lec1_step5-Search_ALG_Basics

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    : 2020-10-14
# # @Author  : Hiroaki Wagatsuma
# # @Site    : https://github.com/hirowgit/2A_python_basic_course
# # @IDE     : Python 3.7.7 (default, Mar 10 2020, 15:43:27) [Clang 10.0.0␣
 ↪(clang-1000.11.45.5)] on darwin
# # @File    : lec1_step5.py
```

[ ]:
```
# Practice 2-2 (page 21/28)
# https://www.slideshare.net/tadahirotaniguchi0624/2-46861654
```

[8]:
```
# open list and closed list
```

[10]:
```
# first idea
OpenList=[1,2,3,4]
```

[5]:
```
OpenList[1]
```

[5]:
```
2
```

[6]:
```
OpenList[0]  # note array start from [0] like C, C++
```

[6]:
```
1
```

[10]:
```
# As you see in Fig 2.9, open list and closed list should be defined at each␣
 ↪node.
# Therefore those lists require multiple open and closed lists for each node.
# It implies dictionary is a good option.
TargetGraph={
    'S':'A','B',
    'A':'S','C','D',
    'B':'S','C',
    'C':'A','B','D',
    'D':'A','C',
#    'G':'unknown now
}
```

```
  Cell In [10], line 5
    'S':'A','B',
              ^
SyntaxError: ':' expected after dictionary key
```

[11]:
```
TargetGraph={
    'S':['A','B'],
    'A':['S','C','D'],
    'B':['S','C'],
    'C':['A','B','D'],
    'D':['A','C']
#    'G':'unknown now
}
```

[111]:
```
TargetGraph['S']
```

[111]:
```
['A', 'B']
```

[112]:
```
TargetGraph['S'][0]
```

[112]:
```
'A'
```

[3]:
```
TargetGraph['S'].append('G')
```

[23]:
```
print(TargetGraph)
```

```
{'S': ['A', 'B', 'G'], 'A': ['S', 'B'], 'B': ['A', 'B'], 'C': ['A', 'B'], 'D':
['A', 'B']}
```

[114]:
```
# If you want to delete the last item
del TargetGraph['S'][-1]
print(TargetGraph)
```

```
{'S': ['A', 'B'], 'A': ['S', 'C', 'D'], 'B': ['S', 'C'], 'C': ['A', 'B', 'D'],
'D': ['A', 'C']}
```

[115]:
```
tList=[]
if tList:
    print('Not Empty')
else:
    print('Empty')
```

```
Empty
```

[4]:
```
tList=[1,2,3,4,5]
while tList:
    del tList[0]
    print(tList)
print('completed')
```

```
[2, 3, 4, 5]
[3, 4, 5]
[4, 5]
[5]
[]
completed
```

[117]:
```
OpenList=['S']
OpenList.insert(0,['A','B'])
print(OpenList)
```

```
[['A', 'B'], 'S']
```

[118]:
```
sList=['A','B']
[d for d in sList]
```

[118]:
```
['A', 'B']
```

[119]:
```
TargetGraph['A']
```

[119]:
```
['S', 'C', 'D']
```

[6]:
```
OpenList=['S']
sList=['A','B']
```

```
OpenList.insert(0, sList[:])
OpenList=[d for d in OpenList]
print(OpenList)
OpenList=[item for i in OpenList for item in i]
print(OpenList)
```

```
[['A', 'B'], 'S']
['A', 'B', 'S']
```

[13]:
```
OpenList=['S']
sList=['A','B']
OpenList.append(sList)
OpenList
```

[13]:
```
['S', ['A', 'B']]
```

[12]:
```
OpenList=['S']
sList=['A','B']
OpenList.insert(0, sList)
OpenList
```

[12]:
```
[['A', 'B'], 'S']
```

[14]:
```
OpenList=['S']
sList=['A','B']
OpenList.extend(sList)
OpenList
```

[14]:
```
['S', 'A', 'B']
```

[78]:
```
if 'A' in ['A', 'B', 'S']:
    print('Yes')
```

```
Yes
```

[79]:
```
if 'A' not in ['A', 'B', 'S']:
    print('Yes')
```

[88]:
```
tList=[]
addList=['A', 'B', 'S']
ClosedList=['S']
activeNode=[item for item in addList if item not in ClosedList]
activeNode
```

[88]:
```
['A', 'B']
```

[134]:
```
OpenList=['S']
state='S'
OpenList.insert(0, TargetGraph[state])
print(OpenList)

