Matemàtica computacional i analítica de dades
Algorítmia i combinatòria en grafs ...
Curs 2020–21

9 Treballar amb un mapa

Una de les aplicacions més clares del algoritmes de cerca de camins més curts en un graf consisteix a buscar el recorregut més curt (per carrers o carreteres) entre dos punts d'un mapa. En aquest problema el mapa queda reduït a un conjunt de *nodes* (del quals es coneix, com a mínim, la seva latitud i longitud) i el carrers (o carreteres) queden determinats com una successió d'aquests nodes de forma que, amb mentalitat de grafs, hi ha una aresta (bidireccional si no s'imposen restriccions de sentit únic als carrers) per a cada parell de nodes consecutius d'una d'aquestes *vies*.

9.1 Distància entre dos punts

Per tal de posar en pràctica qualsevol algoritme que pretengui calcular camins òptims sobre un mapa caldrà, d'entrada, poder localitzar nodes concrets i, a més, calcular la distància entre ells a partir de les dades de la seva posició (latitud i longitud). Aquests càlculs de distàncies hauran de servir tant per establir el pes de cada aresta del graf que es proposi com a part del camí, com per establir una estimació de la distància que falta per arribar a l'objectiu quan s'està utilitzant un algoritme del tipus A^*.

En els exercicis que es proposen a continuació, utilitzarem les dades que proporciona el projecte **OpenStreetMap** (`https://www.openstreetmap.org/`), que permet descarregar (Exportar) les dades d'una regió que ens interessi. El fitxer `Nodes.csv` amb el que treballarem conté la informació d'una part del centre de Sabadell. Encara que la informació que s'obté quan es descarrega una regió del mapa és molt més rica, (de moment) s'ha guardat només (amb una mica de `awk` i algun retoc addicional) una llista de nodes on la informació que proporciona cada línia, separant els camps per punts-i-coma, consisteix en un identificador (format per 10 xifres), la latitud i la longitud. Les primeres línies del fitxer són:

```
0255400656;41.542621000000;2.110951200000
0255400665;41.541808300000;2.106842500000
0255400702;41.540602000000;2.107219200000
0255401013;41.544904700000;2.106070000000
0255401208;41.538617400000;2.109676200000
0255401548;41.541000300000;2.108541000000
0255401549;41.542065200000;2.108301600000
```

Exercicis

Afegiu com a les dues primeres línies del programa i en format comentari els vostres `Nom`, `Cognom` i `NIU`.

El nom dels programes que contenen el codi ha de ser de la forma `PrXExY.c`, on `X` fa referència a la pràctica i `Y` a l'exercici. Per exemple, el nom del programa de l'apartat següent hauria de ser `Pr9Ex911.c`.

Exercici 9.1.1: Definiu un tipus d'estructura `node` com:

```c
typedef struct{
    long int id;
    double latitud,longitud;
}node;
```

per tal de poder definir un *vector* que contingui la informació de tots els nodes que defineixen el mapa. Feu que el programa llegeixi el fitxer `Nodes.csv` i guardi cada línia a una posició del vetor que heu declarat. Acabeu el programa fent que es vegi al terminal el llistat d'aquesta informació (així es podrà comprovar si tot funciona bé). Noteu que haureu de calcular en primer lloc quants elements s'han de guardar i, després de dimensionar el punter corresponent, tornar a llegir el fitxer i omplir les dades. Fixeu-vos també que es demana que l'identificador es guardi com a `long int` (és per assegurar-nos que càpiguen enters d'11 dígits).

Exercici 9.1.2: Una part molt important dels algoritmes que volem implementar consisteix a localitzar un node i extreure'n les seves dades. Implementeu una funció amb el nom i prototipus:

```c
unsigned buscapunt(long int ident , node l[] , unsigned nnodes)
```

que retorni la `i` tal que `l[i].id` és el node que té identificació el nombre `ident`. Preveieu que el node demanat pot ser que no aparegui a la llista. Teniu en compte que, per sort(?), els identificadors apareixen ordenats i, per tant, es pot preveure realitzar la cerca binària en comptes de seqüencial (estratègia que pot fer que es necessitin moltes menys comparacions per localitzar l'element). Canvieu la part final del programa per una que permeti introduir l'identificador d'un node i doni com a resultat les seves coordenades.

Exercici 9.1.3: Afegiu una funció que calculi la distancia geogràfica entre dos nodes qualssevol (tenint en compte només les seves coordenades geogràfiques). Podem programar-ho amb una funció que amb el prototipus:

```c
double distancia(node , node)
```

Inseriu al final del programa que es demani les identificacions de dos nodes i, a continuació, calculi i mostri la distància entre els dos punts segons l'explicació següent.

