



Posicionament de restaurants

PROCESSAMENT DE DADES ESPAIALS

MÀSTER EN ENGINYERIA INFORMÀTICA

Oscar Galera Alfaro i Meriem Abjil Bajja

14 de gener de 2019

Índex

| | | |
|----------|--|-----------|
| 1 | Introducció | 4 |
| 2 | Objectiu | 5 |
| 3 | Definició del problema | 6 |
| 3.1 | Filtrat de dades | 6 |
| 3.2 | Assignar un pes a cada atracció | 9 |
| 3.3 | Dividir el mapa en àrees | 10 |
| 3.3.1 | Estratègies | 10 |
| 3.3.2 | Diagrama de <i>Voronoi</i> de primer nivell | 12 |
| 3.3.3 | Capa de <i>Voronoi</i> de primer nivell capa 2 | 12 |
| 3.4 | Definir les zones candidates | 13 |
| 3.5 | Ubicar els restaurants | 13 |
| 4 | Pre-processament de dades | 16 |
| 5 | Treball futur | 17 |
| 5.1 | El càlcul de distàncies | 17 |
| 5.2 | El càlcul dels diagrames de Voronoi | 17 |
| 5.3 | El càlcul de la importància d'una regió | 17 |
| 6 | Conclusions | 18 |

| | |
|------------------------------|----|
| A Diagrames de Voronoi | 20 |
| B Algoritme A* | 21 |
| C Algoritme latitud longitud | 23 |

Índex de figures

| | | |
|----|---|----|
| 1 | Mapa de <i>Disney Land París</i> | 4 |
| 2 | <i>Magic Bands</i> | 5 |
| 3 | PM Quadtree | 7 |
| 4 | QuadTree: Ordre de Morton | 7 |
| 5 | Exemple PM QuadTree | 8 |
| 6 | Grid amb Punts | 8 |
| 7 | Pesos de cada atracció | 9 |
| 8 | Proporció de pesos de cada atracció | 10 |
| 9 | Diagrama de <i>Voronoi</i> d'ordre 1 | 12 |
| 10 | Mapa amb les possibles ubicacions | 13 |
| 11 | Restaurant en el diagrama de <i>Voronoi</i> superposat de la capa 1 i 2 | 14 |
| 12 | Mapa d'atraccions amb les ubicacions dels restaurants | 15 |
| 13 | Regió <i>Voronoi</i> | 20 |
| 14 | Diagrama de <i>Voronoi</i> | 20 |

1 Introducció

Els dissenyadors de *Disney Land París* han descobert com mantenir el seu univers feliç: mitjançant l'ús de polseres *Magic Bands* per fer un seguiment de les identitats, els moviments i l'estat financer dels usuaris.

El *MyMagic* + “sistema de gestió de vacances” pot fer un seguiment dels convidats a mesura que es mouen per *Disney Land París* i analitzar els seus hàbits de compra.

Degut a l'èxit generat per *Disney Land París* (a partir d'ara *DPL*), el propietari d'aquest ha decidit obrir un nou parc temàtic similar (veure *Figura ??*) però aquest cop en el territori Gironí. Ens han encarregat indicar el posicionament dels restaurants amb l'objectiu de maximitzar els beneficis generats per aquests. Per ubicar aquests restaurants, es realitzarà un estudi de la popularitat de les atraccions que hi ha a *DLP* a fi de determinar les zones més visitades, i d'aquesta manera, fer la distribució dels restaurants més cars en aquestes zones per maximitzar els beneficis generals del parc.



Figura 1: Mapa de *Disney Land París*

2 Objectiu

L'objectiu del projecte és **fer la distribució dels restaurants més cars a partir d'unes determinades zones “remarcades” per maximitzar els beneficis del parc.** Aquestes zones les anomenarem “candidates”, i seran aquelles on es podran ubicar els restaurants.

Per aconseguir aquest objectiu es considerarà que el mapa d'atraccions del nou territori és el mateix que el de *DLP*; que les *Magic Bands* (veure *Figura ??*) recullen i emmagatzemem la informació de la posició dels seus usuaris cada 5 minuts, i que disposem de tantes zones candidates (o més) com restaurants es vulguin ubicar.

