



**Politecnico
di Torino**

Politecnico di Torino

Corso di Laurea in Ingegneria Gestionale

Classe L-8

A.A. 2022/2023

Sessione di Laurea Settembre 2023

Tesi di Laurea Triennale

Applicazione per la creazione di un itinerario turistico nella città di Torino

Relatore:
Prof. Fulvio Corno

Candidato:
Mattia Ghigo 284429

Indice

1 Proposta di progetto	1
1.1 Studente proponente	1
1.2 Titolo della proposta	1
1.3 Descrizione del problema proposto	1
1.4 Descrizione della rilevanza gestionale del problema	1
1.5 Descrizione dei data-set per la valutazione	1
1.6 Descrizione preliminare degli algoritmi coinvolti.....	2
1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software	2
2 Descrizione del problema affrontato	3
3 Descrizione del data-set.....	5
3.1 Alberghi	5
3.2 Luoghi di culto	6
3.3 Musei	6
3.4 Teatri.....	6
3.5 Altri luoghi	7
3.6 Toret	7
4 Descrizione delle strutture dati e degli algoritmi	8
4.1 Strutture dati utilizzate.....	8
4.1.1 Package controller	8
4.1.2 Package db	8
4.1.3 Package model.....	8
4.2 Algoritmi utilizzati.....	9
4.2.1 Algoritmo di ricerca	9
4.2.2 Algoritmo ricorsivo	12
5 Diagramma delle classi principali.....	14
6 Schermate dell'applicazione	16
7 Risultati sperimentali	18
7.1 Esempio 1	18
7.2 Esempio 2	19
7.3 Esempio 3	20
8 Valutazioni sui risultati ottenuti.....	21

1 Proposta di progetto

1.1 Studente proponente

S284429 Ghigo Mattia

1.2 Titolo della proposta

Applicazione per la creazione di un itinerario turistico nella città di Torino

1.3 Descrizione del problema proposto

Il problema riguarda la creazione di un itinerario nella città di Torino. Partendo da un albergo, scelto in funzione di quelli presenti all'interno di una lista creata in base alle richieste dell'utente, l'applicazione creerà un cammino passante tra vari punti di interesse (quali possono essere chiese, monumenti, teatri e molto altro) che rispetti i filtri inseriti ed altri vincoli preimpostati (ad esempio per evitare un elevato quantitativo di luoghi dello stesso genere).

1.4 Descrizione della rilevanza gestionale del problema

Dal punto di vista gestionale, l'applicazione creata potrà risultare utile alle aziende operanti nel settore turistico che, grazie ad esso, potranno minimizzare il tempo necessario per la programmazione di un itinerario e massimizzare così i profitti, potendo soddisfare un numero di clienti più elevato. Inoltre, risulterebbe utile anche al singolo utilizzatore, che non dovrà ricercare su decine di siti differenti che cosa vedere, quando e come. Per quanto riguarda la pianificazione di un viaggio, infatti, è spesso difficile scegliere con cura i luoghi di interesse, dato che la posizione dell'albergo influisce sui luoghi visitabili.

1.5 Descrizione dei data-set per la valutazione

I data-set utilizzati nel progetto derivano dal sito 'aperTO', ovvero il portale del comune di Torino contenente diversi dati riguardanti la città, e dall'associazione 'i love toret', che si occupa della tutela e della valorizzazione dei toret, le fontanelle verdi con testa di toro tipiche di Torino.

Riporto di seguito la lista dei database utilizzati con relativi link:

- alberghi (<http://aperto.comune.torino.it/dataset/alberghi-2020>)
- chiese (<http://aperto.comune.torino.it/dataset/chiese-e-altri-luoghi-di-culto>)
- luoghi storici (<http://aperto.comune.torino.it/dataset/luoghi-storici>)
- musei (<http://aperto.comune.torino.it/dataset/musei>)
- teatri (<http://aperto.comune.torino.it/dataset/teatri>)
- toret (database condiviso dallo staff di 'i love toret', <https://ilovetoret.it/api>)

I data-set relativi ai luoghi da visitare potrebbero venire uniti, al fine di creare un unico database complessivo contenente tutti i luoghi di interesse, riportando un attributo aggiuntivo che specifica la classe di appartenenza (chiese, luoghi storici, musei, teatri e così via) per un migliore filtraggio.

Verrebbero aggiunti, a mano, altri luoghi particolari che meritano di essere citati. Inoltre, verrebbe effettuato un attento controllo, al fine di eliminare tuple superflue (come, ad esempio, nel database relativo agli alberghi, dove sono presenti strutture fuori Torino), completare i campi lasciati vuoti (in alcuni casi, ad esempio, non sono presenti le informazioni relative alle coordinate,

campo molto utile per la creazione del grafo) ed altre piccole migliorie utili (quale, ad esempio, la durata di visita del luogo, rintracciata sul web).

1.6 Descrizione preliminare degli algoritmi coinvolti

Innanzitutto, mediante un algoritmo di ricerca, il software genererà una lista di alberghi che rispetti le preferenze espresse dall'utente, ad esempio il budget, le stelle dell'albergo, se è permesso l'ingresso ai cani e così via. Successivamente alla selezione dell'albergo, sarà possibile selezionare delle preferenze per la futura creazione dell'itinerario (livello di gradimento delle tipologie di luogo, tempo massimo di cui si dispone), a scelta avvenuta un algoritmo creerà un grafo tenendo conto dell'hotel e dei luoghi che rispettano i filtri. Il grafo generato avrà come vertici i luoghi visitabili e l'albergo (che sarà il punto di partenza, e di arrivo, dell'itinerario), e come archi i tragitti da un luogo ad un altro, pesati in base al tempo di spostamento tra i due vertici, calcolato tramite le coordinate (la velocità di spostamento da un luogo a quello successivo dipenderà dalla distanza tra essi, luoghi vicini saranno raggiungibili a piedi, per luoghi più distanti, invece, si potrà prendere uno dei molti mezzi pubblici disponibili per la città). Infine, tramite un algoritmo ricorsivo, si calcolerà l'itinerario migliore, ovvero quello con il maggior numero di luoghi che rispettino le scelte dell'utente.

1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'utente avrà a disposizione un'interfaccia dove dovrà effettuare delle scelte, in funzione delle quali verrà popolata una tendina contenente gli alberghi disponibili per la selezione. Successivamente si abiliteranno ulteriori campi per il filtraggio dei luoghi di interesse che si intende visitare, oltre ad un campo che permetta di inserire il tempo a disposizione per il tour. A scelte effettuate, verrà visualizzato l'elenco rappresentante l'itinerario, in un'area testuale, con tutte le informazioni necessarie.

