

Dipartimento di Ingegneria Gestionale e della Produzione

Laurea Triennale in Ingegneria Gestionale – Classe L8

A.A. 2023/24

**Applicazione per la simulazione dei servizi di un’ agenzia di viaggi**

**Relatore** Prof. Fulvio Corno

**Candidato** Andrea Gaudino, s296109

Sommario

[1. Proposta di progetto 3](#_Toc177147630)

[1.1. Titolo della proposta 3](#_Toc177147631)

[1.2. Descrizione del problema proposto 3](#_Toc177147632)

[1.3. Descrizione della rilevanza gestionale del problema 3](#_Toc177147633)

[1.4. Descrizione dei data-set per la valutazione 3](#_Toc177147634)

[1.5. Descrizione preliminare degli algoritmi coinvolti 4](#_Toc177147635)

[1.6. Descrizione preliminare delle funzionalità previste per l’applicazione software 4](#_Toc177147636)

[2. Descrizione dettagliata del problema affrontato 5](#_Toc177147637)

[3. Descrizione del dataset utilizzato 5](#_Toc177147638)

[4. Descrizione delle strutture dati e degli algoritmi utilizzati 8](#_Toc177147639)

[4.1. Pattern MVC e DAO 8](#_Toc177147640)

[4.2. Algoritmi principali utilizzati 9](#_Toc177147641)

[4.2.1. Selezione dei viaggi 9](#_Toc177147642)

[4.2.2. Valutazione di un viaggio passato 9](#_Toc177147643)

[4.2.3. Creazione di un viaggio personalizzato e ricorsione 11](#_Toc177147644)

[4.3. Auto compilazione dei dati utente 14](#_Toc177147645)

[5. Diagramma delle classi principali 15](#_Toc177147646)

[6. Videate dell’applicazione 17](#_Toc177147647)

[7. Risultati sperimentali 19](#_Toc177147648)

[8. Valutazioni sui risultati ottenuti e conclusioni 20](#_Toc177147649)

[9. Licenza 21](#_Toc177147650)

# Proposta di progetto

## Titolo della proposta

Applicazione per la Simulazione dei Servizi di un’ Agenzia di Viaggi

## Descrizione del problema proposto

L’applicazione che ho sviluppato su permetterà all’utente di cercare le soluzioni migliori per il viaggio che intenderà organizzare. Verrà richiesto di inserire una serie di parametri quali il periodo temporale, la destinazione e il costo. Il programma sarà in grado di fornire all’utente una serie di offerte di viaggio con eventuali sconti. Verrà quindi chiesto di scegliere l’opzione che rispecchi meglio le proprie richieste e successivamente si procederà con l’acquisto del pacchetto selezionato dopo aver inserito i propri dati anagrafici. Nel caso in cui l’utente non trovi un pacchetto viaggio che gli piaccia, gli verrà data la possibilità di crearselo, inserendo ulteriori parametri e lasciando che l’applicazione trovi la miglior soluzione. L'utente avrà anche la possibilità di dare una valutazione ai viaggi effettuati in precedenza per poter aiutare altri viaggiatori alla ricerca di una vacanza che faccia il caso loro.

## Descrizione della rilevanza gestionale del problema

Il progetto è molto rilevante dal punto di vista gestionale, perché risponde a esigenze reali nella pianificazione dei viaggi. Automatizzando la selezione dei pacchetti e permettendo di creare viaggi personalizzati, il sistema semplifica la ricerca e la prenotazione, risparmiando tempo e fatica per gli utenti. Per l'agenzia, gestire un ampio numero di pacchetti diventa molto più semplice grazie a un'interazione rapida ed efficace con il database. Inoltre, le valutazioni degli utenti offrono spunti preziosi per migliorare e adattare l'offerta, aiutando l'agenzia a ottimizzare il proprio business plan in base ai feedback ricevuti.