Per determinar el grau d'interès de cada atracció, es compta amb una gran quantitat de dades generades per les *Magic Bands* (veure *Figura ??*) que es reparteixen a l'entrada del parc temàtic, i que tenen per objectiu la recollida de dades públic a fi d'estudiar el seu comportament.



Figura 2: *Magic Bands*

3 Definició del problema

Per aconseguir l'objectiu que s'ha plantejat es seguiran 6 passos, que són:

1. Preprocessament de dades (veure **secció ??**).
2. Filtrat de dades.
3. Assignar un pes a cada atracció.
4. Dividir el mapa en àrees.
5. Definir les zones candidates.
6. Ubicar els restaurants.

S'assumeixen les següents entrades¹:

- Un fitxer amb els punts² que defineixen els límits del mapa, juntament amb els punts on hi ha situades les entrades de les atraccions i la quantitat de gent que hi ha pujat.
- Un fitxer amb les zona prohibida³ definides pels seus punts i arestes (s'assumeix que les àrees estan definides per polígons).
- Un fitxer amb els punts que representen les persones i la hora en què s'han recollit.

3.1 Filtrat de dades

En aquest apartat es volen filtrar els punts de les persones situades en àrees prohibides. Per això, es creará una estructura *PM QuadTree* a partir dels polígons que regeixen els obstacles i que s'usarà per eliminar tots aquells punts que es trobin a dins.

¹Aquestes entrades vindran proporcionades per la direcció del parc

²Els punts es defineixen per la latitud i la longitud.

³Per zona prohibida s'entén l'àrea que cobreixen les atraccions, així com els obstacles típics que es poden trobar en un parc d'atraccions.

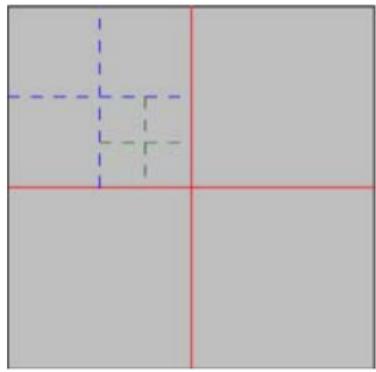


Figura 3: PM Quadtree

Un *PM QuadTree* (*Figura ??*) és una estructura de dades molt utilitzada en informàtica gràfica i es basa en una descomposició jeràrquica dels objectes de l'espai que descriuen. Es crea a partir d'una caixa englobant de forma quadrada que conté tots els polígons (coordenades) que s'estan descriuint. Aquest espai que hi ha, es divideix en quatre quadrats iguals (quadrants) generant així quatre fills que, juntament amb el pare, formen un arbre 4-àri. Tot seguit es reparteixen tots els polígons en el fill corresponent dependent de la seva posició. Aquest procés es va repetint fins que cada node concret conté tan sols un polígon.

L'ordre que s'estableix per a assignar quin número té cada fill és segons l'ordre de *Morton* (veure *Figura ??*), que dóna prioritat al fill superior-esquerre.

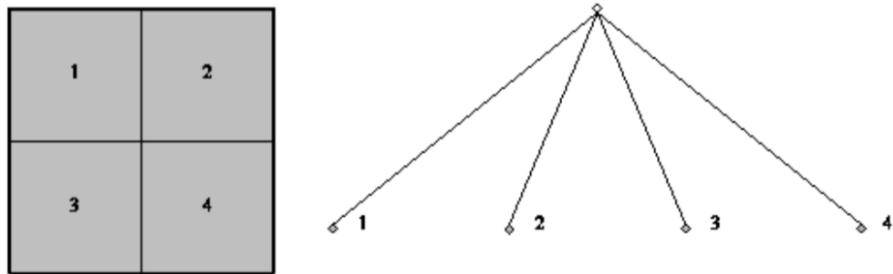


Figura 4: QuadTree: Ordre de Morton

A la *Figura ??* es pot veure un exemple molt senzill d'un QuadTree amb 5 polígons juntament

amb el seu arbre generat.

