Politecnico di Torino

Corso di Laurea Triennale in Ingegneria Gestionale
Classe L8 – Ingegneria dell'Informazione
A.A. 2023/2024

Algoritmo ricorsivo: realizzazione di un applicativo di supporto alle vendite per una concessionaria

Relatore
Prof. Fulvio Corno

Candidato
Rosario Interlandi

Indice

Elenco delle figure

Capitolo 1

Proposta di progetto

1.1 Studente proponente

s284166 Interlandi Rosario

1.2 Titolo proposta

Algoritmo ricorsivo: realizzazione di un applicativo di supporto alle vendite per una concessionaria

1.3 Descrizione del problema proposto

L'avvento dell'era digitale ha avuto un impatto significativo in tutti i settori economici, portando alla necessità di interfacciarsi con i vari dispositivi ICT sviluppando software volti alla promozione e alla gestione delle proprie attività.

Un settore in cui sono state applicate queste nuove soluzioni è quello automobolistico, il quale ha sfruttato il potenziale di questi dispositivi per la promozione delle loro auto. Nel seguente progetto è stato sviluppato un software dedicato: alla ricerca avanzata delle auto di una concessionaria secondo vincoli imposti dall'utente, con lo scopo di migliorare l'esperienza dei clienti nella ricerca dell'auto; una ricerca intelligente che, in base a dei parametri di ricerca selezionati dal cliente, mostri un insieme di auto da acquistare che massimizzi il parametro scelto. Le auto di tale insieme sono presenti all'interno della concessionaria e devono rispettare il tetto massimo imposto dall'utente, tale implementazione ha come scopo la promozione di "cluster di

auto" che potrebbero interessare terzi rivenditori interessati a comprare più auto.

1.4 Descrizione della rilevanza gestionale del problema

La rilevanza gestionale di questo problema consiste nell' ottimizzazione dei tempi di ricerca, utile al miglioramento dell'esperienza del cliente il quale si concretizza in miglioramento della reputazione dell'azienda. Il software potrebbe essere utilizzato anche per la raccolta dati utili per previsioni future in base alle richieste dei clienti e per comprendere quali prodotti promuovere maggiormente. Attraverso la ricerca intelligente si cerca di incrementare i ricavi spingendo il cliente a comprare più auto

1.5 Descrizione dei data-set per la valutazione

Il dataset utilizzato per questa analisi è stato ottenuto da Kaggle[1], una piattaforma che fornisce dataset opensource. La tabella principale, denominata "car details v4.csv", è stata modificata con l'aggiunta di una colonna ID per fornire un codice univoco a ciascun veicolo e semplificare la gestione delle chiavi. Inoltre, i dati della colonna "Max Power" sono stati suddivisi in due colonne separate, "bhp" e "rpm", per renderli confrontabili mentre la colonna "Max torque" è stata rimossa.

Il dataset risultante presenta le seguenti colonne:

1. **Id (INT):** Un codice univoco che identifica l'auto nel database.
2. **Make (VARCHAR):** Stringa che identifica la casa automobilistica.
3. **Model (VARCHAR):** Stringa che identifica il modello del veicolo.
4. **Price (INT):** Valore intero rappresentante il prezzo del veicolo in rupie.
5. **Year (INT):** Anno di produzione del veicolo.
6. **Kilometer (INT):** Numero di chilometri percorsi dal veicolo.
7. **Fuel_Type (VARCHAR):** Tipo di carburante utilizzato dal motore.

[1] `https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho?select=car+details+v4.csv`

8. **Transmission (VARCHAR):** Tipo di cambio del veicolo.
9. **Location (VARCHAR):** Luogo di provenienza del veicolo.
10. **Color (VARCHAR):** Colore del veicolo.
11. **Owner (VARCHAR):** Indicazione se l'auto è nuova o usata.
12. **Seller_Type (VARCHAR):** Tipo di venditore.
13. **Engine (INT):** Numero di cavalli del motore.
14. **bhp (INT):** Potenza massima in bhp.
15. **rpm (INT):** Potenza massima in rpm.
16. **Drivetrain (VARCHAR):** Tipo di trasmissione del veicolo.
17. **Length (DOUBLE):** Lunghezza del veicolo in mm.
18. **Width (DOUBLE):** Larghezza del veicolo in mm.
19. **Height (DOUBLE):** Altezza del veicolo in mm.
20. **Seating_Capacity (INT):** Numero massimo di sedili.
21. **Fuel_Tank_Capacity (DOUBLE):** Massima capacità del serbatoio in litri.

1.6 Descrizione preliminare degli algoritmi coinvolti

Per la realizzazione del progetto il linguaggio usato è Java, compilato rispettando il pattern MVC (Model View Control) usando il framework JavaFX, attraverso il quale vengono separati la logica applicativa dall'interfaccia utente, e il DAO (Data Access Object) per separare l'accesso al database dall'applicazione attraverso l'API JDBC.

Inizialmente verrà richiesto all'utente di inserire alcuni vincoli, non è obbligato a compilarli tutti ma può anche selezionarne una parte. E' stato scelto di non usare tutti gli attributi del veicolo nel filtro per evitare un set di auto poco numeroso in quanto su quel set bisogna sviluppare algoritmi ricorsivi.

Gli algoritmi ricorsivi usati nel software hanno lo scopo di mostrare a un possibilie rivenditore di auto un insieme di auto da acquistare. Nello specifico

viene consetito all'utente di selezionare uno dei tre parametri da applicare alla ricerca ricorsiva, in base al parametro scelto l'algoritmo ricorsivo andrà a massimizzare o minimizzare metriche differenti.

