Start-up Seminar
個人発表①

-最短経路探索-

37-196023　熊野 孝彦

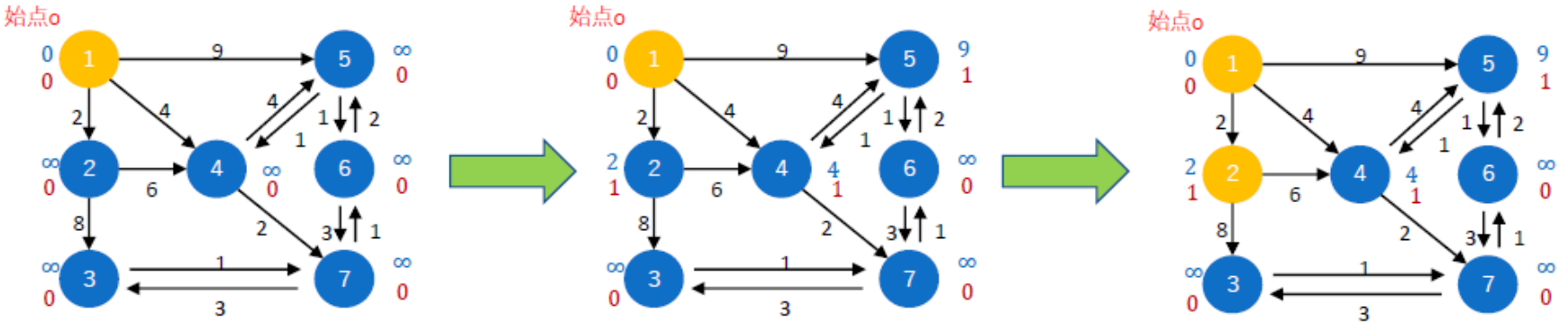
Step.1

Q. Dijkstra法について理解した上で，配布したコードを解読してください

Dijkstra法：起点から隣接するノードへの最小交通費用を計算

→前段階より小さければ更新する

→更新されなくなったら起点を次のノードへ移す



©出原

Step.1

Q. Dijkstra法について理解した上で，配布したコードを解読してください

```python
### ネットワークデータから隣接行列を作成するコード
### for BinN start up seminar 2019
### author Shoma DEHARA
### 2019/03/31

#####################################################
###パッケージの読み込み######
#####################################################

import pandas as pd
import numpy as np

#####################################################
###データ読み込み######
#####################################################

link = pd.read_csv('input/Shibuya_link.csv') # リンクデータをインプット
node = pd.read_csv('input/Shibuya_node.csv') # ノードデータをインプット
# print(link.head())
# print(node.head())
nl = len(link)
nn = max(node['nodeID'])
print(nl) # リンク数
print(nn) # ノード数

#####################################################
###実行######
#####################################################

# 「型」となる零行列を用意します（ノード数×ノード数）.
mat = np.zeros([nn,nn])

# 各列の読み込み
olist = link['n1'] #起点リスト
dlist = link['n2'] #終点リスト
clist = link['LinkCost'] #リンクコストリスト

# 一行ずつ読み込んで零行列の要素を更新
for i in range(nl):
    onode = olist[i]
    dnode = dlist[i]
    mat[onode-1,dnode-1] = clist[i]


#####################################################
###出力######
#####################################################
np.savetxt('input/Shibuya_matrix.csv', mat, delimiter=',' )
```

リンク数 → 4832

ノード数 → 1612

隣接行列を作成

Step.1

Q. Dijkstra法について理解した上で，配布したコードを解読してください．

```python
import numpy as np
from scipy.stats import rankdata
#from numpy import *
import pandas as pd
import time
import itertools
import math

start_time = time.time()

"""
m = [
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0]
]
"""
m = np.genfromtxt( "input/Shibuya_matrix.csv", delimiter="," , filling_values=(0))

m_node = len(m)
print(m_node)

OD = pd.DataFrame(np.loadtxt( "input/ODlist.csv", delimiter="," ,skiprows=1))
OD.columns = ["O", "D"]

Olist = OD['O']
Dlist = OD['D']
```