OpenList=['S']
ClosedList=['S']
state='S'
print(TargetGraph[state])
activeNodes=[item for item in TargetGraph[state] if item not in ClosedList]
OpenList.insert(0, activeNodes)
OpenList=[item for i in OpenList for item in i if item not in ClosedList]
print(OpenList)
```

```
[['A', 'B'], 'S']
['A', 'B']
['A', 'B']
```

[13]:
```
OpenList=['S']
ClosedList=[]
while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList.append(state)
    print(state)
    if state=='G':
        break
 #   activeNodes=TargetGraph[state]
    activeNodes=[item for item in TargetGraph[state] if item not in ClosedList]
    OpenList.insert(0, activeNodes)
#    OpenList=[item for i in OpenList for item in i]
    OpenList=[item for i in OpenList for item in i if item not in ClosedList]
print('completed')
```

```
S
A
C
B
D
completed
```

[59]:
```
OpenList=['S']
ClosedList=[]
while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList.extend(state)
```

```
    print(state)
    if state=='G':
        break
 #   activeNodes=TargetGraph[state]
    activeNodes=list(set(TargetGraph[state]) -set(ClosedList))
    OpenList.extend(activeNodes)
#    OpenList=[item for i in OpenList for item in i]
 #   OpenList=set(OpenList) -set(ClosedList)
#    OpenList=[item for i in OpenList for item in i if item not in ClosedList]
#print('completed')
```

```
S
B
A
C
D
C
D
completed
```

[7]:
```
# completed version of Depth-first search

OpenList=['S']
ClosedList=[]
while OpenList:
    state=OpenList[-1]
    del OpenList[-1]
    ClosedList.extend(state)
    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)
  #  OpenList=list(set(OpenList))
    print(['OpenList(2)',OpenList])
    print('')
print('Completed')
```

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

['state', 'B']
```

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

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

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

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

Completed
```

[13]:
```
# completed version of Depth-first search

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)
  #  OpenList=list(set(OpenList))
    print(['OpenList(2)',OpenList])
    print('')
print('Completed')
```

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

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

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

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

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

Completed
```

[14]:
```
# completed version of Breadth-first search

OpenList=['S']
ClosedList=[]
while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList=ClosedList+[state]
    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)
  #  OpenList=list(set(OpenList))
    print(['OpenList(2)',OpenList])
    print('')
print('Completed')
```

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

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

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

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

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

Completed
```

[16]:
```
activeNodes
```

[16]:
```
[]
```

[15]:
```
# completed version of Breadth-first search

OpenList=['S']
ClosedList=[]
while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList.extend(state)
    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)
  #  OpenList=list(set(OpenList))
    print(['OpenList(2)',OpenList])
    print('')
```

```
print('Completed')
```

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

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

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

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

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

Completed
```

[138]:
```
tmpL= ['D', 'C',  'A', 'S', 'C', 'A', 'B', 'A', 'B']
tmpL= [ 'A', 'S', 'C', 'A', 'B', 'B', 'A']
list(set(tmpL))
```

[138]:
```
['S', 'B', 'A', 'C']
```

[120]:
```
ClosedList=['S', 'B', 'A', 'C', 'D', 'C', 'D']
set(ClosedList)
```

[120]:
```
{'A', 'B', 'C', 'D', 'S'}
```

[116]:
```
OpenList=['S']
ClosedList=[]
while OpenList:
    state=OpenList[-1]
    del OpenList[-1]
    ClosedList.extend(state)
    print(state)
```

```
    if state=='G':
        break
 #   activeNodes=TargetGraph[state]
    activeNodes=list(set(TargetGraph[state]) -set(ClosedList))
    #print(state)
    OpenList.extend(activeNodes)
#    OpenList=[item for i in OpenList for item in i]
 #   OpenList=set(OpenList) -set(ClosedList)
#    OpenList=[item for i in OpenList for item in i if item not in ClosedList]
print('completed')
```

```
S
A
C
B
D
D
B
completed
```

[114]:
```
OpenList=['S']
ClosedList=[]
state=OpenList[0]
del OpenList[0]
ClosedList.extend(state)
print(state)
activeNodes=list(set(TargetGraph[state]) -set(ClosedList))
print(activeNodes)
print(OpenList)
OpenList=activeNodes.extend(OpenList)
print(OpenList)
```

```
S
['B', 'A']
[]
None
```

