lec1_step9-Search_ALG_Advanced

October 6, 2022

[ ]:
```
## Python basics for novice data scientists, supported by Wagatsuma Lab@Kyutech
#
# The MIT License (MIT): Copyright (c) 2020 Hiroaki Wagatsuma and Wagatsuma␣
 ↪Lab@Kyutech
#
# Permission is hereby granted, free of charge, to any person obtaining a copy␣
 ↪of this software and associated documentation files (the "Software"), to␣
 ↪deal in the Software without restriction, including without limitation the␣
 ↪rights to use, copy, modify, merge, publish, distribute, sublicense, and/or␣
 ↪sell copies of the Software, and to permit persons to whom the Software is␣
 ↪furnished to do so, subject to the following conditions:
# The above copyright notice and this permission notice shall be included in␣
 ↪all copies or substantial portions of the Software.
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR␣
 ↪IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,␣
 ↪FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE␣
 ↪AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER␣
 ↪LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING␣
 ↪FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS␣
 ↪IN THE SOFTWARE. */
#
# # @Time    : 2022-10-6
# # @Author  : Hiroaki Wagatsuma
# # @Site    : https://github.com/hirowgit/2A_python_basic_course
# # @IDE     : Python 3.9
# # @File    : lec1_step9.py
```

[ ]:
```
# Practice 3-2 (page 11/29)
# https://www.slideshare.net/tadahirotaniguchi0624/3-46861684
```

[4]:
```
TargetGraph={
    'S':['A','B'],
    'A':['S','B','C'],
    'B':['S','A','E' ,'F'],
    'C':['A','E','D'],
    'D':['C','E' ,'G'],
    'E':['B','C' ,'D' ,'G'],
```

```
    'F':['B'],
    'G':['D','E']
}
```

[5]:
```
N=7
Node=[chr(i) for i in range(65,65+N)]
Node=['S']+Node
print(Node)
```

```
['S', 'A', 'B', 'C', 'D', 'E', 'F', 'G']
```

[6]:
```
C=[[0, 2, 6, 0, 0, 0, 0, 0],
      [2, 0, 2, 1, 0, 0, 0, 0] ,
      [6, 2, 0, 0, 0, 5, 4, 0] ,
      [0, 1, 0, 0, 5, 2, 0, 0] ,
      [0, 0, 0, 5, 0, 1, 0, 1] ,
      [0, 0, 5, 2, 1, 0, 0, 5] ,
      [0, 0, 4, 0, 0, 0, 0, 0] ,
      [0, 0, 0, 0, 1, 5, 0, 0]
]
```

[15]:
```
OpenList=['S','A','E','F']
state='B'
key=Node.index(state)
Cost=C[key]
indexList=[Node.index(L)  for L in OpenList]
CList=[C[Node.index(state)][i] for i in indexList]
print(indexList)
print(CList)
```

```
[0, 1, 5, 6]
[6, 2, 5, 4]
```

[11]:
```
print(OpenList)
print(CList)
print(' ')
keys = ['node','cost']
# keys2 = ['node','cost','h']
d_all=[]
for i in range(len(OpenList)):
    values=[OpenList[i],CList[i]]
    d = {k: v for k, v in zip(keys, values)}
    d_all.append(d)
print(d_all)
d_all.sort(key=lambda x: x['cost'])
print(d_all)
print([d['node'] for d in d_all])
```

```
['S', 'A', 'E', 'F']
[6, 2, 5, 4]

[{'node': 'S', 'cost': 6}, {'node': 'A', 'cost': 2}, {'node': 'E', 'cost': 5},
{'node': 'F', 'cost': 4}]
[{'node': 'A', 'cost': 2}, {'node': 'F', 'cost': 4}, {'node': 'E', 'cost': 5},
{'node': 'S', 'cost': 6}]
['A', 'F', 'E', 'S']
```

