Universitario: Ceron Beimar Miguel

3 objetos de busqueda

Carrera: Ing. de Sistemas

Repositorio git hub:

https://github.com/Beimar98/SIS420/tree/main/1er%20par
(https://github.com/Beimar98/SIS420/tree/main/1er%20pa

Problema 5 Construir un programa que permita encontrar el camino mas corto para identificas dos objetos en un laberinto.

 # Problema 5.2 Construir un programa que permita encontrar el camino mas corto para identificas tres objetos en un laberinto.

In [27]:

```python
# %%
import heapq

# %% Clase padre de todas las funciones
class ColaPrioridadLimitada(object):
    # inicializar las intancias constructores limite de la lista

    def __init__(self, limite=None, *args):
        #limite o terminacion
        self.limite = limite
        # cola en forma de lista
        self.queue = list()

    def __getitem__(self, val):
        return self.queue[val]

    def __len__(self):
        return len(self.queue)

    def push(self, x):
        heapq.heappush(self.queue, x)
        if self.limite and len(self.queue) > self.limite:
            self.queue.remove(heapq.nlargest(1, self.queue)[0])

    def pop(self):
        return heapq.heappop(self.queue)

    def extend(self, iterable):
        for x in iterable:
            self.append(x)

    def clear(self):
        for x in self:
            self.queue.remove(x)

    def remove(self, x):
        self.queue.remove(x)

    # Metodo para saber si la cola esta vacia
    def empty(self):
        if not self.heap:
            return True
        else:
            return False

    def sorted(self):
        return heapq.nsmallest(len(self.queue), self.queue)

# %%
class NodoBusqueda(object):
    '''Nodo para el proceso de busqueda.'''

    def __init__(self, estado, padre=None, accion=None, costo=0, problema=None, profun
        self.estado = estado
        self.padre = padre
        self.accion = accion
        self.costo = costo
        self.problema = problema or padre.problema
        self.profundidad = profundidad
```

```
60
61      def expandir(self, busqueda_local=False):
62          '''Crear sucesores.'''
63          nodos_nuevos = []
64          for accion in self.problema.acciones(self.estado):
65              estado_nuevo = self.problema.resultado(self.estado, accion)
66              costo = self.problema.costo(self.estado, accion, estado_nuevo)
67              fabrica_nodos = self.__class__
68              nodos_nuevos.append(fabrica_nodos(estado=estado_nuevo,
69                                                padre=None if busqueda_local else self,
70                                                problema=self.problema,
71                                                accion=accion,
72                                                costo=self.costo + costo,
73                                                profundidad=self.profundidad + 1))
74          return nodos_nuevos
75
76      def camino(self):
77          '''Camino (lista de nodos y acciones) desde el nodo raiz al nodo actual.'''
78          nodo = self
79          camino = []
80          while nodo:
81              camino.append((nodo.accion, nodo.estado))
82              nodo = nodo.padre
83          return list(reversed(camino))
84
85      def __eq__(self, otro):
86          return isinstance(otro, NodoBusqueda) and self.estado == otro.estado
87
88      def estado_representacion(self):
89          return self.problema.estado_representacion(self.estado)
90
91      def accion_representacion(self):
92          return self.problema.accion_representacion(self.accion)
93
94      def __repr__(self):
95          return 'Node <%s>' % self.estado_representacion().replace('\n', ' ')
96
97      def __hash__(self):
98          return hash((
99              self.estado,
100             self.padre,
101             self.accion,
102             self.costo,
103             self.profundidad,
104         ))
105
106 # %%
107 class NodoBusquedaHeuristicaOrdenado(NodoBusqueda):
108     def __init__(self, *args, **kwargs):
109         super(NodoBusquedaHeuristicaOrdenado, self).__init__(*args, **kwargs)
110         self.heuristica = self.problema.heuristica(self.estado)
111
112     def __lt__(self, otro):
113         return self.heuristica < otro.heuristica
114
115 # %%
116 class NodoBusquedaAEstrellaOrdenado(NodoBusquedaHeuristicaOrdenado):
117     def __lt__(self, otro):
118         return self.heuristica + self.costo < otro.heuristica + otro.costo
119
120 # %%
```

```python
class ProblemaBusqueda(object):
    '''Clase base abstracta, para representar y manipular los espacio de busqueda
    de un problema. IEn esta clase, el espacio de búsqueda debe representarse
    implícitamente como un gráfico.
    Cada estado corresponde con un estado del problema(es decir, una configuración vál
    y cada accion del problema(es decir, una transformación válida a una configuración
    Para utilizar esta clase se debe implementar metodos requeridos by el algoritmo de
    que se utilizara.'''

    def __init__(self, estado_inicial=None):