隣接行列を読み込み

ノード数

発地リスト

着地リスト

```python
def search(m, ori, des):
    max_cost = float('inf')
    unchecked = [False] * m_node
    cost = [max_cost] * m_node
    f_prev = [None] * m_node
    cost[ori-1] = 0
    f_prev[ori-1] = ori
    now = ori

    while True:
        min = max_cost
        next = -1
        unchecked[now-1] = True
        for i in range(m_node):
            if unchecked[i]: continue
            if m[now-1][i]:
                tmp_cost = m[now-1][i] + cost[now-1]
                if cost[i] > tmp_cost:
                    cost[i] = tmp_cost
                    f_prev[i] = now
            if min > cost[i]:
                min = cost[i]
                next = i+1
        now = next
        if next == -1: break
    return [get_path(ori, des, f_prev), cost[des-1]]

def print_path(f_prev, cost):
    for i in range(len(f_prev)):
        print("%d, prev = %d, cost = %d" % (i, f_prev[i], cost[i]))
```

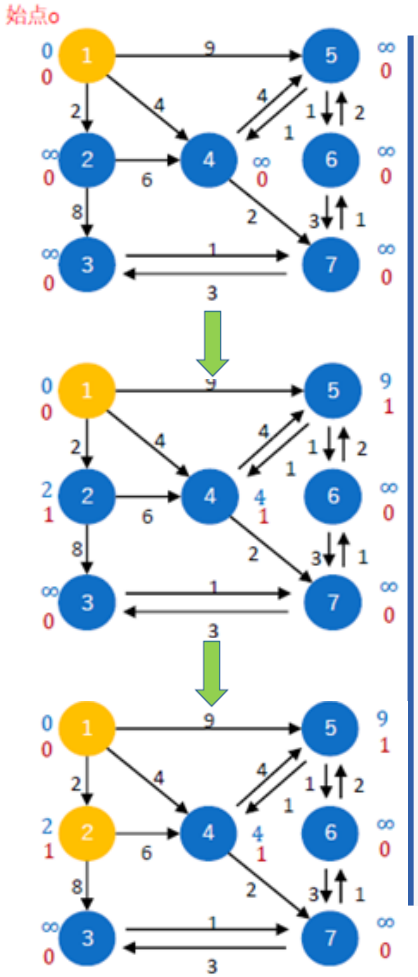
初期コスト=∞

最小費用の更新

起点を次のノードに

Step.1

Q. Dijkstra法について理解した上で，配布したコードを解読してください．




```
31  def search(m, ori, des):
32      max_cost = float('inf')
33      unchecked = [False] * m_node
34      cost = [max_cost] * m_node
35      f_prev = [None] * m_node
36      cost[ori-1] = 0
37      f_prev[ori-1] = ori
38      now = ori
39
40      while True:
41          min = max_cost
42          next = -1
43          unchecked[now-1] = True
44          for i in range(m_node):
45              if unchecked[i]: continue
46              if m[now-1][i]:
47                  tmp_cost = m[now-1][i] + cost[now-1]
48                  if cost[i] > tmp_cost:
49                      cost[i] = tmp_cost
50                      f_prev[i] = now
51              if min > cost[i]:
52                  min = cost[i]
53                  next = i+1
54          now = next
55          if next == -1: break
56      return [get_path(ori, des, f_prev), cost[des-1]]
57
58  def print_path(f_prev, cost):
59      for i in range(len(f_prev)):
60          print("%d, prev = %d, cost = %d" % (i, f_prev[i], cost[i]))
```

初期コスト=∞

最小費用の更新

起点を次のノードに

Step.1

Q. Dijkstra法について理解した上で，配布したコードを解読してください．

```
61
62 def get_path(ori, des, f_prev):
63     path = []
64     now = des
65     path.append(now)
66     while True:
67         path.append(f_prev[now-1])
68         if f_prev[now-1] == ori: break
69         now = f_prev[now-1]
70     path.reverse()
71     return path
72
73 def myround(x, d=0):
74     p = 10 ** d
75     return float(math.floor((x * p) + math.copysign(0.5, x)))/p
76
77 for v in range(len(Olist)):
78     origin = int(Olist[v])
79     destination = int(Dlist[v])
80     print(search(m,origin,destination))
81
82 execution_time = time.time() - start_time
83 print(execution_time)
```

