

PROJECTE 1

Navegador

SO: Sessió d'Orientació

Projecte 1 - Part 1

Intel·ligència Artificial

2023-2024

Universitat Autònoma de Barcelona

1. Introducció

En aquesta pràctica resoldrem un problema simple de navegació sobre un mapa, on donades unes coordenades d'**inici** i de **final**, trobarem la millor ruta entre els dos punts.

Per a fer-ho utilitzarem quatre mètodes de cerca explicats a teoria:

1. Cerca en profunditat (Depth First Search)
2. Cerca en amplada (Breadth First Search)
3. Cerca de cost uniforme (Uniform Cost Search)
4. Cerca A* (A-Star)

En aquesta **1a Part de la pràctica** ens centrarem en els mètodes de

Cerca No Informada i No Òptima

ATENCIÓ! Guardarem els camins en **ordre invers** a com ho hem fet a la teoria.

- Primer element és l'**arrel** (estat inicial)
- Últim element és el node **fulla** (estat actual del camí).

2. Fitxers necessaris

Per a realitzar la pràctica haureu de descarregar les següents carpetes:

1. **CityInformation:** Arxius que representen el mapa de la ciutat.

- A. **InfoVelocity.txt:** Conté la informació sobre la **velocitat** a la qual viatja cada metro.
- B. **Stations.txt:** Conté els **IDs** de cada estació, **nom**, número de **línia** i les **coordenades** on es troba.
- C. **Time.txt:** Taula on podem veure el **temps** que es triga per anar d'una estació a una altra. No hi ha connexió entre dues estacions si el valor de la taula és zero.
- D. **Lyon_city.jpg:** Imatge del **mapa** de la ciutat.

Vel. line 1 : 10
Vel. line 2 : 14
Vel. line 3 : 45
Vel. line 4 : 3

InfoVelocity.txt

1	MASSENA 1	67	79		
2	CHARPENNES	1	140	56	
3	REPUBLIQUE	1	167	64	
4	LE TONKIN	2	140	27	
5	CHARPENNES	2	140	56	
6	COLLEGE BELLECOMBE	2	140	115	
7	THIERS-LAFAYETTE	2	140	157	
8	PART-DIEU	2	108	206	
9	PARTDIEU SERVIENT	2	82	217	
10	CHARPENNES	3	140	56	
11	BROTTEAUX	3	108	134	
12	PART-DIEU	3	108	206	
13	PART-DIEU	4	108	206	
14	DAUPHINE LACASSAGNE	4	152	230	

Stations.txt

```
0.0000 9.05376 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
9.05376 0.00000 4.21603 0.00000 20.00000 0.00000 0.00000 0.00000
0.00000 4.21603 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 5.42857 0.00000 0.00000 0.00000
0.00000 20.00000 0.00000 0.00000 5.42857 0.00000 7.14286 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 4.21429 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 6.03739
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 15.00000 0.00000 0.00000 0.00000 8.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000 0.00000
```


Time.txt



Lyon_city.jpg

2. Fitxers necessaris

Per a realitzar la pràctica haureu de descarregar les següents carpetes:

2.  **Code:** Fitxers de Python amb funcions per a la pràctica.

A. **utils.py**: Conté una sèrie de **funcions** que us poden ser útils per entendre que fa el que programeu.

B. **SubwayMap.py**: Conté les dues classes principals amb les quals treballarem:

- Map (Conté tota la informació sobre la **ciutat**).
- Path (Classe que guarda la informació sobre una **ruta** o successió de parades).

C. **TestCases.py**: Arxiu per comprovar si les funcions que programeu donen el **resultat esperat**.

D. SearchAlgorithm.py: **Arxiu on haureu de programar tota la pràctica.**

Hem proporcionat dues versions de mapes:

1. **Lyon_bigCity**: Tota la ciutat
2. **Lyon_smallCity**: Simplificació del mapa

3. Preparació

Abans de començar a programar és molt recomanable entendre les **dues classes** amb les quals treballarem:

Map i Path.

Map és una classe per mantenir totes les dades relacionades amb les estacions i les seves connexions.

class **Map**:

Atributs

self.stations = {} *Diccionari de diccionaris amb el format {station_id: {"name": name_value, "line": line_value, ...}}*
self.connections = {} *Diccionari de diccionaris que conté tot la informació de les connexions.*
self.velocity = {} *Diccionari amb la velocitat de cada transport*

Funcions

add_station(self, id, name, line, x, y) *Afegeix una estació al diccionari self.stations*
add_connection(self, connections) *Afegeix una connexió al diccionari self.connections*
combine_dicts(self)
add_velocity(self, velocity) *Afegeix una velocitat al diccionari self.velocity*

3. Preparació

Abans de començar a programar és molt recomanable entendre les **dues classes** amb les quals treballarem:

Map i Path.

Path és una classe per mantenir la informació de la ruta des de l'estació d'inici fins a l'estació ampliada.

class **Path**:

Atributs

```
self.route = [] Llista amb la ruta  
self.head = Station() Node inicial  
self.penultime = Station() Penúltim node  
self.last = Station() Útim node  
self.g = int Cost real  
self.h = int Cost heurístic  
self.f = int Combinació dels dos costos
```

Funcions

```
__eq__(self, other) Compara dues rutes per veure si son iguals  
update_h(self, h) Actualitza el cost heurístic  
update_g(self, g) Actualitza el cost real  
update_f(self) Actualitza la combinació dels dos costos  
add_route(self, children) Afegir una nova estació a la ruta
```

3. Preparació

Per confirmar que heu entès aquestes classes i abans de seguir, hauríeu de ser capaços de:

1. Accedir a tota la informació d'una parada en concret (Número de línia, coordenades i velocitat)
2. Accedir a les connexions d'una parada de metro i al seu valor.
3. Entendre i poder crear un Path.

4. Què s'ha de programar?

Funcions que heu de programar per aquesta primera part d'aquesta pràctica:

4.1 Funcions prèvies

- Expand
- remove_cycles

4.2 Algorisme de Cerca en Profunditat

- Insert_depth_first_search
- Depth_first_search

4.3 Algorisme de Cerca en Amplada

- Insert_breadth_first_search
- Breadth_first_search

4.4 Una funció final

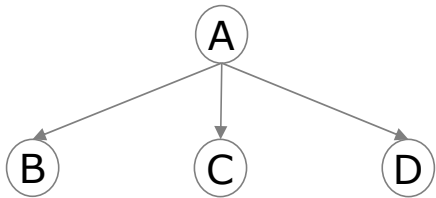
- distance_to_stations

4.1 Funcions prèvies

Expand

Sessió de Teoria

A la classe de teoria vam veure que la funció expandir retorna la llista dels camins resultants d'expandir el node de més profunditat del camí.



Si fem `Expand([A])` retorna `[[B,A], [C,A], [D,A]]`

Sessió de Pràctica

En aquesta pràctica:

- Els camins es representen en **ordre invers** al que hem fet a la teoria.
- Els **nodes** representen estacions de metro.
- Les **branques** son les connexions entre parades.

La funció expand:

- **Input:** Path pare i el Map.
- **Retorna:** Una llista de Paths.

Exemple:

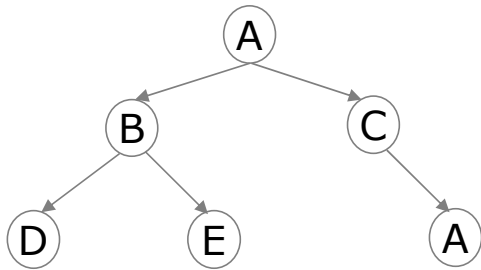
- Crida funció: `expand([14,13,8,12],MAP)`
- Retorna:
`[[14,13,8,12,8], [14,13,8,12,11], [14,13,8,12,13]]`

4.1 Funcions prèvies

remove_cycles

Sessió de Teoria

A la classe de teoria vam veure que la funció `remove_cycles` elimina els camins expandits que tenen cicles.



Si fem `EliminarCicles([[D,B,A], [A,C,A], [E,B,A]])`
Retorna `[[D,B,A], [E,B,A]]`

Sessió de Pràctica

En aquesta pràctica:

- La funció elimina de la llista de camins el conjunt de camins que inclouen alguns cicles en el seu recorregut.

La funció expand:

- **Input:** Llista Path.
- **Retorna:** Una llista de Path sense cicles.

Exemple:

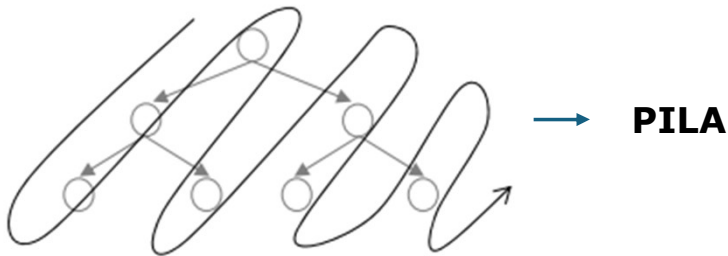
- Crida funció:
`remove_cycles([[14,8,12,8],[14,8,12,11], [14,8,12,14]])`
- Retorna: `[[14,8,12,11]]`

4.2 Algorisme de Cerca en Profunditat

Insert_depth_first_search:

Sessió de Teoria

A la classe de teoria vam veure que a la cerca en profunditat expandim els camins d'esquerra a dreta. Això es pot implementar inserint els nous camins al principi de la llista



Sessió de Pràctica

La funció insert_depth_first_search:

Input:

- Llista de *Path* fills
- Llista de *Path* que estem explorant

Retorna:

- La unió de les dues llistes de *Path*, posant **els fills a davant** dels camins que estem explorant

4.2 Algorisme de Cerca en Profunditat

Depth_first_search:

Sessió de Teoria

Funció **CERCA_profunditat** (NodeArrel, NodeObjectiu)
1. Llista=[[NodeArrel]];
2. Fins que (Cap(Cap(Llista))=NodeObjectiu o bé (Llista=NIL) fer
 a) C=Cap(Llista);
 b) E=Expandir(C);
 c) E=EliminarCicles(E);
 d) Llista=**Inserir_davant**(E,Cua(Llista));
3. Ffinsque;
4. Si (Llista<>NIL) Retornar(Cap(Llista));
5. Sinó Retornar("No existeix Solucio");
Ffuncio

Sessió de Pràctica

La funció depth_first_search:

Input:

- ID de l'estació origen
- ID de l'estació final
- Mapa

Retorna:

- *Path* que és correspon amb la ruta entre l'estació d'origen i destí

Exemple de crida:

- depth_first_search(2,7,map)

Retorna:

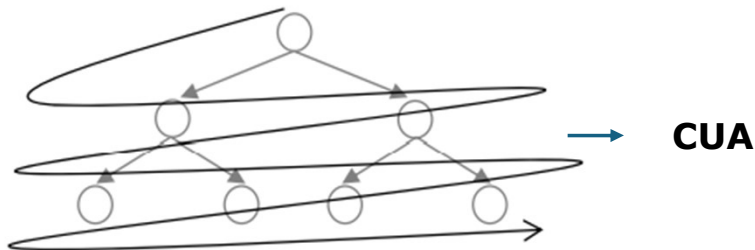
- [2, 5, 6, 7]

4.3 Algorisme de Cerca en Amplada

Insert_breadth_first_search:

Sessió de Teoria

A la classe de teoria vam veure que a la cerca en amplada expandim els camins de dalt cap a baix, nivell a nivell. Això es pot implementar inserint els nous camins al final de la llista.



