POLITECNICO DI TORINO

DIPARTIMENTO DI INGEGNERIA GESTIONALE E DELLA PRODUZIONE

Corso di Laurea in Ingegneria Gestionale Classe L8 – Ingegneria dell'Informazione

Relazione dell'Esame Finale

Ottimizzazione di un corredino in base al budget e alla stagione di nascita

Relatore
Prof. Fulvio Corno

Candidato
Chiara Lanza
s234834

A.A. 2018/2019

Sommario

1. Proposta di progetto

1.1. Descrizione del problema proposto

Il problema da risolvere è la formazione del corredino perfetto, affrontato come tipico problema dello zaino con pesi (prezzo del singolo prodotto).
Un'applicazione JavaFX mostrerà un set di corredini che non sforino il budget di input e tengano conto della stagione di nascita.
Sarà possibile anche affinare i risultati in base alle preferenze del cliente o del venditore, mostrando il corredino che rispetti la sua idea di perfezione.

1.2. Descrizione della rilevanza gestionale del problema

L'applicazione può essere usata con due modalità: cliente o commerciante.
Commercianti e clienti hanno obiettivi diversi e di conseguenza vanno considerati separatamente.
Un cliente nel momento dell'acquisto del corredino può voler spendere il meno possibile (rimanendo comunque nel range del budget di input), o massimizzare le categorie di prodotti acquistati, anche se con singole quantità di prodotto minori.
Un commerciante invece punta a guadagnare il più possibile (ciò non corrisponde sempre alla massima spesa da parte del cliente) o a vendere più prodotti possibili (può avere necessità di svuotare il magazzino a prescindere dal guadagno maggiore o minore).
Entrambi (commerciante e cliente) possono semplicemente voler vedere tutte le combinazioni possibili.

1.3. Descrizione del data-set utilizzato

I dati sono fittizi e inventati, nonostante ci sia una base di conoscenza preliminare del settore, quindi verosimili. Sono presenti due tabelle:

La prima contenente i prodotti (*Listino*) è strutturata come segue:

- nome dell'articolo;
- categoria (sulla quale lavorerà per il problema dello zaino)
- stagione (dove assente è perché parte essenziale del corredino indipendentemente dalla stagione)
- prezzo al dettaglio;
- prezzo all'ingrosso.

La seconda presenta le *categorie* necessaria per le proporzioni da rispettare:

- categoria;
- minima quantità per questa categoria;
- massima quantità per questa categoria;
- proporzione da rispettare.

1.4. Descrizione preliminare degli algoritmi coinvolti

Attraverso un algoritmo ricorsivo con i dovuti vincoli si risolverà il problema dello zaino che avrà però più soluzioni candidate.
Poiché la perfezione è soggettiva la soluzione dipende dalla scelta dell'utente.
I vincoli più forti saranno di necessità (ad esempio non possono mancare body o tutine in un corredino) e di proporzione.

1.5. Descrizione preliminare delle funzionalità previste per l'applicazione software

In output ci sarà una tabella accompagnata da una casella di testo.
La tabella sarà diversa a seconda se si avrà scelto modalità cliente o commerciane.
In quella cliente avremo le colonne *Prodotto*, *Quantità*, *Prezzo*.
Nella modalità commerciante si aggiunge la colonna *Costo venditore*.
La tabella sarà popolata con i vari corredini trovati dall'algoritmo.

