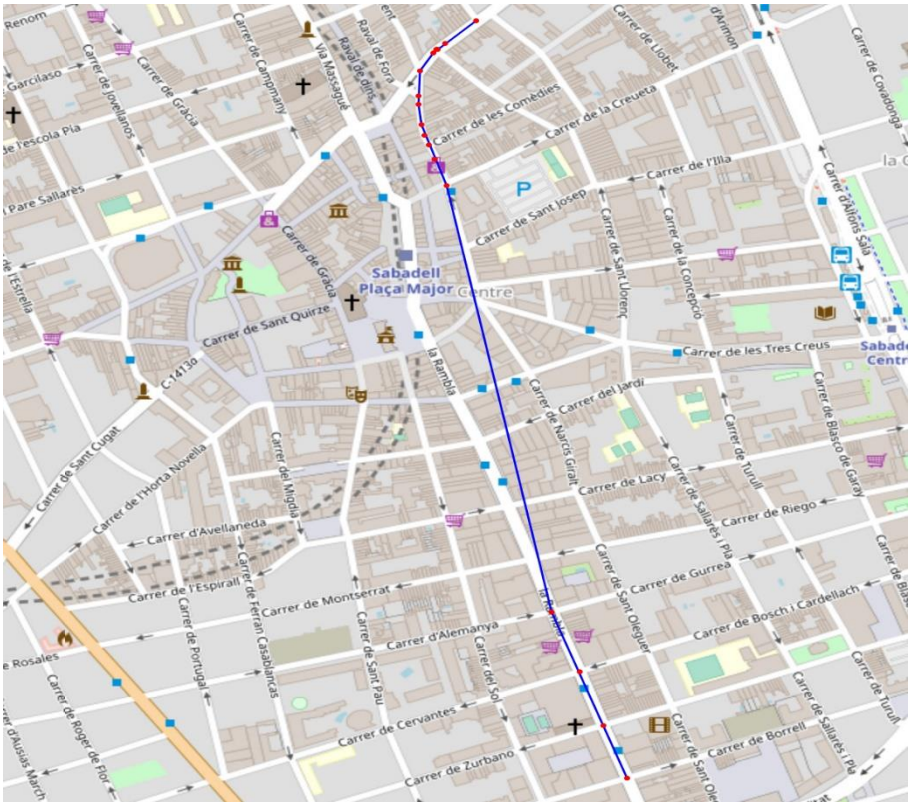
Manel Carrillo Maíllo, 1633426
1r de MatCad.

PROGRAMACIÓ D'UN ALGORITME A*

ALGORISMIA I COMBINATORIA EN GRAFS…



ÍNDEX

1. INTRODUCCIÓ DEL PROBLEMA. 3

2. EXPLICACIÓ DE LA SINTAXI D'EXECUCIÓ. 3

3. EXPLICACIÓ DE L'ESTRUCTURA DEL CODI. 3

4. FUNCIONS UTILITZADES 6

4.1FUNCIÓ BUSCAPUNT 6

4.2 FUNCIÓ DISTÀNCIA 6

4.3 FUNCIÓ ENCUA (Cua, node, unsigned) 7

4.4 FUNCIÓ DESENCUA (Cua) 7

4.5 FUNCIÓ REQUEUE_WITH_PRIORITY 8

4.6 FUNCIÓ ADD_WITH_PRIORITY 8

4.7 FUNCIÓ MOSTRACAMI 8

5. EXEMPLES DE FUNICONAMENT. 10

5.1 PROBA 1, PROBA EXEMPLE 10

5.2 PROBA 2, PROBA AMB NODES INVENTATS 10

5.3 PROBA 3, PROBA AMB 3 ARGUMENTS PASSATS 10

5.4 PROBA 4 , AMB ELS MATEIXOS NODES COM ARGUMENTS 10

5.5 PROBA 5, 1R CAMÍ 11

5.6 SEGÓN CAMÍ 12

1. INTRODUCCIÓ DEL PROBLEMA.

Aquesta pràctica consisteix en la programació d'un gps que calcula la distància més petita entre dos punts implementant l'algorisme A*.

