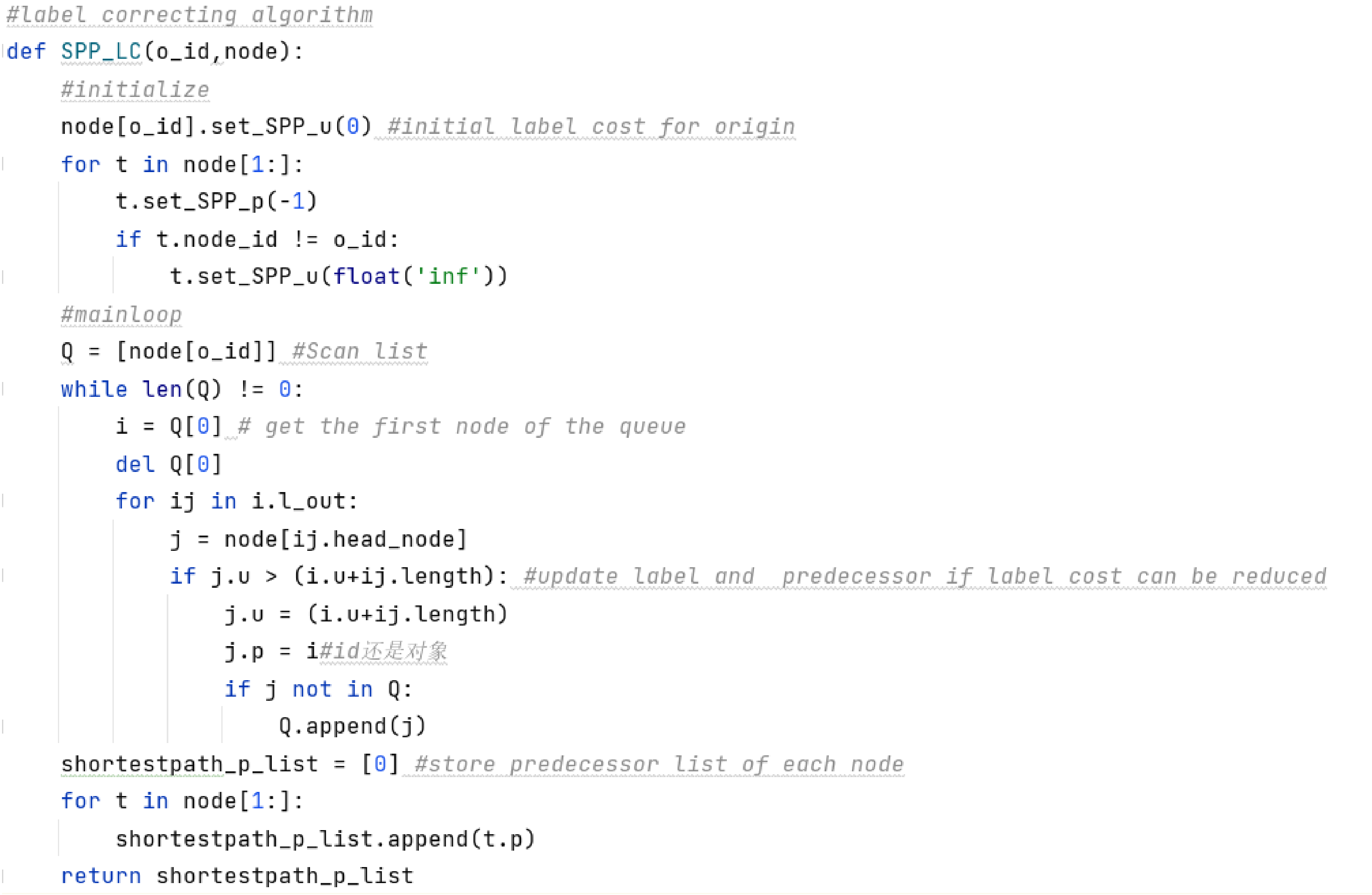
```python
def SPP_GLC(o_id,node,link):
    #initialize
    node[o_id].set_SPP_u(0) #initial label cost for origin
    for t in node[1:]:
        t.set_SPP_p(-1)
        if t.node_id != o_id:
            t.set_SPP_u(float('inf')) #each label sets infinity except origin
    #mainloop
    for k in node[1:]: # loop by num of nodes
        for ij in link[1:]: # loop by num of links
            j = node[ij.head_node]
            i = node[ij.tail_node]
            if j.u > (i.u+ij.length):#update label and  predecessor if label cost can be reduced
                j.u = (i.u+ij.length)
                j.p = i#id还是对象
    shortestpath_p_list = [0] #store predecessor list of each node
    for t in node[1:]:
        shortestpath_p_list.append(t.p)
    return shortestpath_p_list
```