2. Descrizione dettagliata del problema affrontato

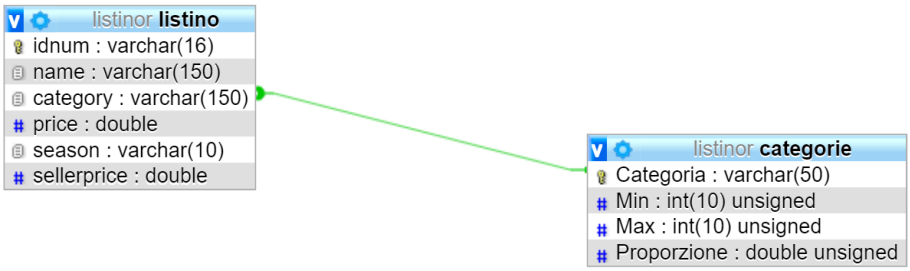
Il momento di acquisto di un corredino è molto delicato, soprattutto se per il primo figlio. Ci si lascia influenzare dai familiari (non esperti del settore) o dagli ospedali, i quali spesso grazie a sovvenzioni o incentivi dati da determinate aziende consigliano acquisti non propriamente indispensabili.
I dati alla base della realizzazione di questa applicazione sono forniti da un negozio di articoli per l'infanzia, senza espliciti riferimenti pubblicitari.
In questo periodo storico in cui bisogna avere la risposta a tutto all'istante e sentiamo la necessità che la tecnologia ci accompagni per ogni nostra azione, questa applicazione ha si presenta come un tool che aiuti i clienti in acquisti sensati, ma che allo stesso tempo può essere un supporto che velocizzi il compito del venditore. L'idea iniziale era quella di utilizzare il data-set completo fornitomi per poter fare un'analisi non solo quantitativa ma anche qualitativa del corredino. Durante la realizzazione però, la dimensione del database ha influito negativamente sulle prestazioni dell'applicazione poiché l'ordine di grandezza delle operazioni svolte non era gestibile. Ho quindi optato per snellire la tabella *Listino* e lavorare solo sulle quantità e sulle proporzioni del corredino.
Passiamo quindi all'applicazione.
Impostati budget e stagione di nascita, l'app calcola tutte le possibili combinazioni:

- tenendo conto di un margine sul budget (non è possibile spendere esattamente quanto prestabilito) senza mai sforarlo
- rispettando i vincoli presenti nella tabella *Categorie.*

Il problema dello zaino si basa sulle specifiche della tabella *Categorie*: dovrà rispettare le proporzioni indicate (anche qui con una tolleranza preimpostata che può essere modificata nel controller), rimanendo sempre nel range tra massimo e minimo quantitativo (dati della tabella).
La mole di lavoro si concentra tutta in questa parte, poiché le combinazioni 'elette' che possono essere sezionate dall'utente saranno già selezionate nella ricorsione generale.

3. Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati

3.1 Database



3.2. Algoritmi

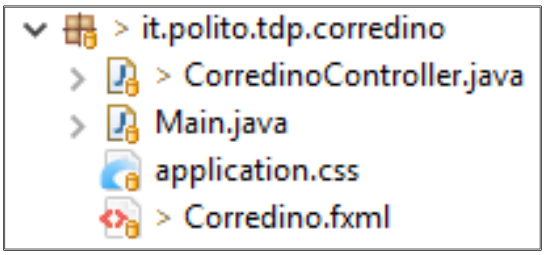
L'applicazione che utilizza il linguaggio Java, sfrutta il pattern MVC (Model-View-Controller), il quale attraverso i principi della programmazione a oggetti, consente un'architettura software separata in 3 ruoli principali:

- il model che fornisce i metodi per accedere ai dati utili all'applicazione;
- il view visualizza i dati e si occupa dell'interazione con l'utente;
- il controller riceve i comandi dell'utente attraverso il view e li esegue utilizzando il model.

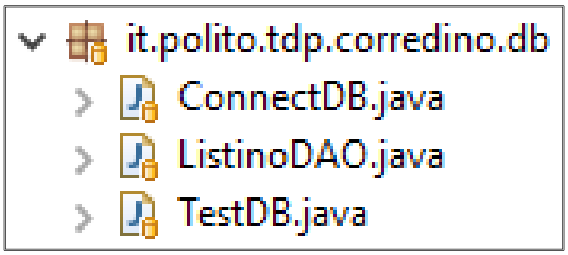
A questo è associato un secondo pattern, il DAO (Data Access Object), che permette di accedere ai dati sul database attraverso query all'interno di metodi e renderli fruibili per l'applicazione.

