
Treball Pràctic Individual

Programació d'un algoritme A*

Marta Matute de Amores
1496672

1 | Introducció

L'algorisme A* és un algoritme de cerca de camins (*path search algorithm*) desenvolupat l'any 1968, i fins la data un dels més òptims. Un dels punts clau d'aquest algoritme és que fa servir una estratègia a mitges entre *breath first search* i *depth first search*: es té en compte tant la distància en línia recta des d'un node concret fins el destí (el que anomenarem funció heurística o h), com la llargada del camí recorregut fins el moment (el que anomenarem g). D'aquesta manera troba, sempre que existeixi, el camí amb la distància total recorreguda menor de tots els camins possibles.

Durant les pràctiques que hem fet durant el curs, hem introduït conceptes i processos necessaris per a poder programar un algoritme A*, com ara les cues o la lectura d'un fitxer de carrers. Fent servir aquests coneixements doncs, hem aconseguit un programa en C que, donats dos nodes i fent servir els fitxers corresponents, troba el camí òptim i indica per pantalla els nodes que s'han seguit per a aconseguir-ho.

2 | Sintaxi d'execució

El nostre programa de l'algoritme A* depèn de dos arxius de dades d'on s'extraurà la informació per a la representació del mapa:

1. **Arxiu de nodes.** És un fitxer que consta de tantes línies com nodes tingui el nostre mapa, on cada línia té exactament tres camps separats per un punt i coma: la identificació del carrer (un nombre enter), la latitud i la longitud (ambdues expressades en graus).
2. **Arxiu de carrers.** Aquest segon fitxer té tantes línies com carrers hi hagi. Cada línia comença amb una cadena de 10 caràcters (el nom del carrer) i un seguit de nodes separats de nou per punts i coma. Aquests nodes són, en l'ordre en que estan escrits, els punts del mapa tal que si els unim amb una línia obtenim el carrer.

El nom d'aquests dos arxius **no es passa per argument**. Si fos necessari canviar-ne algun d'ells, aleshores hauríem d'accedir directament al codi i canviar-ho allà (a la secció Estructura del codi s'especifica com canviar aquests dos arxius).

Per a que l'execució del programa funcioni correctament és necessari tenir els dos fitxers a la mateixa carpeta que el programa que executarem.

A banda dels fitxers, el programa requereix dos arguments. Aquests seran dos variables del tipus `long int` que representaran els `id` dels nodes origen i destí respectivament. En cas que falti algun dels arguments o els nodes introduïts no es trobin entre els nodes als quals hi tenim accés, aleshores s'imprimirà un missatge d'error i s'acabarà l'execució. Si volem tornar a provar dos nodes diferents haurem de tornar a executar.

Per tant, si les instruccions anteriors es compleixen correctament, l'execució del programa es durà a terme fent servir la comanda:

```
./AEstrella idNodeOrigen idNodeDesti
```

3 | Estructura del codi

Per a explicar el codi diferenciarem tres parts: les estructures, les funcions externes i la funció `main`.

3.1 | Estructures

infoaresta: Representa una aresta que uneix dos nodes del mapa. Cada aresta està vinculada a un sol node, i conté la posició dins el vector `nodes` de

l'altre extrem de l'aresta. Dins aquesta estructura hi ha també una cadena de caràcters per a representar el nom del carrer al qual pertany (notem que cada aresta pertany a un i només un carrer), i un nombre real que fa referència a la llargada en metres de l'aresta.

node: Representa un node del mapa. Aquesta estructura conté un nombre enter que representa la *id* del node, dos nombres reals que fan referència a la latitud i la longitud, un altre enter que ens indica amb quants nodes més està connectat, i un vector del tipus **infoaresta** que, evidentment, són les arestes del node.

També té informació que serà necessària per a l'algoritme A^* . Tindrem dues variables de tipus **bool** que indiquen si el node es troba a la cua oberta, o bé a la tancada, o bé a cap de les dues, i per últim cada node tindrà també un apuntador a un altre node, que farem servir per a indicar el node d'on venim.

mapa: Aquesta estructura la farem servir només per a guardar la informació que llegim dels fitxers. Consta d'un vector de nodes i del nombre total de nodes al fitxer.