## Descrizione dei data-set per la valutazione

Il database utilizzato è stato trovato in una repository pubblica su GitHub ([link\_allo\_script\_originale](https://github.com/pavlosdais/Travel-Agency-Database/blob/main/hw2/database/database.sql)). Questo dataset contiene dati fittizi pensati per un'agenzia di viaggi. Verrà opportunamente modificato poiché, sebbene la struttura sia ottimale per gli scopi di questo progetto, i dati presentano alcuni errori sporadici e alcune tabelle risultano superflue per l’applicazione, quindi verranno eliminate. In particolare, la tabella "travel\_agency\_branch" non sarà utilizzata, e la sua chiave primaria, presente in altre tabelle, verrà rimossa. Sarà inoltre aggiunta una nuova tabella chiamata "rating", che conterrà le valutazioni dei viaggi precedenti. Queste valutazioni saranno generate in modo casuale, con una tendenza positiva, così da mostrare all'utente il grado di soddisfazione dei clienti che hanno già partecipato a un determinato viaggio. Le tabelle “travel\_group” e “traveler\_has\_group” saranno eliminate. La colonna "email" della tabella "traveler" verrà modificata per diventare la chiave primaria, permettendo così l'identificazione univoca dei clienti. Infine, verranno generati nuovi pacchetti di viaggio con date aggiornate per la stagione 2024/25, rendendo l'applicazione il più verosimile possibile. La tabella “guided\_tour” verrà rinominata in “trip\_package\_has\_attraction”.

## Descrizione preliminare degli algoritmi coinvolti

Per la ricerca dei pacchetti viaggio già esistenti l’applicazione svolge delle interrogazioni del database mediante query SQL, basate sui dati di input dell’utente. Nel caso di creazione del viaggio, il programma costruisce un grafo (grazie alla libreria NetworkX) con tutte le attrazioni dei paesi scelti dall’utente e restituisce il miglior risultato secondo un algoritmo di ricorsione.

## Descrizione preliminare delle funzionalità previste per l’applicazione software

L'applicazione presenta tre schede principali: una prima per prenotare un viaggio esistente, una seconda per crearne uno personalizzato e una terza per valutare una vacanza passata. La pagina di prenotazione si apre con una serie di menù a tendina, dai quali l'utente può selezionare i parametri di ricerca per i viaggi. Alla pressione di un pulsante, verranno visualizzati tutti i pacchetti che rispettano i criteri scelti. Dopo aver selezionato il viaggio desiderato, si aprirà una pagina per l'inserimento dei dati personali, e, una volta inviati, verrà confermata la prenotazione. Nella scheda per la creazione di un viaggio, sono presenti campi di input e menù a tendina simili alla prima scheda. Tuttavia, in questo caso, l'applicazione fornirà una singola proposta, che rappresenterà il miglior itinerario in base ai parametri inseriti dall'utente. Per lasciare una valutazione, all'utente verrà richiesto di inserire la propria email. A quel punto, verrà visualizzato l'elenco delle prenotazioni passate, dal quale sarà possibile selezionare il viaggio che si desidera valutare.

# Descrizione dettagliata del problema affrontato

Il progetto si propone di risolvere il problema della scelta e della personalizzazione dei viaggi attraverso un sistema che simula le funzionalità di un'agenzia di viaggi. In particolare, l'obiettivo è fornire agli utenti una piattaforma in grado di presentare pacchetti di viaggio predefiniti, selezionabili in base a specifici parametri di preferenza. Qualora nessun pacchetto soddisfi completamente le esigenze dell'utente, il sistema offre la possibilità di creare un viaggio personalizzato, sfruttando un algoritmo avanzato per proporre la soluzione più vantaggiosa in termini di costo e qualità. Inoltre, il sistema include una funzione di valutazione dei viaggi già effettuati, permettendo agli utenti di condividere feedback e recensioni, contribuendo così a migliorare l'esperienza di viaggio complessiva e a facilitare le scelte future per gli altri utenti. Questo approccio integra aspetti di selezione automatica, personalizzazione e interazione con l’utente, risolvendo le sfide legate alla pianificazione dei viaggi. Un sotto-problema chiave è la gestione della personalizzazione dei viaggi. Gli input di questo processo includono i dettagli specifici forniti dagli clienti, come destinazioni, attrazioni e budget. Gli output sono pacchetti di viaggio su misura che rispondono a queste specifiche.