2. EXPLICACIÓ DE LA SINTAXI D'EXECUCIÓ.

El lliurament consisteix en implementar un algoritme A* amb el graf dels fitxers Nodes.csv i Carrers.csv per a calcular el camí més curt entre dos nodes donats.

Aquest camí s'ha de donar com una llista de nodes que siguin adjacents segons la informació del fitxer Carrers.csv i la llargada del camí serà la suma de les distàncies entre els nodes de la llista considerant que s'uneixen per una línia recta.

A diferència de l'algoritme de Dijkstra, pot ser que un node que ha entrat i sortit de la llista oberta, torni a entrar més endavant a la llista perquè s'ha trobat un camí més curt entre l'origen i aquest node.

De manera que si la funció heurística és admissible , tindrem que quan toqui expandir el node destí (que té funció heurística zero), haurem trobat el camí més curt entre l'origen i el destí.

3. EXPLICACIÓ DE L'ESTRUCTURA DEL CODI.

La funció principal rep dos arguments introduïts pel teclat els quals són: el punt des d'on sortim i el punt on volem arribar.

Aquests arguments estan en **char[]** i per poder treballat amb ells, al ser unes id' s de nodes, els hem de transformar a **long long int** , de manera que utilitzem la funció **atol** per a realitzar aquesta conversió.

El primer argument, el guardem a una variable anomenada origen, ja que és el punt de sortida i el segon argument a una variable anomenada destí.

El següent pas a fer, és la lectura dels dos fitxers imprescindibles per a la realització de la pràctica (part feta de la pràctica 9).

El fitxer, Nodes.csv té una llista amb tots els nodes del programa, amb la seva ID, la seva longitud i latitud.

Per poder guardar la informació, obrim el fitxer amb **fopen**, i el fem és la lectura del fitxer sencer per contar quantes nodes hi ha. Així doncs, anem contant cada cop que saltem de línia amb l'objectiu de crear una llista amb tots els nodes i guardar la seva informació.

Per tornar a posar-nos a l'inici del fitxer, utilitzem la comanda **rewind**, així podem llegir-ho de nou i assignar la informació de cada node i anar guardant la informació de cada node. A diferència de la pràctica 9, he afegit més informació ja que he ampliat l'estructura de nodes i he inicialitzat la distancia de cada node, el nombre d'arestes que connecten el node i el valor que indica si un node ha estat a la cua.

El fitxer amb nom carrers.csv és una llista de carrers que concatenen entre ells és a dir els adjacents.

Un cop tenim totes les dades guardades comencem amb la implementació de l'algorisme A* per poder trobar el camí més curt entre node i node.

Com bé s'ha indicat abans, s'entren els dos nodes d'origen i destí per arguments a l'executar el programa, de manera que per saber quins són i guardar la seva informació en una variable, utilitzem la funció **buscapunt** dos cops per trobar els dos nodes a la llista de nodes que hem creat al llegir el fitxer i guardar en aquests la posició dintre d'aquesta llista.

Abans de continuar amb el programa, ens hem d'assegurar que el node inicial i el node final introduïts des de l'execució siguin diferents ja que sinó no hi ha camí a buscar.

Per poder treballar amb l'algorisme A*, s'ha de crear una cua, la qual inicialitzem amb NULL i NULL, perquè encara no hem entrat encara cap element de la cua (els quals seran els nodes).

Per entrar el primer node a la cua, el qual correspon al primer argument entrat, utilitzem la funció **add_with_priority**, i inicialitzem el cost del primer node a 0 i indiquem que aquest no té un node pare o anterior.

En aquest cas no caldria implementar la funció encua ja que al no haver cap element a la cua, aquest **add_with_priority** fa la mateixa funció.

Ara que tenim un valor a la cua i aquesta no està buida, comencem amb el while que englobarà tot el funcionament del algorisme i que acabarà quan la cua principal estigui

buida, és a dir, quan haguem expandit tots els nodes que ens condueixen al node final de manera més ràpida.