- Se viene scelto il parametro "Numero" allora la funzione ricorsiva avrà lo scopo di cercare l'insieme con il maggiore numero di auto presenti, andando quindi a massimizzare il numero di elementi presenti nell'insieme
- L'algoritmo ricorsivo con parametro selezionato "Valore" mira a massimizzare il valore delle auto presente nel set di dati, andando quindi a massimizzare il prezzo totale dato dalla somma del prezzo di ogni singola auto contenuta nel set.
- Il parametro "Migliore" è differente dai primi due in quanto i primi due erano parametri quantitativi e l'attributo da massimizzare è facilmente deducibile. In questo caso per migliore viene inteso che vengono cercate le auto in migliori condizioni in quanto se sono in migliori condizioni sono molto più facile da rivendere (obiettivo del cliente). E' stato scelto quindi di minimazzare la media dei chilometri percorsi dell'insieme delle auto, dal momento che minori chilometri percorsi implicano meno utilizzo da cui è presumibile che siano meno usurate e in migliori condizioni.

1.7 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'utente inizialmente dovrà inserire alcuni vincoli, non è obbligato a inserirli tutti ma potrà anche selezionarne solo alcuni. Dopo di che verrà popolata una tendina con tutte le auto che sono presenti nel set di auto filtrate, le quali soddisfano i vincoli imposti dall'utente. Dopo averne scelta una, aver inserito il budget disponibile e aver scelto uno dei tre parametri alla pressione di un bottone verrà calcolato il migliore insieme di auto secondo il modello descritto nel punto precedente.

Capitolo 2

Descrizione dettagliata del problema affrontato

2.1 Contesto operativo/aziendale

Nell'era moderna ogni persona è dotata di un dispositivo ICT che usa quotidianamente per molte ore al giorno, è diventato quindi necessario per qualsiasi azienda che venda prodotti, interfacciarsi con questa nuova realtà per stare al passo con i tempi e non rimanere indietro. L'introduzione di questi dispostivi ha portato numerosi vantaggi e cambiamenti rilevanti al mercato odierno e al mondo del lavoro. Oggi avere competenze informatiche e familiarità con tali dispositivi è un requisito richiesto dalla maggior parte, se non la totalità, delle aziende per trovare un impiego. Il mondo del lavoro è stato radicalmente rivoluzionato grazie a tali tecnologie, ogni azienda, oltre al prodotto che vende, deve fornire ai clienti servizi web dedicati alla promozione e al supporto della clientela per monitorare l'andamento del mercato e trarne vantaggi sui numerosi competitor.

In questo progetto è stato realizzato un software di una ipotetica concessionaria che mira proprio alla fornitura di tale servizio, andando a mostrare i veicoli che più soddisfano i requisiti di un cliente, andando a creare un filtro, al fine di ottimizzare i tempi di ricerca del prodotto da acquistare. Realizzare un software user-friendly è uno di quegli accorgimenti che fa la differenza tra un e-commerce e un'altro.

2.2 Input, output e criticità

Nei campi di input l'utente potrà selezionare qualsiasi vincolo vuole che la macchina abbia, non è forzato a inserire vincoli non richiesti. In particolare

gli sarà concesso di selezionare: il numero massimo di chilometri percorsi dal veicolo, andando a filtrare il db selezionando le sole auto che hanno un chilometraggio minore al valore inserito; il prezzo massimo che l'auto dovrà avere; il tipo di carburante desiderato (es. diesel, petrol etc.); il colore di cui vuole l'auto; la marca dell'auto e se la vuole di prima o seconda mano. Potevano essere aggiunti altri vincoli come i cavalli del motore o altro ma si è scelto di mettere solamente questi per evitare che la dimensione del set di auto filtrate sia troppo limitata. Una volta trovato l'insieme di veicoli, dopo la pressione del bottone "Cerca veicolo", la tabella sottostante sarà popolata con i veicoli che soddisfano le condizione scelte dall'utente andando a indicare per ognuna di esse: ID, marca, modello, cavalli del motore, anno di produzione, chilometri percorsi e il prezzo.

Nella ricerca intelligente, denominata "Migliori auto", la quale sarà fruibile solamente dopo aver fatto la ricerca, l'utente dovrà selezionare un'auto tra quelle presenti nell'insieme dei veicoli filtrati tramite un apposita tendina, il budget disposto a usare per l'acquisto del cluster di vetture e uno dei tre parametri da applicare alla ricerca (migliore, valore e numero) . Alla pressione del bottone "Trova combinazione migliore" verrà avviato l'algoritmo ricorsivo che, partendo dall'auto selezionata, cerca il miglior cluster di auto, ricercandolo tra l'insieme di auto restituito dalla ricerca filtrata, che: minimizzi la media dei chilometri percorsi rispettando il vincolo imposto sul budget se il parametro selezionato sia "Migliore"; massimizzi il numero di veicoli presenti nell'insieme se il parametro selezionato sia "Numero"; nel caso il parametro selezionato fosse "Valore" il valore da ottimizzare sarebbe il prezzo totale dell'insieme . Il risultato sarà stampato all'interno di una text area, indicando inizialmente in prezzo totale del cluster trovato, indicando per ogni auto presente nell'insieme il suo ID, la marca, il modello e l'anno di produzione tra parentesi tonde.

Nonostante all'utente sia lasciata la facoltà di scegliere quali vincoli usare è consigliabile applicare più filtri possibili in quanto, per la seconda sezione, se vengono applicati pochi filtri si ha un insieme con molti record e il costo computazionale per applicare l'algoritmo ricorsivo potrebbe risultare eccessivamente oneroso rischiando che il programma smetta di funzionare.

Capitolo 3

Descrizione del data-set utilizzato per l'analisi

Il dataset utilizzato contiene informazioni riguardanti le vetture vendute da una concessionaria raccolte tutte in una unica tabella, presa da Kaggle a cui sono state apportate le modifiche descritte in 1.5. Nella tabella tutte le unità di riferimento sono quelle del sistema internazionale di unità di misura a eccezione della distanza percorsa (km al posto del metro) mentre il prezzo è espresso in rupie indiane.