2 Descrizione del problema affrontato

La nostra bella Italia è una delle mete più richieste dai turisti grazie al suo enorme patrimonio culturale: opere d'arte, monumenti, paesaggi, tradizioni, cucina e molto altro. Disponiamo infatti di luoghi senza eguali sia naturali che artificiali, come le innumerevoli città d'arte, luoghi affascinanti che però da visitare in modo esaustivo sono difficili, dato che comprendono numerosi luoghi di elevata importanza.

Per programmare un viaggio ci si può affidare ad un'agenzia viaggi oppure procedere alla creazione autonoma di un itinerario, non sempre però questo compito è di semplice risoluzione: trovare tutte le informazioni necessarie può richiedere svariate ore di lavoro dato che raramente si arriva ad una conclusione con una semplice ricerca sul web, molti siti riportano solamente le attrazioni più turistiche, tralasciandone di molto più caratteristiche ed attraenti. I luoghi che troviamo, inoltre, sono quasi sempre presenti in liste ordinate per nome o tipologia e, questo, durante la creazione di un itinerario può risultare scomodo, dato che per massimizzare un tour bisogna muoversi con intelligenza, minimizzando la durata degli spostamenti.

L'applicazione sviluppata tratta nello specifico la città di Torino, una delle più belle città presente nel nostro Paese, capoluogo del Piemonte e prima capitale d'Italia con un'importantissima storia alle spalle, sia antica che moderna, luogo che porta chi la visita a rimanere ammagliati dalla sua bellezza, tra residenze reali, palazzi, musei, parchi, ristoranti e caffè storici, viali alberati e vie porticate, colline ed ovviamente il fiume Po.

Il problema considerato all'interno di questo elaborato tratta la creazione di un itinerario per la città di Torino, che permette all'utente di visitare il maggior numero di luoghi d'interesse, partendo da un albergo scelto, in base alle proprie necessità e preferenze.

Inizialmente si dovrà selezionare un albergo di partenza scegliendo da una lista generata automaticamente, si potranno applicare dei filtri al fine di soddisfare, per quanto possibile, le esigenze dell'utilizzatore: prezzo che si è disposti a pagare per un pernottamento, numero di stelle minimo della struttura, massima distanza dal centro della città, se l'albergo dispone di camere adatte ai disabili, se sono ammessi animali domestici e se la struttura è bike friendly.

Successivamente si dovranno impostare dei filtri per quanto riguarda la disponibilità di tempo per il tour e le preferenze sui luoghi da visitare: livello di interesse nei luoghi di intrattenimento, nei luoghi di culto e nei musei.

A scelte effettuate l'applicazione produrrà un output contenente l'itinerario rispettante tutti i vincoli inseriti, se è stato possibile generarlo, inoltre creerà anche un secondo itinerario che in questo caso non rispetta i filtri ma massimizza i luoghi visitabili nel tempo dato. Verranno inoltre suggeriti dei luoghi particolari come un locale storico nelle immediate vicinanze dell'albergo scelto, un parco dove potersi rilassare immersi nella natura ed un toret dove potersi rinfrescare nelle calde giornate estive o anche solo riempire la propria borraccia.

Nello specifico, come sottolineato, l'applicazione mira a risolvere problemi legati alla generazione di un itinerario nella città di Torino, nulla vieta però una futura espansione del codice al fine di coprire così molteplici città e Paesi. Con calcolatori più potenti si potrà anche velocizzare il calcolo dell'output.

Questo progetto può risultare utile sia al singolo individuo sia alle agenzie di viaggio, queste ultime infatti, disponendo di questo applicativo, potrebbero minimizzare il tempo impiegato per la programmazione dei tour e, di conseguenza, massimizzare i profitti potendo soddisfare più clienti.

3 Descrizione del data-set

I dati utilizzati per la creazione dell'applicazione derivano da 5 differenti database, 4 di essi provengono dal sito web 'aperTO', il portale degli open data della città di Torino, mentre l'ultimo è stato gentilmente fornito da 'i love toret', associazione che si occupa della tutela e della valorizzazione dei toret, le fontanelle verdi con testa di toro tipiche di Torino.

Tutti i database sono stati revisionati e modificati al fine di renderli il più adatti possibile all'interfacciamento con l'applicazione. Sono stati aggiunti dati utili dove mancavano o dove il valore era 'null', alcune tuple sono state eliminate e, inoltre, per alcuni database, è stato aggiunto l'attributo relativo alla durata della visita, dato fondamentale per il corretto funzionamento dell'applicazione.

Per un migliore utilizzo è stato creato un database unico, chiamato 'provafinale', che contenesse tutti questi dati suddivisi in tabelle, in base alla loro tipologia.

3.1 Alberghi

Le informazioni relative agli alberghi derivano dalla tabella 'alberghi', contenente esattamente 179 tuple, ognuna di esse rappresentante un singolo albergo.

Ogni albergo possiede 10 attributi: identificativo numerico, nome, indirizzo, prezzo al giorno per una mezza pensione (informazione reperita tramite ricerca sul web, oppure nel caso in cui non sia stato possibile rintracciarlo, tramite il calcolo della media dei prezzi di hotel simili), numero di stelle, latitudine, longitudine, bike friendly (sì/no), disponibilità di camere per disabili (sì/no), animali ammessi (sì/no).

Inizialmente il database comprendeva più di 1900 alberghi, la maggior parte dei quali sparsi per tutto il Piemonte, ed ogni tupla aveva molti attributi non utili alla creazione dell'applicazione. Per questi motivi si è deciso di alleggerire la tabella finale, utilizzando query specifiche al fine di minimizzare il tempo necessario a tale scopo, ottimizzando così la lettura dei dati utili.

La fonte del database è 'aperTO'.


#	Nome	Tipo di dati
 1	ID	INT
2	Nome	VARCHAR
3	Indirizzo	VARCHAR
4	Mezza_pensio...	DOUBLE
5	Stelle	INT
6	Latitudine	VARCHAR
7	Longitudine	VARCHAR
8	Bike_friendly	INT
9	Disabili	INT
10	Animali_dome...	INT

Figura 1: struttura dati tabella 'alberghi'

3.2 Luoghi di culto

Le informazioni relative ai luoghi di culto derivano dalla tabella 'chiese', contenente 174 tuple, ognuna di esse rappresentante una singola chiesa.

Ogni chiesa possiede 6 attributi: identificativo numerico, nome, latitudine, longitudine, indirizzo, stima della durata della visita.

In questo caso sono stati eliminati solamente attributi non necessari, il numero di tuple non è variato.

La fonte del database è 'aperTO'.

#	Nome	Tipo di dati
 1	ID	INT
2	Nome	VARCHAR
3	Latitudine	VARCHAR
4	Longitudine	VARCHAR
5	Indirizzo	VARCHAR
6	Durata	INT

Figura 2: struttura dati tabella 'chiese'

3.3 Musei

Le informazioni relative ai musei derivano dalla tabella 'musei', contenente 47 tuple, ognuna di esse rappresentante un museo.

Ogni museo possiede 6 attributi: identificativo numerico, nome, latitudine, longitudine, indirizzo, stima della durata della visita (fondamentale per questa tipologia di luogo perché richiedono un'ampia disponibilità di tempo).

Anche in questo caso sono stati eliminati attributi non necessari, inoltre sono stati aggiunti alcuni importanti musei che mancavano nel database iniziale.

La fonte del database è 'aperTO'.

#	Nome	Tipo di dati
 1	ID	INT
2	Nome	VARCHAR
3	Latitudine	VARCHAR
4	Longitudine	VARCHAR
5	Indirizzo	VARCHAR
6	Durata	INT

Figura 3: struttura dati tabella 'musei'

3.4 Teatri

Le informazioni relative ai teatri derivano dalla tabella 'teatri', contenente 26 tuple, ognuna di esse rappresentante un teatro.

Ogni teatro possiede 6 attributi: identificativo numerico, nome, latitudine, longitudine, indirizzo, stima della durata della visita (purtroppo quasi tutti i teatri non sono visitabili internamente, se non durante le aperture dovute a spettacoli. Sono stati aggiunti ugualmente, dato che spesso si trovano all'interno di edifici storici, simbolo della città, e perciò meritevoli di essere visionati. Inoltre, citandoli all'interno dell'output, l'utente, se interessato, potrà controllare autonomamente la possibilità di visionare uno spettacolo, così da godere pienamente del luogo).

Non sono state effettuate modifiche importanti al database iniziale, solamente alcuni attributi sono stati eliminati.

La fonte del database è 'aperTO'.

#	Nome	Tipo di dati
 1	ID	INT
2	Nome	VARCHAR
3	Latitudine	VARCHAR
4	Longitudine	VARCHAR
5	Indirizzo	VARCHAR
6	Durata	INT

Figura 4: struttura dati tabella 'teatri'

3.5 Altri luoghi

Le informazioni relative a cinema, fontane, gallerie, locali storici, luoghi particolari, monumenti, parchi, piazze e ponti derivano dalla tabella 'altriluoghi', contenente 176 tuple, ognuna di esse rappresentante un luogo.

Ogni luogo presente in questa tabella possiede 7 attributi: identificativo numerico, tipo di luogo, nome, latitudine, longitudine, indirizzo, stima della durata della visita (non tutti i cinema presenti sono visitabili, ma anche in questo caso, come specificato per i teatri, molti di essi si trovano in edifici storici e meritano di essere visitati anche solo esternamente. L'utente può comunque acquistare autonomamente un biglietto per uno spettacolo così da visitare anche internamente la struttura).

Inizialmente il database era composto da sole 22 tuple e per questo motivo sono stati aggiunti ulteriori luoghi, soprattutto quelle particolarità che difficilmente un turista che visita per la prima volta Torino riesce a vedere, in quanto prevalentemente sconosciute.

La fonte del database è 'aperTO'.

3.6 Toret

Le informazioni relative ai toret derivano dalla tabella 'toretti', contenente 815 tuple, ognuna di esse rappresentante un toret.

Ogni toretto presente in questa tabella possiede 5 attributi: identificativo numerico, latitudine, longitudine, indirizzo, stima durata della visita.

Inizialmente il database conteneva più di 900 tuple ed una parte di queste, ovvero quella relativa ai toretti localizzati fuori Torino, è stata eliminata per velocizzare i calcoli effettuati dall'applicazione.

La fonte del database è 'i love toret'.

#	Nome	Tipo di dati
 1	ID	INT
2	Tipo	VARCHAR
3	Nome	VARCHAR
4	Latitudine	VARCHAR
5	Longitudine	VARCHAR
6	Indirizzo	VARCHAR
7	Durata	INT

Figura 5: struttura dati tabella 'altriluoghi'


#	Nome	Tipo di dati
 1	ID	INT
2	Latitudine	VARCHAR
3	Longitudine	VARCHAR
4	Indirizzo	VARCHAR
5	Durata	INT

Figura 6: struttura dati tabella 'toretti'

4 Descrizione delle strutture dati e degli algoritmi

4.1 Strutture dati utilizzate

L'applicazione è stata sviluppata in linguaggio Java, seguendo il pattern MVC (Model-View-Controller) ed il pattern DAO (Data-Access-Object) per l'accesso al database, al fine di disaccoppiare le diverse parti del codice per migliorarne la gestione. A tal fine il progetto è stato suddiviso in tre differenti packages: 'controller' che gestisce l'interfaccia utente, 'dao' per l'accesso al database e 'model' che contiene la logica applicativa.

4.1.1 Package controller

Il package, utile alla gestione dell'interfaccia utente, contiene 3 classi: EntryPoint, FXMLController e Main.

- EntryPoint gestisce il caricamento della scena e crea il controller per gestirla
- FXMLController permette all'interfaccia utente di interagire con il Model, trasformando gli input inseriti dall'utente in dati comprensibili dal Model e gli output prodotti dal Model in informazioni comprensibili dall'utente.
- Main avvia l'applicazione.

4.1.2 Package db

Anche in questo caso il package, creato per connettere Model e database rispettando le regole del pattern DAO, contiene 3 classi: DBConnect, provaFinaleDAO e TestDAO.

- DBConnect crea il collegamento con il database.
- provaFinaleDAO contiene i metodi utili alla lettura dei dati presenti nel database, richiamati dal Model.
- TestDAO è utile in fase di scrittura del codice al fine di eseguire dei test per verificare il collegamento con il database ed il corretto funzionamento dei metodi presenti nella classe provaFinaleDAO.

4.1.3 Package model

Il package, che si occupa della logica applicativa, contiene 8 classi: Albergo, Altro, Chiesa, Luogo, Model, Museo, Teatro, TestModel, Toretto.

- Albergo definisce l'oggetto Albergo, corrispondente ad una singola tupla della tabella alberghi.
- Altro definisce l'oggetto Altro, corrispondente ad una singola tupla della tabella altriLuoghi.
- Chiesa definisce l'oggetto Chiesa, corrispondente ad una singola tupla della tabella chiese.
- Luogo definisce l'oggetto Luogo che può corrispondere ad uno dei precedenti oggetti creati: albergo, chiesa, museo, teatro, toretto o ad altro luogo. Questa classe è stata creata al fine di unire in un'unica lista tutti i luoghi visitabili, utile a semplificare la progettazione degli algoritmi.
- Model gestisce la parte algoritmica del progetto, disponendo di tutte le strutture dati e dei metodi necessari alla creazione degli output.
- Museo definisce l'oggetto Museo, corrispondente ad una singola tupla della tabella musei.
- Teatro definisce l'oggetto Teatro, corrispondente ad una singola tupla della tabella teatri.

- TestModel è utile in fase di scrittura del codice al fine di eseguire dei test per verificare il corretto funzionamento dei metodi presenti nella classe Model.
- Toretto definisce l'oggetto Toretto, corrispondente ad una singola tupla della tabella toretti.

Le classi Albergo, Altro, Chiesa, Luogo, Museo, Teatro e Toretto dispongono di un costruttore e dei relativi metodi getters, setters, hashCode, equals e toString.

4.2 Algoritmi utilizzati

I principali algoritmi utilizzati nello sviluppo dell'applicativo sono 2: il primo è un algoritmo di ricerca che si occupa della lettura dei dati presenti all'interno del database 'provafinale', il secondo è invece un algoritmo ricorsivo che si occupa della creazione dell'itinerario.

4.2.1 Algoritmo di ricerca

Questa prima sequenza di funzioni viene eseguita all'avvio dell'applicazione al fine di popolare le liste ed il grafo necessari al corretto funzionamento della stessa, questo è possibile tramite il collegamento al database 'provafinale' e alla successiva lettura dei dati presenti nelle varie tabelle.

Il codice seguente mostra come sono state popolate le liste relative agli alberghi ed ai luoghi da visitare.

```
public Model() {
    dao = new provaFinaleDAO();

    allAlberghi = new ArrayList<>(dao.readAlberghi());

    allLuoghi = new ArrayList<>();

    allChiese = new ArrayList<>(dao.readChiese());
    for (Chiesa c : allChiese) {
        Luogo l=new Luogo(c.getNome(),c.getTipo(),c.getIndirizzo(),c.getCoordinate(),c.getVisita());
        allLuoghi.add(l);
    }

    allAltri = new ArrayList<>(dao.readAltriLuoghi());
    for (Altro a : allAltri) {
        Luogo l=new Luogo(c.getNome(),c.getTipo(),c.getIndirizzo(),c.getCoordinate(),c.getVisita());
        allLuoghi.add(l);
    }

    allMusei = new ArrayList<>(dao.readMusei());
    for (Museo m : allMusei) {
        Luogo l=new Luogo(c.getNome(),c.getTipo(),c.getIndirizzo(),c.getCoordinate(),c.getVisita());
        allLuoghi.add(l);
    }

    this.allTeatri = new ArrayList<>(dao.readTeatri());
    for (Teatro t : allTeatri) {
        Luogo l=new Luogo(c.getNome(),c.getTipo(),c.getIndirizzo(),c.getCoordinate(),c.getVisita());
        allLuoghi.add(l);
    }

    this.allToretti = new ArrayList<>(dao.readToretti());
    for (Toretto t : allToretti) {
        Luogo l=new Luogo(c.getNome(),c.getTipo(),c.getIndirizzo(),c.getCoordinate(),c.getVisita());
        allLuoghi.add(l);
    }
}
```

Codice 1: metodi del Model per l'aggiunta di alberghi e luoghi

Di seguito vengono riportati i metodi utili alla lettura delle tuple contenute nelle varie tabelle presenti nel database.

```

public List<Albergo> readAlberghi() {

    private LatLng coordinateCentro = new LatLng(45.07121307478032, 7.685087280059961);

    final String sql = "SELECT ID, Nome, Indirizzo, Mezza_pensione, Stelle,"
        + " Latitudine, Longitudine, Bike_friendly, Disabili, Animali_domestici "
        + "FROM alberghi";

    List<Albergo> alberghi = new ArrayList<>();

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

        while (rs.next()) {
            LatLng coord = new LatLng(rs.getDouble("Latitudine"), rs.getDouble("Longitudine"));
            Albergo a = new Albergo(rs.getInt("ID"), rs.getString("Nome"),
                rs.getString("Indirizzo"), rs.getDouble("Mezza_pensione"), rs.getInt("Stelle"),
                coord, rs.getInt("Bike_friendly"), rs.getInt("Disabili"),
                rs.getInt("Animali_domestici"),
                LatLngTool.distance(coordinateCentro, coord, LengthUnit.KILOMETER));
            alberghi.add(a);
        }
        st.close();
        conn.close();

    } catch (SQLException e) {
        e.printStackTrace();
        throw new RuntimeException("Errore di connessione al Database.");
    }

    return alberghi;
}

```

Codice 2: metodo del provaFinaleDAO per la lettura della tabella 'alberghi'

Le coordinateCentro, usate per calcolare la distanza dal centro della città degli alberghi, si riferiscono al centro di Piazza Castello.

I restanti metodi presenti nel provaFinaleDAO non si riportano in conseguenza della loro elevata somiglianza con il codice sopra visibile, le uniche differenze riguardano la query rivolta al database e la tipologia di oggetti da inserire nella lista creata. Di seguito vengono riportare le query effettuate per ogni tabella del database:

- chiese: "SELECT Nome, Indirizzo, Latitudine, Longitudine, Durata FROM chiese".
- altriluoghi: "SELECT Nome, Tipo, Indirizzo, Latitudine, Longitudine, Durata FROM altriluoghi".
- musei: "SELECT Nome, Indirizzo, Latitudine, Longitudine, Durata FROM musei".
- teatri: "SELECT Nome, Indirizzo, Latitudine, Longitudine, Durata FROM teatri".
- toretti: "SELECT ID, Indirizzo, Latitudine, Longitudine, Durata FROM toretti".

L'utente può decidere di restringere la lista di alberghi tra cui scegliere mediante dei filtri presenti all'interno dell'interfaccia, il seguente codice mostra come avviene il filtraggio.

```

public void creaListaAlberghi(double prezzo,int stelle,double distanza,boolean bici,boolean disabili,
    boolean animali) {
    alberghiFiltrati = new ArrayList<>();

    if(bici==true) {
        if(disabili==true) {
            if(animali==true) {
                for(Albergo a : allAlberghi) {
                    if(a.getPrezzo()<=prezzo && a.getStelle()>=stelle
                        && LatLngTool.distance(coordinateCentro,a.getCoordinate(),LengthUnit.KILOMETER)<=distanza
                        && a.getBici()==bici && a.getDisabili()==disabili && a.getAnimali()==animali) {
                            alberghiFiltrati.add(a);
                        }
                    }
                }
            }
            if(animali==false) {
                for(Albergo a : this.allAlberghi) {
                    if(a.getPrezzo()<=prezzo && a.getStelle()>=stelle
                        && LatLngTool.distance(coordinateCentro,a.getCoordinate(),LengthUnit.KILOMETER)<=distanza
                        && a.getBici()==bici && a.getDisabili()==disabili) {
                            alberghiFiltrati.add(a);
                        }
                    }
                }
            }
        }
        if(disabili==false) {
            if(animali==true) {
                for(Albergo a : allAlberghi) {
                    if(a.getPrezzo()<=prezzo && a.getStelle()>=stelle
                        && LatLngTool.distance(coordinateCentro,a.getCoordinate(),LengthUnit.KILOMETER)<=distanza
                        && a.getBici()==bici && a.getAnimali()==animali) {
                            alberghiFiltrati.add(a);
                        }
                    }
                }
            }
            if(animali==false) {
                for(Albergo a : allAlberghi) {
                    if(a.getPrezzo()<=prezzo && a.getStelle()>=stelle
                        && LatLngTool.distance(coordinateCentro,a.getCoordinate(),LengthUnit.KILOMETER)<=distanza
                        && a.getBici()==bici) {
                            this.alberghiFiltrati.add(a);
                        }
                    }
                }
            }
        }
    }
    if(bici==false) {
        if(disabili==true) {
            if(animali==true) {
                for(Albergo a : allAlberghi) {
                    if(a.getPrezzo()<=prezzo && a.getStelle()>=stelle
                        && LatLngTool.distance(coordinateCentro,a.getCoordinate(),LengthUnit.KILOMETER)<=distanza
                        && a.getDisabili()==disabili && a.getAnimali()==animali) {
                            alberghiFiltrati.add(a);
                        }
                    }
                }
            }
            if(animali==false) {
                for(Albergo a : allAlberghi) {
                    if(a.getPrezzo()<=prezzo && a.getStelle()>=stelle
                        && LatLngTool.distance(coordinateCentro,a.getCoordinate(),LengthUnit.KILOMETER)<=distanza
                        && a.getDisabili()==disabili) {
                            alberghiFiltrati.add(a);
                        }
                    }
                }
            }
        }
        if(disabili==false) {
            if(animali==true) {
                for(Albergo a : allAlberghi) {
                    if(a.getPrezzo()<=prezzo && a.getStelle()>=stelle
                        && LatLngTool.distance(coordinateCentro,a.getCoordinate(),LengthUnit.KILOMETER)<=distanza
                        && a.getAnimali()==animali) {
                            alberghiFiltrati.add(a);
                        }
                    }
                }
            }
            if(animali==false) {
                for(Albergo a : allAlberghi) {
                    if(a.getPrezzo()<=prezzo && a.getStelle()>=stelle
                        && LatLngTool.distance(coordinateCentro,a.getCoordinate(),LengthUnit.KILOMETER)<=distanza {
                            alberghiFiltrati.add(a);
                        }
                    }
                }
            }
        }
    }
}

```

Codice 3: metodo del Model per la creazione della lista di alberghi filtrata

4.2.2 Algoritmo ricorsivo

Alla pressione del bottone 'Crea itinerario' presente nell'interfaccia utente l'applicazione esegue una ricorsione al fine di trovare la combinazione ideale di luoghi da visitare.

L'albergo scelto viene aggiunto 2 volte alla lista di luoghi visitabili modificandone la tipologia, nel primo caso avrà come valore 'Partenza' mentre nel secondo 'Arrivo'. L'albergo con tipologia pari a 'Partenza' viene aggiunto all'interno dell'itinerario in prima posizione come punto di partenza, mentre l'albergo con tipologia pari ad 'Arrivo' viene usato come condizione di uscita dall'algoritmo dato che corrisponde al punto di arrivo.

Il metodo presente nel Model, con il quale interagisce in controller, richiede l'immissione di 4 input: tempo a disposizione, livello di interesse per i luoghi di intrattenimento, livello di interesse per i luoghi di culto e livello di interesse per i musei. Questi dati vengono utilizzati per la creazione del grafo sul quale l'algoritmo ciclerà, questo passaggio viene effettuato a questo punto del codice a fini di ottimizzazione.

Vengono generati 2 itinerari simultaneamente, il primo genera come risultato una lista contenente il maggior numero possibile di posti da visitare rispettando solamente il primo dato, ovvero il tempo disponibile, mentre il secondo cerca di rispettare le preferenze dell'utente, cosa che purtroppo non sempre accade dato che i luoghi non sono uniformemente distribuita sulla superficie della città.

```
public void creaItinerario(double tempoDisponibile,int stelleIntrattenimento,int stelleCulto,
                          int stelleMusei) {

    creaGrafo(stelleIntrattenimento, stelleCulto, stelleMusei);

    itinerarioMigliore = new ArrayList<>();
    durata = Double.MAX_VALUE;
    itinerarioMiglioreFiltrato = new ArrayList<>();
    durataFiltrata = Double.MAX_VALUE;

    List<Luogo> parziale = new ArrayList<>();
    Luogo partenza = null;
    Luogo arrivo = null;
    for(Luogo l : luoghiVicini) {
        if(l.getTipo().compareTo("PARTENZA")==0) {
            partenza=l;
        }
        else if(l.getTipo().compareTo("ARRIVO")==0) {
            arrivo = l;
        }
    }
    parziale.add(partenza);
    ricorsione(parziale,getAdiacenti(partenza),arrivo,0.0,false,false,0,false,tempoDisponibile,
              stelleIntrattenimento,stelleCulto,stelleMusei);
}
```

Codice 4: metodo del Model per l'inizializzazione delle strutture dati necessarie alla ricorsione

La vera ricorsione viene effettuata tramite un algoritmo racchiuso all'interno di un metodo privato, ovvero raggiungibile solamente dall'interno della classe in cui si trova.

Quando la condizione di uscita è soddisfatta, ovvero l'itinerario comprende il punto di arrivo, il programma controlla che l'itinerario appena creato contenga più luoghi del migliore, senza filtri, creato fino a quel momento, se questa verifica da esito positivo l'itinerario appena creato diventa il migliore. Vengono inoltre verificate le condizioni per le quali l'itinerario può diventare anche il migliore che rispetta i filtri, questo come anticipato non si verifica sempre.

Se il luogo di arrivo non è contenuto nella lista si cercano di aggiungere altri luoghi ciclando tra quelli adiacenti al luogo in ultima posizione, calcolati mediante un metodo specifico.

```

private void ricorsione(List<Luogo> parziale,List<Luogo> adiacenti,Luogo arrivo,double cont,
    boolean cinema,boolean teatri,int chiese,boolean musei,
    double tempoDisponibile,int stelleIntrattenimento,int stelleCulto,
    int stelleMusei) {
    if(parziale.get(parziale.size()-1).equals(arrivo)) {
        if((parziale.size()-1)>itinerarioMigliore.size()
            || (parziale.size()==itinerarioMigliore.size() && cont<durata)) {
            itinerarioMigliore = new ArrayList<>(parziale);
            durata = cont;
        }
    }

    for(Luogo l : adiacenti) {
        if(!parziale.contains(l)) {
            DefaultWeightedEdge arcoPrecedente = grafo.getEdge(parziale.get(parziale.size()-1),l);
            double precedente = grafo.getEdgeWeight(arcoPrecedente);
            double successivo = 0.0;
            double distanza = LatLngTool.distance(l.getCoordinate(),albergoScelto.getCoordinate(),
                LengthUnit.KILOMETER);

            if(distanza<=1.5) {
                successivo = distanza*60/4;
            }
            else {
                successivo = distanza*60/20;
            }
            if((cont+precedente+l.getVisita()+successivo)<=tempoDisponibile){
                parziale.add(l);
                ricorsione(parziale,this.getAdiacenti(l),arrivo,cont+precedente+l.getVisita(),
                    cinema,teatri,chiese,musei,tempoDisponibile,stelleIntrattenimento,
                    stelleCulto,stelleMusei);
                parziale.remove(l);
            }
        }
    }
}

```

Codice 5: metodo del Model per l'esecuzione della ricorsione

```

public List<Luogo> getAdiacenti(Luogo partenza) {
    List<Luogo> adiacenti = Graphs.successorListOf(grafo,partenza);
    return adiacenti;
}

```

Codice 6: metodo del Model per il calcolo dei luoghi adiacenti ad un luogo dato

Altri controlli sono presenti all'interno codice dell'algoritmo per cercare di velocizzare il calcolo dell'itinerario, non vengono però riportati per semplicità di visualizzazione e comprensione.

5 Diagramma delle classi principali

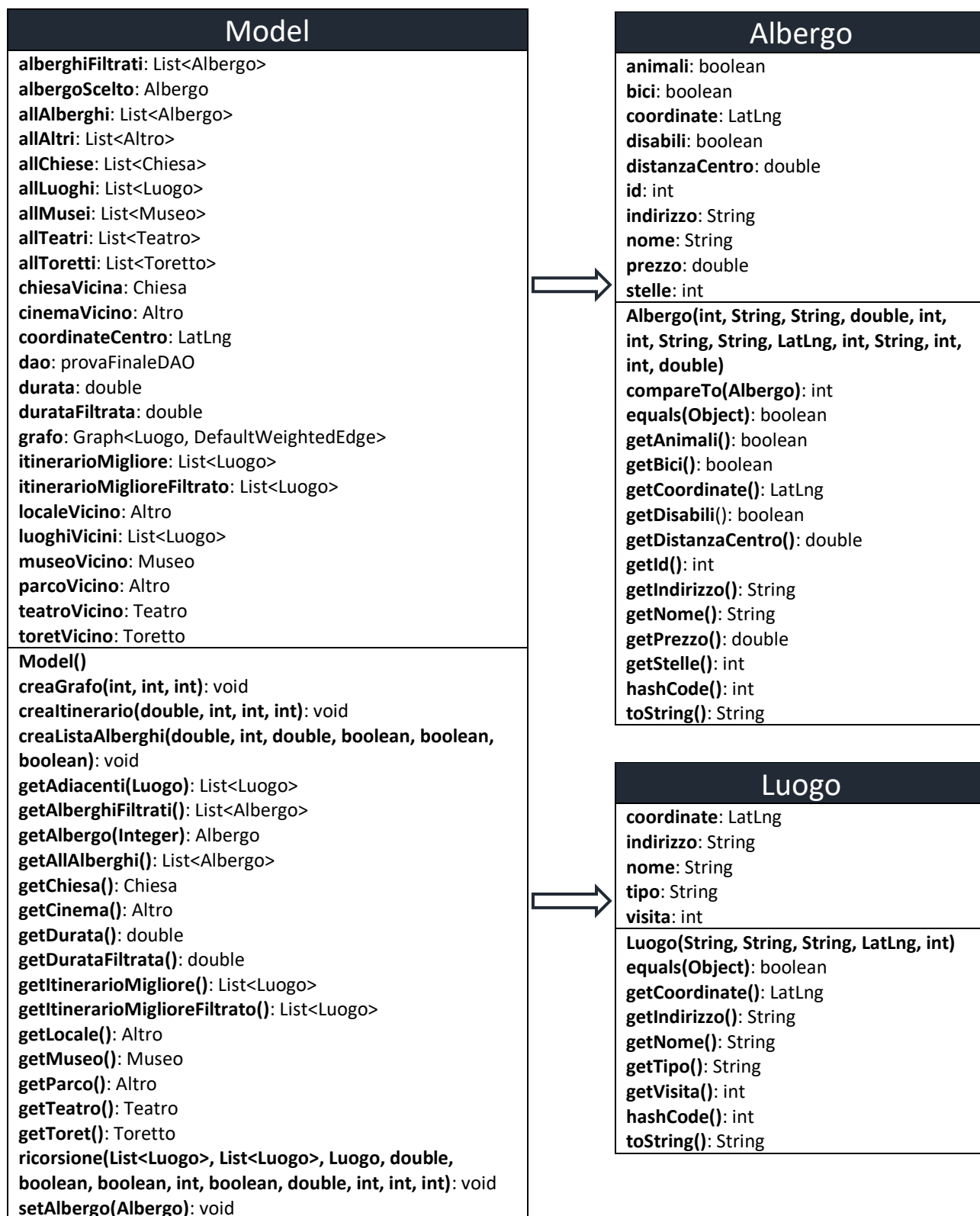


Diagramma delle classi principali contenute nel package 'model'

provaFinaleDAO
coordinateCentro: LatLng
readAlberghi(): List<Albergo>
readAltriLuoghi(): List<Altro>
readChiese(): List<Chiesa>
readMusei(): List<Museo>
readTeatri(): List<Teatro>
readToretti(): List<Toretto>

Diagramma delle classi principali contenute nel package 'db'

6 Schermate dell'applicazione

The screenshot shows a web application window titled "Prova finale" with standard window controls. The main heading is "Generatore di itinerari per la città di Torino".

Alberghi Section:

- A dropdown menu labeled "Scegli un albergo..." is highlighted with a blue border.
- Buttons: "Imposta albergo", "Imposta filtri", and "Elimina filtri".

Filters Section:

- Labels: "Prezzo max", "Stelle min", "Dist. centro max", "Accesso animali", "Accesso bici", "Accesso disabili".
- Each label has a corresponding input field (dropdown or checkbox).

Luoghi Section:

- Labels: "Tempo disponibile", "Intrattenimento", "Luoghi di culto", "Musei".
- Each label has a corresponding dropdown menu.

Actions:

- A button labeled "Calcola itinerario".

Results Area:

- Text: "179 alberghi trovati".
- Text: "Selezionare un albergo".

Footer:

- Logos: "aperTo" (with a blue cube icon) and "I ♥ TORET" (with a bull head icon).

Figura 7: schermata iniziale dell'interfaccia utente

L'interfaccia è stata realizzata utilizzando Scene Builder.

All'interno dell'area testuale, dove successivamente sarà presentato il risultato, vengono segnalate le istruzioni da seguire per un corretto utilizzo dell'applicazione.

La sezione di interfaccia contenente i filtri applicabili ai luoghi visitabili viene sbloccata in conseguenza alla scelta dell'albergo.

Prova finale

Generatore di itinerari per la città di Torino

Alberghi Hotel Parco Europa... Imposta albergo Imposta filtri Elimina filtri

Prezzo max 150.0 € Stelle min 3 Dist. centro max 5.0 km Accesso animali ☒ Accesso bici ☐ Accesso disabili ☐

Luoghi

Tempo disponibile 2 h Intrattenimento Non interessato Luoghi di culto Mediamente interes... Musei Mediamente interes...

Calcola itinerario

Albergo selezionato: Hotel Parco Europa
Indirizzo: VIALE XXV APRILE 193 C
Prezzo a notte: 96.0 €
Numero di stelle: 3
Distanza dal centro: 4.62 km

Itinerario creato:
S. Pietro in Vincoli, indirizzo = Via S. Rocco 29
Ponte Principessa Isabella, indirizzo = Ponte Principessa Isabella
Fontana dei 12 Mesi, indirizzo = Corso Federico Sclopis 6
Condominio 25 Verde, indirizzo = Via Gabriele Chiabrera 25
Monumento all'Autiere d'Italia, indirizzo = Corso Unità d'Italia 40


aperT  **I ♥ TORET**™

Figura 8: schermata dell'interfaccia utente con risultato

Nell'area di testo viene presentato il risultato generato, esso è composto da informazioni relative all'albergo scelto ed all'itinerario generato, nello specifico contiene:

- nome dell'albergo
- indirizzo dell'albergo
- prezzo dell'albergo a notte
- numero di stelle dell'albergo
- distanza dell'albergo dal centro
- lista rappresentante l'itinerario che rispetta i filtri
- numero di luoghi di cui è composto l'itinerario filtrato
- durata dell'itinerario filtrato
- lista rappresentante un itinerario alternativo che non rispetta i filtri
- numero di luoghi di cui è composto l'itinerario non filtrato
- durata dell'itinerario non filtrato
- informazioni su ulteriori luoghi specifici presenti nelle vicinanze dell'albergo

Link al video dimostrativo: <https://youtu.be/XZ1HuocmU7A>

7 Risultati sperimentali

7.1 Esempio 1

Input:

Albergo: Hotel Original
Filtri inseriti:
Prezzo max: 100.0 €
Stelle min: 2
Dist. Centro max: 8.0 km
Accesso animali: selezionato
Accesso bici: non selezionato
Accesso disabili: non selezionato
Tempo disponibile: 2 h
Intrattenimento: Mediamente interessato
Luoghi di culto: Molto interessato
Musei: Non interessato

Output:

Albergo selezionato: Hotel Original
Indirizzo: Via Farinelli Arturo 4
Prezzo a notte: 65.0 €
Numero di stelle: 3
Distanza dal centro: 7.59 km

Itinerario creato:

S. Marco Evangelista, indirizzo = Via Edoardo Daneo 19
Piazzale Grande Torino, indirizzo = Piazzale Grande Torino
'Punti di vista' di Tony Cragg, indirizzo = Corso Sebastopoli 123
Monumento ai Caduti di Nassiriya, indirizzo = Corso IV Novembre 80
Arco Olimpico di Torino, indirizzo = Via Giordano Bruno 181
Piazza Bengasi, indirizzo = Piazza Bengasi
Teatro Agnelli (visitabile solo esternamente), indirizzo = Via Paolo Sarpi 111/A
S. Giovanni Bosco, indirizzo = Via Paolo Sarpi 117
Numero di posti visitabili: 8
Durata itinerario: 110 minuti

Itinerario alternativo, senza filtri:

S. Marco Evangelista, indirizzo = Via Edoardo Daneo 19
Piazza Bengasi, indirizzo = Piazza Bengasi
Arco Olimpico di Torino, indirizzo = Via Giordano Bruno 181
Monumento ai Caduti di Nassiriya, indirizzo = Corso IV Novembre 80
'Punti di vista' di Tony Cragg , indirizzo = Corso Sebastopoli 123
Piazzale Grande Torino, indirizzo = Piazzale Grande Torino
Colonna del Centenario dell'Unità d'Italia, indirizzo = Corso Unità d'Italia 70
Teatro Agnelli (visitabile solo esternamente), indirizzo = Via Paolo Sarpi 111/A
Numero di posti visitabili: 8
Durata itinerario: 102 minuti

Cinema nelle vicinanze: UCI Cinemas Torino Lingotto, indirizzo = Via Nizza 262

Locale storico nelle vicinanze: Caffè Platti, indirizzo = Corso Vittorio Emanuele II 72

Parco nelle vicinanze: Parco Sangone, indirizzo = Str. Castello di Mirafiori 272

Toret, simbolo di Torino, nelle vicinanze: Toret n. 144, indirizzo = Via Plava - Corso Unione Sovietica (giardino) Sud