El primer que fem dintre d'aquest while, és comprovar si l'índex (posició a la llista de nodes) del primer element de la cua és el mateix que el del node final el qual hem guardat amb la funció **buscapunt** a l'inici de la implementació de l'algorisme.

Si això es compleix, vol dir que hem trobat el camí que va del primer node al final, de manera que per imprimir –ho, cridem a la funció **mostracami** per veure per quins llocs hem passat per arribar al nostre destí i trenquem el while amb un break i acabar amb el programa.

En canvi, si no es compleix, vol dir que encara no hem arribat al final del trajecte.

A continuació, s' inicialitza un for amb l'objectiu de calcular la distancia total del node, és a dir el seu cost.

Per al càlcul d'aquesta distància final, es necessita del càlcul de tres distancies, la primera és la distancia entre el node pare i l'adjunt, implementant la funció **distància** entre aquests dos. La segona distància a tindre en compte, és la distància entre el node adjunt i el node final (en línia recta). I per últim, per saber la distancia total, hem de sumar aquestes dues distàncies calculades i la distància del node del qual estem mirant els seus adjunts.

Un cop calculada la distància total del node, mirem si aquesta distància calculada és més petita que el cost del mateix. Si es compleix, assignem el nou cost calculat del node al seu cost (hem trobat un camí més curt), actualitzant també la distància d'aquest (suma de la distancia del node pare i la distancia entre ells dos).

Per altra banda assignem que el pare del node és el node anterior a l'adjunt (assignes de qui t'extens).

Un cop s'han assignada la nova distancia, es mira si aquests han estat (o hi són) a la cua, o no.

Per a fer-ho hem afegit una variable a l'estructura node que diu si ha estat dins (1 si ha estat i 0 si no).

Si no ha estat, encuem amb prioritat dintre de la llista, en canvi, si ja ha estat, reencuem amb prioritat, així tenim la cua ordenada de manera que el node que s'expandirà després és aquell amb la distancia mínima.

4. FUNCIONS UTILITZADES

4.1FUNCIÓ BUSCAPUNT (long long int, node [], unsigned)

La funció buscapunt rep tres paràmetres, el primer d'ell és un **long long int**, el qual en aquest programa serà la ID d'un carrer o node., el segon argument entrat, és una llista de tots els nodes per poder fer la cerca dins el programa, i el tercer és un **unsigned** que serà el nombre total de nodes que hi ha al fitxer.

L'objectiu d'aquesta funció com bé diu el seu nom, és buscar un punt dintre d'una llista de punts, de manera que per programar-la, es fa una cerca amb un for fins que el node de la llista de la posició del iterat és la ID del node. Si es troba es retorna l'iterat i sinó retornem un missatge que no s'ha trobat el node i fem un return 0.

4.2 FUNCIÓ DISTÀNCIA (node node)

Aquesta funció té la finalitat de calcular la distancia entre dos nodes (els dos introduïts com a paràmetres).

En aquesta funció inicialitzem 7 variables double , 6 de posició dels nodes(x,y,z de cada node) i l'últim per a guardar el mòdul de la distància

Per poder trobar amb precisió la distancia entre les coordenades x,y,z definim

$x = R \cos(\theta) \cos(\phi)$

$y = R \sin(\theta) \cos(\phi)$

$z = R \sin(\phi)$,

R = 6371 km (radi aproximat de la Terra).

Un cop sabem el que hem de buscar utilitzem les formules anteriors per trobar els sis punts (x1,y1,z1,x2,y2,z2) utilitzem les formules per trobar aquests punts tenint en compte que hem de passar els angles a radiants, per això hem definit una constant que fa el traspassament directament i que té el valor aproximat de PI/180.

Trobats els punts només queda calcular la distància entre aquests, o el mòdul, resultat del qual guardarem a la última variable creada i la retornem.