Il progetto è suddiviso in tre packages:

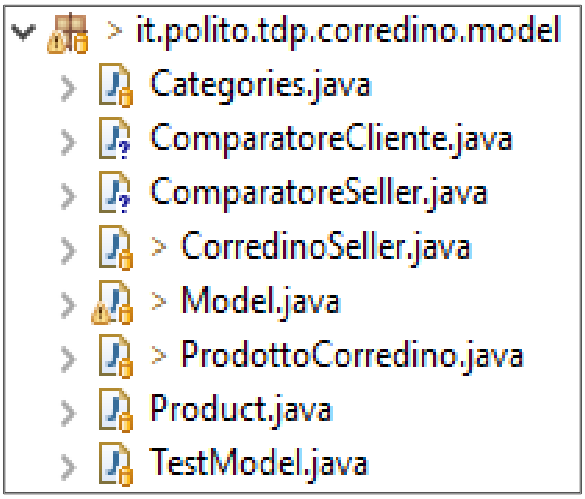
- *corredino* (generale): contenente la classe *Main* da lanciare per l'avvio dell'applicazione, l'unico controller *CorredinoController* che definisce i metodi necessari per l'interazione dell'utente, *Corredino.fxml* per la definizione dell'interfaccia grafica e il file .css;



- *db*: contenente la classe *ConnectDB* utilizzata per la connessione al database, la classe *ListinoDAO* che, tramite query SQL, permette il caricamento dei dati e la classe *TestDB* di testing per la connessione.



- *Model*: contenente la classe *Model*, che possiede tutta la logica applicativa del software, la classe *ProdottoCorredino*, che definisce il singolo prodotto da inserire nella classe *CorrdinoSeller*, della quale sono presenti due comparatori *ComparatoreSeller* e *ComparatoreCliente* che ordinano i corredini in base alla modalità selezionata. Le classi *Product* e *Categories* riprendono i dati presenti nel DataBase.



Il metodo *calcola* della classe *Model* inizializza tutte le variabili e dà inizio alla ricorsione, ha come input il budget e la stagione selezionati dall'utente nell'interfaccia.

```
public void calcola(double budget, String season){
    categories=dao.getAllCat(season);
    combinazioni=new ArrayList<>();
    best=new ArrayList<>();
    totBest=Double.MAX_VALUE;
    quantitaCat=new ArrayList<>();
    double tot=0;
    parziale=new ArrayList<>();
    allP=new ArrayList<>();
    for(Categories c: categories) {
        //aggiungo in ordine di categoria tutti i prodotti, stesso ordine avranno le liste di categorie
        allP.addAll(dao.getAllProductCat(season, c));
        //inizializzo a 0 le quantita per ogni categoria
        this.quantitaCat.add(0);
    }

    for(int i=0;i<allP.size();i++) {
        parziale.add(0);
    }
    this.prova(0, 0, tot, budget);
}
```

Il metodo *prova* è la vera e propria ricorsione.

Ha come input l'indice nella *List<Product >allP* del prodotto analizzato nel livello corrente della ricorsione, l'indice della categoria nella *List<Categories> categories* del prodotto analizzato nel livello precedente (necessario per controllare il vincolo di proporzione), il totale attuale del corredino e il budget dell'utente.

Essendo molto lungo meglio analizzarlo in più parti.

Nella prima parte se sono stati analizzati tutti i prodotti presenti nel listino, controlla che il corredino sia completo e rispetti le proporzioni con il metodo *isCompleta(parziale)*. Se rispetta i vincoli viene aggiunta alla lista di soluzioni, e se ha un totale minore di quello salvato viene aggiornato il corredino *best*.

```
public void prova(int indexP, int indexC, double tot, double budget) {

    double margine= this.marg*budget;
    if(indexP>parziale.size()-1) {
        if(budget-tot<margine && this.isCompleta(parziale)) {
            List<Integer> add= new ArrayList<Integer>(parziale);
            combinazioni.add(add);
            if(tot<totBest) {
                totBest=tot;
                best=new ArrayList<>(parziale);
            }
        }
        return;
    }
```