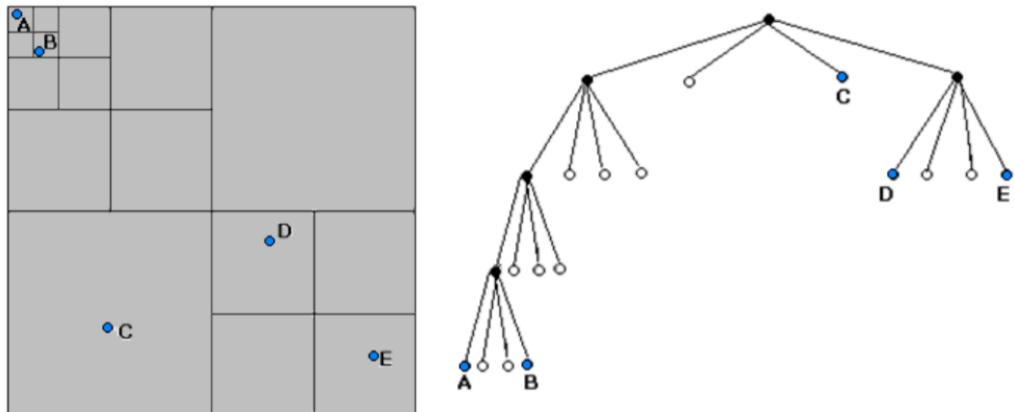


Figura 5: Exemple PM QuadTree

Un cop es té creada l'estructura, s'avaluarà cada un dels punts per veure en quina regió està. Si aquesta regió conté part d'un obstacle s'haurà de comprovar si el punt està o no dins d'aquest. Si el punt no es troba dins d'un obstacle, es considerarà una dada vàlida i es posicionarà a sobre d'una grid, de manera que, a la grid tindrem posicionades totes les dades vàlides. Podem veure a la següent *Figura ??* un exemple de com seria la grid.

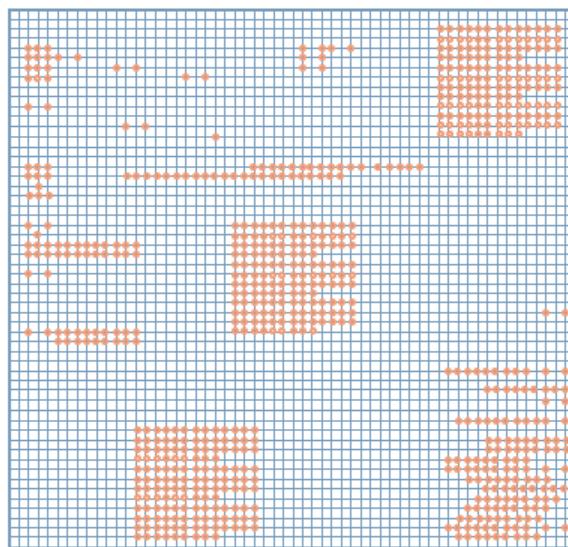


Figura 6: Grid amb Punts

3.2 Assignar un pes a cada atracció

El pes d'una atracció es definirà a partir de la quantitat de gent que ha pujat a aquesta. Per cada pivot, s'assigna un pes p_i tal que:

$$p_i = \frac{q_i}{t}$$

on:

- q_i és la quantitat de gent que ha pujat a l'atracció i .
- t és el total de gent.

En la següent *Figura ??* es pot veure el mapa amb els pesos totals per a cada una de les atraccions.



Figura 7: Pesos de cada atracció

En la següent *Figura ??* es pot veure el mapa amb la proporció de pesos per a cada una de les atraccions.



Figura 8: Proporció de pesos de cada atracció

3.3 Dividir el mapa en àrees

A partir del pes de cada atracció (ubicat en l'entrada d'aquesta), es poden seguir diferents estratègies per calcular la regió d'influència per cada una de les atraccions a partir del nombre de persones que hi ha al voltant.

3.3.1 Estratègies

Algunes d'aquestes estratègies són:

- Utilitzar un diagrama de *Voronoi* per definir la regió d'interès de cada atracció (cada persona dins d'aquesta àrea computa 1).
- Utilitzar l'estratègia del punt anterior, i per cada atracció, per cada una de les àrees no definides pel pivot, computar les persones que hi ha com $\frac{1}{1+d}$ on d correspon a la distància en metres des de la persona a l'entrada de l'atracció.
- Utilitzar l'estratègia del punt anterior, però aquesta vegada comptant la distància (d) des de la persona al segment que defineix l'àrea regida pel pivot.
- Utilitzar diagrames de *Voronoi* de nivell amb capa (k on $2 \leq k \leq n$ i n és el nombre d'atraccions), el diagrama de primer nivell defineix la regió de màxim interès de l'atracció (computant 1 per cada persona) i les capes $k-1$ **capes superposades**, determinen zones d'interès però no tan importants (computen $\frac{1}{k}$ per cada persona).

