RELAZIONE PROGETTO – CORSO "GESTIONE DELL'INFORMAZIONE GEO-SPAZIALE" –

Studente: **Mattia Marchionna**

Matricola: **945736**

E-mail: **mattia.marchionna@studenti.unimi.it**

Testo 2

1. Implementare l'algoritmo di ***StayPoint Detection*** – discusso nel corso - per l'individuazione di soste significative in una traiettoria. Ogni sosta ha un identificatore univoco. Usare un linguaggio di programmazione a scelta. L'articolo di riferimento è riportato nel seguito.

2. Acquisire una traiettoria arbitrariamente lunga che comprenda delle soste e testare l'algoritmo. La durata e l'estensione spaziale delle soste sono a scelta dello studente. Si richiede di visualizzare la sequenza di soste.

In seguito, descriverò le soluzioni adottate per entrambi i punti presenti nel testo sopra citato.

Soluzione

Uno stay point S rappresenta una regione geografica in cui un utente rimane per un po' di tempo. Pertanto, ogni punto di permanenza ha il suo significato semantico. Ad esempio, i luoghi di vita e di lavoro, il ristorante e il centro commerciale che visitiamo, il luogo in cui viaggiamo, ecc.

Usiamo la longitudine e la latitudine dei punti GPS all'interno della regione per costruire uno stay point. In genere, tali "stay point" si verificano quando l'individuo si aggira in alcuni luoghi, come un parco, un campus, ecc.

I dati raccolti dai dispositivi GPS costituiscono una sequenza di punti $P = \{p1, p2, p3, ..., pn\}$. Ogni punto $pi \in P$ è caratterizzata da una longitudine ($pi.\ Lngt$), una latitudine ($pi.\ Lat$) e da uno timestamp ($pi.\ T$). Si può notare dalla figura 1 che è possibile collegare i punti GPS ordinati temporalmente, in maniera tale da ottenere una traiettoria GPS.

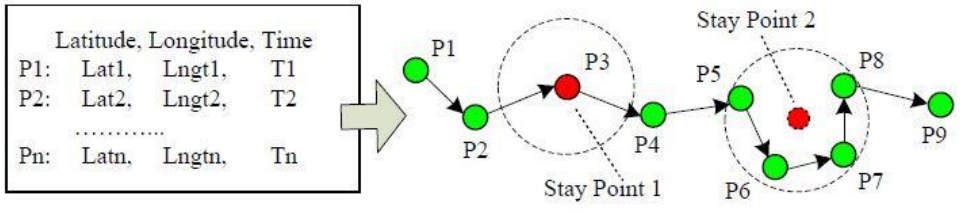

Figura 1

PSEUDOCODICE ALGORITMO STAY POINT DETECTION:

Input:

- un insieme di punti *P*
- una soglia di distanza *distThreh*
- una soglia di intervallo di tempo *timeThreh*

Output:

- un insieme di stay points *SP*={*S*}

```
1. i=0, pointNum = |P|;
2. while i < pointNum do,
3.      j:=i+1;
4.      while j < pointNum do,
5.           dist=Distance(pi, pj);
6.      if dist > distThreh then
7.           ΔT=pj.T-pi.T;
8.           if ΔT>timeThreh then
9.                S.coord=ComputMeanCoord({pk | i<=k<j})
10               S.arvT= pi.T; S.levT=pj.T ;
11.              SP.insert(S);
12.          i:=j; break;
13.     j:=j+1;
14. return SP
```

In pratica, l'algoritmo "Stay Point Detection" non fa altro che considerare inizialmente un punto Pi della traiettoria (alla prima iterata sarà, ovviamente, il primo punto) e per ogni punto Pj successivo a Pi confronta la distanza tra i due; se quest'ultima risulta essere minore della soglia di distanza data in input all'algoritmo, allora si considera il punto Pj successivo a quello precedente, altrimenti, viene calcolato l'intervallo di tempo tra Pj e Pi; se tale intervallo è maggiore della soglia di intervallo di tempo, l'algoritmo ha individuato uno Stay Point contenente tutti i punti Pk compresi tra Pi e Pj, incluso Pi ed escluso Pj. Il successivo punto Pi sarà Pj.

L'algoritmo procede fintantoché ci sono ancora punti Pi della traiettoria da analizzare.

Come linguaggio di programmazione da utilizzare per implementare l'algoritmo SPD si è deciso di utilizzare il Python, per diversi motivi; infatti è possibile:

- avviare automaticamente codice Python all'avvio di QGIS
- esegui comandi nella console Python all'interno di QGIS

- creare ed usare plugin in Python
- Creare un'applicazione personalizzata basata sulle API di QGIS

Inoltre, è possibile sfruttare la console Python integrata in QGIS, accessibile dal menu *Plugins ▸ Python Console*.

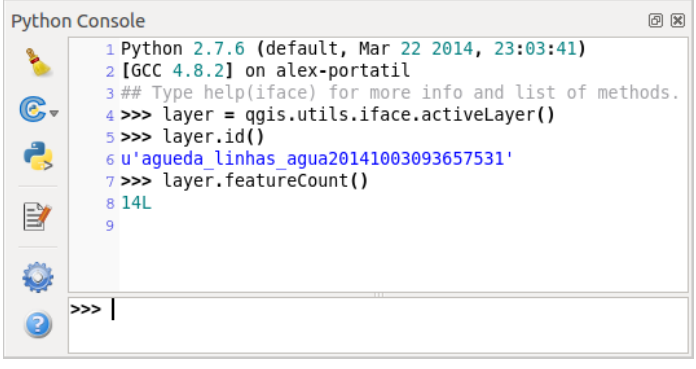

Figura 2

Come si può notare dalla figura 2, per l'interazione con l'ambiente di QGIS, esiste una variabile *iface*, che è un'istanza della classe *QgsInterface*, un'interfaccia che consente di accedere alla mappa, ai menu, ai pannelli e alle parti di QGIS.

Esaminiamo brevemente il codice Python:

```
def rand_color():
    return "#" + "".join(random.sample("0123456789abcdef", 6))

# Haversine Formule
def calculate_distance(lat1,lon1,lat2,lon2):
    R = 6373.0
    lat1 = radians(lat1)
    lon1 = radians(lon1)
    lat2 = radians(lat2)
    lon2 = radians(lon2)
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = (sin(dlat/2))**2 + cos(lat1) * cos(lat2) * (sin(dlon/2))**2
    c = 2 * atan2(sqrt(a), sqrt(1-a))
    distance = R * c
    return distance
```

Figura 3

La funzione rand_color() genera, come si può comprendere dal nome, la stringa esadecimale di colore casuale. Ci sarà utile per categorizzare gli eventuali stay point rilevati. (Figura 3)

Mentre, la funzione calculate_distance() riceve in input la latitudine e la longitudine di due punti e ne calcola la distanza, utilizzando la formula di Haversine. (Figura 3)

```
def categorized(stay_point, stay_point_time, stay_point_distance):
    new_fn = 'C:\\Users\\matti\\Desktop\\Progetto GIG\\StayPoint.shp'
    layerFields = QgsFields()
    layerFields.append(QgsField('id', QVariant.Int))
    layerFields.append(QgsField('id_SP', QVariant.Int))
    writer = QgsVectorFileWriter(new_fn, 'UTF-8', layerFields, QgsWkbTypes.Point, QgsCoordinateReferenceSystem('EPSG:4326'), 'ESRI Shapefile')
    feat = QgsFeature()
    i = 0
    j = 0
    for s in stay_point:
            print(f'SP {j}: from {stay_point_time[j][0]} to {stay_point_time[j][1]}, range: {stay_point_distance[j] * 1000} m')
            for p in s:
                feat.setGeometry(QgsGeometry.fromPointXY(p))
                feat.setAttributes([i, j])
                writer.addFeature(feat)
                i+=1
            j+=1
    lyr = iface.addVectorLayer(new_fn, 'StayPoint', 'ogr')
    del(writer)
    d = dict()
    for n in range(0,j):
        d[n] = list()
    feats = lyr.getFeatures()
    for f in feats:
        geom = f.geometry()
        x = geom.asPoint()
        d[f['id_SP']].append(x)
    targetField = 'id_SP'
    rangeList = []
    opacity = 1
    min = 0
    max = len(d)
    i = 0
    minVal = min
    while i < max:
        lab = f'Stay Point {i}'
        maxVal = minVal+1
        rangeColor = QtGui.QColor(rand_color())
        symbol2 = QgsSymbol.defaultSymbol(lyr.geometryType())
        symbol2.setColor(rangeColor)
        symbol2.setOpacity(opacity)
        range2 = QgsRendererRange(minVal, maxVal-1, symbol2, lab)
        rangeList.append(range2)
        minVal+=1
        i+=1
    groupRenderer = QgsGraduatedSymbolRenderer('', rangeList)
    groupRenderer.setMode(QgsGraduatedSymbolRenderer.EqualInterval)
    groupRenderer.setClassAttribute(targetField)
    lyr.setRenderer(groupRenderer)
    QgsProject.instance().addMapLayer(lyr)
```

Figura 4

La funzione categorized(stay_point, stay_point_time, stay_point_distance) categorizza i vari stay_points, assegnando ad ognuno di essi un colore diverso, in maniera tale da distinguerli nella maniera più opportuna. (Figura 4)

```
def calculate_stay_points(fn, p_dist, minuti):
    p_dist = p_dist/1000 # conversion from meters to kilometers
    layer = iface.addVectorLayer(fn, '', 'ogr')
    feats = layer.getFeatures()
    p_time = minuti * 60 # seconds
    stay_point = list()
    stay_point_time = list()
    stay_point_distance = list()
    points = []
    points_time = []
    S = list()
    for feat in feats:
        geom = feat.geometry()
        x = geom.asPoint()
        points.append(x)
        points_time.append(feat['time'])

    flag = True
    i = 0
    while i < len(points):
        flag = True
        j = i+1
        while j < len(points):
            if calculate_distance(points[i].y(), points[i].x(), points[j].y(), points[j].x()) > p_dist:
                flag = False
                d2 = datetime.datetime.strptime(points_time[j-1], '%Y/%m/%d %H:%M:%S.%f')
                d1 = datetime.datetime.strptime(points_time[i], '%Y/%m/%d %H:%M:%S.%f')
                if (d2 - d1).seconds > p_time:
                    for index in range(i, j):
                        S.append(points[index])
                    stay_point_time.append([points_time[i], points_time[j-1]])
                    stay_point_distance.append(calculate_distance(points[i].y(), points[i].x(), points[j-1].y(), points[j-1].x()))
                    stay_point.append(S)
                    S = list()
                i = j
                break
            j+=1
        if flag:
            i+=1

    categorized(stay_point, stay_point_time, stay_point_distance)
```

Figura 5

La funzione calculate_stay_points(fn, p_dist, minuti), calcola, appunto, gli stay points della traiettoria ricevuta in input, utilizzando l'algoritmo descritto in precedenza attraverso lo pseudocodice. Inoltre utilizza la funzione categorized(stay_point, stay_point_time, stay_point_distance) per categorizzare il risultato. (Figura 5)

```
calculate_stay_points('C:\\Users\\matti\\Desktop\\Progetto gestione informazione geospaziale\\Traiettorie2\\traj.shp', 100, 1.2)
```

Figura 6

La seguente istruzione permette di chiamare la funzione stay_point_detection(fn, p_dist, minuti), che farà quanto descritto sopra. (Figura 6)

La funzione prende in input il path in cui è presente la traiettoria da analizzare, la soglia di distanza espressa in metri e la soglia di intervallo di tempo espressa in minuti.

Il risultato finale sarà il seguente:

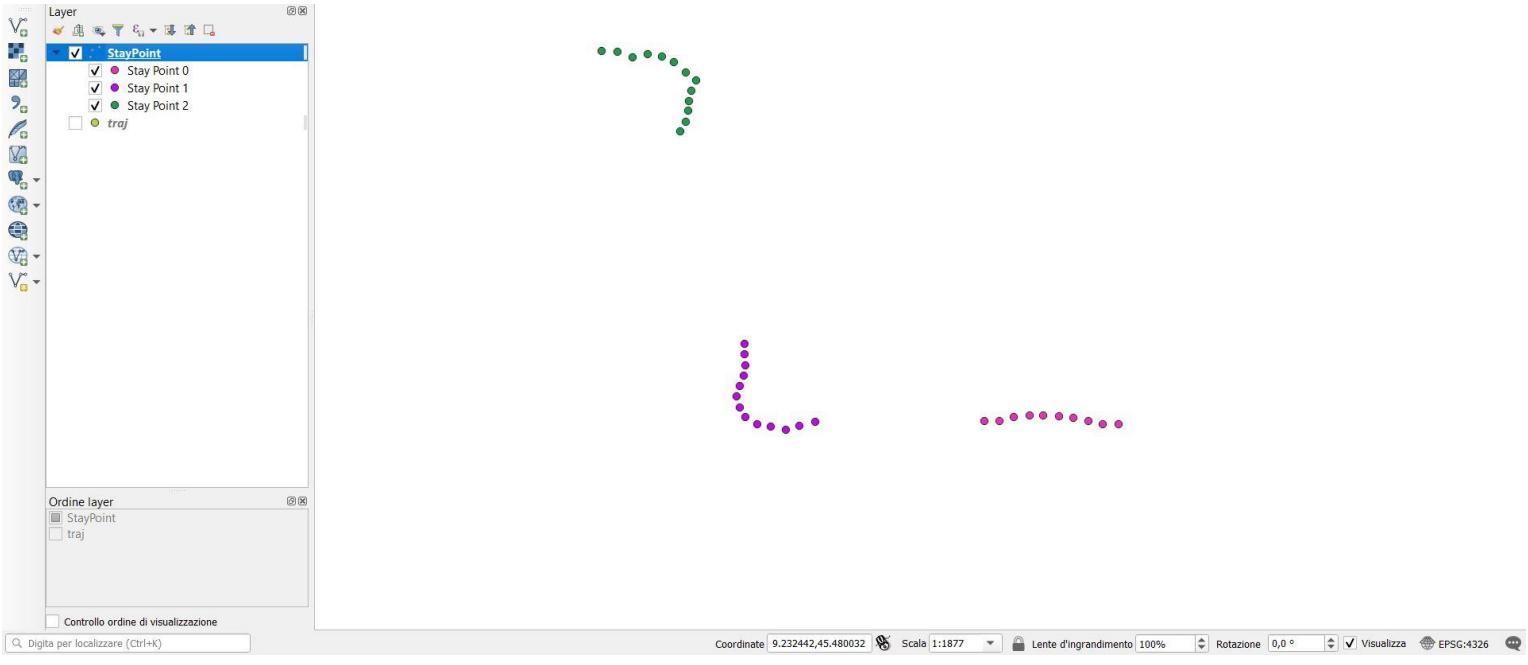

Figura 7

Per l'acquisizione della traiettoria è stata utilizzata l'app consigliata durante il corso dalla Professoressa M.L. Damiani: Geo Tracker, scaricabile dal seguente link:

https://play.google.com/store/apps/details?id=com.ilyabogdanovich.geotracker&hl=it

Icona App Geo Tracker

Figura 8