UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

TAREA 3: Algoritmo básico de búsqueda.

Sergio González Velázquez
Andrés Gutiérrez Cepeda
David Carneros Prado

Asignatura: Sistemas Inteligentes

Grupo de Trabajo: BC1 - 01

Repositorio de código: https://github.com/DavidCarneros/BC1-01

Titulación: Grado en Ingeniería Informática

Fecha: 10 de noviembre de 2018

➢ 1. Objetivo de la tarea

El objetivo de la tarea 3 consiste en definir el algoritmo básico de búsqueda necesario para implementar las siguientes estrategias:

-Anchura

-Profundidad: Simple, acotada e iterativa

-Costo Uniforme

A su vez se incluye la poda de los nodos cuyos estados ya estén en otros nodos generados y que tengan coste mayor.

➢ 2. Herramientas utilizadas

- En la clase Estado se hace uso del módulo *hashlib,* perteneciente a la librería estándar y que permite realizar cifrados en MD5.

  https://docs.python.org/3.7/library/hashlib.html

- En la clase Problema se hace uso del módulo *json,* para decodificar una cadena en formado JSON.

  https://docs.python.org/3.7/library/json.html

- En la clase AlgoritmoBusqueda se hace uso del módulo *stack,* para retornar la solución de manera ordenada.

  https://docs.python.org/3.1/tutorial/datastructures.html

➢ 3. Implementación

▪ 3.1 Clase Espacio de Estados

La clase EspacioDeEstados tiene como entrada un fichero .graphml que utilizaremos para crear un objeto de la clase Grafo que implementamos en la Tarea1. Esto nos permitirá representar el espacio de estados como un grafo y hacer uso de algunas funciones propias de grafos.

-***esta(estado).*** *No sufre ningún cambio con respecto a la versión anteriormente presentada.*

-***sucesores(estado).*** Es una función que devuelve una lista de sucesores para el estado determinado como parámetro de la función. Para ello partimos de una lista donde se encuentran todos los nodos del grafo que son adyacentes al estado actual y otra lista con todos los nodos que quedan por recorrer partiendo del estado actual. A partir de estas dos listas generaremos la lista de sucesores de ese estado.

Iremos recorriendo la lista de nodos adyacentes para generar los estados sucesores a dicho estado, de aquellos estados sucesores meteremos a la lista de estados por visitar a aquellos estados adyacentes que no se encuentren en la lista de nodos por recorrer del estado anterior.

```
def sucesores(self,estado):

    listaSucesores = []

    listaDeAdyacentes = self.__grafo.adyacentesNodo(estado.getNode())

    listaNodosPorRecorrer = estado.getListNodes()

    for ady in listaDeAdyacentes:
        nombreCalle = ady[2]

        listaNodosNueva = []

        for i in listaNodosPorRecorrer:
            if not i == ady[1]:
                listaNodosNueva.append(i)

        estadoNuevo = Estado(ady[1],sorted(listaNodosNueva))
        coste = ady[3]
        accM= '{} --> {} ({}) coste: {}'.format(estado.getNode(),ady[1],nombreCalle,ady[3])
        listaSucesores.append([accM,estadoNuevo,coste])

    return listaSucesores
```

- **3.2 AlgoritmoBusqueda**

La clase AlgoritmoBusqueda tiene como entrada un fichero .json que utilizaremos para la obtención y creación del fichero solucion.txt que contendrá los datos referentes a la solución encontrada. Esta clase contendrá el algoritmo de búsqueda junto con la poda.

-***escribirSolucion(solucion,n_final,estrategia).*** *Este método escribe en un fichero de texto la solución encontrada al problema del fichero problema.json. En dicho fichero se colocara:*

- *La estrategia utilizada.*
- *Profundidad de la solución.*
- *El numero de nodos generados.*
- *El costo del camino.*
- *Y la secuencia de nodos por los que se pasa hasta encontrar la solución.*

```
def escribirSolucion(solucion,n_final,estrategia):

    with open(archivoSolucion,'w') as f:
        f.write("La solucion es: \nEstrategia: {}\n".format(estrategia))
        f.write("Total de nodos generados: {}\n".format(nodosGenerados))
        f.write("Costo: {}\n".format(n_final.getCosto()))
        f.write("Profundidad: {}\n\n\n".format(n_final.getProfundidad()+1))

        for nodo in solucion:
            f.write(nodo.getAccion())
            f.write("\nEstoy en {} y tengo que visitar:{} \n\n".format(nodo.getEstado().getNode(),
                nodo.getEstado().getListNodes()))
        f.close()
```

-***crea_nodo(padre,estado,costo,estrategia,accion).*** Es una función auxiliar utilizada por el método busqueda_acotada que permite la creación de un objeto del tipo nodo que corresponde con el nodo inicial.

```
def crea_nodo(padre, estado, costo, estrategia,accion):
    nodo=Nodo.Nodo(padre,estado,costo,estrategia,accion)
    return nodo
```

-***crearListaNodosArbol(Ls,padre,prof_Max,estrategia).*** Para cada acción de la lista de acciones disponibles para el nodo padre se generan todos los nodos descendientes del nodo padre, como consecuencia de aplicar las acciones de esa lista. Como añadido aremos uso de una variable global nodosGenerados que utilizaremos para llevar la cuenta de los nodos que se generan en el proceso de búsqueda de la solución.

```
def crea_nodo(padre, estado, costo, estrategia,accion):
    nodo=Nodo.Nodo(padre,estado,costo,estrategia,accion)
    return nodo
```

-***crearSolucion(nodo).*** Se obtienen todos los antecesores del nodo objetivo hasta llegar al nodo inicial. Todos estos nodos son introducidos en una pila FIFO para que al sacarlos de la pila e introducirlos en la lista solución estos se retornen de manera ordenada.

```
def crearSolucion(nodo):
    pila = stack.Stack()

    nodoArbol = nodo

    while (not nodoArbol.getPadre()==None):
        #accion = nodoArbol.getAccion()
        pila.push(nodoArbol)
        nodoArbol = nodoArbol.getPadre()

    pila.push(nodoArbol)

    solucion = []

    while not pila.isEmpty():
        solucion.append(pila.pop())

    return solucion
```

-***poda(diccionarioPoda,Ln).*** Método que implementa la poda de los nodos cuyos estados ya están en otros nodos generados y tienen un f mayor a ellos. Para la ejecución de la poda hacemos uso de un diccionario, para cada nodo de la lista de nodos sucesores comprobamos si el estado ya existe en el diccionario, de encontrarse se comprueba si el valor de f del nuevo nodo es menor que el valor del f en los nodos ya generados anteriormente para ese estado, si eso es así no podaremos el nodo y actualizaremos el valor de f. Si por el contrario el estado del nodo no se encuentra en el diccionario significa que no había sido visitado y se actualiza el diccionario para futuras podas.

```
def poda(diccionarioPoda, Ln):

    ListaNodosPoda = []

    for nodo in Ln:
        estadoNodo = nodo.getEstado()
        idEstado = estadoNodo.getId()
        F_estado = nodo.getF()

        if( idEstado in diccionarioPoda):
            if F_estado < int(diccionarioPoda.get(idEstado)):
                diccionarioPoda.update({idEstado:F_estado})
                ListaNodosPoda.append(nodo)
        else:
            diccionarioPoda.update({idEstado:F_estado})
            ListaNodosPoda.append(nodo)

    return ListaNodosPoda,diccionarioPoda
```

-***busqueda_acotada(prob,estrategia,prof_Max).*** Es el algoritmo de búsqueda que nos permite la generación del árbol de búsqueda a partir del nodo inicial y la función sucesor, la raíz del árbol de búsqueda viene definida por el problema.

```
def busqueda_acotada (prob, estrategia, prof_Max):

    diccionarioPoda = {}

    frontera = Frontera.Frontera()
    estado_inicial = prob.getEstadoInicial()
    n_inicial = crea_nodo (None, estado_inicial, 0, estrategia, '0.0 0 0.0')
    frontera.insertar(n_inicial)
    solucion = None

    while ((solucion == None) and (not(frontera.esVacia()))):
        n_actual=frontera.elimina()
        estadoActual = n_actual.getEstado()
        if prob.esObjetivo(estadoActual):
            solucion=True
        else:
            Ls=prob.getEspacioEstados().sucesores(n_actual.getEstado())
            Ln=crearListaNodosArbol(Ls, n_actual, prof_Max, estrategia)
            ListaNodosPoda,diccionarioPoda = poda(diccionarioPoda, Ln)
            frontera.insertarLista(ListaNodosPoda)

    if (solucion==None):
        return None,None
    else:
        return crearSolucion(n_actual),n_actual
```

-***busqueda(prob,estrategia,prof_Max,inc_Prof).*** Es el algoritmo básico de búsqueda que nos permitirá implementar las distintas estrategias:

- Anchura.
- Costo Uniforme
- Profundidad (Simple, acotada e iterativa).

```
def Busqueda(prob, estrategia, prof_Max, inc_Prof):

    prof_Actual = inc_Prof
    solucion = None

    while ((solucion == None) and (prof_Actual<= prof_Max)):
        global nodosGenerados
        nodosGenerados=0
        solucion,n_final = busqueda_acotada (prob, estrategia, prof_Actual)
        prof_Actual = prof_Actual + inc_Prof

    return solucion,n_final
```

-***main.*** Se obtiene por linea de argumentos la profundidad máxima, el incremento y la estrategia, se ejecuta el algoritmo de búsqueda y si esta da una solución se escribe en el fichero de texto.

```python
if __name__=="__main__":

    #Comprobacion de los argumentos de entrada
    if not len(sys.argv)==4:
        print('Uso: ./AlgoritmoBusqueda <profMaxima> <incremento> <estrategia>')
        exit()
    elif (sys.argv[3].lower()) not in estrategiasBusqueda:
        print('Error. Estrategia de búsqueda desconocida')
        print(estrategiasBusqueda)
        exit()


    Prof_Max = int (sys.argv[1])
    Inc_Prof= int(sys.argv[2])
    Estrategia=(sys.argv[3]).lower()

    Prob = Problema.Problema (archivoJSON)

    solucion,n_final=Busqueda(Prob, Estrategia, Prof_Max, Inc_Prof)

    if(solucion is not None):
        escribirSolucion(solucion,n_final,Estrategia) #Se escribe la solucion en un archivo .txt
        print("Algoritmo finalizado...")
    else:
        print("Sin solucion...")
```

➢ 4. Ejecución

Para la correcta ejecución del programa se deberá de ejecutar la siguiente linea en la consola de comandos localizada en el directorio del código fuente del programa:

- ./AlgoritmoBusqueda <profMaxima> <incremento> <estrategia>

Donde profMaxima hace referencia a la profundidad máxima, el incremento que hace referencia al incremento de profundidad por iteración y la estrategia a seguir en el algoritmo de búsqueda.