Segons l'estratègia que es volgués seguir, s'obtindria una distribució en àrees o un altre. En aquest cas, s'ha optat per utilitzar l'última proposta amb una $k = 2$, ja que es creu que és una bona aproximació pel problema que es vol resoldre.

Un cop entès que és un diagrama de *Voronoi* sobre el pla (veure Annex ??), podem aplicar els mateixos conceptes sobre aquest projecte.

A partir de la funció distància de cada punt es pot obtenir diferents diagrames de *Voronoi*: el més proper i el segon més proper.

Per trobar els diferents diagrames de *Voronoi*, cal definir de quina manera es calcularan les distàncies entre dos punts. Per calcular la distància entre un punt i un pivot s'utilitzarà l'**algoritme A*** (veure Annex ??) per cada punt de la grid, de manera que el primer pivot trobat es considera el punt generador més proper i el segon trobat, es considera el segon punt generador més proper.

Des de cada punt situat a la grid es buscarà el primer i segon pivot més proper tenint en compte que a la grid hi han camins habilitats i no habilitats (els camins no habilitats corresponen als camins on es troba un obstacle). Per això, abans d'aplicar A*, aplicarem l'**algoritme Dijkstra**

sobre la grid començant des d'un pivot per eliminar tots els camins que no estan habilitats, és a dir, que hi hagin obstacles.

3.3.2 Diagrama de *Voronoi* de primer nivell

En la següent *Figura ??* es pot veure una possible representació del diagrama de *Voronoi* d'ordre 1 per l'escenari plantejat.



Figura 9: Diagrama de *Voronoi* d'ordre 1

3.3.3 Capa de *Voronoi* de primer nivell capa 2

Per afinar més els càlculs, es vol calcular la capa 2 del diagrama de *Voronoi* de primer nivell, i a partir d'aquesta obtenir una aproximació de la influència de les persones que estan pròximes a les àrees definides en el primer nivell.

Per obtenir la capa 2, cal realitzar l'algoritme del **diagrama de *Voronoi* de primer nivell capa 2** utilitzant, com a pivot del diagrama de *Voronoi*, el segon pivot més proper.

3.4 Definir les zones candidates

Un cop es disposa de la regió d'influència de cada atracció (veure Secció ??). Es vol fer el posicionament dels restaurants, però abans, cal que la direcció del parc proporcioni un mapa tot indicant quines són les possibles ubicacions per aquests restaurants. En la següent *Figura ??* es mostra un possible mapa.



Figura 10: Mapa amb les possibles ubicacions

3.5 Ubicar els restaurants

En aquest punt, la direcció ha de proporcionar una llista amb els restaurants que es volen ubicar amb un valor associat, aquest valor ha de reflectir quant car és cada restaurant.

Una vegada es tenen els restaurants a ubicar i les zones candidates, cal calcular el benefici de cada configuració a partir del benefici de cada restaurant i la influència que rep d'acord a les atraccions que té al voltant.

Per calcular el benefici de cada candidat:

- Es defineix l'àrea circular (de radi predefinit) que cobreix un restaurant
- Es calcula l'àrea ponderada del pes dels pivots dels diagrames de *Voronoi* superposats de capa 1 i capa 2

A continuació es podrà veure un exemple de com seria aquest càlcul. Se suposa que hi ha un restaurant situat en el diagrama de *Voronoi* resultat de superposar les capes 1 i 2 com es veu a la *Figura ??*