Le criticità possono comprendere la complessità dell'algoritmo necessario per ottimizzare le opzioni in base a molteplici variabili. Tuttavia, le potenzialità sono elevate, poiché una personalizzazione efficace può migliorare significativamente la soddisfazione dell'utente e la qualità del servizio. Un altro aspetto rilevante è il sistema di valutazione dei viaggi, che raccoglie le recensioni degli utenti sui viaggi già effettuati, mentre gli output consistono in valutazioni aggregate e suggerimenti per future offerte. Le criticità qui includono la necessità di gestire recensioni autentiche e di evitare manipolazioni. Le potenzialità sono ampie, poiché le recensioni possono fornire indicazioni preziose per migliorare continuamente l'offerta.

# Descrizione del dataset utilizzato

Il database che include le opportune modifiche effettuate si trova nella repository del mio progetto al seguente link :

<https://github.com/TdP-prove-finali/GaudinoAndrea/blob/main/database/script_finale.sql>.

Qui sotto riporto il diagramma ER (Entità-Relazione) del database.

Immagine che contiene testo, diagramma, schermata, Carattere

Descrizione generata automaticamente

Come si può notare, ci sono molte interazioni tra le tabelle, il che ha reso la ricerca e l'inserimento dei dati più complessi. Tuttavia, questa complessità contribuisce a una maggiore sicurezza della struttura del database, poiché la rimozione di un dato diventa un'operazione non banale a causa del sofisticato meccanismo delle chiavi esterne. In grassetto sono riportate le chiavi primarie.

Di seguito vengono descritte le tabelle utilizzate con le loro caratteristiche:

* Trip\_package
  + Trip\_package\_id: identificativo del pacchetto viaggi
  + Trip\_start: data di inizio del viaggio
  + Cost\_attraction: somma dei costi delle attrazioni turistiche del viaggio
  + Cost\_accomodation: costo alloggio
  + Package\_cost\_category\_id: categoria di costo dell’alloggio
* Package\_cost
  + Package\_cost\_id: categoria di costo dell’alloggio
  + Package\_cost\_category: categoria di costo visualizzabile dall’utente (4 possibili valori $, $$, $$$, $$$$)
* Trip\_package\_has\_destination
  + Trip\_package\_id: identificativo del pacchetto viaggi
  + Destination\_id: identificativo della destinazione
* Destination
  + Destination\_id: identificativo della destinazione
  + Name: nome della località
  + Country: stato in cui si trova la località
  + Local\_language: lingua locale
* Offer
  + Offer\_id: identificativo dell’offerta
  + Trip\_package\_id: identificativo del pacchetto viaggi
  + Offer\_start: data di inizio della validità della offerta
  + Offer\_end: data di fine della validità della offerta
  + Cost: costo della offerta
  + Offer\_name: nome offerta da far visualizzare all’utente
  + Offer\_info\_category: categoria dell’offerta
* Reservation
  + Reservation\_id: identificativo della prenotazione
  + Customer\_id: identificativo del cliente
  + Trip\_package\_id: identificativo del pacchetto viaggi
  + Date: data di acquisto del pacchetto viaggi
  + Offer\_id: eventuale identificativo dell’offerta utilizzata
* Ratings
  + Customer\_id: identificativo del cliente
  + Trip\_package\_id: identificativo del pacchetto viaggi
  + Rating: voto da 1 a 5
* Trip\_package\_has\_attraction
  + Trip\_package\_id: identificativo del pacchetto viaggi
  + Tourist\_attraction\_id: identificativo della attrazione turistica
  + Travel\_guide\_employee\_AM: eventuale matricola della guida turistica
* Tourist\_attraction:
  + Tourist\_attraction\_id: identificativo della attrazione turistica
  + Name: nome della attrazione turistica
  + Attraction\_info\_cat: categoria della attrazione
  + Destination\_id: identificativo della località
  + Cost: costo attrazione
* Travel\_guide
  + Travel\_guide\_employee\_AM: matricola della guida turistica
  + Cost\_percentage: percentuale di costo da aggiungere al costo della attrazione turistica
* Travel\_guide\_has\_languages
  + Travel\_guide\_employee\_AM: matricola della guida turistica
  + Languages\_id: identificativo della lingua parlata dalla guida
* Languages
  + Languages\_id: identificativo della lingua parlata dalla guida
  + Name: nome della lingua
* Employees
  + Employees\_AM: matricola dei dipendenti
  + Name: nome dei dipendenti
  + Surname: cognome
  + Address: indirizzo
  + Salary: stipendio mensile
* Traveler
  + Traveler\_id: identificativo del cliente
  + Email: email del cliente
  + Name: nome dell’utente
  + Surname: cognome del cliente
  + Age: età del cliente
  + Address: indirizzo di casa
  + Phone: numero di telefono
  + Gender: genere sessuale cliente

