CSE 512 – Winter 2018 – Lab 3

Instructor: Kerstin Voigt
Tuesdays 1:30-3:20pm in JB 359

In this lab, we will build the foundation for our later implementation of "Heuristic Search" (lecture to come). Below, you will see a listing of code for search functions of increasing levels of sophistication. Relatively to where we will be going over the next weo weeks, these functions are all basic building blocks.

Begin with a "starter" copy of file `graphsearch_lab3.py` and copy into it the function definitions shown below. While you do this, we will interrupt periodically in order to experiment and discuss the specifics of each function individually.

Exercise 1: Copy functions dfs_ and bfs_traverse, dfs_ and bfs_ search, dfs_ and bfs_search_path, and best_search_path.

Exercise 2: Only if you are done with Exercise 1, and you have a good understanding of best_search_path, attempt the definition of a related function best_search_ALLpaths. This new function is to list not only the first solution path found, but list paths that are possible between a given start and goal node.

Credit for this lab: Work conscientiously on the exercises and sign up on this week's signup sheet.

Code listing:

```
#graphsearch_lab3.py
# by Kerstin Voigt, Jan 2018, for lectures and lab on GRAPHSEARCH

import random

#directed, acyclic graph of "nodes"
# example from Poole et.al. "Computational Intelligence" Oxford Univ Press,,
# 1998,pp. 117, 121

GRAPH = {'o103':['o109','ts','l2d3'],
         'o109':['o111','o119'],
         'ts':['mail'],
         'o111':[],
         'mail':[],
         'o119':['store','o123'],
         'store':[],
         'o123':['o125','r123'],
```

```
         'o125':[],
         'r123':[],
         'l2d3':['l2d1','l2d4'],
         'l2d1':['l3d2','l2d2'],
         'l2d4':['o109'],
         'l2d2':['l2d4'],
         'l3d2':['l3d3','l3d1'],
         'l3d3':[],
         'l3d1':['l3d3']}

COST = {('o103','ts'):8, ('o103','o109'):12, ('o103','l2d3'):4,\
        ('ts','mail'):6,\
        ('o109','o111'):4, ('o109','o119'):16,\
        ('o119','store'):7, ('o119','o123'):9,\
        ('o123','r123'):4, ('o123','o125'):4,\
        ('l2d1','l3d2'):3, ('l2d1','l2d2'):6,\
        ('l2d2','l2d4'):3, ('l2d3','l2d1'):4,\
        ('l2d3','l2d4'):7, ('l2d4','o109'):7,\
        ('l3d2','l3d3'):6, ('l3d2','l3d1'):4,\
        ('l3d1','l3d3'):8}

def dfs_traverse(start):
  open = [start]
  closed = []
  while open != []:
    nxt = open[0]
    open = open[1:]
    closed.append(nxt)
    print nxt
    succ = GRAPH[nxt]
    random.shuffle(succ)  # WHY DO THIS?
    for x in succ:
      if not x in closed:
        open = [x] + open
  return closed

def bfs_traverse(start):
  open = [start]
  closed = []
  while open != []:
    nxt = open[0]
    open = open[1:]
    closed.append(nxt)
    print nxt
    succ = GRAPH[nxt]
    random.shuffle(succ)
    for x in succ:
      if not x in closed:
        open.append(x)
  return closed

def dfs_search(start,goal):
  open = [start]
  closed = []
  while open != []:
    nxt = open[0]
    if nxt == goal:
```

```
      return goal
    open = open[1:]
    closed.append(nxt)
    print nxt
    succ = GRAPH[nxt]
    #random.shuffle(succ)
    for x in succ:
      if not x in closed:
        open = [x] + open
  return None

def bfs_search(start,goal):
  open = [start]
  closed = []
  while open != []:
    nxt = open[0]
    if nxt == goal:
      return goal
    open = open[1:]
    closed.append(nxt)
    print nxt
    succ = GRAPH[nxt]
    random.shuffle(succ)
    for x in succ:
      if not x in closed:
        open.append(x)
  return None

def dfs_search_path(start,goal):
  open = [[start,[start]]]
  closed = []
  k = 1
  while open != []:
    nxt= open[0]
    nxtnode = nxt[0]
    nxtpath = nxt[1]
    if nxtnode == goal:
      return nxt
    open = open[1:]
    closed.append(nxtnode)
    print "%d. %s" % (k,nxt)
    succ = GRAPH[nxtnode]
    random.shuffle(succ)
    for x in succ:
      if not x in closed:
        open = [[x,addpath(nxtpath,x)]] + open
    k += 1
  return None

def bfs_search_path(start,goal):
  open = [[start,[start]]]
  closed = []
  k = 1
  while open != []:
    nxt = open[0]
    nxtnode = nxt[0]
    nxtpath = nxt[1]
```

```
    if nxtnode == goal:
      return nxt
    open = open[1:]
    closed.append(nxtnode)
    print "%d. %s" % (k,nxt)
    succ = GRAPH[nxtnode]
    random.shuffle(succ)
    for x in succ:
      if not x in closed:
        open.append([x,addpath(nxtpath,x)])
    k += 1
  return None

def addpath(path,x):
  newpath = path[:]
  newpath.append(x)
  return newpath

def best_search_path(start,goal):
  open = [[start,[start],0]]
  closed = []
  k = 1
  while open != []:
    nxt = open[0]
    nxtnode = nxt[0]
    nxtpath = nxt[1]
    nxtcost = nxt[2]
    if nxtnode == goal:
      return nxt
    open = open[1:]
    closed.append(nxtnode)
    print "%d. %s" % (k,nxt)
    succ = GRAPH[nxtnode]
    random.shuffle(succ)
    for x in succ:
      if not x in closed:
        xcost = COST[(nxtnode,x)]
        open.append([x,addpath(nxtpath,x),nxtcost+xcost])
    open.sort(lambda x,y: CostCmp(x,y))
    k += 1
  return None

# x,y are lists that are compared by
# magnitude of their third elements;
def CostCmp (x,y):
  if x[2] < y[2]:
    return -1
  elif x[2] == y[2]:
    return 0
  else:
    return 1
```