4.3 FUNCIÓ ENCUA (Cua, node, unsigned)

La funció encua és una funció utilitzada a les cues per inserir un objecte a la part posterior de la cua. Rep com a paràmetre una cua a la qual li entrarem un element de cua que serà un ID en el nostre cas, també rep una llista de nodes i un índex. Arguments que serveixen per indicar que el índex del nostre element cua és l'índex del node a la posició índex entrat com a paràmetre.

L'element cua que estem utilitzant té un element següent de manera que com no hem

introduït cap més el guardem com un NULL.

Per últim si la cua ja té un element inici, guardem l'element la cua->final->següent. Si no, com no hi ha cap element a la cua, guardem l'element a l'inici i al final de la cua.

Al final no he utilitzat aquesta funció a la pràctica però es una alternativa al primer add_with_priority del programa.

4.4 FUNCIÓ DESENCUA (Cua)

Aquesta funció fa el contrari que la funció encua. Si la cua no està buida en retira el seu primer element. En cas contrari torna un codi d'error.

La funció desencua rep com a paràmetre la cua a la qual li volem treure l'element.

El primer pas a fer, és la comprovació per veure que la llista no esta buida mirant si el primer element de la cua és diferent de NULL. Si és NULL retornem i sinó creem una estructura d'element cua al qual li guardem l'element cua inicial de la cua que és el que hem de treure.

Per altra banda, hem de posar el següent element del node alliberat com a primer element de la cua i per a fer-ho diem que l'element inicial de la cua, és el següent element de l'element inicial de la cua (amb les estructures es veu de manera més clara).

I finalment, per a perdre el primer element de la cua, alliberem l'element cua creat dins de la funció que conté l'element que volem perdre.

4.5 FUNCIÓ REQUEUE_WITH_PRIORITY (unsigned, Cua, node) {

A aquesta funció entra la ID del node, la cua i una llista de nodes i comprova si l'índex entrat és el mateix del node que hi ha a l'inici de la cua.

Si no, recorre la cua fins trobar quest node i s'elimina de la cua. Per tornar a posar-ho dintre, cridem a la funció add_with_priority

4.6 FUNCIÓ ADD_WITH_PRIORITY (unsigned, Cua, node []) {

Aquesta funció rep tres arguments, el primer és l'índex del node que es vol afegir a la cua, el segon és la cua on volem afegit el node i l'últim és la llista de nodes.

L'objectiu d'aquesta funció és afegir el node de forma ordenada de manera que el primer element de la cua sigui aquell amb el cost mínim.

Per a fer-ho comprovem si hi ha cap element a la cua, si no hi ha, fa la funció d'un encua.

Sinó, fem un for per ordenar cada node segons el cost corresponent, de manera que si el cost és ,és gran que el del següent node, es recorre una posició més fins que s'incompleix la condició, és a dir fins que el cost sigui més petit que el cost del següent.

En aquesta part assignem o canviem la variable de dintre_cua a 1 per indicar que ha estat a la cua.

4.7 FUNCIÓ MOSTRACAMI (node[],unsigned,unsigned,unsigned) { (ADAPTACIÓ DE GERRES).

Aquesta funció, és bàsicament la utilitzada al problema de transvasament d'aigua de les gerres per trobar configuracions però molt més senzilla.

En aquest cas en comptes de tindre configuracions tenim la llista de nodes, també passem la quantitat de nodes, i el punt d'inici i el punt del final que són els trobats amb la funció buscapunts a l'inici del main.

L'objectiu d'aquesta funció, és carregar un vector amb tots els nodes pare a partir del node final, és a dir tindre un vector que comenci pel node d'arribada i acabi al node inicial amb tots els nodes que utilitzem o que passem per unir aquests dos punts, els quals uns seran els pares o anteriors del altres.

Per carregar un vector sense haver de fer un malloc (no faig un malloc ja que podem saber amb un while petit quants nodes hi haurà en aquest), hem de saber quants nodes o elements hi haurà en aquest vector.

Per saber això podem recórrer tots el nodes pare del node final fins que el node anterior dels anteriors del node final sigui el node inicial, i contem quantes iteracions hem fet fins trobar-ho.