# Descrizione delle strutture dati e degli algoritmi utilizzati

## Pattern MVC e DAO

Per questo progetto mi sono avvalso del pattern MVC (Model, View, Controller) come descritto nel seguito:

* *View:* si occupa della rappresentazione grafica dei dati. Ha il solo scopo di gestire l’interfaccia utente, senza manipolare le informazioni ricevute.
* *Controller:* ha il compito di gestire l’interazione tra model e view, passando i dati da un lato all’altro senza manipolarli.
* *Model:* è la logica applicativa del programma. Gestisce l’accesso ai dati e la loro manipolazione.

Il *DAO (Data Access Oject)* ha invece lo scopo di gestire le interazioni con il database, interrogandolo con delle query SQLsql. I dati da qui prelevati vengono immediatamente inviati al model che si occupa della loro gestione.

## Algoritmi principali utilizzati

### Selezione dei viaggi

Il primo algoritmo utilizzato nel programma consiste nella ricerca di tutti i pacchetti viaggio che rispettano i valori di input impostati dall'utente. A quest'ultimo viene chiesto di specificare, tramite dei menù a tendina, il mese e l'anno scelti per l'inizio del viaggio, la destinazione primaria e la categoria di costo dell'alloggio. Alla pressione del bottone “Ricerca viaggi” verranno mostrati tutti i pacchetti disponibili con l’eventuale dicitura “offerta disponibile” nel caso in cui, nel giorno della prenotazione, siano disponibili sconti. La ricerca dei viaggi è effettuata tramite la seguente query SQL parametrica, nella quale i parametri sono mese, anno e destinazione del viaggio:

**select** **distinctrow** tp.\*

**from** trip\_package *tp*, trip\_package\_has\_destination *th*, destination *d*

**where** **month**(tp.trip\_start) = %s

**and** **year**(tp.trip\_start) = %s

**and** th.trip\_package\_id = tp.trip\_package\_id

**and** th.destination\_id = d.destination\_id

**and** d.country = %s

Per ricercare le eventuali offerte ho utilizzato una ulteriore interrogazione SQL:

**select** *o*.\*

**from** offer *o*

**where** o.trip\_package\_id = %s

**and** o.offer\_start <= %s

**and** o.offer\_end >= %s

### Valutazione di un viaggio passato

Una delle schede della applicazione permette ai clienti dell’agenzia di valutare uno dei viaggi passati. In questa pagina viene chiesto all’utente di inserire la propria mail, e alla pressione del bottone “Cerca i tuoi viaggi” vengono mostrati due menù a tendina: nel primo vengono elencati i viaggi prenotati dal cliente e già effettuati mentre nel secondo ci sono 5 opzioni che corrispondono ai voti che si possono dare per ogni viaggio (all’utente sono mostrati come stelline, mentre nel database sono salvati come interi da 1 a 5).

All’inizio della procedura è richiesto all’utente di inserire il suo indirizzo di mail. Successivamente il software controlla se venga inserita una mail inesistente nel database il programma. In caso affermativo, l’applicazione risponde con un alert nel quale evidenzia questo problema. Se invece l’utente è già registrato, ma non ha viaggi da valutare, viene mostrato sempre un alert, ma con un messaggio diverso dal precedente. Per recuperare tutti i viaggi valutabili dall’utente viene eseguita la seguente query:

**select** *tp*.trip\_package\_id , *d*.name , *tp*.trip\_start

**from** reservation *r*, traveler *t*, trip\_package *tp* , trip\_package\_has\_destination *tphd* , destination *d*

**where** *r*.Customer\_id = *t*.traveler\_id

**and** t.email = %s

**and** tp.trip\_start < %s

**and** r.trip\_package\_id = tp.trip\_package\_id

**and** tp.trip\_package\_id = tphd.trip\_package\_id

**and** d.destination\_id = tphd.destination\_id

**order** **by** *tp*.trip\_start **asc**

I parametri qui sono l’email dell’utente e la data di partenza del viaggio.