Dimensioni grafo: 46 vertici e 401 archi

Tempo per il calcolo del risultato: 23.733 secondi

7.2 Esempio 2

Input:

Albergo: Hotel CHC Torino Castello

Filtri inseriti:

Prezzo max: 250.0 €

Stelle min: 4

Dist. Centro max: 0.5 km

Accesso animali: non selezionato

Accesso bici: non selezionato

Accesso disabili: selezionato

Tempo disponibile: 0.5 h

Intrattenimento: Mediamente interessato

Luoghi di culto: Non interessato

Musei: Mediamente interessato

Output:

Albergo selezionato: Hotel CHC Torino Castello

Indirizzo: Via XX Settembre 70

Prezzo a notte: 126.0 €

Numero di stelle: 4

Distanza dal centro: 0.13 km

Itinerario creato:

Piazza Castello, indirizzo = Piazza Castello

Monumento a Vincenzo Gioberti, indirizzo = Piazza Carignano 4

Piazza Carignano, indirizzo = Piazza Carignano

Teatro Carignano - Teatro Stabile Torino (visitabile solo esternamente), indirizzo = Piazza Carignano 6

Numero di posti visitabili: 4

Durata itinerario: 30 minuti

Itinerario alternativo, senza filtri:

Piazza Castello, indirizzo = Piazza Castello

Monumento ai Cavalieri d'Italia, indirizzo = Piazza Castello

Piazza Carlo Alberto, indirizzo = Piazza Carlo Alberto

Monumento a Carlo Alberto, indirizzo = Piazza Carlo Alberto 3

Piazza Carignano, indirizzo = Piazza Carignano

Monumento a Vincenzo Gioberti, indirizzo = Piazza Carignano 4

Numero di posti visitabili: 6

Durata itinerario: 28 minuti

Cinema nelle vicinanze: Cinema Lux, indirizzo = Galleria S. Federico 33

Museo nelle vicinanze: Palazzo Madama - Museo Civico di Arte Antica, indirizzo = Piazza Castello 0, durata visita = 90

Locale storico nelle vicinanze: Ristorante Del Cambio, indirizzo = Piazza Carignano 2

Parco nelle vicinanze: Parco Archeologico Torri Palatine, indirizzo = Via Porta Palatina

Toret, simbolo di Torino, nelle vicinanze: Toret n. 100, indirizzo = Via Porta Palatina

Dimensioni grafo: 37 vertici e 633 archi

Tempo per il calcolo del risultato: 0.682 secondi

7.3 Esempio 3

Input:

Albergo: Hotel Gardenia

Filtri inseriti:

Prezzo max: 100.0 €

Dist. Centro max: 4.0 km

Accesso animali: non selezionato

Accesso bici: non selezionato

Accesso disabili: non selezionato

Tempo disponibile: 1 h

Intrattenimento: Non interessato

Luoghi di culto: Non interessato

Musei: Non interessato

Output:

Albergo selezionato: Hotel Gardenia

Indirizzo: Via Sabaudia 10

Prezzo a notte: 82.0 €

Numero di stelle: 1

Distanza dal centro: 3.9 km

Itinerario creato:

Monumento al Terzo Reggimento Alpini, indirizzo = Viale Carlo Ceppi 15

Panchina degli innamorati, indirizzo = Parco del Valentino

Monumento ad Amedeo di Savoia, indirizzo = Corso Massimo d'Azeglio 11

Monumento a Felice Govean, indirizzo = Piazzetta Felice Govean

Portone del Melograno, indirizzo = Via Giovanni Argentero 4

Sistema di accumulo San Salvario - The Heat Garden, indirizzo = Via Ernesto Lugaro 38

Numero di posti visitabili: 6

Durata itinerario: 57 minuti

Locale storico nelle vicinanze: Confetteria Avvignano, indirizzo = Piazza Carlo Felice 50

Parco nelle vicinanze: Parco Europa, indirizzo = Via Nuova 28

Toret, simbolo di Torino, nelle vicinanze: Toret n. 1377, indirizzo = Giardino Gianni Rodari (c.so Moncalieri, 262)

Dimensioni grafo: 24 vertici e 246 archi

Tempo per il calcolo del risultato: 0.108 secondi

8 Valutazioni sui risultati ottenuti

L'applicazione, basata principalmente su un algoritmo ricorsivo e connessa ad un database di dimensioni non trascurabili, può richiedere molto tempo per generare l'output, pur essendo presenti svariati filtri e controlli all'interno del codice.

Il tempo necessario al calcolatore per produrre il risultato dipende principalmente da 2 fattori: la distanza dell'albergo selezionato dal centro della città e il tempo di cui l'utente dispone per l'itinerario. Alberghi più vicini al centro dispongono, in media, di più luoghi visitabili nelle loro vicinanze rispetto a quelli presenti in periferia e questo, pur essendo un vantaggio per i turisti, crea sovraccarico a livello computazionale per la creazione dell'output.

Un'idea delle tempistiche necessarie alla generazione del risultato può essere data dalla dimensione del grafo, creato partendo dall'albergo selezionato. Questo, pur avendo in alcuni casi pochi vertici, presenta spesso un elevato numero di archi che rallentano la produzione dell'itinerario.

Un altro fattore che influenza molto i tempi di generazione dell'output è la tipologia di luoghi presente in maggior quantità vicino all'albergo di partenza: luoghi come i musei richiedono più tempo per essere visitati rispetto invece a piazze e monumenti. Un elevato numero di luoghi che richiedono poco tempo rallenterà la generazione dell'itinerario dato che l'algoritmo ricorsivo potrà andare più a fondo ed aggiungere più luoghi ad ogni soluzione parziale.

Il punto debole dell'applicazione è quindi causato dalla ricorsione, limitata dalle prestazioni del calcolatore.

Punto di forza dell'algoritmo è il fatto che cerchi in ogni caso di generare itinerari comprendenti luoghi derivanti da diverse tipologie, questo è utile al fine di tenere vivo l'interesse durante la visita della città. In coda all'output vengono inoltre aggiunti dei luoghi particolari, come parchi, locali storici e toretti, per un'ulteriore eterogeneità e per far conoscere ai visitatori luoghi che non vengono spesso pubblicizzati.

In conclusione, seppur con certe limitazioni, l'applicazione svolge il compito richiesto producendo itinerari per la città di Torino che rispettino i filtri imposti dall'utente.



Quest'opera è distribuita con Licenza [Creative Commons Attribuzione - Non commerciale - Condividi allo stesso modo 4.0 Internazionale](https://creativecommons.org/licenses/by-nc-sa/4.0/).