ElementCua: Representa un element d'una cua. Està format per un apuntador a un node (el propi element), i un apuntador al següent element de la cua.

UnaCua: Aquesta estructura només conté un apuntador al primer element de la cua. Notem que podríem fer servir directament el primer element, però per comoditat i per a que s'entengui millor hem fet servir aquesta estructura anomenada **UnaCua**.

3.2 | Funcions

distancia: Rep dos nodes i torna la distància en línia recta que hi ha entre ells tenint en compte que la terra es rodona. Fa servir la latitud i longitud dels nodes.

posarencua: Rep l'apuntador a una cua i l'apuntador a un node, i afegeix el node a la posició que li correspon segons la seva f . Fem servir una metodologia semblant a la que varem fer servir a la Pràctica 2. D'aquesta manera, la nostra cua oberta sempre tindrà com a primer element aquell amb la menor f , i llavors cada vegada que haguem d'escollir el següent node a expandir, només haurem d'agafar el primer element de la cua oberta. Tot i que el retorn d'aquesta funció

és un valor enter, aquest no el farem servir com a tal. Serveix únicament per a parar la funció cada vegada que haguem afegit el valor. D'aquesta manera ens estalviem moltes instruccions del tipus `if-else`, ja que no seguirà l'execució de la funció un cop es trobi amb un `return`.

buscapunt: Rep un vector de nodes i la identificació d'un node en concret i retorna l'índex en que es troba aquest node dins el vector que hem passat per argument. Si no troba aquest node al vector, aleshores imprimeix un missatge d'error i retorna el valor -1.

treurelprimer: Rep un apuntador a una cua i retorna un apuntador al primer node de la cua. A més a més, modifica la cua de manera que el primer ja no hi és a la cua.

treurelement: Rep un apuntador a una cua i un apuntador a un node. Aquesta funció no retorna cap valor, però modifica la cua de manera que el node passat per argument ja no es troba dins la cua.

llegirFitxers: Rep dos cadenes de caràcters i l'adreça d'una variable del tipus `mapa`. Els dos primers arguments són, en aquest ordre, el nom d'un fitxer de dades on hi ha guardada la informació relativa als nodes i el nom d'un fitxer de dades on hi ha guardada la informació relativa als carrers. En cas que vulguem canviar algun d'aquests dos arxius, aleshores caldrà que canviem els arguments de la funció. Aquesta modificació es faria a la línia 70. La funció no retorna res, sinó que actualitza la variable `mapa` que li hem passat per pantalla: assigna el vector de nodes que hem omplert dins la funció a `mapa.nodes` i el nombre total de nodes trobats a `mapa.nnodes`.

3.3 | Funció main

Dins el `main` tornem a trobar 4 parts diferenciades: comprovació d'errors per a un correcte funcionament del programa, inicialització de les cues, la implementació de l'algoritme A^* , i finalment la impressió per pantalla del resultat obtingut.

La primera part s'assegura que el nombre d'arguments sigui correcte, i que els nodes passats com arguments estiguin dins la llista de nodes que tenim. En cas que alguna d'aquestes condicions no es compleixi, aleshores s'imprimeix un missatge d'error (específic per a cadascun dels casos) i s'atura l'execució. També es comprova que els nodes origen i final no siguin els mateixos, ja que en aquest cas no ens cal trobar cap ruta.

Per a la inicialització de les cues, es crea i s'assigna memòria al primer element, i s'inicialitza la cua oberta (a la qual també li hem assignat memòria prèviament) amb aquest element. També assignem memòria a la cua tancada. En cas que no hi hagi prou espai en memòria per a qualsevol de les cues, o per a l'element, aleshores s'imprimeix un missatge d'error i s'atura l'execució.