Non tutti gli attributi della tabella sono stati presi in considerazione per la realizzazione del programma. Per la prima sezione dove viene fatto il filtraggio dei veicoli vengono utilizzate solo le colonne: Kilometer, Price, Fuel_type, Color, Make e Owner per la parte dove l'utente inserisce gli input, ovvero in base agli input dell'utente il filtro viene applicato solo su questi attributi. Sempre nella stessa sezione vengono utilizzate anche le colonne: id, Make, Model, Engine, Year, Kilometer e Price per visualizzare il risultato della ricerca in una tabella.

Nella seconda sezione, "Migliori auto", vengono usate le colonne: id, Make, Model e Year per popolare la tendina dell'auto da selezionare e per il risultato; Price per il prezzo totale del cluster di auto trovate dall'algoritmo ricorsivo.

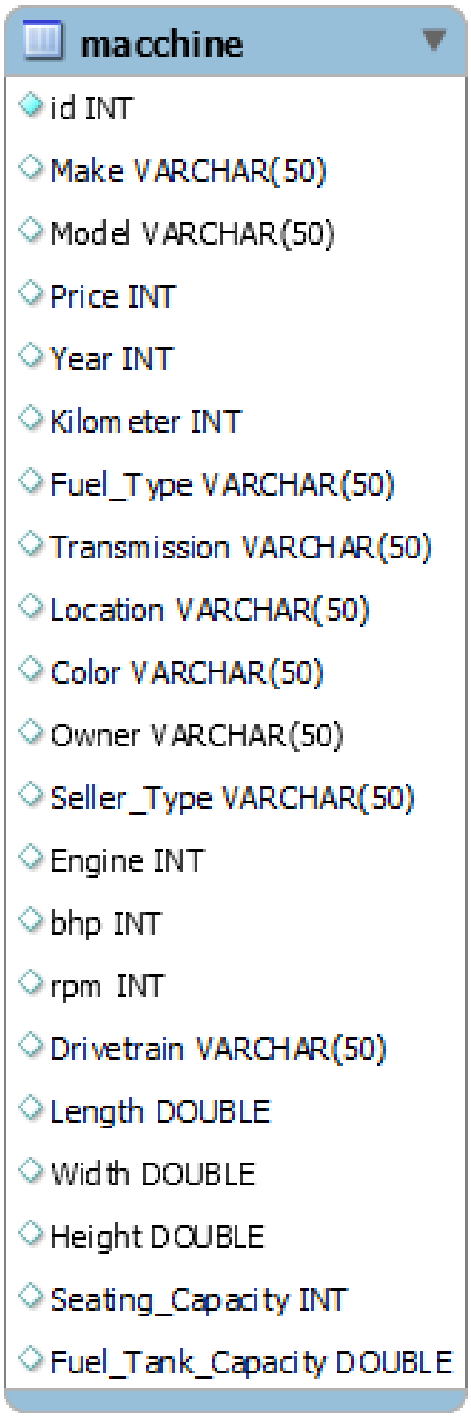

Figura 3.1: Diagramma ER: tabella macchine

Capitolo 4

Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati

4.1 Descrizione architettura e classi

Il programma è stato realizzato in Java rispettando i pattern di programmazione MVC (Model View Control) e DAO (Data Access Object), andando quindi a rendere il programma scalabile separando le varie responsabilità di presentazione, modello e controllo e fornendo un interfaccia con il database. Il progetto è strutturato in tre pacchetti:

- **it.polito.s284166.Tesi**: in questo package viene gestitata l'architettura dell'applicativo, sono presenti tre classi :
  - Main.java: Classe dove viene eseguito il codice che inizializza ed esegue il programma JavaFX.
  - EntryPoint.java: Classe dove viene gestito in controllo dell'applicativo: in questa classe vengono inizializzati il modello, il controller e la scena; il modello viene associato al controller e viene settato il CSS della scena.
  - FXMLController.java: Classe dove viene gestita l'impostazione dell'interfaccia utente (andando a inizializzare gli item dell'interfaccia e popolandoli) e gli eventi scatenati dall' interazione dell'utente con l'interfaccia andando a richiamare metodi del modello.
- **it.polito.s284166.Tesi.db**: in questo pacchetto viene gestito l'accesso e l'interazione tra modello e database, sono presenti due classi:

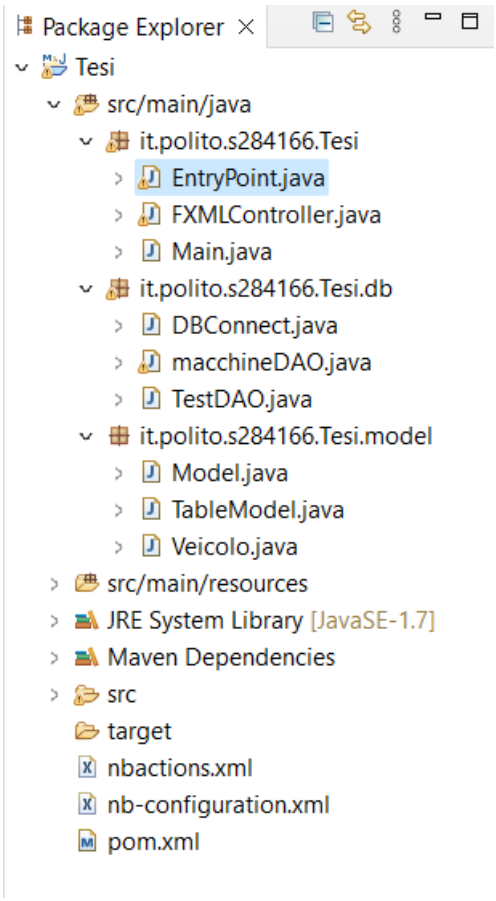

Figura 4.1: Package Explorer

– DBConnect.java: questa è una classe di configurazione dove vengono settati l'URL (Uniform Resource Locator) e le credenziali di accesso del database e vengono impostate delle proprietà nel datasource in modo che possano essere utilizzate delle prepared statement per eseguire query SQL parametriche, evitando in questo modo attacchi di tipo SQL injection.