[16]:
```python
# incompleted version of cost-first search (not summation)

OpenList=['S']
ClosedList=[]
while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList=[state]+ClosedList
    ClosedList=list(set(ClosedList))
    print(['state',state])
    print(['OpenList(1)',OpenList])
    print(['ClosedList',ClosedList])
    if state=='G':
        break
    tmpSt=set(TargetGraph[state]) -set(ClosedList)
    activeNodes=list(tmpSt -set(OpenList))
    OpenList.extend(activeNodes)

    # cost-first sorting
    indexList=[Node.index(L)  for L in OpenList]
    CList=[C[Node.index(state)][i] for i in indexList]

    keys = ['node','cost']
    d_all=[]
    for i in range(len(OpenList)):
        values=[OpenList[i],CList[i]]
        d = {k: v for k, v in zip(keys, values)}
        d_all.append(d)
    d_all.sort(key=lambda x: x['cost'])
    OpenList=[d['node'] for d in d_all]
    # ---

    print(['OpenList(2)',OpenList])
    print('')
print('Completed')
```

```
['state', 'S']
['OpenList(1)', []]
['ClosedList', ['S']]
```

```
['OpenList(2)', ['A', 'B']]

['state', 'A']
['OpenList(1)', ['B']]
['ClosedList', ['S', 'A']]
['OpenList(2)', ['C', 'B']]

['state', 'C']
['OpenList(1)', ['B']]
['ClosedList', ['C', 'S', 'A']]
['OpenList(2)', ['B', 'E', 'D']]

['state', 'B']
['OpenList(1)', ['E', 'D']]
['ClosedList', ['C', 'B', 'S', 'A']]
['OpenList(2)', ['D', 'F', 'E']]

['state', 'D']
['OpenList(1)', ['F', 'E']]
['ClosedList', ['C', 'B', 'A', 'D', 'S']]
['OpenList(2)', ['F', 'E', 'G']]

['state', 'F']
['OpenList(1)', ['E', 'G']]
['ClosedList', ['C', 'F', 'B', 'A', 'D', 'S']]
['OpenList(2)', ['E', 'G']]

['state', 'E']
['OpenList(1)', ['G']]
['ClosedList', ['C', 'F', 'E', 'B', 'A', 'D', 'S']]
['OpenList(2)', ['G']]

['state', 'G']
['OpenList(1)', []]
['ClosedList', ['C', 'F', 'G', 'E', 'B', 'A', 'D', 'S']]
Completed
```

[23]:
```
CList=[]
len(CList)
```

[23]:
```
0
```

[25]:
```
CList=[]
C2=[0]*len(C[0])
print(C2)
C2[1]=1
print(C2)
```

```
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
```

[28]:
```
C[Node.index(state)][i]+C[1][2]
```

[28]:
```
2
```

[29]:
```
d_all
```

[29]:
```
[{'node': 'G', 'cost': 5}]
```

[33]:
```
CList=[0]*len(C[0])
CList
```

[33]:
```
[0, 0, 0, 0, 0, 0, 0, 0]
```

[47]:
```
print(indexList)
# [CList[i]+C[Node.index(state)][i] for i in indexList]
CList

Node.index(state)
```

```
[7]
```

[47]:
```
7
```

[92]:
```
connN
```

[92]:
```
['', '', '', '', '', '', '', '']
```

[95]:
```
# completed version of cost-first search ( summation)

OpenList=['S']
ClosedList=[]
CList=[]
C2=[0]*len(C[0])
connN=['']*len(C[0])
Path=[]

while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList=[state]+ClosedList
    ClosedList=list(set(ClosedList))
    print(['state',state])
    print(['OpenList(1)',OpenList])
    print(['ClosedList',ClosedList])
    if state=='G':
```

```
        Path.append(pre_state)
        Path.append('G')
        break
    tmpSt=set(TargetGraph[state]) -set(ClosedList)
    activeNodes=list(tmpSt -set(OpenList))
    OpenList.extend(activeNodes)

    # cost-first sorting
    indexList=[Node.index(L)  for L in OpenList]
    if len(CList)==0:
        CList=[C[Node.index(state)][i] for i in indexList]
    else:
        print(indexList)
        CList=[C2[Node.index(state)]+C[Node.index(state)][i] for i in indexList]

    keys = ['node','cost']
    d_all=[]
    flagU1=False
    flagU2=True
    for i in range(len(OpenList)):
        values=[OpenList[i],CList[i]]
        d = {k: v for k, v in zip(keys, values)}
        if (C[Node.index(state)][Node.index(values[0])]>0) and ( C2[Node.
↪index(values[0])]>values[1] or C2[Node.index(values[0])]==0):
            # connected node                                                    ␣
↪    and  (not larger than before                          or    blank )
            C2[Node.index(values[0])]=values[1]
            connN[Node.index(values[0])]=state
            flagU1=True
        if   C2[Node.index(values[0])]<values[1]:
            flagU2=False
        d_all.append(d)
    if flagU1 and flagU2:
        Path.append(state)
    d_all.sort(key=lambda x: x['cost'])
    OpenList=[d['node'] for d in d_all]
    print(OpenList)
    print(C2)
    print(['Path',Path])
    # ---
    pre_state=state
    print(['OpenList(2)',OpenList])
    print('')

connN2=[connN,Node]
print('Completed')
print(' ')
```

```
print(C2)
print(['Path',Path])
connN2
```

```
['state', 'S']
['OpenList(1)', []]
['ClosedList', ['S']]
['A', 'B']
[0, 2, 6, 0, 0, 0, 0, 0]
['Path', ['S']]
['OpenList(2)', ['A', 'B']]

['state', 'A']
['OpenList(1)', ['B']]
['ClosedList', ['S', 'A']]
[2, 3]
['C', 'B']
[0, 2, 4, 3, 0, 0, 0, 0]
['Path', ['S', 'A']]
['OpenList(2)', ['C', 'B']]

['state', 'C']
['OpenList(1)', ['B']]
['ClosedList', ['C', 'S', 'A']]
[2, 5, 4]
['B', 'E', 'D']
[0, 2, 4, 3, 8, 5, 0, 0]
['Path', ['S', 'A', 'C']]
['OpenList(2)', ['B', 'E', 'D']]

['state', 'B']
['OpenList(1)', ['E', 'D']]
['ClosedList', ['C', 'B', 'S', 'A']]
[5, 4, 6]
['D', 'F', 'E']
[0, 2, 4, 3, 8, 5, 8, 0]
['Path', ['S', 'A', 'C']]
['OpenList(2)', ['D', 'F', 'E']]

['state', 'D']
['OpenList(1)', ['F', 'E']]
['ClosedList', ['C', 'B', 'A', 'D', 'S']]
[6, 5, 7]
['F', 'E', 'G']
[0, 2, 4, 3, 8, 5, 8, 9]
['Path', ['S', 'A', 'C']]
['OpenList(2)', ['F', 'E', 'G']]
```

```
['state', 'F']
['OpenList(1)', ['E', 'G']]
['ClosedList', ['C', 'F', 'B', 'A', 'D', 'S']]
[5, 7]
['E', 'G']
[0, 2, 4, 3, 8, 5, 8, 9]
['Path', ['S', 'A', 'C']]
['OpenList(2)', ['E', 'G']]

['state', 'E']
['OpenList(1)', ['G']]
['ClosedList', ['C', 'F', 'E', 'B', 'A', 'D', 'S']]
[7]
['G']
[0, 2, 4, 3, 8, 5, 8, 9]
['Path', ['S', 'A', 'C']]
['OpenList(2)', ['G']]

['state', 'G']
['OpenList(1)', []]
['ClosedList', ['C', 'F', 'G', 'E', 'B', 'A', 'D', 'S']]
Completed

[0, 2, 4, 3, 8, 5, 8, 9]
['Path', ['S', 'A', 'C', 'E', 'G']]
```