Qui di seguito riporto la funzione che si occupa di gestire i dati selezionati dal database per inserirli nella interfaccia utente nella pagina di valutazione.

def cercaViaggiUtente(self, e):  
 self.mail\_utente = self.view.mail\_utente.value  
 self.view.ddViaggi.value = None  
 self.view.ddVoti.value = None  
 if self.mail\_utente != "":  
  
 dizio = self.model.getViaggiUtente(self.mail\_utente, datetime.date.today())  
 if len(dizio) != 0:  
 *# dropdown viaggi* lista\_opzioni = []  
 for trip in dizio:  
 data = dizio[trip][0][1]  
 destinazioni = [x[0] for x in dizio[trip]]  
 stringa\_destinazioni = ", ".join(destinazioni) + " -- " + str(data)  
 lista\_opzioni.append(ft.dropdown.Option(key=trip, text=stringa\_destinazioni))  
 self.view.ddViaggi.options = lista\_opzioni  
  
 *#dropdown voto* lista\_voti = []  
 for i in range(1, 6):  
 lista\_voti.append(ft.dropdown.Option(key=i, text=i\*'★'))  
 self.view.ddVoti.options = lista\_voti  
 self.view.contentValuta.controls.append(self.view.rowValutazione)  
 else:  
 self.view.create\_alert(f"Nessun viaggio trovato per l'indirizzo mail: {self.mail\_utente}")  
 else:  
 self.view.create\_alert("Email non inserita!!")  
  
 self.view.update\_page()

La funzione getViaggiUtente del model permette di reperire i dati dal database collegandosi con il DAO ed eseguendo la query sopra riportata. Come si può notare i parametri passati dal controller sono l’email dell’utente e la data di “today”, ovvero quella in cui l’utente effettua la valutazione del viaggio. Gli alert sopra citati vengono creati grazie alla funzione della view “create\_alert”, passandogli come parametro il messaggio da far visualizzare. Nella “rowValutazione” della view oltre ai due “dropdown” è presente anche un bottone per inviare la valutazione al database.

Qui sotto riporto la funzione python che viene chiamata alla pressione di tale bottone.

def valutaViaggio(self, listaP):  
 try:  
 trip\_id = int(self.view.ddViaggi.value)  
 voto = int(self.view.ddVoti.value)  
  
 if not self.model.valutaViaggio(trip\_id, voto, self.mail\_utente):  
 self.view.create\_alert("Viaggio già valutato!")  
 else:  
 self.rigaRisultato = ft.Row(controls=[ft.Text("Valutazione registrata correttamente", color="blue", size=20)], alignment=ft.MainAxisAlignment.CENTER)  
 dlg = ft.AlertDialog(content=self.rigaRisultato)  
 self.view.page.dialog = dlg  
 dlg.open = True  
 self.svuotaParametri(listaP)  
 except TypeError:  
 self.view.create\_alert("Viaggio o valutazione non selezionati!!")  
 self.view.update\_page()

Il blocco “try-except” viene utilizzato per controllare che l’utente abbia effettivamente selezionato i parametri. La funzione del model “valutaViaggio” permette di inserire la valutazione nella tabella “reservation” del database con una query di “insert-into”. In seguito alla pressione del bottone verranno visualizzati degli alert di successo o di insuccesso. Il messaggio di insuccesso viene mostrato solamente se il viaggio selezionato ha già una valutazione registrata.

### Creazione di un viaggio personalizzato e ricorsione

La seconda scheda dell'applicazione offre la possibilità di creare un viaggio personalizzato. Questa pagina si apre automaticamente quando l'utente preme un pulsante sulla pagina principale, quella di prenotazione di un viaggio esistente, se nessuna delle opzioni elencate soddisfa le sue preferenze o se non ci sono viaggi che rispettano le sue esigenze. In ogni caso la scheda è selezionabile dal menù della pagina in ogni momento, nel caso in cui si desideri saltare il processo di ricerca e passare direttamente alla creazione del proprio viaggio. Anche in questa pagina vengono richiesti dei parametri all’utente e alla pressione del bottone “crea viaggio” il programma invia i dati al model nel quale viene creato un grafo. Questa struttura dati, presente nella libreria “NetworkX”, permette di rappresentare e gestire relazioni e connessioni tra oggetti. Un grafo è composto da nodi (oggetti) e archi (collegamenti tra oggetti). In questo programma i nodi sono rappresentati dalle attrazioni turistiche dei paesi selezionati dall’utente. La funzione getAllNodes del DAO esegue la seguente query parametrica dove i 3 parametri sono i 3 stati inseriti dall’utente.