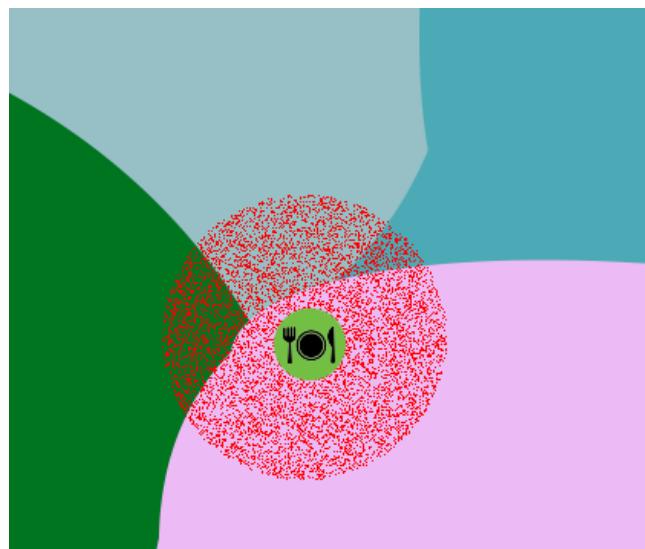


Figura 11: Restaurant en el diagrama de *Voronoi* superposat de la capa 1 i 2

$$B_x = A_1 * P_1 + A_2 * P_2 + A_3 * P_3 + A_4 * P_4$$

on:

- B_x és el benefici del restaurant x
- A_1 és l'àrea de color verd que cobreix el restaurant
- P_1 és el pes de l'àrea de color verd

- A_2 és l'àrea de color blau cel que cobreix el restaurant
- P_2 és el pes de l'àrea de color blau cel
- A_3 és l'àrea de color blau marí que cobreix el restaurant
- P_3 és el pes de l'àrea de color blau marí
- A_4 és l'àrea de color rosa que cobreix el restaurant
- P_4 és el pes de l'àrea de color rosa

Una vegada generada totes les combinacions, s'elegirà la que maximitzi el benefici global del parc. En la següent *Figura ??* es mostra un exemple d'una possible solució final.



Figura 12: Mapa d'atraccions amb les ubicacions dels restaurants

4 Pre-processament de dades

Pel caràcter del problema i per evitar errors, es realitzarà un preprocessament amb els punts recollits, de tal manera que, només es consideraran aquells punts:

- S'han recollit dins d'un conjunt de franges horàries que es consideren d'interès pels visitants alhora de buscar un restaurant per menjar. Així doncs, el llistat de franges horàries a considerar és:
 - Esmorzar: 9h-10h
 - Dinar: 12h-14h
 - Berenar: 17h-18h
 - Sopar: 20h-22h
- No s'hagin recollit a les zones on es troben obstacles. Considerem obstacles a les atraccions, boscos, llacs, etc.

Per realitzar-ho, seguirem els següents passos:

- Convertir els punts que delimiten el mapa.
- Convertir els punts dels pivots i comprovar que estiguin dintre del mapa.
- Convertir els punts i les arestes de les zones prohibides.
- Convertir els punts de les persones.
 - Filtrar els punts per la frange horària. Només es quedarán els punts de les hores que ens interessen.
 - Convertir els punts filtrats.

Nota: Quan es parla de convertir els punts, es fa referència a aplicar l'algorisme de conversió de latitud, longitud a X, Y (veure annex ??) per tal de transformar un punt referenciat per (latitud, longitud) en un punt (X, Y).

5 Treball futur

5.1 El càlcul de distàncies

Es podria mirar de realitzar el càlcul de distàncies amb la programació en paral·lel. El càlcul de distàncies consisteix en realitzar un Dijkstra per eliminar els obstacles i un A* per fer el càlcul de les distàncies entre un punt i un pivot. Tan en el Dijkstra com A*, com que el càlcul de les distàncies entre un punt i un pivot es poden fer independentment, la programació en paral·lel resulta eficaç per millorar l'eficiència del càlcul.

5.2 El càlcul dels diagrames de Voronoi

La construcció dels diagrames de Voronoi es comú fer-ho amb la estructura de la triangulació Delaunay. La triangulació Delaunay es pot realitzar amb algoritmes en paral·lel, per tant, es podria realitzar la construcció dels dos diagrames de Voronoi mitjançant la programació en paral·lel.

5.3 El càlcul de la importància d'una regió

Pel càlcul de la importància d'una regió es podria tenir en consideració diferents criteris, com per exemple, l'edat dels usuaris, si van en grup, el tipus d'atracció on pugen. Es podria aplicar tècniques per aprofitar l'informació rebuda amb les *Magic Bands*.

