Grupo B1-8

- Mohamed Essalhi Ahamyan.
- Carlos Fernández-Aparicio Margotón.
- Mario Puebla Ortiz.

Repositorio: https://github.com/SI-CR/lab-b1-8.git

Índice

1. Descripción del problema

En el laboratorio de Sistemas Inteligentes del curso 22/23, se trata de crear un algoritmo de búsqueda utilizando las estrategias vistas en teoría (anchura, profundidad, coste uniforme, voraz y A*). Partiendo de una localización inicial, deberemos visitar una serie de puntos de interés.

2. Resolución del problema

La resolución del problema se ha realizado en incrementos durante las diferentes tareas planteadas en clase. A continuación, tendremos el trabajo realizado durante dichas tareas.

a. Tarea 1

La tarea uno consiste en la lectura de un archivo XML en el que sus datos corresponden a un grafo, cuyo objetivo es simbolizar a partir de un grafo una ciudad.

Hemos utilizado la librería SAX, con los métodos startElement, characters y endElement para poder leer el grafo. Se han creado como clases auxiliares:

- **Nodos:** almacenaremos los datos que corresponden con: id, id_osm, longitud, latitud y una lista de adyacencia del propio nodo.
- **Aristas:** almacenaremos de forma temporal el origen, destino, id y longitud de la arista.
- **Claves:** indica los id de los data para encontrar los datos que buscamos.
- **Dato:** clase auxiliar en la que guardaremos su key y el dato que contiene.

Como resultado de esta tarea tendremos el archivo CargarDatos.py con los siguientes elementos:

- Lista de los nodos del grafo. Cada uno de los nodos tendrá ordenada su lista de adyacencia.
- Diccionario de aristas que constara de una tupla formada por:
    - Tupla que indica: (nodo origen, nodo destino).
    - Coste de la arista

Último Commit:

- HASH: 61e0c58f23e3bccd4268276e0c15622acd4c8c37
- FECHA: 17/10/2022
- HORA: 20:00

b. Tarea 2

En esta segunda tarea debemos crear los elementos necesarios para poder definir el problema. Para definir un problema necesitamos:

- Espacio de estados
- Estado inicial
- Función objetivo

Hemos creado las siguientes clases para realizar el desempeño de esta tarea:

- **Estado**: Compuesta por:
    - **IDEstado**: Variable que identifica de manera única cada estado cuyo valor se obtiene de la función crearID
    - **Localización**: Variable que identifica el nodo dentro de la ciudad donde nos situemos
    - **NodosPorVisitar**: Variable que almacena una lista de nodos definida por el programa que debemos visitar para llegar a la solución
    - **CrearID:** función que devuelve una cadena criptográfica MD5, a partir de la concatenación de las variables Localización y NodosPorVisitar
    - **FuncionSucesor**: función que devuelve una lista la clase Sucesor de todos los sucesores válidos del estado
- **Sucesores**: Compuesta por:
    - **estadoAdy**: identificador del nuevo estado
    - **accion**: tupla compuesta por el id del nodo de origen y el de destino
    - **costo**: coste de la acción para llegar al nuevo estado

Además, hemos creado las siguientes funciones:

- **ComprobarEstadoInicial**: comprueba que el estado inicial sea correcto. Para ello si tenemos un estado cuya localización se encuentra dentro de los NodosPorVisitar, lo elimina de este último.
  Ejemplo: ((20, [20,50]), para corregirlo → (20[50]))
- **FuncionObjetivo**: comprueba si hemos llegado a la solución, es decir, si la longitud de la lista de NodosPorVisitar es vacía.