[95]:
```
[['', 'S', 'A', 'A', 'C', 'C', 'B', 'D'],
 ['S', 'A', 'B', 'C', 'D', 'E', 'F', 'G']]
```

[13]:
```
CList
```

[13]:
```
[5]
```

[20]:
```
H =[4,4,2,3,1,1,0,0]
```

[100]:
```
# completed version of best-first search

OpenList=['S']
ClosedList=[]
Path=[]

while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList=[state]+ClosedList
    ClosedList=list(set(ClosedList))
```

```
    print(['state',state])
    print(['OpenList(1)',OpenList])
    print(['ClosedList',ClosedList])
    if state=='G':
        Path.append('G')
        break
    tmpSt=set(TargetGraph[state]) -set(ClosedList)
    activeNodes=list(tmpSt -set(OpenList))
    OpenList.extend(activeNodes)

    # heuristic value sorting
    indexList=[Node.index(L)  for L in OpenList]
    HList=[H[i] for i in indexList]

    keys = ['node','h']
    d_all=[]
    for i in range(len(OpenList)):
        values=[OpenList[i],HList[i]]
        d = {k: v for k, v in zip(keys, values)}
        d_all.append(d)
    d_all.sort(key=lambda x: x['h'])
    OpenList=[d['node'] for d in d_all]
    Path.append(state)
    # ---

    print(['OpenList(2)',OpenList])
    print('')
print('Completed')
print(' ')
print(['Path',Path])
```

```
['state', 'S']
['OpenList(1)', []]
['ClosedList', ['S']]
['OpenList(2)', ['B', 'A']]

['state', 'B']
['OpenList(1)', ['A']]
['ClosedList', ['B', 'S']]
['OpenList(2)', ['F', 'E', 'A']]

['state', 'F']
['OpenList(1)', ['E', 'A']]
['ClosedList', ['B', 'S', 'F']]
['OpenList(2)', ['E', 'A']]

['state', 'E']
```

```
['OpenList(1)', ['A']]
['ClosedList', ['E', 'B', 'S', 'F']]
['OpenList(2)', ['G', 'D', 'C', 'A']]

['state', 'G']
['OpenList(1)', ['D', 'C', 'A']]
['ClosedList', ['F', 'G', 'E', 'B', 'S']]
Completed

['Path', ['S', 'B', 'F', 'E', 'G']]
```