程序部分：打印GLC最短路



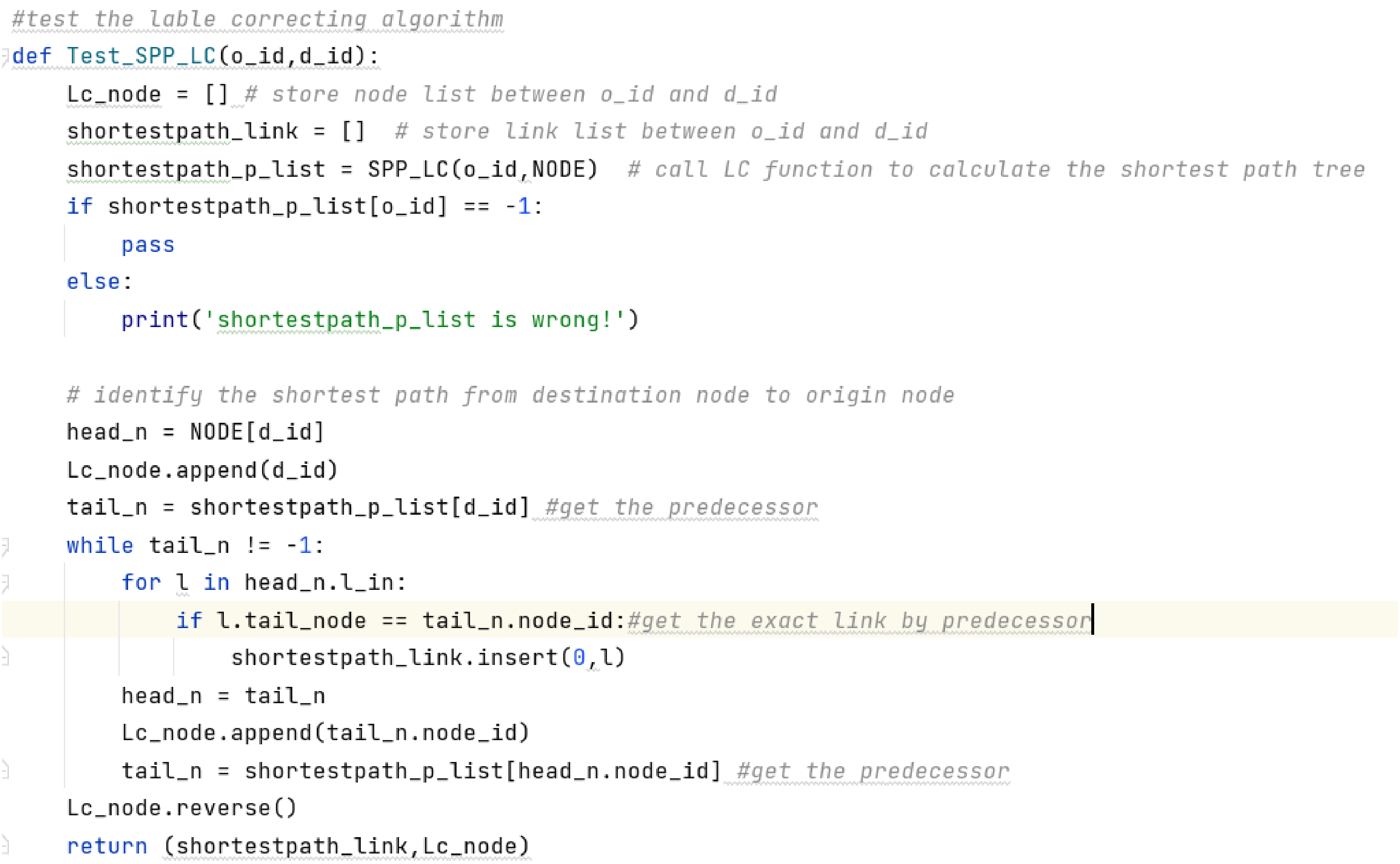
```python
#test general lable correcting algorithm
def Test_SPP_GLC(o_id,d_id):
    Lc_node = [] # store node list between o_id and d_id
    shortestpath_link = []  # store link list between o_id and d_id
    shortestpath_p_list = SPP_GLC(o_id,NODE,LINK) # call glc function to calculate the shortest path tree
    if shortestpath_p_list[o_id] == -1: # check correctness
        pass
    else:
        print('shortestpath_p_list is wrong!')

    #identify the shortest path from destination node to origin node
    head_n = NODE[d_id]
    Lc_node.append(d_id)
    tail_n = shortestpath_p_list[d_id] #get the predecessor
    while tail_n != -1:
        for l in head_n.l_in:
            if l.tail_node == tail_n.node_id: #get the exact link by predecessor
                shortestpath_link.insert(0,l)
        head_n = tail_n
        Lc_node.append(tail_n.node_id)
        tail_n = shortestpath_p_list[head_n.node_id]
    Lc_node.reverse()
    return (shortestpath_link,Lc_node)
```