        self.estado_inicial = estado_inicial

    def acciones(self, estado):
        '''Devuelve las acciones disponibles para realizar a partir de un estado.
        El valor devuelto es íterador sobre acciones.
        Las acciones son específicas del problema y no se debe asumir nada sobre ellas
        '''
        raise NotImplementedError

    def resultado(self, estado, accion):
        '''Debuelve un nuevo estado despues de aplicar una accion a estado.'''
        raise NotImplementedError

    def costo(self, estado, accion, estado2):
        '''Devuelve el costo de aplicar una accion para alcanzar el estado2 a partir d
            El valor devuelto es un numero (intero o de punto flotante),
            por defecto la funcion devuelve 1.
        '''
        return 1

    def es_objetivo(self, estado):
        '''Devuelve True si estado es el estado_objetivo y false caso contrario'''
        raise NotImplementedError

    def valor(self, estado):
        '''Devuelve el valor de `estado`, para motivos de optimizacion.
           valor es un numero (entero o punto flotante).'''
        raise NotImplementedError

    def heuristica(self, estado):
        '''DEvuelve un estimado del costo faltante para alcanzar la solucion a partir
        return 0

    def estado_representacion(self, estado):
        """
        Devuelve un string de representacion de un estado.
        Por defecto devuelve str(estado).
        """
        return str(estado)

    def accion_representacion(self, accion):
        """
        Devuelve un string de representacion de una acción.
        Por defecto devuelve str(acción).
        """
        return str(accion)

# %%
def voraz(problema, busqueda_en_grafo=False, viewer=None):
    '''
    Busqueda voraz.
```

```
182
183     Si se establece busqueda_en_grafo=True, se eliminara la busqueda en estados repeti
184     Requiere redefinir las funciones de la clase ProblemaBusqueda:
185     ProblemaBusqueda.acciones, ProblemaBusqueda.resultado, y
186     ProblemaBusqueda.es_objetivo, ProblemaBusqueda.costo,
187     ProblemaBusqueda.heuristica.
188     '''
189     return _buscar(problema,
190                    ColaPrioridadLimitada(),
191                    busqueda_en_grafo=busqueda_en_grafo,
192                    fabrica_nodos=NodoBusquedaHeuristicaOrdenado,
193                    reemplazar_grafo_cuando_mejor=True)
194
195 # %%
196 def aestrella(problema, busqueda_en_grafo=False, viewer=None):
197     '''
198     Busuqeda A*.
199
200     Si se establece busqueda_en_grafo=True, se eliminara la busqueda en estados repeti
201     Requiere redefinir las funciones de la clase ProblemaBusqueda:
202     ProblemaBusqueda.acciones, ProblemaBusqueda.resultado, y
203     ProblemaBusqueda.es_objetivo, ProblemaBusqueda.costo,
204     ProblemaBusqueda.heuristica.
205     '''
206     return _buscar(problema,
207                    ColaPrioridadLimitada(),
208                    busqueda_en_grafo=busqueda_en_grafo,
209                    fabrica_nodos=NodoBusquedaAEstrellaOrdenado,
210                    reemplazar_grafo_cuando_mejor=True)
211
212 # %%
213 def _buscar(problema, frontera, busqueda_en_grafo=False, limite_profundidad=None,
214             fabrica_nodos=NodoBusqueda, reemplazar_grafo_cuando_mejor=False):
215     '''
216     Algoritmo basico de busqueda, base de todos los demas algoritmos de busqueda.
217     '''
218     memoria = set()
219     nodo_inicio = fabrica_nodos(estado=problema.estado_inicial, problema=problema)
220     frontera.push(nodo_inicio)
221
222     while frontera:
223         nodo = frontera.pop()
224
225         if problema.es_objetivo(nodo.estado):
226             return nodo
227
228         memoria.add(nodo.estado)
229
230         if limite_profundidad is None or nodo.profundidad < limite_profundidad:
231             expandido = nodo.expandir()
232
233             for n in expandido:
234                 if busqueda_en_grafo:
235                     otros = [x for x in frontera if x.estado == n.estado]
236                     assert len(otros) in (0, 1)
237                     if n.estado not in memoria and len(otros) == 0:
238                         frontera.push(n)
239                     elif reemplazar_grafo_cuando_mejor and len(otros) > 0 and n < otro
240                         frontera.remove(otros[0])
241                         frontera.push(n)
242                 else:
```

```
                    frontera.push(n)

# %%
import math



MAPA = """
##############################
# o       #              # x #
# ####    ########   #####   #
#         #              #   #
#    ###  #  ####   ######   #
#      #           #        #
#  #       w  #  #           #
#######   ####    #    #######
#z       #      #    #       #
##############################
"""
MAPA = [list(x) for x in MAPA.split("\n") if x]

COSTOS = {
    "arriba": 1.0,
    "abajo": 1.0,
    "izquierda": 1.0,
    "derecha": 1.0,
    "arriba izquierda": 2.0,
    "arriba derecha": 2.0,
    "abajo izquierda": 2.0,
    "abajo derecha": 2.0,
}