Un cop contats creem aquest vector amb n+1 components, i tornem a començar de nou fent exactament el mateix però ara a part de contar iteracions per després llegir el vector carregat a la inversa, anem carregant el vector on vector[0] serà el node final i vector[1] el seu node pare i així successivament.

Un cop acabat aquest segon while tenim el vector carregat amb el camí a la inversa, de manera que l'hem de recorre des del final fins al principi per mostrar el camí en l'ordre correcte.

Per a fer-ho fem un for utilitzant un dels iterats carregats amb el nombre de nodes del camí i per imprimir-los tots, els imprimim restant un al valor del iterador per anar enrere en el for, el qual pararà quan el valor del iterador sigui 0

5. EXEMPLES DE FUNICONAMENT.

Per a aquesta part he volgut diferenciar alguns exemples, un on només es veu la terminal per veure els errors implementats i un altre on es pot veure el mapa amb el camí seguit

5.1 PROBA 1, PROBA EXEMPLE

Nodes 259184345, 1793441250

```
# La distancia de 259184345 a 1793441250 es de 507.886803 metres.
# Cami Optim:
Id=259184345 | 41.545380 | 2.106830 | Dist=0.000000
Id=259437888 | 41.545752 | 2.106744 | Dist=42.042028
Id=259437905 | 41.546388 | 2.106495 | Dist=115.722403
Id=259438253 | 41.546734 | 2.107858 | Dist=235.476557
Id=965459173 | 41.546963 | 2.107746 | Dist=262.617110
Id=960085142 | 41.547169 | 2.107648 | Dist=286.900380
Id=1944921315 | 41.547281 | 2.107553 | Dist=301.623075
Id=2412854895 | 41.547777 | 2.107092 | Dist=368.879521
Id=1944921533 | 41.548161 | 2.107668 | Dist=433.058491
Id=1944921536 | 41.548240 | 2.107798 | Dist=446.899874
Id=1944921547 | 41.548280 | 2.107864 | Dist=454.047082
Id=1793441253 | 41.548396 | 2.108055 | Dist=474.495886
Id=1944921549 | 41.548399 | 2.108073 | Dist=476.041257
Id=1955175329 | 41.548407 | 2.108110 | Dist=479.200109
Id=1955175330 | 41.548452 | 2.108329 | Dist=498.154671
Id=1793441250 | 41.548481 | 2.108440 | Dist=507.886803
```

5.2 PROBA 2, PROBA AMB NODES INVENTATS

```
> ./A 12345 12334332
ID no trobada, node inventat
```

5.3 PROBA 3, PROBA AMB 3 ARGUMENTS PASSATS

```
> ./A 12345 12334332 899898
Error a la introducció de nodes entrats.
```

5.4 PROBA 4 , AMB ELS MATEIXOS NODES COM ARGUMENTS

```
> ./A 259184345 259184345
Error: No hi ha camí a fer, el node de sortida i el node d'arribada són els mateixos
```