程序部分：Label correcting算法



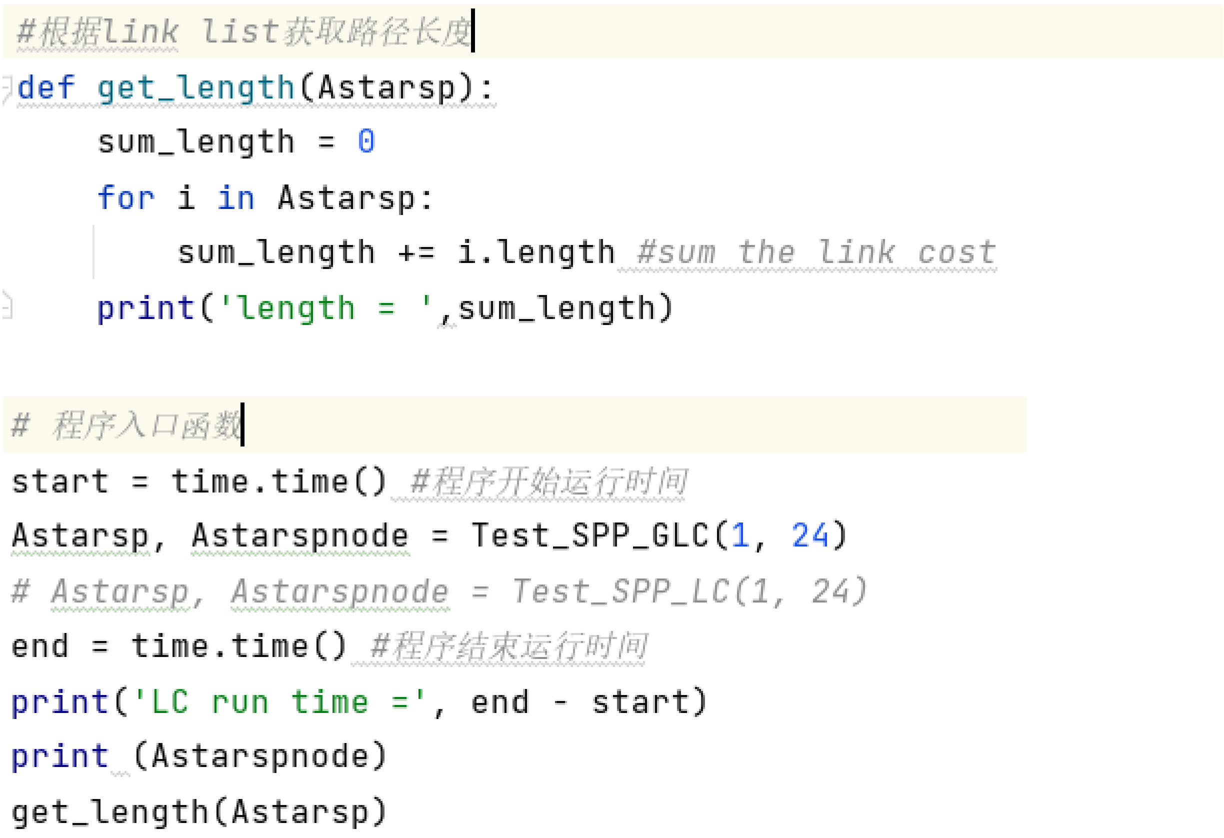
```
#label correcting algorithm
def SPP_LC(o_id,node):
    #initialize
    node[o_id].set_SPP_u(0) #initial label cost for origin
    for t in node[1:]:
        t.set_SPP_p(-1)
        if t.node_id != o_id:
            t.set_SPP_u(float('inf'))
    #mainloop
    Q = [node[o_id]] #Scan list
    while len(Q) != 0:
        i = Q[0] # get the first node of the queue
        del Q[0]
        for ij in i.l_out:
            j = node[ij.head_node]
            if j.u > (i.u+ij.length): #update label and  predecessor if label cost can be reduced
                j.u = (i.u+ij.length)
                j.p = i#id还是对象
                if j not in Q:
                    Q.append(j)
    shortestpath_p_list = [0] #store predecessor list of each node
    for t in node[1:]:
        shortestpath_p_list.append(t.p)
    return shortestpath_p_list
```