経路情報

計算時間

Step.1

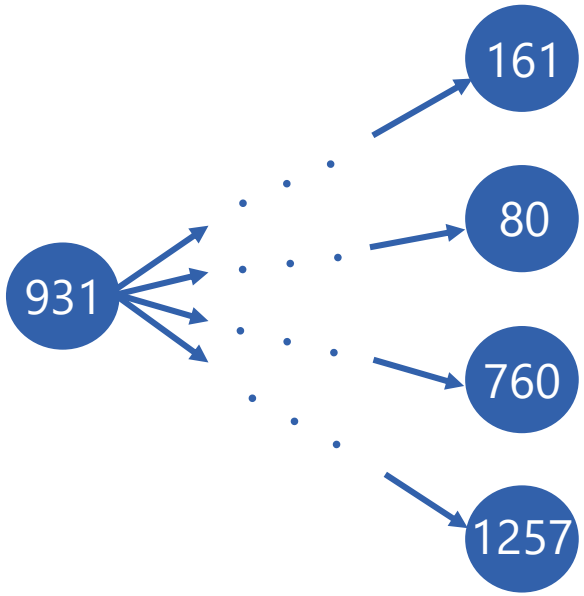
Q. Dijkstra法について理解した上で，配布したコードを解読してください．

ノード数

経路情報

```
1612
[[931, 186, 183, 177, 176, 175, 174, 170, 168, 167, 163, 162, 161], 426.90161687700004]
[[931, 186, 943, 944, 178, 172, 171, 958, 166, 154, 202, 153, 152, 849, 151, 150, 136, 137, 138, 93, 90, 87, 83, 81, 80], 1371.50684226]
[[931, 1233, 1147, 294, 295, 297, 300, 350, 353, 354, 355, 356, 357, 358, 359, 363, 1066, 1115, 392, 399, 420, 429, 759, 1142, 1448, 1447, 760], 1415.9943885799998]
[[931, 1039, 539, 924, 512, 1207, 1042, 507, 1209, 505, 504, 503, 689, 659, 660, 661, 685, 686, 687, 688, 1239, 1240, 1250, 1251, 1257], 1803.7652265199995]
9.441357374191284
```

計算時間



Step.1

Q. 渋谷か松山のネットワークデータを用いて，Dijkstra法による最短経路探索を実行し，結果をGISで表示してください.

Step.2

Q. Bellman-Ford法かA*アルゴリズムについて各自ひとつ調べてまとめてください.

・A*アルゴリズム

Dijkstra法を推定値つきの場合に一般化したアルゴリズム.

一回に計算できるのは辺の重みが非負値のOD１ペアに対する最短経路のみ.

ヒューリスティックス関数という探索の道標となる関数を用いており，最短経路の距離の下限値がわかっている場合，Dijkstra法より高速で求まる.

Step.2

・特徴

二次元の地図での探索では，ヒューリスティック関数hを次のように表せる

$$h = \sqrt{x^2 + y^2}$$（ユークリッド距離）

また，hが恒等的に0のとき，Dijkstra法と一致する．

負の重みのリンクがある場合はBellman-Ford法を用いる．

一つの起点からいくつかの場所への最短経路を求める場合，同時に求めるDijkstra法より計算時間が遅くなってしまう場合もある．

Step.2

Q. 調べた最短経路探索手法をPythonで実装してください.

※渋谷のデータで実装

```python
import time
import numpy as np
from scipy.stats import rankdata
#from numpy import *
import pandas as pd
import itertools
import math

start_time = time.time()

"""
m = [
    [0, 142.2064614, 0, 89.68804473, 0, 0, 0, 0, 0],
    [142.2064614, 0, 314.46613, 0, 204.8164485, 0, 0, 0, 0],
    [0, 314.46613, 0, 0, 0, 81.37164805, 0, 0, 0],
    [89.68804473, 0, 0, 0, 287.8825196, 0, 373.8821118, 0, 0],
    [0, 204.8164485, 0, 287.8825196, 0, 201.4898183, 0, 502.5906807, 0],
    [0, 0, 81.3716480459089, 0, 201.4898183, 0, 0, 0, 496.0501523],
    [0, 0, 0, 373.8821118, 0, 0, 0, 239.2130025, 0],
    [0, 0, 0, 0, 502.5906807, 0, 239.2130025, 0, 129.1523508],
    [0, 0, 0, 0, 0, 496.0501523, 0, 129.1523508, 0],
]
"""
m = np.genfromtxt( "input/Shibuya_matrix.csv", delimiter="," , filling_values=(0) )
m_node = len(m)

OD = pd.DataFrame(np.loadtxt( "input/ODlist.csv", delimiter=",",skiprows=1))
OD.columns = ["O", "D"]

Olist = OD['O']
Dlist = OD['D']

position = pd.DataFrame(np.loadtxt( "input/Shibuya_node.csv", delimiter=",",skiprows=1))
del position[0]
position.columns = ["Lat","Lon"]

Latlist = position['Lat']
Lonlist = position['Lon']

def astar(m,ori,des):
    num = len(m)
    closedset = []
    openset = [ori-1]
    camefrom = {}
    g_score = [float("inf") for i in range(num)]
    g_score[ori-1] = 0
    f_score = {}
    f_score[ori-1] = g_score[ori-1] + heuristic_cost_estimate(ori-1, des-1)
    while openset:
        current = min((f_score[node], node) for node in openset)[1]
        if current == des-1:
            return (reconstruct_path(camefrom,current),g_score[des-1])
        openset.remove(current)
        closedset.append(current)

        for neighbor in neighbor_nodes(current):
            if neighbor in closedset:
                continue
            if neighbor not in openset:
                openset.append(neighbor)
```