[103]:
```
OpenList=['S']
ClosedList=[]
state=OpenList[0]
del OpenList[0]
ClosedList.extend(state)
print(state)
activeNodes=list(set(TargetGraph[state]) -set(ClosedList))

print('activeNodes')
print(activeNodes)
```

```
print('OpenList')
print(OpenList)
print('ClosedList')
print(ClosedList)
OpenList.extend(activeNodes)
print('OpenList')
print(OpenList)
```

```
S
activeNodes
['B', 'A']
OpenList
[]
ClosedList
['S']
OpenList
['B', 'A']
```

[109]:
```
state=OpenList[0]
del OpenList[0]
ClosedList.extend(state)
print(state)
activeNodes=list(set(TargetGraph[state]) -set(ClosedList))

print('activeNodes')
print(activeNodes)
print('OpenList')
print(OpenList)
print('ClosedList')
print(ClosedList)
OpenList.extend(activeNodes)
print('OpenList')
print(OpenList)
```

```
D
activeNodes
[]
OpenList
[]
ClosedList
['S', 'B', 'A', 'C', 'D', 'C', 'D']
OpenList
[]
```

[88]:
```
activeNodes=['B', 'A']
print(activeNodes)
OpenList=[]
```

```
print(OpenList)
activeNodes.extend(OpenList)
print(activeNodes)
print(OpenList)
print(activeNodes.extend(OpenList))
#OpenList=activeNodes.extend(OpenList)
activeNodes.extend(OpenList)
print(OpenList)
print(activeNodes)
activeNodes.extend(OpenList)
```

```
['B', 'A']
[]
['B', 'A']
[]
None
[]
['B', 'A']
```

[66]:
```
state=OpenList[0]
del OpenList[0]
ClosedList.extend(state)
print(state)
activeNodes=list(set(TargetGraph[state]) -set(ClosedList))
print(activeNodes)
#OpenList.extend(activeNodes)
OpenList=activeNodes.extend(OpenList)
OpenList
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/mg/w5t8lkhc8xj79f001s7kzpfh0000gp/T/ipykernel_45436/4057407122.py␣
 ↪in <module>
----> 1 state=OpenList[0]
      2 del OpenList[0]
      3 ClosedList.extend(state)
      4 print(state)
      5 activeNodes=list(set(TargetGraph[state]) -set(ClosedList))

TypeError: 'NoneType' object is not subscriptable
```

[57]:
```
OpenList
activeNodes
```

[57]:
```
{'C'}
```

[51]:
```
OpenList=['S']
del OpenList[0]
activeNodes
OpenList=activeNodes.extend(OpenList)
```

[44]:
```
activeNodes=list(set(TargetGraph[state]) -set(ClosedList))
activeNodes
OpenList=activeNodes.extend(OpenList)
OpenList
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/mg/w5t8lkhc8xj79f001s7kzpfh0000gp/T/ipykernel_45436/4272406267.py␣
 ↪in <module>
      1 activeNodes=list(set(TargetGraph[state]) -set(ClosedList))
      2 activeNodes
----> 3 OpenList=activeNodes.extend(OpenList)
      4 OpenList

TypeError: 'NoneType' object is not iterable
```

[25]:
```
OpenList=['S']
ClosedList=[]
state=OpenList[0]
del OpenList[0]
ClosedList.extend(state)
ClosedList
TargetGraph[state]
OpenList
```

[25]:
```
[]
```

[29]:
```
set(['S','A','B'])-set(['C','A','B'])
```

[29]:
```
{'S'}
```

[28]:
```
set([1,2,3,4,5])-set([2,4])
```

[28]:
```
{1, 3, 5}
```

[136]:
```
TargetGraph={
    'A':['B','C'],
    'B':['A','D','E'],
    'C':['A','F','G','H'],
    'D':['B','I'],
    'E':['B'],
```

```
    'F':['C'],
    'G':['C','J'],
    'H':['C'],
    'I':['D'],
    'J':['G']
#    'G':'unknown now
}
```