**select** *ta*.tourist\_attraction\_id **as** *id*, *ta*.cost **as** *cost* , *d*.country **as** *country* , *ta*.name **as** *nameAtt*, *d*.name **as** *nameDest*, *d*.destination\_id **as** *dest\_id*

**from** tourist\_attraction *ta* , destination *d*

**where** *ta*.destination\_id = *d*.destination\_id

**and** (d.country =%s **or** d.country =%s **or** d.country =%s)

Non c’è l’obbligo di inserire 3 stati, basta che se ne selezioni uno solo. Per questo motivo la query è stata progettata per poter funzionare anche quando uno o due stati sono mancanti (hanno cioè valore NULL).

Due nodi sono collegati tra di loro se hanno un prezzo “simile”, ovvero se il loro costo ha una differenza di al massimo due unità. Qui riporto il codice python che ne descrive il funzionamento.

self.grafo.add\_nodes\_from(DAO.getAllNodes(stato1, stato2, stato3))  
for attrazione in list(self.grafo.nodes):  
 for altra\_attrazione in list(self.grafo.nodes):  
 if attrazione.id != altra\_attrazione.id and abs(  
 attrazione.cost - altra\_attrazione.cost) <= 2:  
 self.grafo.add\_edge(attrazione, altra\_attrazione)

Ciò è stato fatto per velocizzare il processo di ricorsione che verrà eseguito in seguito alla istruzione appena descritta. La ricorsione è una tecnica di programmazione in cui una funzione chiama se stessa più volte per risolvere un problema altrimenti complesso da risolvere in altre maniere. Qui riporto le funzioni in python che portano alla formazione della miglior combinazione possibile di attrazioni turistiche attraverso un processo ricorsivo.

def creaViaggio(self, stato1, stato2, stato3, numeroAttr):  
 self.grafo.clear()  
 self.grafo.add\_nodes\_from(DAO.getAllNodes(stato1, stato2, stato3))  
 for attrazione in list(self.grafo.nodes):  
 for altra\_attrazione in list(self.grafo.nodes):  
 if attrazione.id != altra\_attrazione.id and abs(  
 attrazione.cost - altra\_attrazione.cost) <= 2:  
 self.grafo.add\_edge(attrazione, altra\_attrazione)  
 stati = [stato1, stato2, stato3]  
 statiNonNulli = [x for x in stati if x is not None]  
 self.solBest = []  
 self.bestCosto = float(math.inf)  
  
  
 for i in list(self.grafo.nodes):  
 parziale = [i]  
 if int(numeroAttr) > len(self.grafo.nodes):  
 numeroAttr = len(self.grafo.nodes)  
 self.ricorsione(i, parziale, int(numeroAttr), statiNonNulli, costo=0)

statiUsati = set()  
if self.solBest == []:  
 self.bestCosto = 0  
 for i in list(self.grafo.nodes):  
 if i.country not in statiUsati:  
 self.solBest.append(i)  
 self.bestCosto += i.cost  
 statiUsati.add(i.country)

return self.solBest, self.bestCosto  
  
def ricorsione(self, v, parziale, numeroAttr, stati, costo):  
 if costo >= self.bestCosto:  
 return  
 if len(parziale) == numeroAttr:  
 statiDiversi = set(x.country for x in parziale)  
 if len(statiDiversi) == len(stati):  
 if costo < self.bestCosto:  
 self.solBest = copy.deepcopy(parziale)  
 self.bestCosto = costo  
 print('soluzione trovata')  
  
 vicini = list(nx.neighbors(self.grafo, v))  
 viciniAmmissibili = self.getAmmissibili(vicini, parziale, numeroAttr)  
 for v in viciniAmmissibili:  
 parziale.append(v)  
 newCosto = costo + v.cost  
 self.ricorsione(v, parziale, numeroAttr, stati, newCosto)  
 parziale.pop()  
  