Último Commit:

- HASH: b6be7ce41bcc713d1c36be0347ea68a2f5fc2db6
- FECHA: 24/11/2022
- HORA: 20:00

c. Tarea 3

En esta tarea debemos implementar los siguientes algoritmos de búsqueda:

- Anchura
- En Profundidad
- Coste Uniforme

Para poder construir el árbol de búsqueda hemos creado las siguientes clases

- **NodoBusqueda**. Compuesta por:
    - **ID**: identificador del nodo de búsqueda
    - **nodoPadre**: nodo padre del nodo de búsqueda
    - **Profundidad**: profundidad a la que se encuentra el nodo
    - **Valor**: valor del nodo que dependerá de la estrategia
    - **Heuristica**: heurística que utilizaremos más adelante con otras estrategias. En esta tarea su valor será “X”
    - **Coste**: coste del nodo de búsqueda
    - **Estado**: estado del nodo de búsqueda
    - **__lt__**: función implementada para poder comparar dos nodos por su valor y en caso de empate por el id, necesario para implementar una cola de prioridad.
    - **Camino**: función para dado un nodo devolver una lista con los nodos hasta llegar al nodo inicial
    - **Print**: función para imprimir el nodo de búsqueda con el formato establecido en el enunciado.
- **Visitados**: Compuesta por:
    - **Visitados**: variable de tipo set en la que almacenaremos los nodos que hayamos visitado
    - **Inserción**: función para insertar un estado en caso de que no exista en el set visitados
    - **Pertenece**: función para comprobar si un estado existe en el set visitados

Además, hemos creado las siguientes funciones:

- **ValorEstrategia**: nos indicará cual es el valor de un nodo, dependiendo de la estrategia
- **AlgoritmoBusqueda**: es la función principal que se nos pedía implementar en esta tarea. Para ello debemos pasarle como argumentos la estrategia, la profundidad máxima y el estado inicial del que partimos. Si encuentra solución para el problema imprimirá el camino, en caso contrario nos dirá que no hay solución.

Último Commit:

- HASH: a2b256e0bdd3b0672c2011b6cd41ab6ecfd05728
- FECHA: 25/10/2022
- HORA: 15:00

d. Tarea 4

Para esta tarea se nos ha pedido implementar los algoritmos de búsqueda voraz y A*, para ello, aunque no ha hecho crear nuevas clases, sí que hemos tenido que actualizar métodos ya definidos y crear nuevos métodos.

- **HeuristicaEuclidea**: función para calcular la heurística euclídea, a partir de la distancia euclídea. Elegiremos el mínimo valor entre las distancias entre los nodos por visitar (valor que se mantendrá constante, dado por la función CalcularM1) y la distancia entre la localización en la que nos encontremos con el resto de los nodos por visitar.
- **CalcularM1**: función para calcular las distancias entre los nodos por visitar, que será auxiliar para calcular la heurística.
- **HeurísticaArco**: función para calcular la heurística de arco, a partir de la mínima longitud de arista del grafo (valor que se mantiene constante y que hemos añadido al DiccionarioAristas la clave “min_long” para almacenar dicha longitud) multiplicado por la longitud de la lista de NodosPorVisitar
- **ValorEstrategia**: función que ya teníamos y a la que se le ha actualizado los casos en los que tengamos como estrategia voraz y A*
- **AlgoritmoBusqueda**: función que ya teníamos y a la que hemos introducido los nuevos elementos necesarios para que podamos realizar las estrategias voraz y A*

Hemos realizado finalmente que puedas obtener la salida tanto a través de la salida estándar como en un fichero llamado “Solución.txt” con el formato de los ejemplos de las soluciones del grafo de Ciudad Real.

Para modificar los datos del enunciado debemos dirigirnos al final del archivo ArbolBusqueda.py y modificaremos los datos ProfMax, e0 y la estrategia. Por defecto, se utilizará la HeuristicaEuclidea

Último Commit:

- HASH: 12381acc8a1ba2c27769e1bc87fbf69f571b3f7a
- FECHA: 07/12/2022
- HORA: 17:55

3. Manual de usuario

La ejecución del programa modificará o creará un fichero llamado “Resultados.txt” en el que se expondrán las soluciones, también se imprimirán por la línea de salida del terminal.

Las modificaciones del software que se le permiten al usuario para que el programa funcione con normalidad son las siguientes:

- **Nombre del fichero XML:** Debemos dirigirnos al fichero CargarDatos.py y pegar la ruta del fichero XML en la instrucción 207

```
204    parser = xml.sax.make_parser()
205    parser.setContentHandler(handler)
206    #parser.parse('CR_Capital.graphML.xml')
207    parser.parse('nuevo.graphxml.xml')
208
209    for i in ListaNodos:
210        i.ListaAdyacencia.sort()
211
```

- **Estado inicial:** En el fichero ArbolBusqueda.py, escribiremos en la línea 211 el estado inicial
- **Profundidad Máxima:** En el fichero ArbolBusqueda.py, estableceremos la profundidad es la línea 220
- **Heurística:** En el fichero ArbolBusqueda.py, elegiremos la heurística dejándola sin comentar

```
216    #HEURISTICAS:
217    Heuristica = "Arco"
218    #Heuristica = "Euclidea"
219
220    ProfMax = 400
221    e0 = CargarProblema.Estado(337,[249,431,1076])
222    imprimirSolucion(e0, ProfMax)
223
224
```

- **Estrategias:** En el fichero ArbolBusqueda.py, en el método imprimirSolucion, dejaremos sin comentar las estrategias que queramos elegir

```
200        AlgoritmoBusqueda(estado, "A", ProfMax)
201        AlgoritmoBusqueda(estado, "Uniform", ProfMax)
202        AlgoritmoBusqueda(estado, "Greedy", ProfMax)
203        AlgoritmoBusqueda(estado, "Breadth", ProfMax)
204        #AlgoritmoBusqueda(estado, "Depth", ProfMax)
```

4. Opinión personal y valoración

A continuación, se mostrarán las opiniones personales de los integrantes del grupo y su valoración en cuanto al reparto de trabajo

a. Mohamed

Este laboratorio me ha parecido una actividad interesante, ya que hemos podido aprender como se ejecutan los algoritmos vistos en teoría aplicados en la vida real, lo que me hecho ver la utilidad de los mismos y como se construyen rutas, tal y como lo puede hacer Google Maps.

El seguimiento de las tareas y como se han planificado es el adecuado ya que la carga de trabajo no ha sido demasiado grande como hemos podido tener en otras asignaturas.

Además, dentro del grupo nos hemos coordinado muy bien y no hemos tenido ningún problema a la hora de repartir trabajo o ayudarse unos a otros debido al compañerismo que hemos desarrollado.

En general estoy bastante satisfecho con este laboratorio, le pondría una nota de 9/10.

Mi aportación en el trabajo ha sido: (C) Como todo el mundo

b. Carlos

Mi opinión sobre el laboratorio de Sistemas Inteligentes es variada, ya que como tal me ha gustado el seguimiento que ha llevado el profesor de nuestro trabajo, preguntando en cada sesión sobre los distintos avances que hemos hecho, haciendo que fuésemos al día con el trabajo gracias al seguimiento.

En cuanto a el problema hemos encontrado distintas dificultades que hemos ido solucionando y por mi parte este trabajo me ha costado, ya que no se me dan bien los grafos, pero por lo demás yo creo que bien.

La metodología seguida ha sido dividirnos el trabajo lo más equilibrado posible, aunque a veces no ha sido posible y unos han cubierto a otros, pero al no haber ningún problema interno en el grupo con ese tema, podemos decir que la metodología usada ha sido buena.

Yo creo que mi aportación en el trabajo ha sido: (C) Como todo el mundo

c. Mario

Sobre el laboratorio, me ha parecido interesante como hemos aplicado la teoría que vemos en clase con un código real implementado con nosotros, sobre todo porque se ha aplicado a un caso como es el grafo de Ciudad Real, para así poder observar que tiene una aplicación real lo que venimos trabajando.

Sobre el problema, como he comentado antes, me parece correcto que trabajemos sobre datos reales, para ver que tiene un propósito verdadero lo que vemos en clase y las horas que dedicamos a su realización y plasmación en código.

Sobre la metodología seguida, también me parece la más acertada en el tiempo que llevo en la carrera, que he podido experimentar varias. Digo esto porque en mi opinión, que se dedique una sesión para explicar la siguiente tarea/hito y se deje una o dos semanas de tiempo para su entrega, es la forma idónea para que el alumno pueda plasmar lo visto en la sesión de explicación, y en ese tiempo que tiene hasta la entrega sea capaz de darle vueltas, preguntar dudas y alcanzar la solución con un entendimiento de lo que está haciendo.

Reparto del trabajo -> (C) Como todo el mundo