Aproximació de la distància per coordenades: Com que la regió sobre la que s'està treballant és prou petita per a poder obviar la curvatura de la Terra, la distància entre dos punts amb $(\text{longitud}, \text{latitud}) = (\theta, \varphi)$ es pot aproximar amb prou precisió per la distància euclidiana entre els punts $(x, y, z) \in \mathbb{R}^3$ corresponents,

$$\begin{aligned} x &= R\,\cos(\theta)\,\cos(\varphi) \\ y &= R\,\sin(\theta)\,\cos(\varphi) \\ z &= R\,\sin(\varphi)\,, \end{aligned}$$

on $R = 6371\,\text{km}$ és el radi aproximat de la Terra. Teniu en compte també que les dades de la latitud i la longitud venen donades en graus i que les funcions trigonomètriques de `C` (i de la majoria de sistemes de càlcul) utilitzen els valors dels angles en radiants (per fer la conversió cal multiplicar per $\pi/180 \approx 3.1415926536/180.$, i si voleu utilitzar més xifres de π tampoc passa res).[a]
Podeu trobar fàcilment pàgines web que ofereixen el càlcul de la distància entre dos punts de la Terra a partir de les seves coordenades geogràfiques (per exemple Google Maps, `https://www.movable-type.co.uk/scripts/latlong.html`). Podeu utilitzar qualsevol d'aquests llocs per comprovar l'exactitud dels vostres càlculs.

[a] Per a regions més extenses, aquesta aproximació pot donar estimacions (molt) dolentes!! Si voleu calcular distàncies amb més precisió, busqueu "llei dels cosinus esfèrica/spherical law of cosines" o "haversine".

9.2 Connexions entre els nodes

Quan es treballa amb la informació d'un mapa i es busquen recorreguts òptims cal tenir, a part de la informació de les posicions dels nodes del terreny, la descripció de les *vies/carrers* a través de les quals es pot fer el recorregut. En el projecte **OpenStreetMap**, cada carrer/carretera es descriu com una successió dels identificadors dels nodes consecutius que es van trobant al llarg del seu recorregut. Per tant, cada carrer defineix una aresta (en principi bidireccional) entre cada node i el següent que apareix a la llista corresponent.

Exercicis

Afegiu com a les dues primeres línies del programa i en format comentari els vostres `Nom`, `Cognom` i `NIU`.

El nom dels programes que contenen el codi ha de ser de la forma `PrXExY.c`, on `X` fa referència a la pràctica i `Y` a l'exercici. Per exemple, el nom del programa de l'apartat següent hauria de ser `Pr9Ex921.c`.

Exercici 9.2.1: Modifiqueu el programa que llegeix les posicions dels nodes d'un mapa per tal que, a més, llegeixi la llista de carrers del fitxer `Carrers.csv` i incorpori a la informació de cada node quins són els seus adjacents al llarg d'algun carrer. Recordem les estructures fetes anteriorment (i adaptades a aquest cas):

```
1   #define MAXARST 5 // pot ser s'ha de canviar!!!!
2
3   typedef struct{
4     char carrer[12];
5     unsigned numnode; // fa referència a la posició al vector de nodes
6   }infoaresta;
7
8   typedef struct{
9     long int id;
10    double latitud,longitud;
11    int narst;
12    infoaresta arestes[MAXARST];
```

```
13      }node;
```

Noteu que hi pot haver nodes que només tinguin un node adjacent (si estan en un punt sense sortida), dos nodes adjacents (per exemple, si estan en el recorregut d'un carrer i no corresponen a cap encreuament de carrers) o un nombre indeterminat d'adjacents (quan el node correspon a una cruïlla de dos o més carrers). Això fa que sigui necessari pensar bé com s'ha d'implementar la llista d'adjacents dels nodes, donat que no és immediat descobrir quina longitud ha de tenir.

Exercici 9.2.2: Si us fixeu en les primeres línies del fitxer de carrers, veureu que els nodes amb identificadors

`0259437888;0259437889;0259437890;0259437805;1945672908;`
`1945672910;2988535157;2150815508;1945672929;0259437827;`
`1946430351;1946430357;0259437891`
formen el carrer que té per clau `id=0023936036`.

Feu que, a partir de les dades que teniu guardades (o sigui, sense tornar a llegir el fitxer `Carrers.csv`), el programa demani per pantalla la identificació d'un carrer i mostri els seus nodes separats per punt-i-coma (`;`) en un ordre que hi hagi una aresta entre un node i el següent. En particular, si entreu `0023936036` hauríeu d'obtenir la llista anterior en aquest ordre o en ordre invers.

Exercici 9.2.3: Afegiu un camp `llargada` a la informació de les arestes, quedant:

```
1       typedef struct{
2         char id[12];
3         unsigned numnode;
4         double llargada;
5       }infoaresta;
```

i ompliu-lo amb la distància entre els dos nodes que uneix utilitzant la funció que determina la distància entre dos nodes qualssevol del mapa (veure Exercici 9.1.3).
Feu que el programa demani la identificació d'un carrer i a continuació en mostri per pantalla la llargada del carrer (suma de les llargades dels seus trams).

Instruccions finals

Quan acabeu la pràctica, feu el lliurament dels fitxers de codi (que tenen els noms de la forma `Pr9ExY.c` segons el que hem indicat anteriorment) a través del Campus Virtual des de l'apartat de lliuraments de l'assignatura. Recordeu que al principi de cada fitxer hi ha d'haver el vostre nom i NIU.