6 Conclusions

L'objectiu principal d'aquest projecte és **fer la distribució dels restaurants amb l'objectiu de ubicar aquells que són més cars en les zones més freqüentades**, per això, es parteix de la informació d'un parc d'atracció similar en el qual s'han recollit les dades dels visitants del parc a través de les *Magic Bands*. L'objectiu d'aquest projecte ha estat complert satisfactòriament.

Aquest objectiu, com s'ha pogut comprovar, ha estat complert de forma satisfactòria. Tot el conjunt de requeriments definits des d'un principi han estat assolits al final.

Amb el desenvolupament d'aquest treball hem pogut veure la complexitat dels problemes de dades espaiials. El que en principi semblen problemes simples tenen solucions bastant complicades de trobar.

Finalment creiem que la solució donada compleix els requeriments inicials i dona resposta al problema plantejat, però ens hagués agradat aprofundir una mica més tècnicament en l'algoritme.

Referències

- [1] GIZMODO.COM., *How I Let Disney Track My Every Move*, 2017.
<https://gizmodo.com/how-i-let-disney-track-my-every-move-1792875386>
- [2] OKABE, A.; BOOTS, B.; SUGIHARA, K., *Spatial tessellations. Concepts and Applications of Voronoi Diagrams* 2nde.d.
<http://www.gbv.de/dms/ilmenau/toc/119618044.PDF>
- [3] ABELLANAS, B.; ABELLANAS, M.; VILAS, C., *Modelización de bosques mediante diagramas de Voronoi*, 2018.
<https://www.infor.uva.es/egc07/articulos/28.pdf>
- [4] MOYA CARRASCO, Á., *Generación de trayectorias en tiempo real a partir de diagramas de Voronoi*, 2016.
https://idus.us.es/xmlui/bitstream/handle/11441/44404/TFG_Angel_Moya_Carrasco.pdf?sequence=1&isAllowed=y
- [5] TRINCHET ALMAGUER, D; PÉREZ ROSÉS, H., *Algoritmo para solucionar el problema inverso generalizado de Voronoi*, 2007.
<http://www.redalyc.org/pdf/3783/378343634005.pdf>
- [6] GARCÍA, O., *The state-space approach in growth modelling*, 1994.
https://www.researchgate.net/publication/237870252_The_state-space_approach_in_growth_modelling
- [7] ABC., *El diagrama de Voronoi, la forma matemática de dividir el mundo*, 2017.
https://www.abc.es/ciencia/abci-diagrama-voronoi-forma-matematica-dividir-mundo-201704241101_noticia.html

A Diagrames de Voronoi

Quan es parla dels diagrames de *Voronoi* es fa referència a una estructura geomètrica que representa informació de proximitat sobre un conjunt de punts. Els diagrames de *Voronoi* són una de les estructures fonamentals dins de la Geometria Computacional, d'alguna manera emmagatzemen tota la informació referent a la proximitat entre punts.

Però què és un diagrama de *Voronoi*? Per tal d'entendre millor el concepte introduirem els diagrames de *Voronoi* sobre el pla.

En la següent *Figura ??* podem veure que el punt x està dins de la regió de punts més propers a p . Aquesta regió és la regió de *Voronoi* del punt p , i p és el punt generador de la regió.

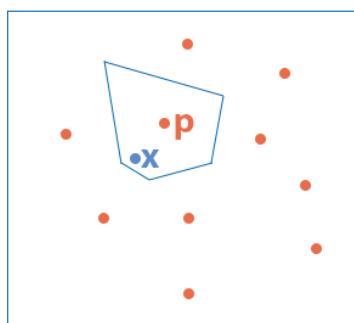


Figura 13: Regió *Voronoi*

La unió de totes les regions de *Voronoi* formen el diagrama de *Voronoi*, tal i com es pot veure a la *Figura ??*

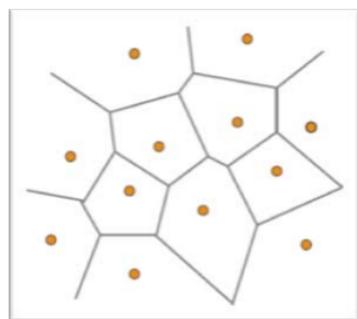


Figura 14: Diagrama de *Voronoi*

B Algoritme A*

La primera operació a realitzar, és dividir l'espai en caselles. En el nostre cas les caselles són quadrades, però realment es pot adaptar l'algoritme de dividir l'espai de la forma que es desitgi (triangles, hexàgons, etc.). A partir d'ara cada casella serà una cel·la espacial (es crearà una estructura de dades que la representi) relacionada amb els seus adjacents i amb el seu "pare"; el pare serà la cel·la per la qual hem arribat a tenir en compte l'actual com a possible candidata a formar part del camí final.

Aspectes a tenir en consideració:

1. Tenir dues llistes de cel·les: una **oberta** i una **tancada**. A la llista **oberta** anirem introduint cel·les que hem d'avaluar, per veure si són bones candidates a formar part del camí final. A la llista **tancada** introduirem les cel·les ja evaluades.
2. L'**avaluació de les cel·les** es fa en base a dos factors: la longitud del camí més curt per arribar des del punt d'origen a la cel·la actual, i l'estimació del camí més curt que podria quedar per arribar a la destinació. L'estimació pel destí es fa usant una funció **heurística**, en concret se sol usar la **Distància Manhattan**, que essencialment consisteix a explorar totes les cel·les que separen el destí vertical i horitzontalment, sense moviments diagonals; com si estiguéssim recorrent "pomes" en una gran ciutat i no poguéssim travessar-les, només vorejar-les (d'aquí ve el nom).

L'algoritme té diferents passos:

1. Afegim la cel·la origen a la llista oberta.
2. Agafem el primer element de la llista oberta i el traitem i l'inserim en la llista tancada.
3. Agafem les cel·les adjacents a la cel·la extreta.
4. Per a cada cel·la adjacent:

- Si la cel·la és el destí, hem acabat. Recorrem inversament la cadena de pares fins a arribar a l'origen per obtenir el camí.
 - Si la cel·la representa un mur o terreny infranqueable; la ignorem.
 - Si la cel·la ja és a la llista tancada, la ignorem.
 - Si la cel·la ja és a la llista oberta, comprovem si la seva nova G (ho veurem més endavant) és millor que l'actual, i en aquest cas recalculem factors (ho veurem més endavant) i posem com a pare de la cel·la, la cel·la extreta. En cas que no sigui millor, la ignorem.
 - Per a la resta de cel·les adjacents, els establim com a pare la cel·la extreta i recalculem factors. Després les afegim a la llista oberta.
5. Ordenem la llista oberta. La llista oberta és una llista ordenada de forma ascendent en funció del factor F de les cel·les.
6. Tornar al pas 1.

Factors a tenir en consideració:

Cada cel·la té 3 factors. G, H i F.

- **G:** És el cost d'anar des de la cel·la origen fins a l'actual. Cada vegada que ens moguem un pas en horitzontal o vertical, afegirem 10 punts de cost. Cada vegada que ens moguem en diagonal, afegirem 14. Per què 14? Perquè encara que geomètricament la proporció exacta hauria de ser $14,14213 (\sqrt{10 * 10 + 10 * 10})$, 14 és una bona aproximació sencera que ens farà guanyar velocitat en evitar l'ús de coma flotant.
- **H:** És la distància mínima i optimista, sense usar diagonals, que resta fins a la destinació. L'heurística basada en Distància Manhattan de la qual hem parlat abans.
- **F:** És la suma de **G** i **H**.