Per a la implementació de l'algoritme A^* seguim l'estructura del pseudocodi proporcionat per a fer la pràctica. La única part que difereix lleugerament de la guia donada és el fet que la cua s'ordena conforme s'hi afegeixen nodes. D'aquesta manera ens estalviem recórrer tota la cua cada vegada que vulguem trobar el valor amb la f mínima. Si cada vegada afegíssim els valors al final, aleshores sempre hauríem de mirar tots els elements (ja que el mínim podria trobar-se en qualsevol posició). De la manera en que ho he fet, aleshores només cal recórrer part de la cua quan vulguem afegir un element, i només recorrerem elements fins que trobem el lloc adequat.

Finalment, per a imprimir la solució, ens assegurem primer que haguem trobat aquesta solució. Això ho fem comprovant que l'últim node que hem expandit sigui el node destí. En cas contrari, aleshores imprimim un missatge d'error i aturem l'execució. Quan sí hem trobat la solució, aleshores tornarem enrere a partir de l'últim node expandit i anirem guardant aquests valors dins un vector que hem definit com `camí`. L'últim pas serà simplement imprimir els nodes que hem guardat a `camí`.

4 | Exemple de funcionament

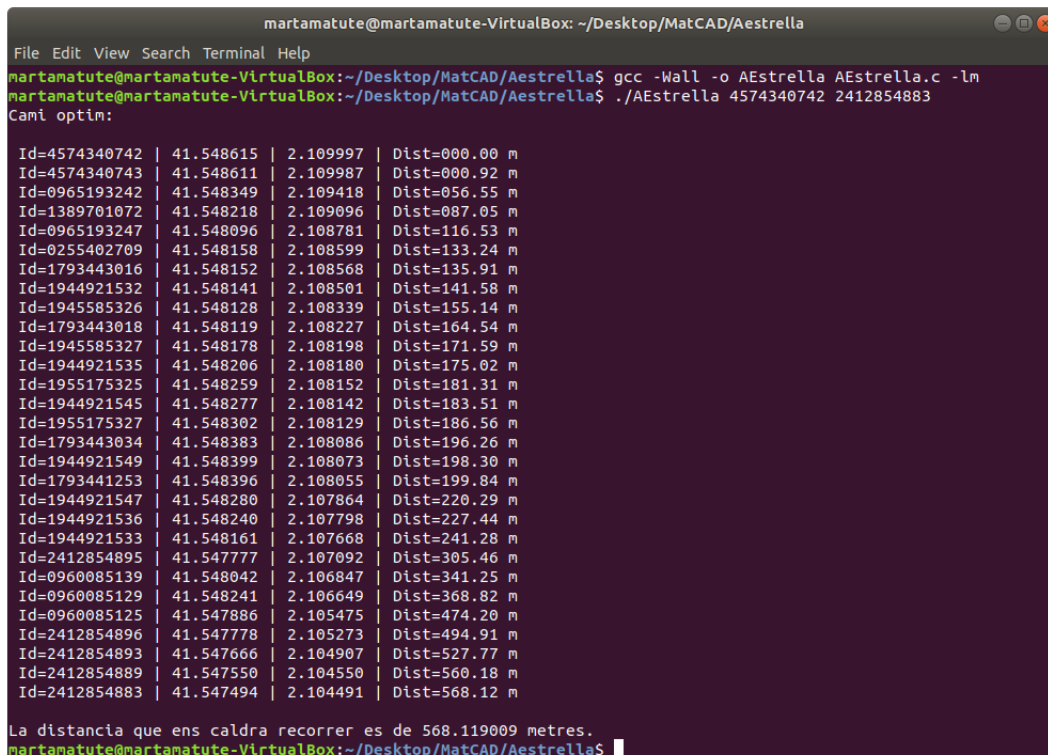
Fem a continuació una simulació de l'execució d'aquest programa. Com ja hem dit, ens assegurem que els fitxers `Nodes.csv` i `Carrers.csv` es troben a la mateixa carpeta que `AEstrella.c`. A continuació compilem el programa fent servir la instrucció

```
gcc -Wall -o AEstrella AEstrella.c -lm
```