  
  
def getAmmissibili(self, vicini, parziale, numeroAttr):  
 if len(parziale) == numeroAttr:  
 return []  
 ammissibili = []  
 parziale\_set = set(parziale)  
 for v in vicini:  
 if v not in parziale\_set:  
 ammissibili.append(v)  
 return ammissibili

Le migliori attrazioni sono state selezionate imponendo alcuni vincoli in modo che per l’utente risulti una soluzione ottima: deve esserci almeno una attrazione per ogni stato selezionato, il numero di attrazioni deve essere pari a quello specificato dall’utente (nel caso in cui il numero di attrazioni richieste superi il totale delle attrazioni disponibili, il numero di attrazioni sarà impostato al totale delle attrazioni disponibili) e deve avere il costo più basso possibile. Questo vincolo è stato impostato basandosi sulle esperienze personali di viaggio, in cui è sempre vantaggioso cercare di ottimizzare i costi quando possibile.

L’algoritmo ricorsivo non si ferma finché non trova una soluzione che rispetti tutti i vincoli, terminando con la restituzione dell’elenco delle attrazioni e del costo complessivo, modo che l’utente possa visualizzare ciò che è stato trovato per il suo viaggio.

Ho inserito un eventuale controllo sulla effettiva riuscita del programma, perché nel caso in cui il grafo non abbia archi o ne abbia meno rispetto ai nodi, la ricorsione non riuscirà a trovare una soluzione, in quanto effettua tante volte la funzione sui successori dell’ultimo nodo inserito. Se questo non ha successori o ne ha solo del proprio stato la funzione non troverà mai una “solBest”, quindi in caso di destinazioni con poche attrazioni la soluzione sarà composta solamente da una attrazione per stato.

### Auto compilazione dei dati utente

Superata la fase di selezione o creazione del viaggio all’utente non resta altro che inserire i propri dati. Per fare ciò si aprirà in automatico una nuova scheda chiamata “inserisci dati” alla pressione dei bottoni per prenotare il viaggio. Questa pagina contiene alcuni campi di input testuale in cui l’utente dovrà inserire i propri dati anagrafici. Nel caso in cui l’utente sia sicuro di aver già effettuato prenotazioni presso l’agenzia di viaggi ha la possibilità di saltare il processo di compilazione dei campi di input. Basterà inserire la propria mail e cliccare sul bottone “Auto-compila” e il programma procederà a compilare i campi in automatico come si può notare dal seguente codice python.

def autoCompila(self, email):  
 if email.value is None or email.value == "":  
 self.view.create\_alert("Email non inserita")  
 return  
 traveler = self.model.autoCompila(email.value)  
 if traveler:  
 self.traveler = traveler  
 else:  
 self.view.create\_alert("Email non presente nel database, compilare a mano")  
 return  
 self.name.value = self.traveler.name  
 self.surname.value = self.traveler.surname  
 self.age.value = self.traveler.age  
 self.address.value = self.traveler.address  
 self.phone.value = self.traveler.phone  
 self.gender.value = self.traveler.gender  
 self.view.update\_page()

La funzione autoCompila del model restituisce un oggetto della classe Traveler oppure None se non trova una corrispondenza tra le mail del database e quella inserita dall’utente. La classe Traveler ha tutti gli attributi dei viaggiatori presenti nella omonima tabella del databse. Se la variabile “traveler” è nulla viene mostrato un alert che esplicita questo e invita a procedere con la compilazione manuale dei dati. Tutte le variabili al fondo della funzione sono i campi di input della pagina in questione, e verranno auto compilati con i dati trovati nel database.

# Diagramma delle classi principali

Riporto qui i diagrammi delle classi principali, ovvero *Model* e *Controller.*

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

Immagine che contiene testo, schermata, documento, Carattere

Descrizione generata automaticamente

# Videate dell’applicazione

Immagine che contiene testo, Carattere, linea, numero

Descrizione generata automaticamente

*Fig. 6.1 Screenshot della videata “Prenota”*

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

*Fig. 6.2 Screenshot della videata “Crea”*

Immagine che contiene testo, schermata, linea, numero

Descrizione generata automaticamente

*Fig. 6.3 Screenshot della videata “Valuta”*

Immagine che contiene testo, software, linea, Carattere

Descrizione generata automaticamente

*Fig. 6.4 Screenshot della videata “Inserisci dati”*

Il video dimostrativo dell’applicazione software è disponibile al seguente link: [video\_dimostrativo](https://www.youtube.com/watch?v=aG2oLluywMc)

# Risultati sperimentali

Il programma ha tempi di esecuzione decisamente brevi, infatti le query sono eseguite pressoché istantaneamente. L’unico algoritmo utilizzato che ha tempi variabili e in alcuni casi molto lunghi è quello ricorsivo per la ricerca del miglior itinerario possibile. A scopo statistico ho voluto riportare in un grafico i tempi di esecuzione dell’algoritmo nel caso in cui l’utente selezioni solo un stato ed ho effettuato la ricerca per ogni stato con un numero di attrazioni che va da 1 a 9. I risultati sono riportati nella figura sottostante

Immagine che contiene diagramma, testo, Diagramma, linea

Descrizione generata automaticamente

*Fig. 7.1 Grafico con i tempi di esecuzione dell’algoritmo ricorsivo*

Praticamente tutti gli stati hanno tempi molto brevi (intorno a 10-2 s), tranne alcuni che ci impiegano decisamente di più. Questo è dovuto all’alto numero di attrazioni in questi stati, ciò aumenta notevolmente le iterazioni dell’algoritmo e di conseguenza i tempi di esecuzione.

# Valutazioni sui risultati ottenuti e conclusioni

Il programma da me realizzato rispecchia gli obiettivi prefissati, utilizzando un'interfaccia grafica semplice e user-friendly. Il suo punto di forza risiede nella rapidità di esecuzione delle interrogazioni al database. Grazie alla struttura ben organizzata del database, è stato possibile garantire un accesso rapido ai dati tramite query relativamente semplici. Dal punto di vista gestionale l’obiettivo è stato raggiunto, ovvero creare una applicazione che sia in grado di aiutare una agenzia di viaggi nella ricerca dei dati all’interno dei propri database e che possa automatizzare il processo di creazione di un viaggio personalizzato.

La gestione degli errori di input dell'utente è stata impostata in modo efficace, con un messaggio generato automaticamente per ogni errore rilevato, che invita l'utente a correggere l'input e riprovare.

L’unico punto di debolezza, già analizzato nel paragrafo precedente, è l’algoritmo ricorsivo che per alcuni casi risulta essere lento, tuttavia i risultati forniti sono comunque sempre i migliori possibili.

# Licenza



Questo documento `e condiviso con licenza Creative Commons BY-NC-SA 4.0.

Tu sei libero di:

* + Condividere — riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare questo materiale con qualsiasi mezzo e formato
  + Modificare — remixare, trasformare il materiale e basarti su di esso per le tue opere

Il licenziante non può revocare questi diritti fintanto che tu rispetti i termini della licenza.

Alle seguenti condizioni:

* + Attribuzione — Devi riconoscere una menzione di paternità adeguata , fornire un link alla licenza e indicare se sono state effettuate delle modifiche . Puoi fare ciò in qualsiasi maniera ragionevole possibile, ma non con modalità tali da suggerire che il licenziante avalli te o il tuo utilizzo del materiale.
  + NonCommerciale — Non puoi utilizzare il materiale per scopi commerciali.
  + StessaLicenza — Se remixi, trasformi il materiale o ti basi su di esso, devi distribuire i tuoi contributi con la stessa licenza del materiale originario.
  + Divieto di restrizioni aggiuntive — Non puoi applicare termini legali o misure tecnologiche che impongano ad altri soggetti dei vincoli giuridici su quanto la licenza consente loro di fare.

Note:

Non sei tenuto a rispettare i termini della licenza per quelle componenti del materiale che siano in pubblico dominio o nei casi in cui il tuo utilizzo sia consentito da una eccezione o limitazione prevista dalla legge. Non sono fornite garanzie. La licenza può non conferirti tutte le autorizzazioni necessarie per l’utilizzo che ti prefiggi. Ad esempio, diritti di terzi come i diritti all’immagine, alla riservatezza e i diritti morali potrebbero restringere gli usi che ti prefiggi sul materiale.

Per una copia della licenza completa, visita <https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>