– macchineDAO.java: in questa classe sono presenti i metodi per recuperare i dati dal database e fornirli al modello organizzati in array. Sono presenti tre metodi: *getColor()*, *getFuelType()* e *getMarcaType()* resitituiscono array di stringhe di tutti i colori, tipi di carburante e case automobilistiche rispettivamente presenti nel database, sono state create per popolare le tendine all'avvio del programma; il metodo *applyFilter()* restituisce un array di veicoli che rispettano i vincoli imposti dall'utente e che quindi compor-

ranno i record dell'insieme di auto filtrate. Per la creazione della query di quest'ultimo metodo è stata utilizzata la classe StringBuilder del package java.lang, la quale consente di modificare una stringa senza creare un nuovo oggetto stringa ad ogni modifica, funzione molto efficiente in termini di memoria che mi permette di modificare la stringa dopo aver fatto controlli sulla presenza di determinati parametri e la rimozione di sottostringhe che porterebbero errori nell'esecuzione della query ("AND" finale o "WHERE" quando non sono stati imposti vincoli sulla ricerca).

- **it.polito.s284166.Tesi.model**: in questo pacchetto sono presenti le classi per la logica funzionale del programma. Sono presenti tre classi:
  - Model.java: classe principale dove vengono salvate stutture dati (array), vengono richiamati metodi del DAO e dove viene svolta la logica applicativa.
  - TableModel.java: classe creata per popolare la tabella di output della prima sezione, ha gli stessi attributi della tabella descritta nel capitolo 3 e per ognuno di essi i getter e i setter.
  - Veicolo.java: questa è la classe creata per rappresentare i vertici, presenta tutti gli attributi delle colonne del database e i getter.

4.2 Ricerca per filtro

All'interno della classe model, troviamo il metodo *doRicerca()* che prende in input i valori passati come parametri dal controller, valori che rappresentano dati che determinano i vincoli da applicare. vengono trovati i record, istanze della classe "Veicolo" del package **it.polito.s284166.Tesi**, attraverso il metodo *applyFilter()* della classe macchineDAO che restituisce una lista di oggetti "Veicolo", salvati in un arraylist.

```
public void doRicerca(Integer kilometer,Integer price, String fuel, String color,String ownerType,String Marca ) {
    this.autoFiltrate = this.dao.applyFilter(kilometer, price, fuel, color,ownerType, Marca);
}
```

Figura 4.2: Codice metodo doRicerca()

```
StringBuilder sqlBuilder = new StringBuilder("SELECT * FROM macchine WHERE");

List<Object> parametriPresenti = new ArrayList<>();

if (kilometer != null) {
    sqlBuilder.append(" Kilometer <= ? AND");
    parametriPresenti.add(kilometer);
}
if (price != null) {
    sqlBuilder.append(" Price <= ? AND");
    parametriPresenti.add(price);
}
if (fuel != null) {
    sqlBuilder.append(" Fuel_Type = ? AND");
    parametriPresenti.add(fuel);
}
if (color != null) {
    sqlBuilder.append(" Color = ? AND");
    parametriPresenti.add(color);
}

if (ownerType != null) {
    sqlBuilder.append(" Owner = ? AND");
    parametriPresenti.add(ownerType);
}
if (marca != null) {
    sqlBuilder.append(" Make = ? AND");
    parametriPresenti.add(marca);
}

if (sqlBuilder.toString().endsWith("AND")) {
    sqlBuilder.setLength(sqlBuilder.length() - 3);
}
if (sqlBuilder.toString().endsWith("WHERE")) {
    sqlBuilder.setLength(sqlBuilder.length()-5);
}
```

Figura 4.3: Codice creazione query del metodo applyFilter()

4.3 Algoritmo ricorsivo

Per la ricerca delle migliori auto simili a quella selezionata è stato utilizzato un algoritmo ricorsivo. Il controller si interfaccia con questo algoritmo attraverso il metodo *getBestSolution()* a cui passerà come parametro l'auto desiderata dall'utente, il budget dell'utente espresso con un integer e uno dei tre parametri che cambiano la funzione obiettivo dell'algoritmo ricorsivo. In questo metodo vengono inizializzati tutte le strutture e i punteggi necessari a far funzionare l'algoritmo, viene chiamato il metodo *ricorsione()* e infine viene restituito il risultato dell'algoritmo in una lista di oggetti "Veicolo".

Il metodo *ricorsione()* non ha valori di ritorno ma va a modificare direttamente il punteggio e la lista "bestPath" che viene restituita dal metodo precedente e riceve in input la lista parziale, che sarà la lista in cui verranno aggiunti ed eliminati veicoli filtrati, il budget imposto dall'utente e il tipo di

```
public List<Veicolo> getBestSolution(Veicolo root, Integer budget, String type) {
    this.bestPath = new ArrayList<>();
    if (type.compareTo("Migliore")==0) {
        this.bestScore = 1000000000.0;
    }else if (type.compareTo("Numero")==0) {
        this.bestScore = 0.0;
    }else if (type.compareTo("Valore")==0) {
        this.bestScore = 0.0;
    }
    this.root = root;
    List<Veicolo> parziale = new ArrayList<>();
    parziale.add(this.root);
    ricorsione(parziale,budget,type);
    return this.bestPath;
}
```

Figura 4.4: Codice metodo getBestSolution()

ricerca descritto in una stringa.

Questo metodo presenta tutte le caratterstiche di una tipica funzione ricorsiva. Inizialmente viene preso l'ultimo record della lista parziale, alla prima iterazione sarà l'auto scelta dall'utente, su cui bisogna controllare se sia possibile aggiungere altri veicoli senza violare i vincoli di uscita. Ha delle condizioni di uscita, ovvero che il budget non venga superato. Per calcolare il prezzo totale dei veicoli presenti nella lista parziale viene chiamato il metodo *actualWeight()* che prende in input la lista parziale e va a sommare il prezzo di ogni veicolo presente nella lista passata come input.

```
private Integer actualWeight(List<Veicolo> parziale) {
    int result = 0;
    Veicolo d_1= this.root;
    for (int i = 1; i<parziale.size();i++) {

        result += this.graph.getEdgeWeight(this.graph.getEdge(d_1,parziale.get(i))
        d_1 = parziale.get(i);

    }
    return result;
}
```

Figura 4.5: Codice metodo actualWeight()

Dopo aver visto che l'ultimo veicolo aggiunto rispetti i vincoli, viene calcolato il punteggio della lista parziale, se tale punteggio fosse minore o maggiore(a seconda del parametro scleto) del punteggio migliore registrato fino a quel momento il bestPath viene sostituito con una copia della lista parziale e il miglior punteggio viene aggiornato. Nel caso il parametro selezzionato fosse "Migliore", il punteggio viene calcolato con il metodo *getScore()* che riceve come parametro la lista parziale e calcola la media dei chilometri percorsi da tutti i veicoli presenti nella lista restituendo la media come un double.

```
private Double getScore(List<Veicolo> parziale) {
    Double score = 0.0;
    for (Veicolo v: parziale) {
        score += v.getKilometer();
    }
    score = score/parziale.size();
    return score;
}
```

Figura 4.6: Codice metodo getScore()

Se il parametro selezionato fosse "Numero" lo score è dato dalla dimensione dell'insieme quindi il punteggio viene calcolato con il metodo size() della classe Arraylist del framework java.util.

Infine se viene selezionato il parametro "Valore" il punteggio viene calcolato con il metodo *actualWeight()* usato per il vincolo sul budget dato che in entrambi i casi si vuole calcolare il prezzo totale dell'insieme di auto.

E' importante notare che dopo l'aggiornamento del miglior percorso non viene terminata l'esecuzione della funzione in quanto potrebbero ancora essere aggiunte auto alla lista che potrebbero minimizzare ulteriormente il punteggio ottenuto.

Arrivati a questo punto vengono presi tutti i veicoli della lista iniziale e, per ognuno di essi, se non sono già presenti nella lista parziale, si prova ad aggiungerlo all lista, si richiama il metodo *ricorsione()* e infine si trova una istruzione che rimuove il record "appena" aggiunto. Si arriva a quest'ultima istruzione quando viene terminato il metodo ricorsione, ovvero quando il valore della lista parziale supera il budget, quindi si vuole rimuovere l'ultima auto aggiunta e provare ad aggiunngerne un'altra che potrebbe rientrare nel budget, tale operazione è chiamata backtracking.

```java
private void ricorsione(List<Veicolo> parziale, Integer budget,String type) {

    if (this.actualWeight(parziale)> budget) {
        return;
    }
    if (type.compareTo("Migliore")==0) {
        if (getScoreMigliore(parziale)< this.bestScore) {
            this.bestScore = getScoreMigliore(parziale);
            this.bestPath = new ArrayList<>(parziale);
        }
    }else if (type.compareTo("Numero")==0) {
        if (parziale.size()>this.bestScore) {
            this.bestScore = Double.valueOf(parziale.size());
            this.bestPath = new ArrayList<>(parziale);
        }
    }else if (type.compareTo("Valore")==0) {
        if (actualWeight(parziale) > this.bestScore) {
            this.bestScore = Double.valueOf(this.actualWeight(parziale));
            this.bestPath = new ArrayList<>(parziale);
        }
    }
    List<Veicolo> successori = new ArrayList<>(autoFiltrate);

    for (Veicolo successor: successori ) {
        if (!parziale.contains(successor)) {
            parziale.add(successor);
            ricorsione(parziale,budget,type);
            parziale.remove(successor);
        }

    }

}
```

Figura 4.7: Codice metodo ricorsione()

Capitolo 5

Diagramma delle classi delle parti principali

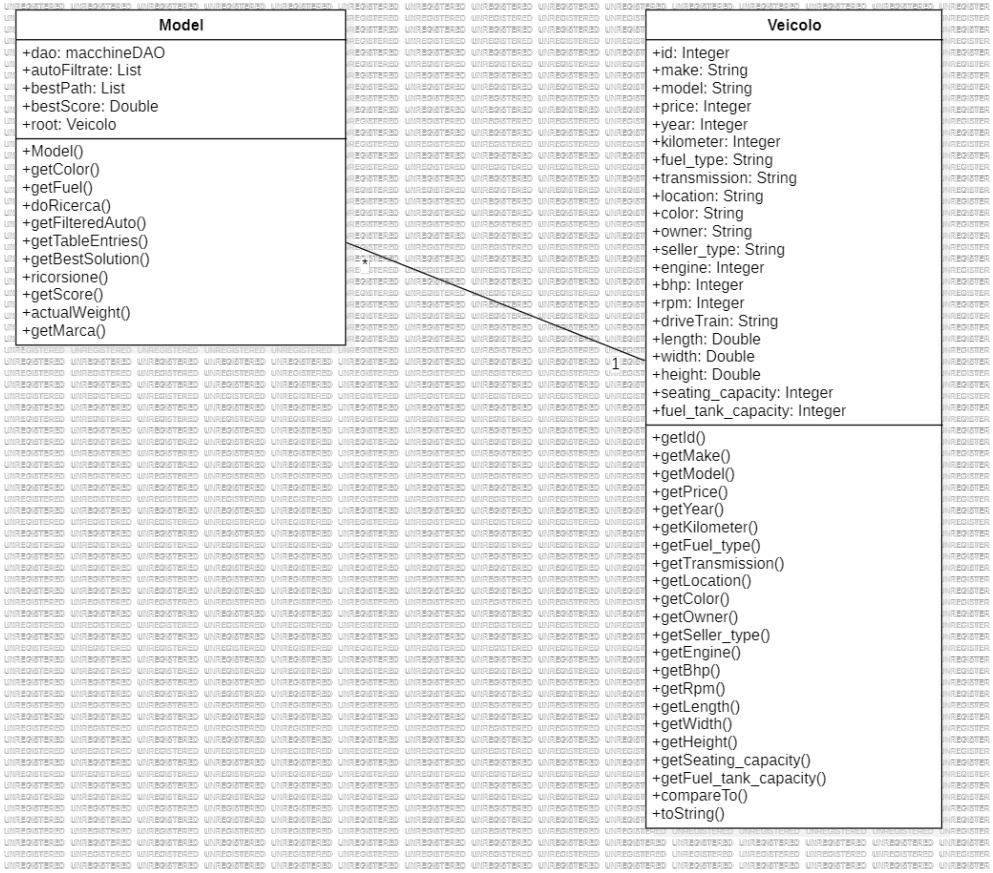

Figura 5.1: Diagramma delle classi

Capitolo 6

Videate dell'applicazione realizzata e link al video dimostrativo del software

6.1 Link video software

Nel seguente link `https://youtu.be/hZkOR1db05c` è presente il video registrato che mostra il funzionamento del software.

6.2 Videata applicazione

All'esecuzione del software si apre la finestra "Ricerca" contente i campi di input per il filtro, il bottone per eseguire la ricerca e la tabella di output.

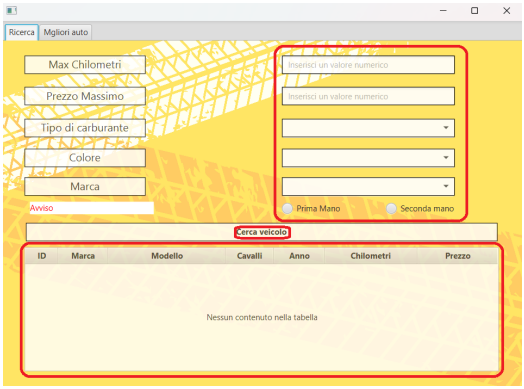

Figura 6.1: Finestra iniziale del software

Mentre la finestra "Migliori auto" presenta una tendina inizialmente vuota, un campo di input, tre radio button, un bottone per far partire la ricorsione e una text area per il risultato finale.

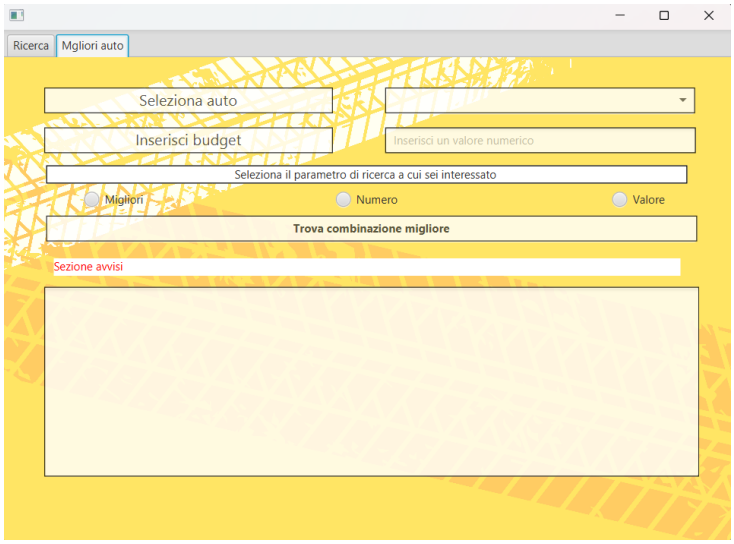

Figura 6.2: Finestra migliori auto simili

Una volta inseriti gli input della finestra "Ricerca" e avviata la ricerca, la tabella e la tendinda della finestra "Migliori auto" vengono popolate.

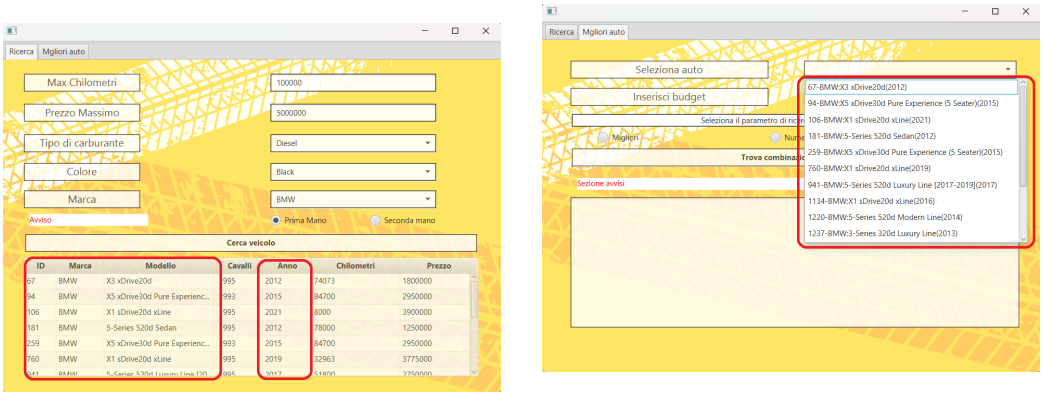

Figura 6.3: Output ricerca e popolamento tendina

Arrivati a questo punto, dovranno essere inseriti gli input della finestra "Miglio auto" e alla pressione del bottone sottostante verrà stampato il risultato della ricorsione.

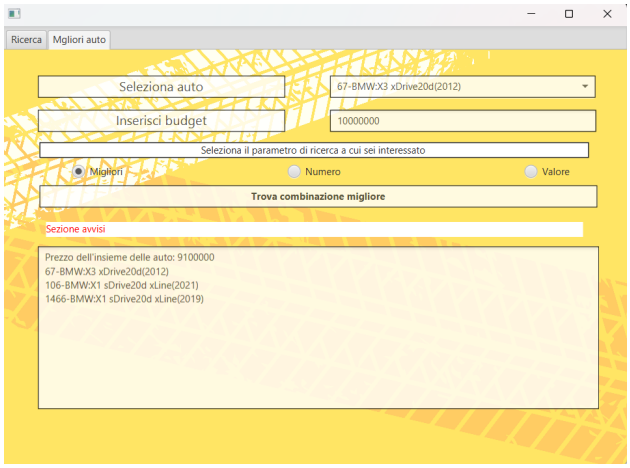

Figura 6.4: Esempio di risultati dell'algoritmo ricorsivo

6.3 Gestione criticità

Se l'utente provasse a compilare i campi in un formato sbagliato, o provasse a inserire come budget un valore minore a quello inserito come prezzo minimo di ricerca viene segnalato attraveso delle label e non viene eseguita la ricerca.

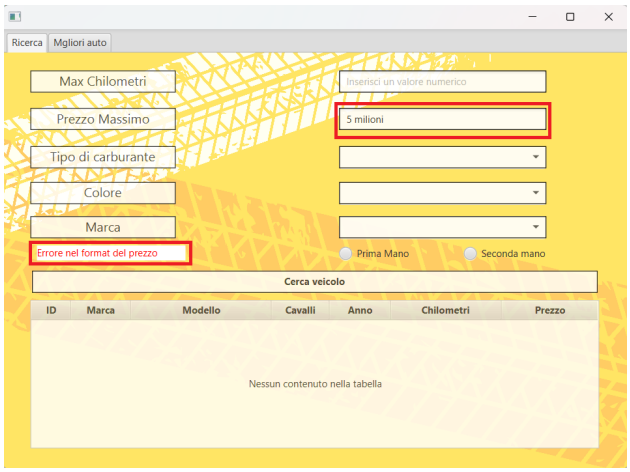

Figura 6.5: Gestione errore formato

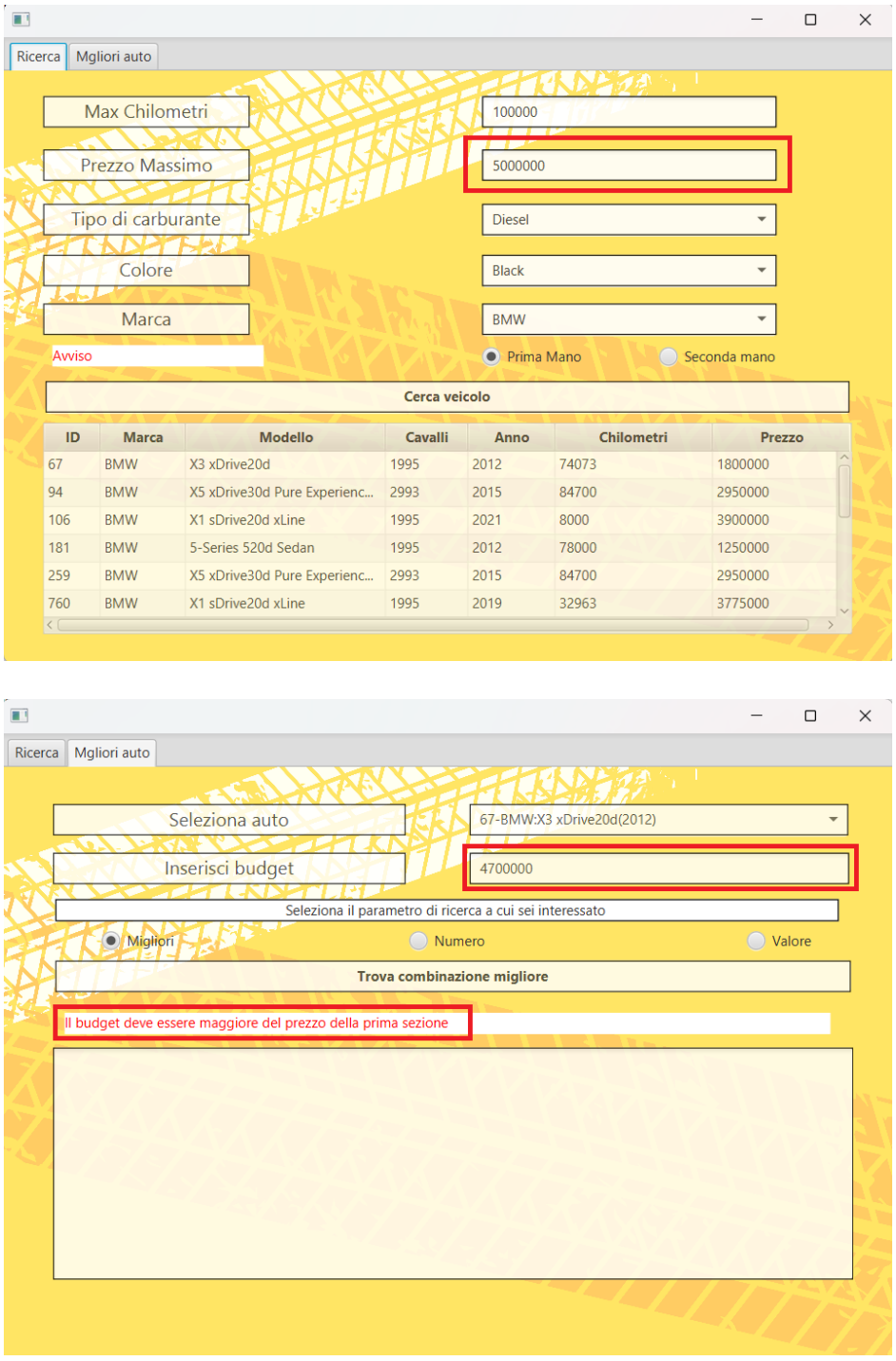

Figura 6.6: Gestione budget inferiore a prezzo di ricerca

Capitolo 7

Valutazioni sui risultati ottenuti

7.1 Risultati dell'algoritmo ricorsivo

Generalmente, a parità di input, le soluzioni offerte dai tre tipi di ricerca sono differenti, come si può osservare negli output seguenti, dal momento che hanno

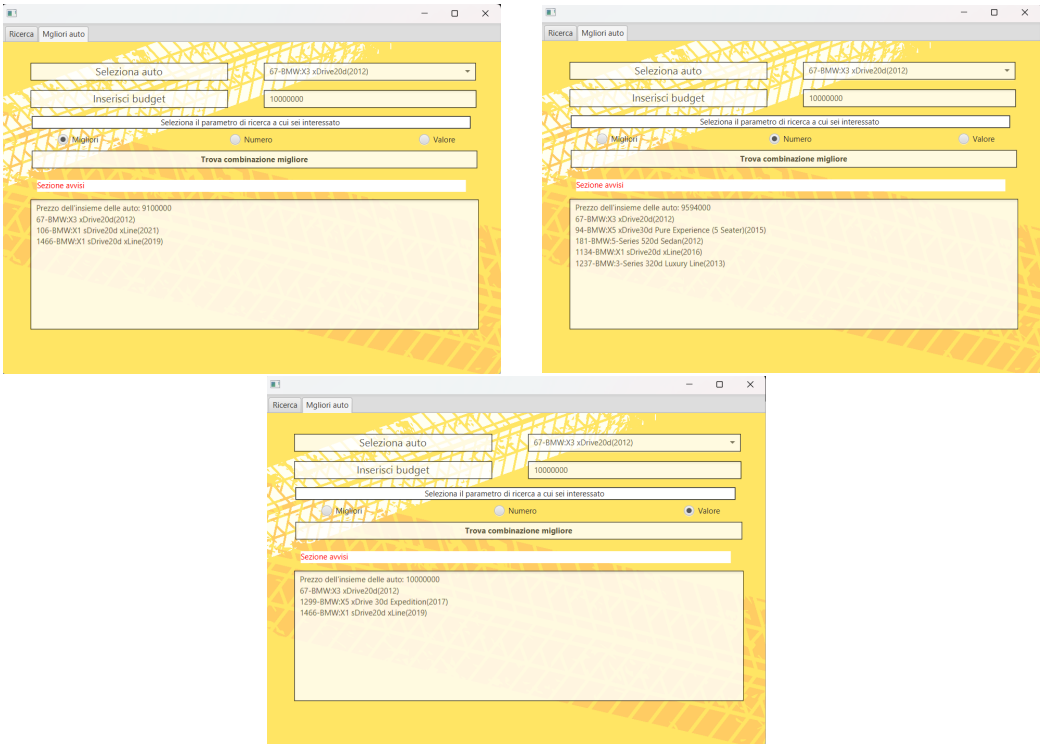

Figura 7.1: Output algoritmi ricorsivi

L'unico caso in cui è certo che i tre algoritmi restituiscono lo stesso risultato è dato dalla ricerca in un dataset di numerosità pari a uno (se non c'è nessun elemento nel dataset il bottone per avviare l'algoritmo ricorsivo è disabilitato).

Un caso che potrebbe capitare frequentemente invece è che il risultato delle ricerche fatte con il parametro "Numero" e "Valore" potrebbero essere equivalenti. I due algoritmi forniscono certamente due soluzioni equivalenti se il vincolo sul budget è maggiore o uguale alla somma del prezzo di tutte le auto presenti nel dataset filtrato (evento probabile quando vengono applicati filtri molto restrittivi).

7.2 Conclusione

Il software è in grado di fornire soluzioni ottime al knpasack problem a funzione obiettivo variabile secondo la scleta dal cliente. Offre una interfaccia semplice e ben strutturata, mostrando tutti i più importanti dettagli del veicolo fornendo quindi un inquadratura del prodotto esaustiva.

I punti critici dell'applicativo sono dati dall'alto costo computazionale quando viene usato un dataset di grandi dimensioni o quando vengono forniti pochi vincoli da applicare.

Il sofware può essere migliorato andando a costruire un sistema che permetta agli utenti di lasciare feedback sul prodotto, i feedback positivi sono importanti per convincere il potenziale cliente all'acquisto del veicolo oppure fornendo un ulteriore servizio al cliente implementando un sistema che tenga conto della tracciabilità dell'ordine in caso di acquisto.

Sitografia

Immagine sfondo software: https://it.freepik.com/vettori-gratuito/sfondo-di-impronta-di-tracce-di-pneumatici-gialli_13124469.htm#query=auto%20background&position=6&from_view=keyword&track=ais&uuid=071a3551-c7b4-4577-8867-df86a09e11df su Freepik.

Licenza