En cas que aquesta comanda no retorni cap error, escollim dos nodes com ara 4574340742 i 2412854883, i aleshores escrivim a la línia de comandes

```
./AEstrella 4574340742 2412854883
```

Això ens tornarà a la terminal el resultat:



```
martamatute@martamatute-VirtualBox: ~/Desktop/MatCAD/Aestrella
File Edit View Search Terminal Help
martamatute@martamatute-VirtualBox:~/Desktop/MatCAD/Aestrella$ gcc -Wall -o AEstrella AEstrella.c -lm
martamatute@martamatute-VirtualBox:~/Desktop/MatCAD/Aestrella$ ./AEstrella 4574340742 2412854883
Camí optim:

Id=4574340742 | 41.548615 | 2.109997 | Dist=000.00 m
Id=4574340743 | 41.548611 | 2.109987 | Dist=000.92 m
Id=0965193242 | 41.548349 | 2.109418 | Dist=056.55 m
Id=1389701072 | 41.548218 | 2.109096 | Dist=087.05 m
Id=0965193247 | 41.548096 | 2.108781 | Dist=116.53 m
Id=0255402709 | 41.548158 | 2.108599 | Dist=133.24 m
Id=1793443016 | 41.548152 | 2.108568 | Dist=135.91 m
Id=1944921532 | 41.548141 | 2.108501 | Dist=141.58 m
Id=1945585326 | 41.548128 | 2.108339 | Dist=155.14 m
Id=1793443018 | 41.548119 | 2.108227 | Dist=164.54 m
Id=1945585327 | 41.548178 | 2.108198 | Dist=171.59 m
Id=1944921535 | 41.548206 | 2.108180 | Dist=175.02 m
Id=1955175325 | 41.548259 | 2.108152 | Dist=181.31 m
Id=1944921545 | 41.548277 | 2.108142 | Dist=183.51 m
Id=1955175327 | 41.548302 | 2.108129 | Dist=186.56 m
Id=1793443034 | 41.548383 | 2.108086 | Dist=196.26 m
Id=1944921549 | 41.548399 | 2.108073 | Dist=198.30 m
Id=1793441253 | 41.548396 | 2.108055 | Dist=199.84 m
Id=1944921547 | 41.548280 | 2.107864 | Dist=220.29 m
Id=1944921536 | 41.548240 | 2.107798 | Dist=227.44 m
Id=1944921533 | 41.548161 | 2.107668 | Dist=241.28 m
Id=2412854895 | 41.547777 | 2.107092 | Dist=305.46 m
Id=0960085139 | 41.548042 | 2.106847 | Dist=341.25 m
Id=0960085129 | 41.548241 | 2.106649 | Dist=368.82 m
Id=0960085125 | 41.547886 | 2.105475 | Dist=474.20 m
Id=2412854896 | 41.547778 | 2.105273 | Dist=494.91 m
Id=2412854893 | 41.547666 | 2.104907 | Dist=527.77 m
Id=2412854889 | 41.547550 | 2.104550 | Dist=560.18 m
Id=2412854883 | 41.547494 | 2.104491 | Dist=568.12 m

La distancia que ens caldra recorre es de 568.119009 metres.
martamatute@martamatute-VirtualBox:~/Desktop/MatCAD/Aestrella$
```

Si a més a més volem representar aquest camí al mapa, aleshores podem fer servir el codi en python proporcionat al campus virtual. Haurem de guardar l'output de

l'execució anterior a un arxiu amb extensió `txt` (observem que haurem d'esborrar les dos primeres i les dos últimes línies), guardar-lo amb el nom corresponent al que hi hagi dins el codi en `python` (en el nostre cas `resultat.txt`) i aleshores executar el codi. Això ho podem fer amb les comandes

```
usuari$ ./AEstrella 4574340742 2412854883 > auxiliar
usuari$ head -n -3 auxiliar | tail -n +3 > resultat.txt
usuari$ python3 plot.py
```

i de manera automàtica s'obrirà al nostre navegador predeterminat una finestra amb el camí seguit representat al mapa. En el nostre cas s'ha obert la representació al mapa següent:

