POLITECNICO DI TORINO

Dipartimento di Ingegneria Gestionale e della Produzione
Corso di Laurea in Ingegneria Gestionale
Classe L-8 Ingegneria dell'Informazione

Cammini minimi nella città di New York

Relatore
Prof. Fulvio Corno

Candidato
Forcignanò Walter (s223216)

Anno Accademico 2017/2018

Indice

Proposta di progetto

Studente proponente

s223216 Walter Forcignanò

Titolo della proposta

Cammini minimi dei Taxi all'interno della città di New York.

Descrizione del problema proposto

Il problema che si vuole risolvere è quello di individuare all'interno di un grafo, costruito analizzando i percorsi dei taxi nella città di New York, i cui dati sono ricavati da un dataset esistente, il percorso che minimizza il tempo di percorrenza o i metri da percorrere date due posizioni inizialmente scelte dall'utente. Nel caso in cui i due luoghi selezionati non siano collegati, il programma visualizza un messaggio di errore.

Nel dettaglio, dati tutti possibili percorsi effettuati dai taxi, individuare tramite un algoritmo ricorsivo la soluzione che porta alla ottimizzazione della funzione obiettivo rappresentata dal tempo totale di percorrenza oppure dalla distanza percorsa in base alla scelta dell'utente.

Descrizione della rilevanza gestionale del problema

Il problema è rilevante dal punto di vista gestionale, in quanto l'ottimizzazione del tempo di percorrenza è alla base di qualsiasi sistema di controllo logistico, sia per il trasporto di persone, che di merci. Nella sua versione più completa, il problema potrebbe essere affrontato includendo altre variabili come l'usura del mezzo (a seconda delle strade percorse), il costo del carburante, la gestione dei turni e l'esperienza di viaggio (ad esempio, a parità di tempo ed altre condizioni, si potrebbe preferire percorsi più panoramici e rilevanti da un punto di vista turistico).

Descrizione dei data-set

La fonte dal quale verranno prelevati i dati è un data-set trovato in rete sul sito www.kaggle.com, è un data-set reale, il cui contenuto è una collezione di dati con varie informazioni, tra cui i cammini dei taxi all'interno della città di New York.

Link al file CSV da cui è tratto il data-set:
https://www.kaggle.com/maheshdadhich/strength-of-visualization-python-visuals-tutorial/data

Il nome del data-set è: New York City Taxi with OSRM

File CSV: fastest_routes_test.csv

Di ogni tratta effettuata dai taxi vi è una sua scomposizione del tragitto nelle varie vie della città che vengono attraversate, e vengono riportate alcune misurazioni tra cui il tempo di percorrenza per ogni step, la distanza in metri del singolo step, e le coordinate spaziali del luogo attraversato che rappresenteranno i nodi del grafo che verrà costruito.

I dati risultano essere grezzi, in particolare nel formato CSV, quindi per poter effettuare qualunque operazione, è necessario prima formattare i dati, rappresentati solitamente da liste, e verificare la loro integrità.

La particolarità del data-set è che i valori riportati in tabella sono delle liste separate da un carattere separatore nel dettaglio '|'.

In particolare, i dati che verranno utilizzati sono:

- number_of_steps: Numero di step effettuati nel percorso.
- street_for_each_step: Lista delle strade in cui avviene ogni manovra. Più manovre possono essere fatte sulla stessa via. Inoltre, la stessa via può apparire più volte.
- -total_travel_time: Tempo totale tra i due punti del grafo che rappresentano partenza e arrivo del percorso misurato in secondi.
- -distance_per_step: Distanza che c'è tra ogni singolo step misurata in metri.
- -travel_time_per_step: Tempo che intercorre per ogni step misurato in secondi.
- -step_location_list: Le coordinate di ogni azione che viene fatta durante il percorso.
- -step_maneuvers: L'azione eseguita sullo step.

Le possibili manovre sono:

- Turn: Svolta.
- New name: Non è effettuata alcuna svolta, ma il nome della strada cambia.
- Depart: Il percorso inizia.
- Arrive: Il percorso ha termine.
- Merge: Fusione all'interno di un'altra strada (e.g. entrare su una autostrada tramite una rampa)
- On ramp: Entrata in autostrada.
- Off ramp: Uscita dall'autostrada.
- fork: Biforcazione.
- End of road: La strada termina con un'intersezione a ' T '.
- continue: Svolta per rimanere sulla stessa strada.
- Roundabout: Rotonda.
- Rotary:Rotatoria (rotonda di grandi dimensioni).
- Roundabout turn: Piccola rotonda che può essere trattata come una regolare svolta.

Descrizione preliminare degli algoritmi coinvolti

Il principale algoritmo che verrà implementato sarà quello della ricorsione. In particolare, dati due punti della città di New York corrispondenti a due vie, identificare se queste sono collegate all'interno del grafo costruito sulla base dei percorsi dei taxi esistenti, e tra tutti i possibili cammini individuare quello che vada a minimizzare la distanza o il tempo di percorrenza, trovando quindi l'ottimo se esiste, altrimenti sarà visualizzato un messaggio di errore.

In particolare, l'esistenza del cammino tra due punti sarà individuata se una volta costruito il grafo dato dalle coordinate delle posizioni di ogni singolo step dei percorsi effettuati da ogni singolo taxi, esisterà un arco che collegherà i due nodi del grafo. Ciò sarà realizzato tramite un opportuno algoritmo di visita.

Costruzione di una struttura dati avanzata, un grafo pesato tenendo conto come peso il tempo di percorrenza del taxi o i metri percorsi tra una posizione e l'altra.

La formattazione dei dati dal formato CSV, in un formato che sia gestibile e che rispetti i principi della programmazione a oggetti.

Descrizione preliminare delle funzionalità previste per l'applicazione software

A livello generale l'utente avrà a disposizione un'interfaccia che presenterà tutte le possibili vie presenti nel database, potrà selezionare quale percorso si vuole effettuare e avviare, definiti i dati di partenza, ovvero le due località, l'algoritmo che calcolerà il cammino minimo effettuato da un taxi.

Descrizione dettagliata del problema affrontato

Il problema che viene affrontato in questa applicazione, affronta la tematica dell'ottimizzazione, che risulta circoscritta, nel presente elaborato, nella ricerca di un cammino minimo tra due posizioni scelte dall'utente, in cui la voce di costo da minimizzare risulta o il tempo di percorrenza o i metri da percorrere a seconda della scelta dell'utente.

L'ottimizzazione è una branca della matematica, che partendo da una problematica di natura pratica come questa, definisce un modello matematico con lo scopo di ricercare, mediante opportuni algoritmi, i punti di massimo o di minimo di una determinata funzione.

Il problema dell'ottimizzazione del tempo di percorrenza o della distanza percorsa è rilevante dal punto di vista gestionale, perché alla base di qualunque tipologia di sistema di controllo logistico, sia per il trasporto di persone, che di merci.

L'affrontare queste tematiche con strumenti opportuni, come applicazioni software realizzate mediante linguaggi di alto livello, come quello ad oggetti di Java, risulta un metodo utile ed efficace per effettuare scelte di natura operativa, che possono portare al risparmio di voci di costo. Nel caso in questione le voci potrebbero essere il carburante oppure semplicemente il tempo speso durante il tragitto.

L'ottimizzazione si concretizza nell'elaborato in questione in un problema di ricerca del cammino minimo all'interno di un grafo orientato e pesato e può essere formalizzato nel seguente modo:

Il grafo costituito da un certo numero di vertici V e archi E è definito come:

$$\mathrm{G} = (\mathrm{V}, \mathrm{E})$$

Il peso dell'arco è definito come:

$$\mathrm{w}\colon \mathrm{E} \rightarrow R+$$

La somma dei pesi di un cammino p, è data dalla somma degli archi che lo compongono secondo l'equazione:

$$w(p) = \sum_{(u,v)\epsilon p} w(u,v)$$

Il cammino minimo è rappresentato come:

$$\delta(u,v)$$

La stima corrente del cammino minimo per un nodo è rappresentata come:

$$d[u]$$

Se v non fosse raggiungibile da u, allora:

$$\delta(u, v) = \infty$$

Il predecessore all'interno del cammino minimo è definito come:

$$\pi[v] \leftarrow u$$

La maggior parte degli algoritmi per la ricerca del cammino minimo sono basati sulla tecnica del rilassamento che si ottiene ogni qual volta che si scopre che un determinato nodo risulta essere più facilmente raggiungibile attraversando un arco che ha un peso minore.

Considerando ad esempio un arco (u,v) con peso w il rilassamento si definisce come segue:

Relax (u, v, w):

if $\mathrm{d[v]} > \mathrm{d[u]} + \mathrm{w(u,v)}$
then
$\mathrm{d[v]} \leftarrow \mathrm{d[u]} + \mathrm{w(u,v)}$
$\pi[\mathrm{v}] \leftarrow \mathrm{u}$

Applicando in maniera iterativa tutte le possibili sequenze di rilassamento si può ottenere un albero dei cammini minimi per ogni nodo del grafo.

La ricerca ed il calcolo del cammino ottimo devono però essere realizzati tramite un opportuno algoritmo. Questo deve essere efficiente, in quanto avendo lo spazio di tutte le possibili soluzioni, una cardinalità esponenziale, non ci si può limitare ad enumerarle tutte, vista anche la grossa mole di dati presente nel data-set.

Così la scelta su quale tipologia di algoritmo utilizzare è ricaduta sull'algoritmo di Dijkstra, algoritmo ricorsivo ideato per affrontare questa tipologia di problema, e poiché il grafo realizzato non presenta archi di peso negativo, unica limitazione per il corretto funzionamento dell'algoritmo, risulta idoneo per l'applicazione software realizzata.

La potenzialità della soluzione di Dijkstra risulta essere molto alta, in quanto è in grado di analizzare e riportare tutti gli alberi di peso minimo a partire da un nodo scelto come radice e definendo la destinazione, è possibile avere informazioni su quale sia il cammino minimo tra i due nodi scelti.

È la soluzione più efficace ed efficiente in quanto ha una complessità computazionale polinomiale, nel particolare:

$$O(|E| + |V| * \log |V|)$$

dove E rappresenta il numero di archi e V rappresenta il numero di vertici.

Proprio per le sue potenzialità e per la consuetudine per cui problematiche di questo tipo affiorano, l'algoritmo di Dijkstra risulta essere già implementato ed ottimizzato all'interno di una libreria di Java, che è stata importata all'interno del progetto.

Lo pseudo-codice dell'algoritmo è qui riportato:

```
 function Dijkstra (Graph, source):
2       dist[source] ← 0                                  // Initialization
3
4       create vertex set Q
5
6       for each vertex v in Graph:
7           if v ≠ source
8               dist[v] ← INFINITY                        // Unknown distance from source to v
9           prev[v] ← UNDEFINED                           // Predecessor of v
10
11          Q.add_with_priority(v, dist[v])
12
13
14      while Q is not empty:                             // The main loop
15          u ← Q.extract_min()                           // Remove and return best vertex
16          for each neighbor v of u:                     // only v that are still in Q
17              alt ← dist[u] + length(u, v)
18              if alt < dist[v]
19                  dist[v] ← alt
20                  prev[v] ← u
21                  Q.decrease_priority(v, alt)
22
23      return dist, prev
```

Per quanto riguarda il funzionamento dell'applicazione, gli input forniti dall'utente sono le due posizioni per le quali calcolare il cammino ottimo, di default in base ai metri da percorrere, ma su decisione dell'utente anche in base al tempo di percorrenza.

In base alla mole di dati, presenti nel database taxi_routes, che si carica all'interno dell'ambiente Java, viene costruito un grafo in cui i vertici rappresentano le posizioni, per le quali i taxi sono passati durante i loro cammini, e gli archi rappresentano il collegamento tra le due posizioni col relativo peso.

Se un percorso all'interno del grafo costruito è possibile, allora questo verrà calcolato tramite un opportuno algoritmo e verranno riportate a schermo delle statistiche sul percorso trovato, altrimenti verrà segnalato un messaggio di errore.

Attraverso un radio-button l'utente può scegliere che tipologia di informazioni ottenere, se riguardanti i metri da percorrere oppure il tempo di percorrenza scomposto in ore minuti e secondi.

L'output che viene generato e che risulta dipendente dal grafo costruito, è il cammino ottimo, ovvero la lista dei luoghi che devono essere attraversati per giungere dalla posizione di partenza alla destinazione che minimizza la funzione di costo.

Del cammino trovato vengono riportate statistiche in una tabella divisa in più colonne, all'interno di un grafico a torta, e in opportuni campi visibili all'utente. Nel particolare, all'interno della tabella le informazioni sono relative alle vie attraversate durante il percorso, le manovre effettuate, le coordinate geografiche delle posizioni attraversate, e in base alla scelta dell'utente, informazioni relative alla distanza percorsa oppure al tempo necessario per raggiungere la destinazione.

All'interno del grafo a torta che risulta essere espandibile al click dell'utente, vengono riportate invece informazioni relative alla distribuzione dei tempi o delle distanze percorse da parte del taxi nell'effettuare il cammino dalla posizione di partenza alla destinazione, scomponendo l'informazione per ogni singola via attraversata.

Descrizione del data-set utilizzato per l'analisi.

Il data-set è stato realizzato a partire da un file CSV di grandi dimensioni, in cui erano presenti informazioni grezze sui cammini dei singoli taxi all'interno della città di New York.

I dati sono stati ripuliti e inseriti all'interno di un database denominato "taxi_routes" in cui sono presenti due tabelle:

- una tabella "percorso" in cui sono presenti tutte le informazioni inerenti al percorso effettuato dai vari taxi.
- una tabella "step" che invece contiene tutte le informazioni per ogni step del percorso e che risulta la tabella più importante del data-set in quanto da questa sono determinati tramite delle opportune elaborazioni dei dati, il numero di nodi e archi del grafo che poi viene generato.

Nell'immagine sottostante sono riportate le colonne della tabella percorso.

#	Nome	Tipo di dati
1	**percorso_id**	**INT**
2	taxi_id	VARCHAR
3	starting_street	VARCHAR
4	end_street	VARCHAR
5	total_distance	DOUBLE
6	total_travel_time	DOUBLE
7	number_of_steps	INT

1. *percorso_id*: Identificativo del percorso.
2. *taxi_id*: Identificativo del taxi.
3. *starting_street*: Nome della strada di partenza da cui il percorso del taxi ha inizio.
4. *end_street*: Nome della strada in cui il percorso del taxi ha fine.
5. *total_distance*: Distanza totale misurata in metri percorsa.
6. *total_travel_time*: Tempo di percorrenza totale misurato in secondi.

7. *number_of_steps*: Numero di step effettuati durante il percorso.

Nell'immagine sottostante sono riportate le colonne della tabella step.

#	Nome	Tipo di dati
1	taxi_id	VARCHAR
2	percorso_id	INT
3	step_id	INT
4	street_for_each_step	VARCHAR
5	travel_time_per_step	DOUBLE
6	distance_per_step	DOUBLE
7	step_maneuvers	VARCHAR
8	step_direction	VARCHAR
9	step_location_list	VARCHAR

1. *taxi_id*: Identificativo del taxi.
2. *percorso_id*: Identificativo del percorso.
3. *step_id*: Identificativo dello step del percorso.
4. *street_for_each_step*: Nome della strada attraversata nello step del percorso.
5. *travel_time_per_step*: Tempo totale da percorrere per raggiungere lo step successivo.
6. *distance_per_step*: Distanza totale da percorrere per raggiungere lo step successivo.
7. *step_maneuvers*: Manovra effettuata allo step del percorso.
8. *step_direction*: Direzione presa per la manovra effettuata sullo step.
9. *step_location_list*: Coordinate per ogni manovra effettuata

Descrizione ad alto livello delle strutture dati e algoritmi utilizzati

L'applicazione è stata realizzata utilizzando il linguaggio Java e JavaFX.

Implementa tre pattern applicativi, nel particolare il pattern MVC (Model-View-Controller), il pattern DAO (Data-Access-Object) ed il pattern ORM (Object-Relational-Mapping).

La struttura dell'applicazione software è divisa in tre package:

- Main:
  contiene la classe Main per l'avvio dell'applicazione, le due classi controller(*Controller e PieChartController*) che definiscono i metodi che permettono l'interazione dell'utente e i file .fxml (*piechart e base_Interface*) per la definizione dell'interfaccia grafica.

- Database:
  contiene la classe ConnectDB utilizzata per la connessione al database, e la classe TaxiDAO che tramite query SQL permette il caricamento dei dati presenti nel database, all'interno dell'ambiente Java.
  Il package contiene anche le due classi *CostruzioneTabellaPercorso* e *CostruzioneTabellaStep* tramite le quali è possibile creare il DB a partire dal file CSV.
  Ed infine la classe TestConnection che permette una veloce verifica della connessione col DB.

- Model:
  contiene la classe omonima(*Model*) che possiede tutta la logica applicativa dell'applicazione. Contiene anche varie classi *Posizione*, *Percorso*, *Step*, con le relative IdMap, più due classi *newRow* e *PosizionePiuPeso* che invece sono più legate con la rappresentazione dei dati sull'interfaccia grafica.

Tutti i dati sono caricati all'avvio dell'applicazione tramite i metodi, presenti nella classe TaxiDAO, loadAllPercorsi e loadAllStep, tuttavia l'utente mediante uno slider presente nell'interfaccia, può decidere quante informazioni devono essere caricate.

```java
/**
 * Carica tutti i percorsi all'interno della percorsoIdMap passata come parametro.
 *
 * @param percorsoIdMap
 */
public List<Percorso> loadAllPercorsi(PercorsoIdMap percorsoIdMap) {

    List<Percorso> percorsi = new ArrayList<>();

    String sql = "select distinct percorso_id,taxi_id,starting_street,end_street,total_distance,total_travel_time,number_of_steps from percorso as p order by percorso_id";

    try {
        Connection conn = ConnectDB.getConnection();
        PreparedStatement st = conn.prepareStatement(sql);
        ResultSet rs = st.executeQuery();

        while (rs.next()) {

            percorsi.add(percorsoIdMap.get(new Percorso(rs.getInt("percorso_id"), rs.getString("taxi_id"),
                    rs.getString("starting_street"), rs.getString("end_street"), rs.getDouble("total_distance"),
                    rs.getDouble("total_travel_time"), rs.getInt("number_of_steps"))));

        }

        conn.close();
        return percorsi;

    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Errore connessione al database");
        throw new RuntimeException("Error Connection Database");
    }

}
```

```java
/**
 * Metodo che carica tutti gli step all'interno della stepIdMap. Il numero di
 * step caricati è pari a numeroStep.
 *
 * @param stepIdMap
 * @param numeroStep
 * @return
 */
public List<Step> loadAllStep(StepIdMap stepIdMap, int numeroStep) {

    List<Step> steps = new ArrayList<>();

    String sql = "select step_id,taxi_id,percorso_id,street_for_each_step,travel_time_per_step,distance_per_step,step_maneuvers,step_direction,step_location_list "
            + "from step order by percorso_id limit ?";

    try {
        Connection conn = ConnectDB.getConnection();
        PreparedStatement st = conn.prepareStatement(sql);
        st.setInt(1, numeroStep);
        ResultSet rs = st.executeQuery();

        while (rs.next()) {

            steps.add(
                    stepIdMap.get(new Step(rs.getString("taxi_id"), rs.getInt("percorso_id"), rs.getInt("step_id"),
                            rs.getString("street_for_each_step"), rs.getDouble("travel_time_per_step"),
                            rs.getDouble("distance_per_step"), rs.getString("step_maneuvers"),
                            rs.getString("step_direction"), rs.getString("step_location_list"))));
        }

        conn.close();
        return steps;

    } catch (SQLException e) {
        e.printStackTrace();
        System.out.println("Errore connessione al database");
        throw new RuntimeException("Error Connection Database");
    }

}
```

Nel secondo metodo il numero di step da caricare è definibile dall'utente perché la mole di dati presente risulta essere molto alta, ed è richiesta una maggiore disponibilità di RAM per poter caricarli tutti.

I dati vengono presi ed inseriti all'interno delle IdMap presenti nel package model per far sì che eseguendo il pattern ORM vengano eliminate eventuali duplicati di oggetti.

In particolare, vengono riempite le IdMap degli step e dei percorsi e successivamente all'interno dei singoli oggetti vengono inseriti i riferimenti gli uni degli altri.

In base ai dati inseriti viene generata una struttura di alto livello, ovvero un grafo che all'avvio dell'applicazione viene generato con l'informazione relativa a cento mila step e con pesi corrispondenti alla distanza percorsa in metri. Tuttavia, l'utente può scegliere il numero di step da inserire all'interno del sistema e quindi indirettamente gestire anche le dimensioni del grafo stesso, e può anche definire che tipologia di pesi assegnare agli archi del grafo e la scelta comprende oltre alla distanza in metri, anche il tempo di percorrenza in ore, minuti e secondi.

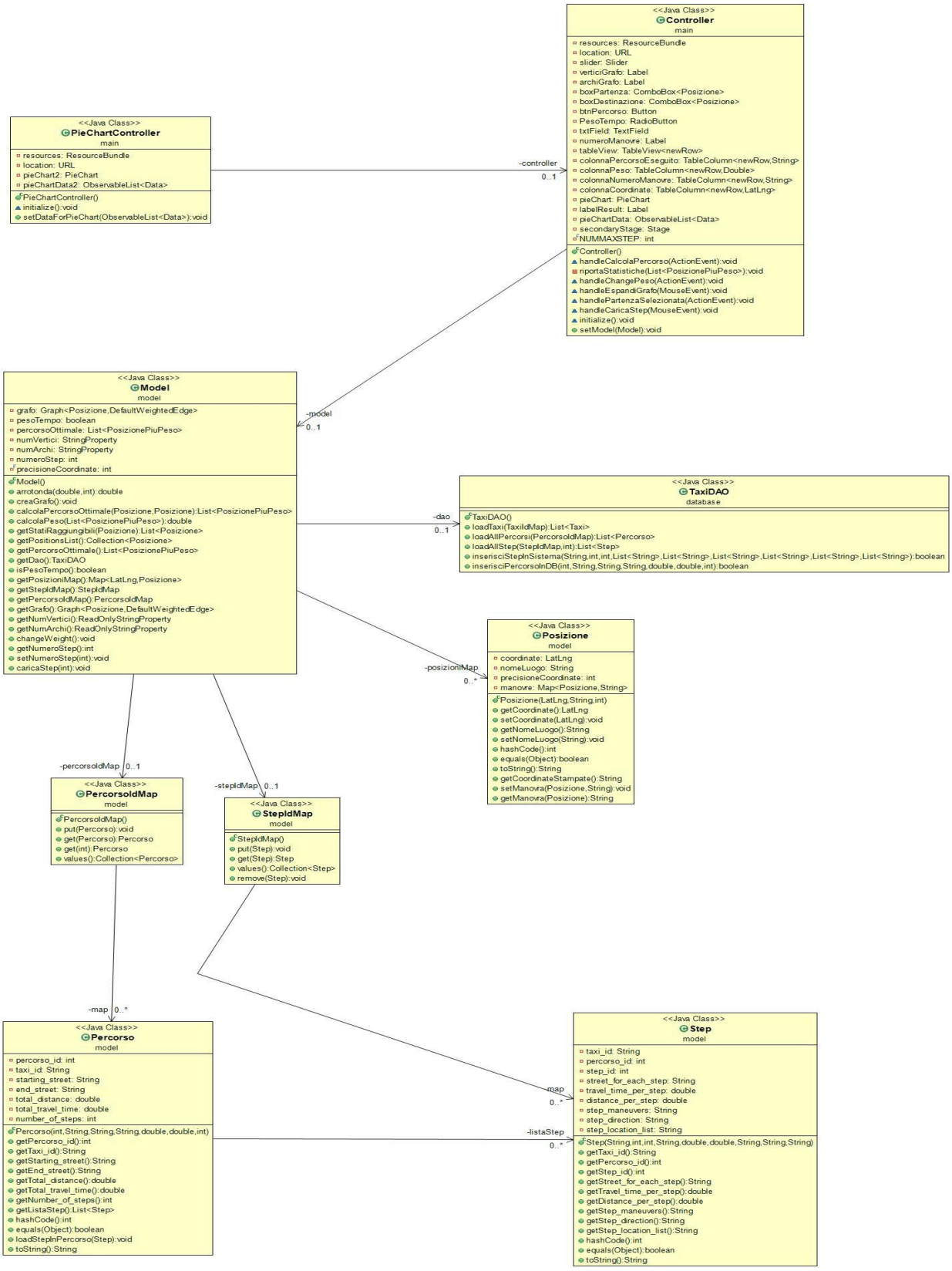
Tramite il metodo creaGrafo() della classe model, a partire dall'informazione relativa agli step caricati, è possibile definire i vertici che saranno degli oggetti di tipo Posizione che hanno la peculiarità di essere univocamente determinati dalle coordinate geografiche approssimate alla quarta cifra decimale per far sì che due manovre effettuate in un raggio di circa 11 metri vengano considerate come avvenute nella stessa posizione, e successivamente gli archi che permetteranno di creare i collegamenti tra le varie posizioni del grafo generato, andando quindi a realizzare a livello astratto una mappa stradale della città di New York.

Il grafo ha la peculiarità di essere semplice orientato e pesato, proprio per modellizzare la topologia della città, e questo ha anche il vantaggio di consentire l'utilizzo dell'algoritmo di Dijkstra per la ricerca del cammino minimo.

Altre strutture dati presenti sono liste, array, set e mappe, nelle quali vengono inserite svariate informazioni, utili per l'applicazione.

Vengono utilizzati algoritmi di ordinamento per far sì che le liste di oggetti siano ordinate secondo svariati criteri.

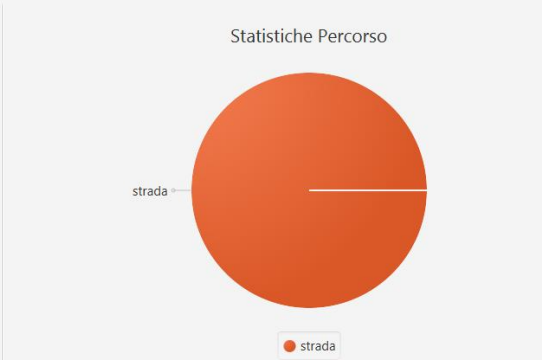
Diagramma delle classi delle parti principali dell'applicazione



Alcune videate dell’applicazione realizzata e link al video dimostrativo del software

Il link al video di presentazione dell'applicazione è:

Mappa stradale della città di New York



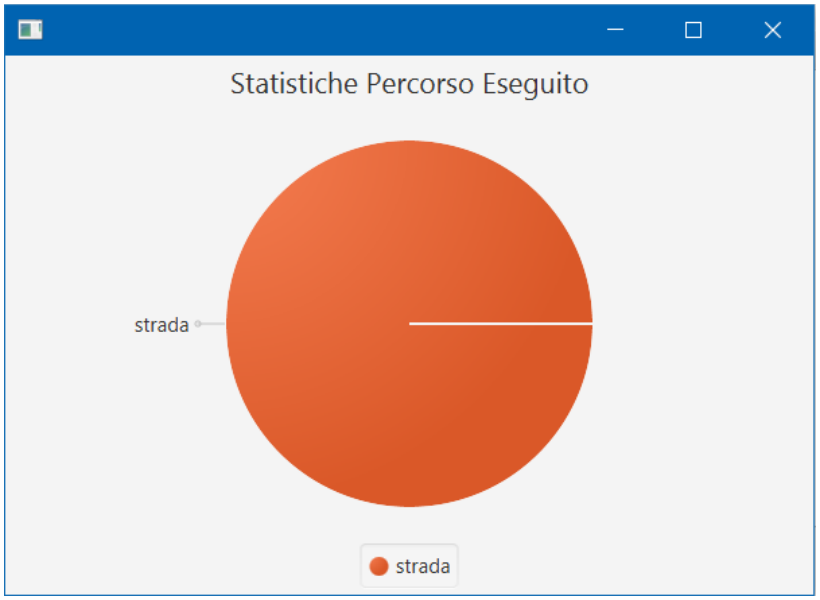
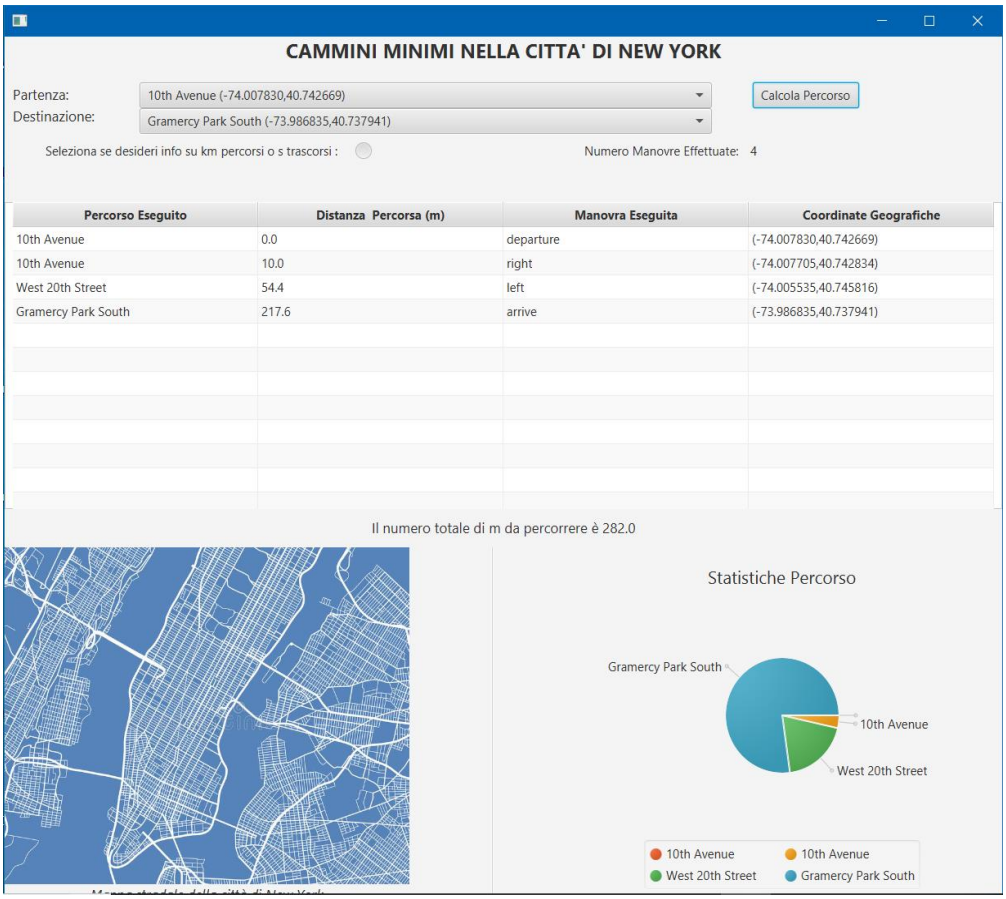
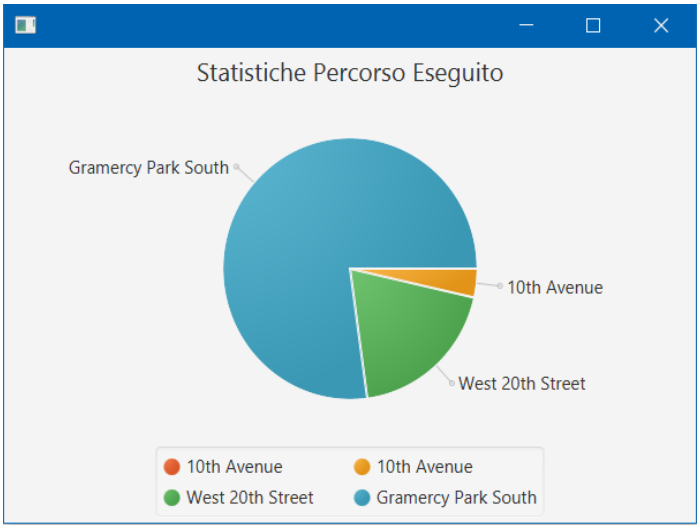


Tabelle con risultati sperimentali ottenuti

L'immagine sottostante invece rappresenta cosa avviene se selezionato una posizione di partenza e una di destinazione si preme il pulsante calcola percorso.



Cliccando sul grafico a torta, verrà aperta una nuova finestra con il grafo di grandi dimensioni.



Nell'immagine sottostante viene messa in evidenza la tabella con le informazioni inerenti al percorso che minimizza questa volta il tempo di percorrenza.

Percorso Eseguito	Distanza Percorsa (m)	Manovra Eseguita	Coordinate Geografiche
10th Avenue	0.0	departure	(-74.007830,40.742669)
10th Avenue	10.0	right	(-74.007705,40.742834)
West 20th Street	54.4	left	(-74.005535,40.745816)
Gramercy Park South	217.6	arrive	(-73.986835,40.737941)

Nell'immagine sottostante viene messa in evidenza la tabella con le informazioni inerenti il percorso che minimizza questa volta il tempo di percorrenza.

Percorso Eseguito	Tempo Percorrenza (s)	Manovra Eseguita	Coordinate Geografiche
10th Avenue	0.0	departure	(-74.007830,40.742669)
10th Avenue	21.2	right	(-74.007705,40.742834)
10th Avenue	81.0	left	(-74.007238,40.743470)
West 16th Street	270.4	left	(-74.004433,40.742289)
6th Avenue	825.1	left	(-73.995874,40.738686)
6th Avenue	223.2	right	(-73.994593,40.740442)
6th Avenue	74.8	right	(-73.994165,40.741031)
Park Avenue South	616.6	uturn	(-73.987767,40.738341)
Gramercy Park South	90.3	arrive	(-73.986835,40.737941)

Valutazioni sui risultati ottenuti e conclusioni.

I risultati ottenuti ovviamente risultano essere verosimili, tanto più attendibili quanto più è alta la mole di dati che viene caricata all'interno dell'applicazione, ma non rappresentano necessariamente il miglior cammino tra le due posizioni nella realtà, questo perché il grafo è costruito con dei dati basati solo su dei cammini effettuati da dei taxi all'interno della città di New York, quindi non tiene conto di svariati fattori come la percentuale di traffico sul percorso scelto oppure eventuali incidenti, lavori in corso su determinate strade, tutte informazioni che se aggiunte porterebbero ad un perfezionamento dell'applicazione.

A causa della mancanza di queste informazioni, l'applicazione vuole fornire un'idea indicativa della distanza minima da percorrere e del tempo necessario per percorrere un determinato percorso tra una posizione e l'altra della città di New York in condizioni indicativamente rientranti nella normalità.

Un punto di forza è sicuramente il fatto che un'applicazione come quella di Google Maps, fonda la sua struttura su un tipo di struttura di questo tipo, ovviamente, molto più ottimizzata e con una mole di dati di gran lunga superiore, ed inoltre questi vengono gestiti specifici calcolatori che effettuano un gran numero di operazioni al secondo.

Un altro punto di forza dell'applicazione è che si fonda su dei dati reali, i cammini di taxi all'interno della grande mela, il che rende i risultati ottenuti verosimili.

Un punto di debolezza è che l'applicazione per poter funzionare correttamente deve caricare in memoria un gran numero di informazioni che per i cammini di una singola città magari è ancora sostenibile, ma all'aumentare dei cammini possibili, richiede un grande spazio di memoria.

Per questo se venisse esteso a più città l'ideale sarebbe richiedere al database solo le informazioni che risultano necessarie per effettuare il cammino e quindi sarebbe necessario salvare all'interno dei database i cammini minimi tra le posizioni maggiormente attraversate.

Il vantaggio che si otterrebbe sarebbe quello di avere una applicazione più snella lato client, più carica lato server ma anche più sicura in quanto all'utente verrebbero fornite solo le informazioni richieste ed in questo modo si potrebbero evitare eventuali operazioni non lecite sui dati effettuate da un utente non benevolo.

In conclusione, possedendo una banca dati di dimensioni superiori, ovviamente l'applicazione potrebbe fornire dei risultati maggiormente corrispondenti con la realtà, ma l'applicazione per come implementata fornisce dati realistici.

Fonti:

- Link per informazioni su pseudocodice dell'algoritmo di Dijkstra:

  https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Pseudocode

- Link per informazioni inerenti alla ricerca del cammino minimo:

  https://elite.polito.it/files/courses/03FYZ/2018/slide/06-04-graphs-shortestpaths.pdf