Se la categoria del prodotto corrente è diversa da quella del successivo, poiché dal database sono estratti in ordine di categoria, controlla che siano rispettate le proporzioni dettate nella tabella categorie del dataset con il metodo *controllaProp()* al quale passo il corredino attuale e l'indice della categoria attuale.

```
//inizializzo indexCAtt perche se non lo setto esco con il return
int indexCAtt=-1;
//cerco quale categoria di prodotto sto testando, per inviarla come dato nella ricorsione
//(serve per il testing intermedio delle proporzioni)
//NB: indexC e del prodotto precedente e serve per il test
if(!(indexP>parziale.size()-1)) {
    indexCAtt= categories.indexOf(allP.get(indexP).getCategory());
}

//controllo ce liindice della categoria sia diverso per poter fare un controllo delle categorie
//fin ora complete e bloccare in anticipo eventuali soluzioni inadatte
//indexCAtt deve essere >=1 perche per poter fare il confronto servono almeno due categorie(0,1)
//NB:sono in ordine anche se e' inutile passare la lista parziale perhè tanto è stata dichiarata
//globalmente, indexCAtt è necessario
if(indexCAtt!=indexC && indexCAtt>=1) {
    if(!this.controllaProp(parziale, indexCAtt))
        return;
}
```

Calcola la quantità massima di prodotti che possono essere aggiunti in base alla categoria e ai vincoli sul budget.

```
//capisco quanti prodotti posso aggiungere in base al massimo della categoria,
//ma tengo conto anche di quelli gia inseriti per quella categoria
int max=categories.get(indexCAtt).getMax()-this.quantitaCat.get(indexCAtt);

//quanti ne posso aggiugere ancora senza sforare il budget?
int max2= (int) ((int)(budget-tot)/allP.get(indexP).getPrice());

if(max2<max)
    max=max2;

//controllo se posso aggiungere ancora o è inutile
if(budget-tot<min)
    return;
```

La vera e propria ricorsione si risolve in poche righe di codice

```
    //vera e propria ricorsione
    for(int q=0;q<max;q++) {

        //aggiungo la quantita di prodotti alla lista di quantita dei prodotti
        this.parziale.set(indexP, q);
        //aggiorno tot
        tot += allP.get(indexP).getPrice()*q;
        //aggiorno la quantita della categoria in corso
        int att= this.quantitaCat.get(indexCAtt);
        this.quantitaCat.set(indexCAtt, att+q);
        if(indexP==35)
            System.out.println("Uffa");
        if(indexP<allP.size())
        this.prova(indexP+1, indexCAtt, tot, budget);

        //ritorno alle condizioni precedenti

        this.quantitaCat.set(indexCAtt, att);
        tot -= allP.get(indexP).getPrice()*q;
    }
```

I metodi *isCompleta()* e *controllaProp()* sono molto simili con l'accorgimento che il primo deve valere per ogni categoria. Riporto quindi solo *controllaProp()* poiché più breve e perciò più immediato.

```
private boolean controllaProp(List<Integer> parziale, int indexCAtt) {
    //il controllo lo faccio tra la categoria su indexCatt e la precedente,
    //se il suo minimo è diverso da 0, altrimenti con una delle precedenti

    for(int i =0;i<indexCAtt;i++) {
        for(int j=0;j<indexCAtt;j++) {
            if(i!=j) {
            //per facilitare debug ho salvato in delle variabli
            double qi_Min=categories.get(i).getMin();
            double qj_Min=categories.get(j).getMin();
            double qi_Att=quantitaCat.get(i);
            double qj_Att=quantitaCat.get(j);
            double prop_i=categories.get(i).getProporzione();
            double prop_j=categories.get(j).getProporzione();
            if(qi_Min!=0 && qj_Min!=0)
                if(qi_Att>0 && qj_Att>0)
                    if(!(((double)(qi_Att/qj_Att))<(1+this.tolleranza)*(prop_i/prop_j) &&
                            ((double)(qi_Att/qj_Att))>(1-this.tolleranza)*(prop_i/prop_j)))
                        return false;
            if((qi_Min==0 || categories.get(i).getMax()==1)  && qj_Min!=0 && qj_Att!=0)
                if(qi_Att!=0 && !(((double)(qi_Att/qj_Att))<(1+this.tolleranza)*(prop_i/prop_j) &&
                        ((double)(qi_Att/qj_Att))>(1-this.tolleranza)*(prop_i/prop_j)))
                    return false;
            if((qj_Min==0 || categories.get(j).getMax()==1)  && qi_Min!=0 && qi_Att!=0)
                if(qj_Att!=0 && !(((double)(qi_Att/qj_Att))<(1+this.tolleranza)*(prop_i/prop_j) &&
                        ((double)(qi_Att/qj_Att))>(1-this.tolleranza)*(prop_i/prop_j)))
                    return false;
        }}
    }
    return true;
}
```