程序部分：打印LC最短路

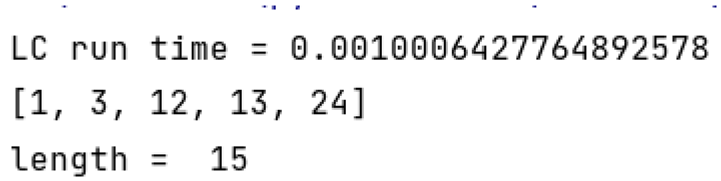


```
#test the lable correcting algorithm
def Test_SPP_LC(o_id,d_id):
    Lc_node = [] # store node list between o_id and d_id
    shortestpath_link = []  # store link list between o_id and d_id
    shortestpath_p_list = SPP_LC(o_id,NODE)  # call LC function to calculate the shortest path tree
    if shortestpath_p_list[o_id] == -1:
        pass
    else:
        print('shortestpath_p_list is wrong!')

    # identify the shortest path from destination node to origin node
    head_n = NODE[d_id]
    Lc_node.append(d_id)
    tail_n = shortestpath_p_list[d_id] #get the predecessor
    while tail_n != -1:
        for l in head_n.l_in:
            if l.tail_node == tail_n.node_id:#get the exact link by predecessor
                shortestpath_link.insert(0,l)
        head_n = tail_n
        Lc_node.append(tail_n.node_id)
        tail_n = shortestpath_p_list[head_n.node_id] #get the predecessor
    Lc_node.reverse()
    return (shortestpath_link,Lc_node)
```

程序部分：调用定义运行函数



```
#根据link list获取路径长度
def get_length(Astarsp):
    sum_length = 0
    for i in Astarsp:
        sum_length += i.length #sum the link cost
    print('length = ',sum_length)

# 程序入口函数
start = time.time() #程序开始运行时间
Astarsp, Astarspnode = Test_SPP_GLC(1, 24)
# Astarsp, Astarspnode = Test_SPP_LC(1, 24)
end = time.time() #程序结束运行时间
print('LC run time =', end - start)
print (Astarspnode)
get_length(Astarsp)
```



```
LC run time = 0.0010006427764892578
[1, 3, 12, 13, 24]
length =  15
```

第一次大作业：

最短路算法分析报告（word形式）：

1、读取Sioux Falls (sf)、ChicagoSketch (cs)、ChicagoRegional(cr)网络

2、依据GLC的代码，**编写LC 和LS算法函数的编写**；

3、针对sf、cs和cr网络，分别挑选**20个O-D对计算最短路并打印输出**；

4、在cs和cr网络上，**对比GLC/LC/LS三个算法效率**，以图表形式体现，O-D对数不限，并对**算法复杂度进行分析。**

- ❑ 报告以word形式提交，代码在python环境下实现提交
- ❑ 严禁抄袭，一旦发现计0分