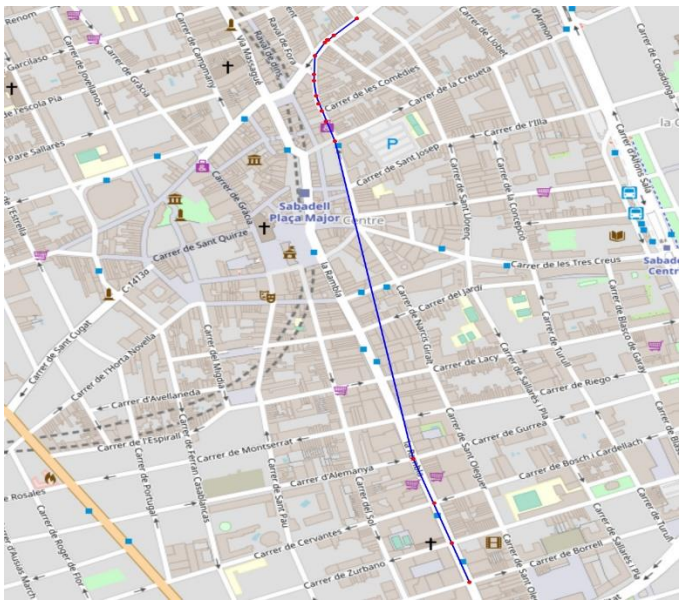
5.5 PROBA 5, 1R CAMÍ

Node sortida 1446249442

Node arribada 1626526528

La distancia entre 1446249442 i 1626526528 és de 1193.795510 metres

```
# La distancia entre 1446249442 i 1626526528 és de 1193.795510 metres
# Cami Óptim:
 Id=1446249442 | 41.550568 | 2.109855 | Dist=0.000000
 Id=1389701176 | 41.550261 | 2.109395 | Dist=51.312689
 Id=1389701174 | 41.550188 | 2.109287 | Dist=63.447756
 Id=1446298035 | 41.550174 | 2.109272 | Dist=65.339259
 Id=1389701172 | 41.550126 | 2.109218 | Dist=72.407099
 Id=1389701167 | 41.549900 | 2.109030 | Dist=101.985710
 Id=1446297818 | 41.549552 | 2.109020 | Dist=140.701788
 Id=1446297970 | 41.549439 | 2.109017 | Dist=153.202934
 Id=1446297816 | 41.549166 | 2.109048 | Dist=183.645001
 Id=1446297793 | 41.549026 | 2.109087 | Dist=199.621516
 Id=1446297687 | 41.548890 | 2.109155 | Dist=215.713108
 Id=1446297574 | 41.548704 | 2.109248 | Dist=237.767614
 Id=965193242 | 41.548349 | 2.109418 | Dist=279.749836
 Id=255400656 | 41.542621 | 2.110951 | Dist=929.334165
 Id=255742676 | 41.541818 | 2.111350 | Dist=1024.551198
 Id=259189436 | 41.541095 | 2.111687 | Dist=1109.778458
 Id=1626526528 | 41.540386 | 2.112038 | Dist=1193.795510
```



5.6 SEGÓN CAMÍ

Node sortida = 1389701052

Node Arribada = 1944921549

La distancia entre 1389701052 i 1944921549 és de 221.654287 metres

```
# La distancia entre 1389701052 i 1944921549 és de 221.654287 metres
# Cami Óptim:
 Id=1389701052 | 41.547515 | 2.109688 | Dist=0.000000
 Id=1389701062 | 41.547484 | 2.109220 | Dist=39.112376
 Id=1389701038 | 41.547476 | 2.108898 | Dist=65.908920
 Id=1449178113 | 41.547900 | 2.108861 | Dist=113.132195
 Id=3737291337 | 41.548032 | 2.108823 | Dist=128.132035
 Id=255402708 | 41.548108 | 2.108818 | Dist=136.594789
 Id=965193247 | 41.548096 | 2.108781 | Dist=139.887120
 Id=255402709 | 41.548158 | 2.108599 | Dist=156.596562
 Id=1793443016 | 41.548152 | 2.108568 | Dist=159.264004
 Id=1944921532 | 41.548141 | 2.108501 | Dist=164.933980
 Id=1945585326 | 41.548128 | 2.108339 | Dist=178.498519
 Id=1793443018 | 41.548119 | 2.108227 | Dist=187.890705
 Id=1945585327 | 41.548178 | 2.108198 | Dist=194.943659
 Id=1944921535 | 41.548206 | 2.108180 | Dist=198.377915
 Id=1955175325 | 41.548259 | 2.108152 | Dist=204.663496
 Id=1944921545 | 41.548277 | 2.108142 | Dist=206.866031
 Id=1955175327 | 41.548302 | 2.108129 | Dist=209.919026
 Id=1793443034 | 41.548383 | 2.108086 | Dist=219.616777
 Id=1944921549 | 41.548399 | 2.108073 | Dist=221.654287
```