隣接行列を読み込み

Step.2

Q. 調べた最短経路探索手法をPythonで実装してください.

```
61              tentative_g_score = g_score[current] + m[current][neighbor]
62              if tentative_g_score >= g_score[neighbor]:
63                  continue
64              camefrom[neighbor] = current
65              g_score[neighbor] = tentative_g_score
66              f_score[neighbor] = g_score[neighbor] + heuristic_cost_estimate(neighbor,des-1)
67      return False
68
69  def reconstruct_path(came_from, current_node):
70      path = [current_node + 1]
71      while current_node in came_from:
72          current_node = came_from[current_node]
73          path.append(current_node + 1)
74      return list(reversed(path))
75
76  def neighbor_nodes(p):
77      neighbors = []
78      for i in range(m_node):
79          if m[p][i] != 0:
80              neighbors.append(i)
81      return neighbors
82
83  def heuristic_cost_estimate(p1, p2):
84      return manhattan_distance(p1, p2)
85
86  def manhattan_distance(p1, p2):
87      return abs(Latlist[p1]-Latlist[p2]) + abs(Lonlist[p1]-Lonlist[p2])
88
89  for v in range(len(Olist)):
90      origin = int(Olist[v])
91      destination = int(Dlist[v])
92      print(origin, destination)
93      print(astar(m,origin,destination))
94
95  execution_time = time.time() - start_time
96  print(execution_time)
```

経路情報

計算時間

Step.2

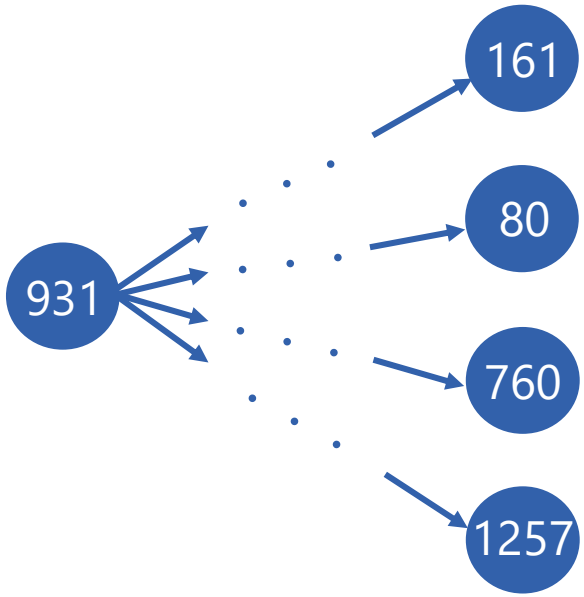
Q. 調べた最短経路探索手法をPythonで実装してください.

経路情報

```
931 161
([931, 186, 183, 177, 176, 175, 174, 170, 168, 167, 163, 162, 161], 426.90161687700004)
931 80
([931, 186, 943, 944, 178, 172, 171, 958, 166, 154, 202, 153, 152, 849, 151, 150, 136, 137, 138, 93, 90, 87, 83, 81, 80], 1371.50684226)
931 760
([931, 1233, 1147, 294, 295, 297, 300, 350, 353, 354, 355, 356, 357, 358, 359, 363, 1066, 1115, 392, 399, 420, 429, 759, 1142, 1448, 1447, 760], 1415.9943885799998)
931 1257
([931, 1039, 539, 924, 512, 1207, 1042, 507, 1209, 505, 504, 503, 689, 659, 660, 661, 685, 686, 687, 688, 1239, 1240, 1250, 1251, 1257], 1803.7652265199995)
13.806444883346558
```

計算時間

経路はDijkstraのときと同じ



Step.2

Q. Dijkstra法と計算時間の比較をしましょう.

Dijkstra法 [s]	A* [s]
9.441357	13.80644

考察： 今回は 1 点(ハチ公前)から 4 点への最短経路を求めたため,
A*の方が長い時間がかかった.