Sessió de Pràctica

La funció insert_breadth_first_search:

Input:

- Llista de *Path* fills
- Llista de *Path* que estem explorant

Retorna:

- La unió de les dues llistes de *Path*, posant **els fills a darrera** de la llista de camins que estem explorant

4.3 Algorisme de Cerca en Amplada

Breadth_first_search:

Sessió de Teoria

Funció CERCA_amplada (NodeArrel, NodeObjectiu)

1. Llista=[[NodeArrel]];
2. Fins que (Cap(Cap(Llista))=NodeObjectiu o bé (Llista=NIL) fer
 - a) C=Cap(Llista);
 - b) E=Expandir(C);
 - c) E=EliminarCicles(E);
 - d) Llista=**Inserir_darrera(E,Cua(Llista));**
3. Ffinsque;
4. Si (Llista<>NIL) Retornar(Cap(Llista));
5. Sinó Retornar("No existeix Solucio");

Ffuncio

Sessió de Pràctica

La funció breadth_first_search:

Input:

- ID de l'estació origen
- ID de l'estació final
- l'objecte Mapa

Retorna:

- *Path* que és correspon amb la ruta entre l'estació d'origen i destí

Exemple de crida:

- breadth_first_search(13,1,map)

Retorna:

- [13, 12, 11, 10, 2, 1]

4.4 Una funció final

distance_to_stations

Sessió de Pràctica

Aquesta darrera funció ens servirà per calcular la distància entre qualsevol punt del mapa i totes les estacions de metro d'aquest.

La funció distance_to_stations:

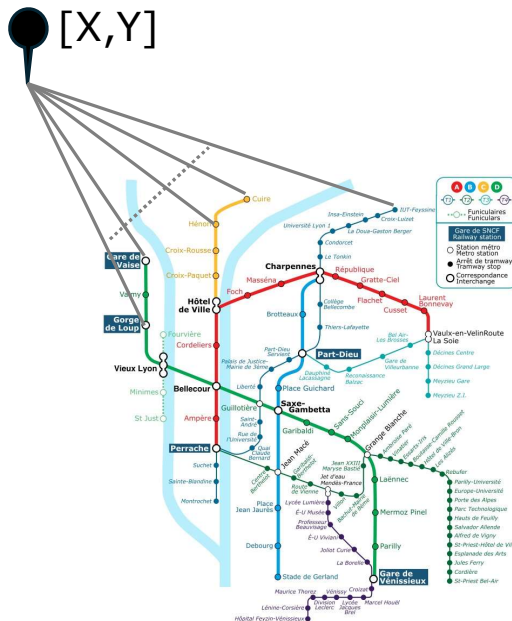
- **Input:**
 - Coord (llista amb dos valors reals; les coordenades).
 - l'objecte Mapa.
- **Retorna:**
 - Un diccionari que conté:
 - Claus: tots els IDs de totes les estacions del mapa,
 - Valors: la distància entre cada estació i el punt de coordenades.

Aquest diccionari està ordenat primer per distància (ascendent), i segon per id de l'estació (ascendent).

Exemple:

- Crida funció: distance_to_station([100,200],MAP)
- Retorna: {8:10.0, 12:10.0, 13:10.0, 9:24.76, 7:58.73, 14:60.03, 11:66.48, 6:93.94, 1:125.42, 2:149.45, 5:149.45, 10:149.45, 3:151.61, 4:177.56}

Goal:



5. Entrega de la Part 1

Per a l'avaluació d'aquesta primera part de la pràctica haureu de pujar al Campus Virtual el vostre fitxer **SearchAlgorithm.py** que ha de contenir el vostre **NIU** a la variable authors i el vostre **grup** a la variable group (a l'inici de l'arxiu).

L'entrega s'ha de fer abans del dia **03/03/2023 a les 23:55**

ATENCIÓ! és important que tingueu en compte els següents punts:

1. La **correcció** del codi es fa de manera **automàtica**, per tant, assegureu-vos de penjar els arxius amb la nomenclatura i format correctes. Si no ho poseu bé la nota serà un 0.
2. El codi està sotmès a **detecció automàtica** de plagis durant la correcció.
3. Qualsevol part del codi que no estigui dins de les funcions de l'arxiu SearchAlgorithm.py **no** podrà ser **avaluada**, per tant, no modifiqueu res fora d'aquest arxiu.
4. Per evitar que el codi entri en bucles infinits hi ha un **límit de temps** per a cada exercici, per tant si les vostres funcions triguen massa les considerarà incorrectes.