[99]:
```python
# completed version of A* search

OpenList=['S']
ClosedList=[]
CList=[]
C2=[0]*len(C[0])
connN=['']*len(C[0])
Path=[]

while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList=[state]+ClosedList
    ClosedList=list(set(ClosedList))
    print(['state',state])
    print(['OpenList(1)',OpenList])
    print(['ClosedList',ClosedList])
    if state=='G':
#         Path.append(pre_state)
        Path.append('G')
        break
    tmpSt=set(TargetGraph[state]) -set(ClosedList)
    activeNodes=list(tmpSt -set(OpenList))
    OpenList.extend(activeNodes)

    # heuristic value sorting
    indexList=[Node.index(L)  for L in OpenList]
    HList=[H[i] for i in indexList]

    # cost-first sorting
    if len(CList)==0:
        CList=[C[Node.index(state)][i] for i in indexList]
    else:
        print(indexList)
        CList=[C2[Node.index(state)]+C[Node.index(state)][i] for i in indexList]
```

```
    keys = ['node','cost','h','f']
    d_all=[]
    flagU1=False
    flagU2=True
    for i in range(len(OpenList)):
        values=[OpenList[i],CList[i],HList[i],CList[i]+HList[i]]
        d = {k: v for k, v in zip(keys, values)}
        if (C[Node.index(state)][Node.index(values[0])]>0) and ( C2[Node.
↪index(values[0])]>values[1] or C2[Node.index(values[0])]==0):
            # connected node                                                  ␣
↪    and  (not larger than before                          or    blank )
            C2[Node.index(values[0])]=values[1]
            connN[Node.index(values[0])]=state
            flagU1=True
        if   C2[Node.index(values[0])]<values[1]:
            flagU2=False
        d_all.append(d)
    if flagU1 and flagU2:
        Path.append(state)
    d_all.sort(key=lambda x: x['f'])
    OpenList=[d['node'] for d in d_all]
    print(OpenList)
    print(C2)
    print(['Path',Path])
    # ---
    pre_state=state
    print(['OpenList(2)',OpenList])
    print('')

connN2=[connN,Node]
print('Completed')
print(' ')
print(C2)
print(['Path',Path])
connN2
```

```
['state', 'S']
['OpenList(1)', []]
['ClosedList', ['S']]
['A', 'B']
[0, 2, 6, 0, 0, 0, 0, 0]
['Path', ['S']]
['OpenList(2)', ['A', 'B']]

['state', 'A']
['OpenList(1)', ['B']]
['ClosedList', ['S', 'A']]
```

```
[2, 3]
['B', 'C']
[0, 2, 4, 3, 0, 0, 0, 0]
['Path', ['S', 'A']]
['OpenList(2)', ['B', 'C']]

['state', 'B']
['OpenList(1)', ['C']]
['ClosedList', ['B', 'S', 'A']]
[3, 5, 6]
['C', 'F', 'E']
[0, 2, 4, 3, 0, 9, 8, 0]
['Path', ['S', 'A']]
['OpenList(2)', ['C', 'F', 'E']]

['state', 'C']
['OpenList(1)', ['F', 'E']]
['ClosedList', ['B', 'C', 'S', 'A']]
[6, 5, 4]
['F', 'E', 'D']
[0, 2, 4, 3, 8, 5, 8, 0]
['Path', ['S', 'A', 'C']]
['OpenList(2)', ['F', 'E', 'D']]

['state', 'F']
['OpenList(1)', ['E', 'D']]
['ClosedList', ['C', 'F', 'B', 'A', 'S']]
[5, 4]
['E', 'D']
[0, 2, 4, 3, 8, 5, 8, 0]
['Path', ['S', 'A', 'C']]
['OpenList(2)', ['E', 'D']]

['state', 'E']
['OpenList(1)', ['D']]
['ClosedList', ['C', 'F', 'E', 'B', 'A', 'S']]
[4, 7]
['D', 'G']
[0, 2, 4, 3, 6, 5, 8, 10]
['Path', ['S', 'A', 'C', 'E']]
['OpenList(2)', ['D', 'G']]

['state', 'D']
['OpenList(1)', ['G']]
['ClosedList', ['C', 'F', 'E', 'B', 'A', 'D', 'S']]
[7]
['G']
[0, 2, 4, 3, 6, 5, 8, 7]
```

```
['Path', ['S', 'A', 'C', 'E', 'D']]
['OpenList(2)', ['G']]

['state', 'G']
['OpenList(1)', []]
['ClosedList', ['C', 'F', 'G', 'E', 'B', 'A', 'D', 'S']]
Completed

[0, 2, 4, 3, 6, 5, 8, 7]
['Path', ['S', 'A', 'C', 'E', 'D', 'G']]
```

[99]:
```
[['', 'S', 'A', 'A', 'E', 'C', 'B', 'D'],
 ['S', 'A', 'B', 'C', 'D', 'E', 'F', 'G']]
```