C Algorithme latitud longitud

```
public class MapService {  
    // CHANGE THIS: the output path of the image to be created  
    private static final String IMAGE_FILE_PATH = "/some/user/path/map.png";  
  
    // CHANGE THIS: image width in pixel  
    private static final int IMAGE_WIDTH_IN_PX = 300;  
  
    // CHANGE THIS: image height in pixel  
    private static final int IMAGE_HEIGHT_IN_PX = 500;  
  
    // CHANGE THIS: minimum padding in pixel  
    private static final int MINIMUM_IMAGE_PADDING_IN_PX = 50;  
  
    // formula for quarter PI  
    private final static double QUARTERPI = Math.PI / 4.0;  
  
    // some service that provides the county boundaries data in longitude and latitude  
    private CountyService countyService;  
  
    public void run() throws Exception {  
        // configuring the buffered image and graphics to draw the map  
        BufferedImage bufferedImage = new BufferedImage(IMAGE_WIDTH_IN_PX,  
                                                       IMAGE_HEIGHT_IN_PX,  
                                                       BufferedImage.TYPE_INT_RGB);  
  
        Graphics2D g = bufferedImage.createGraphics();  
        Map<RenderingHints.Key, Object> map = new HashMap<RenderingHints.Key, Object>();  
        map.put(RenderingHints.KEY_INTERPOLATION, RenderingHints.VALUE_INTERPOLATION_BICUBIC);  
        map.put(RenderingHints.KEY_RENDERING, RenderingHints.VALUE_RENDER_QUALITY);  
        map.put(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);  
        RenderingHints renderHints = new RenderingHints(map);  
        g.setRenderingHints(renderHints);  
  
        // min and max coordinates, used in the computation below  
        Point2D.Double minXY = new Point2D.Double(-1, -1);  
        Point2D.Double maxXY = new Point2D.Double(-1, -1);  
  
        // a list of counties where each county contains a list of coordinates that form the county boundary  
        Collection<Collection<Point2D.Double>> countyBoundaries = new ArrayList<Collection<Point2D.Double>>();  
  
        // for every county, convert the longitude/latitude to X/Y using Mercator projection formula
```

```

for (County county : countyService.getAllCounties()) {
    Collection<Point2D.Double> lonLat = new ArrayList<Point2D.Double>();

    for (CountyBoundary countyBoundary : county.getCountyBoundaries()) {
        // convert to radian
        double longitude = countyBoundary.getLongitude() * Math.PI / 180;
        double latitude = countyBoundary.getLatitude() * Math.PI / 180;

        Point2D.Double xy = new Point2D.Double();
        xy.x = longitude;
        xy.y = Math.log(Math.tan(QUARTERPI + 0.5 * latitude));

        // The reason we need to determine the min X and Y values is because in order to draw the map,
        // we need to offset the position so that there will be no negative X and Y values
        minXY.x = (minXY.x == -1) ? xy.x : Math.min(minXY.x, xy.x);
        minXY.y = (minXY.y == -1) ? xy.y : Math.min(minXY.y, xy.y);

        lonLat.add(xy);
    }

    countyBoundaries.add(lonLat);
}

// readjust coordinate to ensure there are no negative values
for (Collection<Point2D.Double> points : countyBoundaries) {
    for (Point2D.Double point : points) {
        point.x = point.x - minXY.x;
        point.y = point.y - minXY.y;

        // now, we need to keep track the max X and Y values
        maxXY.x = (maxXY.x == -1) ? point.x : Math.max(maxXY.x, point.x);
        maxXY.y = (maxXY.y == -1) ? point.y : Math.max(maxXY.y, point.y);
    }
}

int paddingBothSides = MINIMUM_IMAGE_PADDING_IN_PX * 2;

// the actual drawing space for the map on the image
int mapWidth = IMAGE_WIDTH_IN_PX - paddingBothSides;
int mapHeight = IMAGE_HEIGHT_IN_PX - paddingBothSides;

// determine the width and height ratio because we need to magnify the map to fit into the given image dimension
double mapWidthRatio = mapWidth / maxXY.x;

```

```

        double mapHeightRatio = mapHeight / maxXY.y;

        // using different ratios for width and height will cause the map to be stretched. So, we have to determine
        // the global ratio that will perfectly fit into the given image dimension
        double globalRatio = Math.min(mapWidthRatio, mapHeightRatio);

        // now we need to readjust the padding to ensure the map is always drawn on the center of the given image dimension
        double heightPadding = (IMAGE_HEIGHT_IN_PX - (globalRatio * maxXY.y)) / 2;
        double widthPadding = (IMAGE_WIDTH_IN_PX - (globalRatio * maxXY.x)) / 2;

        // for each country, draw the boundary using polygon
        for (Collection<Point2D.Double> points : countyBoundaries) {
            Polygon polygon = new Polygon();

            for (Point2D.Double point : points) {
                int adjustedX = (int) (widthPadding + (point.getX() * globalRatio));

                // need to invert the Y since 0,0 starts at top left
                int adjustedY = (int) (IMAGE_HEIGHT_IN_PX - heightPadding - (point.getY() * globalRatio));

                polygon.addPoint(adjustedX, adjustedY);
            }

            g.drawPolygon(polygon);
        }

        // create the image file
        ImageIO.write(bufferedImage, "PNG", new File(IMAGE_FILE_PATH));
    }
}

```