[143]:
```
OpenList=['A']
ClosedList=[]
k=1
while OpenList:
    state=OpenList[0]
    del OpenList[0]
    ClosedList.append(state)
    print(str(k)+": "+state)
    if state=='Goal':
        break
 #   activeNodes=TargetGraph[state]
    activeNodes=[item for item in TargetGraph[state] if item not in ClosedList]
    OpenList.insert(0, activeNodes)
#    OpenList=[item for i in OpenList for item in i]
    OpenList=[item for i in OpenList for item in i if item not in ClosedList]
    k=k+1
print('completed')
```

```
1: A
2: B
3: D
4: I
5: E
6: C
7: F
8: G
9: J
10: H
completed
```

[140]:
```
activeNodes=[item for item in TargetGraph[state] if item not in ClosedList]
activeNodes
```

[140]: []

[144]:
```
TargetG=['A','B','C','D','E','F']
ClosedList=['C','F']
OpenList=['A']
activeNodes=[item for item in TargetG if item not in ClosedList]
```

```
activeNodes
print('activeNodes',activeNodes)
OpenList.insert(0, activeNodes)
print('OpenList',OpenList)
```

```
activeNodes ['A', 'B', 'D', 'E']
OpenList [['A', 'B', 'D', 'E'], 'A']
```

[145]:
```
TargetG=['A','B','C','D','E','F']
ClosedList=['C','F']
OpenList=['A']
activeNodes=[item for item in TargetG if item not in ClosedList]
activeNodes
print('activeNodes',activeNodes)
OpenList.append(activeNodes)
print('OpenList',OpenList)
```

```
activeNodes ['A', 'B', 'D', 'E']
OpenList ['A', ['A', 'B', 'D', 'E']]
```

[146]:
```
TargetG=['A','B','C','D','E','F']
ClosedList=['C','F']
OpenList=['A']
activeNodes=[item for item in TargetG if item not in ClosedList]
activeNodes
print('activeNodes',activeNodes)
OpenList.extend(activeNodes)
print('OpenList',OpenList)
```

```
activeNodes ['A', 'B', 'D', 'E']
OpenList ['A', 'A', 'B', 'D', 'E']
```

[148]:
```
TargetG=['A','B','C','D','E','F']
ClosedList=['C','F']
OpenList=['A']
activeNodes=[item for item in TargetG if item not in ClosedList]
print('activeNodes',activeNodes)
```

```
activeNodes ['A', 'B', 'D', 'E']
```

[149]:
```
TargetG=['A','B','C','D','E','F']
ClosedList=['C','F']
OpenList=['A']
activeNodes=list(set(TargetG)-set(ClosedList))
print('activeNodes',activeNodes)
```

```
activeNodes ['D', 'B', 'A', 'E']
```

[154]:
```
import numpy as np
from pytictoc import TicToc
t = TicToc() #create instance of class
t.tic() #Start timer
t.toc() #Time elapsed since t.tic() Elapsed time is 2.612231 seconds.
```

```
Elapsed time is 0.000015 seconds.
```

[167]:
```
t = TicToc() #create instance of class
loopN=100
NofD_target=10000
NofD_clist=5000
t.tic() #Start timer
for i in range(0,loopN):
    rD_target=np.random.randint(0,NofD_target,size=NofD_target)
    rD_clist=np.random.randint(0,NofD_target,size=NofD_clist)
    rDL_target=rD_target.tolist()
    rDL_clist=rD_clist.tolist()
    activeNodes=[item for item in rDL_target if item not in rDL_clist]

    # print(rDL_target)
    # print(rDL_clist)

t.toc()
```

```
Elapsed time is 45.894154 seconds.
```

[168]:
```
t = TicToc() #create instance of class
loopN=100
NofD_target=10000
NofD_clist=5000
t.tic() #Start timer
for i in range(0,loopN):
    rD_target=np.random.randint(0,NofD_target,size=NofD_target)
    rD_clist=np.random.randint(0,NofD_target,size=NofD_clist)
    rDL_target=rD_target.tolist()
    rDL_clist=rD_clist.tolist()
    activeNodes=list(set(rD_target)-set(rD_clist))
    # print(rDL_target)
    # print(rDL_clist)

t.toc()
```

```
Elapsed time is 0.214059 seconds.
```

[171]:
```
sD = np.arange(10).reshape(2, 5)
sD = sD/2
print(sD)
```

```
np.savetxt('csv_data_d.csv', sD, delimiter=',', fmt='%d')
np.savetxt('csv_data_f.csv', sD, delimiter=',', fmt='%.5f')
```

```
[[0.  0.5 1.  1.5 2. ]
 [2.5 3.  3.5 4.  4.5]]
```