class JuegoLaberinto(ProblemaBusqueda):

    def __init__(self, tablero, objetivo):
        self.tablero = tablero
        self.estado_objetivo = (0, 0)
        for y in range(len(self.tablero)):
            for x in range(len(self.tablero[y])):
                if self.tablero[y][x].lower() == "o":
                    self.estado_inicial = (x, y)
                elif self.tablero[y][x].lower() == objetivo:
                    self.estado_objetivo = (x, y)

        super(JuegoLaberinto, self).__init__(estado_inicial=self.estado_inicial)
# Creacion de paredes
    def acciones(self, estado):
        acciones = []
        for accion in list(COSTOS.keys()):
            nuevox, nuevoy = self.resultado(estado, accion)
            if self.tablero[nuevoy][nuevox] != "#":
                acciones.append(accion)
        return acciones

    def resultado(self, estado, accion):
        x, y = estado

        if accion.count("arriba"):
            y -= 1
        if accion.count("abajo"):
```

```
            y += 1
        if accion.count("izquierda"):
            x -= 1
        if accion.count("derecha"):
            x += 1

        estado_nuevo = (x, y)
        return estado_nuevo

    def es_objetivo(self, estado):
        return estado == self.estado_objetivo

    def costo(self, estado, accion, estado2):
        return COSTOS[accion]

    def heuristic(self, estado):
        x, y = estado
        gx, gy = self.estado_objetivo
        return math.sqrt((x - gx) ** 2 + (y - gy) ** 2)

## manjador o controlador del mapa
def main():
    problema = JuegoLaberinto(MAPA, "x")

    resultado1 = aestrella(problema, busqueda_en_grafo=True)
#     resultado = voraz(problema, busqueda_en_grafo=True)
    camino1Count = 0

    camino1 = [x[1] for x in resultado1.camino()]



    problema2 = JuegoLaberinto(MAPA, "w")

    resultado2 = aestrella(problema2, busqueda_en_grafo=True)
#     resultado = voraz(problema, busqueda_en_grafo=True)
    camino1Count2 = 0

    camino2 = [x[1] for x in resultado2.camino()]


#     resultado = voraz(problema, busqueda_en_grafo=True)
    problema3 = JuegoLaberinto(MAPA, "z")
    resultado3 = aestrella(problema3, busqueda_en_grafo=True)
#     resultado = voraz(problema, busqueda_en_grafo=True)
    camino1Count3 = 0

    camino3 = [x[1] for x in resultado3.camino()]




    for y in range(len(MAPA)):
        for x in range(len(MAPA[y])):
            #estado inicial o
            if (x, y) == problema.estado_inicial:
                print("o", end='')

            #estado objetivo x, w , z
            elif (x, y) == problema.estado_objetivo:
                print("x", end='')
```

```
            elif (x, y) == problema2.estado_objetivo:
                print("w", end='')
            elif (x, y) == problema3.estado_objetivo:
                print("z", end='')

            #caminos
            #primer camino
            elif (x, y) in camino1:
                print("·", end='')
                camino1Count += 1
            #segundo camino
            elif (x, y) in camino2:
                print(".", end='')
                camino1Count2 += 1
            #tercer camino

            elif (x, y) in camino3:
                print(".", end='')
                camino1Count3 += 1
            else:
                print(MAPA[y][x], end='')
        print()

    ####$$$
#    for n in range(len(MAPA)):
#        for m in range(len(MAPA[n])):
#            if (m,n) == problema.estado_inicial:
#                print("z", end='')
#            elif (m, n) == problema.estado_objetivo:
#                print("x", end='')

#            elif (m,n) == problema2.estado_objetivo:
#                print("w", end='')
#            elif (m, n) in camino1:
#                print("·", end='')
#                camino1Count += 1
#            elif (m, n) in camino2:
#                print(".", end='')
#                camino1Count2 += 1
#            else:
#                print(MAPA[n][m], end='')
#        print()

    camino1=len(camino1)
    camino2=len(camino2)
    camino3=len(camino3)
    print(camino1)
    print(camino2)
    print(camino3)


    if(camino1>camino2 ):
        print("el camino mas corto es el de w: ",camino2," pasos")
    elif(camino2 > camino1 ) :
        print("el camino mas corto es el de x: ",camino1," pasos")
    else:

        print("son iguales")
###$$$
    if(camino1<camino2 and camino3<camino2):
        print("el camino mas largo es el de w: ",camino2," pasos")
```

```
    elif(camino2 < camino1 ) :
        print("el camino mas largo es el de x: ",camino1," pasos")




if __name__ == "__main__":
    main()

# %%
```

```
##############################
# o····   #             # x #
#.####.·  ########   #####·  #
# ..   .· #             #·  #
#   .###.·#  ####   ######·  #
#    . # .··········#    ·   #
#  #  .   w  #  #   ·····    #
#######.  ####    #    #######
#z.....  #      #    #       #
##############################
28
9
14
el camino mas corto es el de w:  9  pasos
el camino mas largo es el de x:  28  pasos
```

In [ ]:

```
1
```

In [ ]:

```
1
```