Il metodo getAll() riporta le combinazioni di quantità trovate con la ricorsione in veri corredini con prodotti, e salva anche le liste elette che possono essere richieste nel controller

```
public void getAll() {

    CorredinoSeller b = new CorredinoSeller();
    maxCat=0;
    double tot=0;
    int att=0;
    double income=0;
    ris= new LinkedList<>();
    int indexCatVecchia=0;
    for(List<Integer> li : combinazioni) {
        //risfrutto la variabile che utilizzo in modalità seller anche se non hanno lo stesso scopo
        item=0;
        tot=0;
        att=0;
        income=0;
        b = new CorredinoSeller();
        for(int i=0;i <li.size();i++) {
            if(li.get(i)!=0) {
                ProdottoCorredino temp= new ProdottoCorredino(allP.get(i).getName(),li.get(i),
                        (double)allP.get(i).getPrice(),(double)allP.get(i).getSellerPrice());
                b.addP(temp);
                item +=li.get(i);
                income += (double)(allP.get(i).getPrice()-allP.get(i).getSellerPrice())*li.get(i);
                tot += temp.getQuantita()*temp.getCosto();
                if(i!=0)
                //contatore di categorie nel corredino
                if(!allP.get(i).getCategory().equals(allP.get(indexCatVecchia).getCategory())) {
                    att++;
                    indexCatVecchia=i;}
            }
        }
```

```
        b.setTot((Math.round( tot * Math.pow( 10, 2 ) )/Math.pow( 10, 2 )));
        b.setTotProdotti(item);
        b.setIncomeTot((Math.round( income * Math.pow( 10, 2 ) )/Math.pow( 10, 2 )));
        ris.add(b);
        if (att>maxCat) {
            maxCat=att;
            maxItC = new CorredinoSeller(b.getP(),(double)b.getTot());}

        if(item>maxItem) {
            maxItem=item;
            maxIt= new CorredinoSeller(b.getP(),(double)b.getTot(),b.getIncomeTot());
        }
    }
}
```

I restanti metodi del Model restituiscono solo le variabili richieste.

Altre classi interessanti sono i due comparatori della classe *CorredinoSeller*:

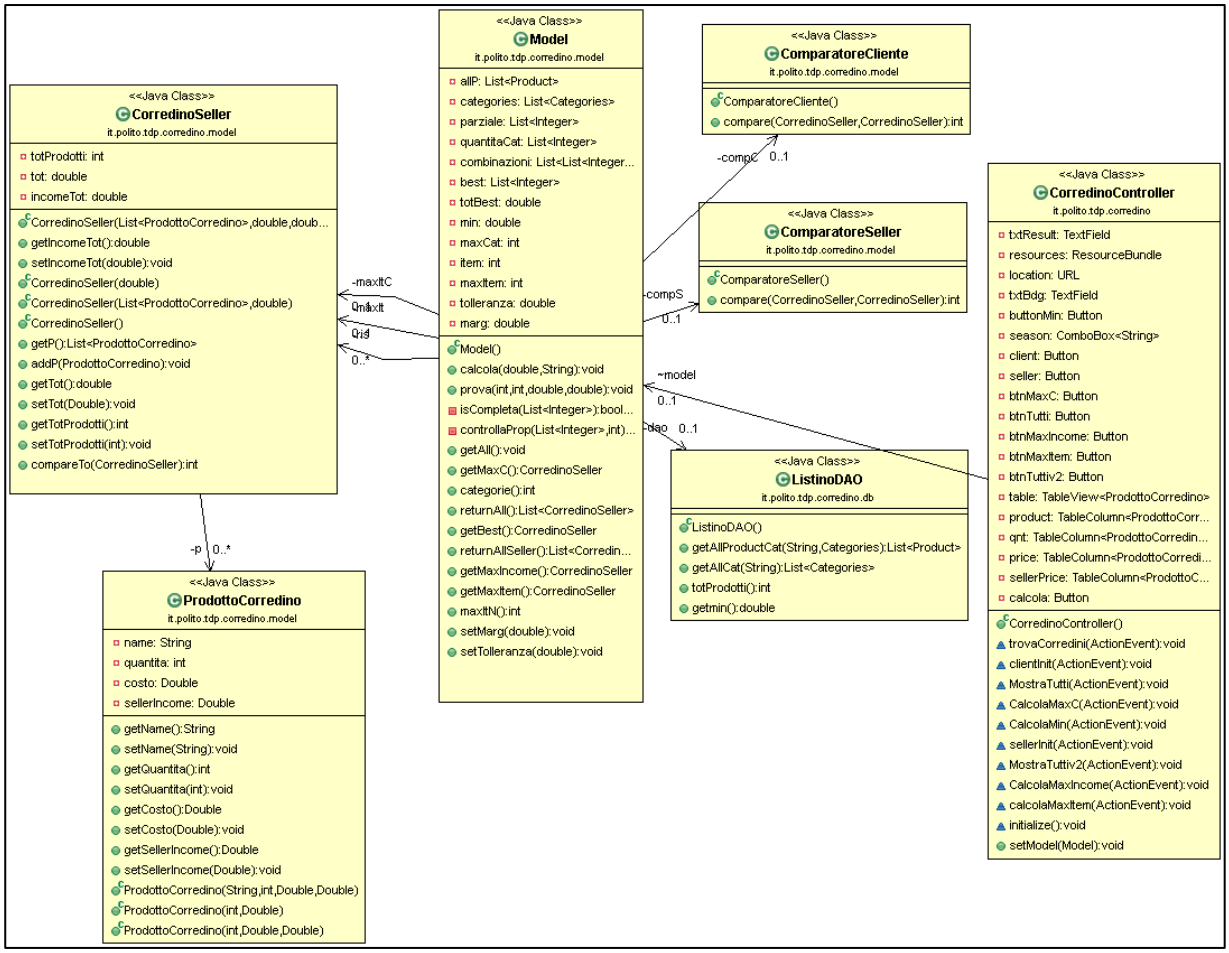
uno per ordinare la lista con costo crescente

```
package it.polito.tdp.corredino.model;

import java.util.Comparator;

public class ComparatoreCliente implements Comparator<CorredinoSeller>{

    public int compare(CorredinoSeller cs1, CorredinoSeller cs2) {

            return (int) (cs1.getTot()*100-cs2.getTot()*100);
        }

}
```

e l'altro per ordinarla con ricavo decrescente (output per il commerciante).

```
package it.polito.tdp.corredino.model;

import java.util.Comparator;

public class ComparatoreCliente implements Comparator<CorredinoSeller>{

    public int compare(CorredinoSeller cs1, CorredinoSeller cs2) {

            return (int) (cs1.getTot()*100-cs2.getTot()*100);
        }

}
```

4. Diagramma delle classi delle parti principali dell'applicazione



5. Esempi di risultati con videate

Schermata di avvio:

Alla pressione del pulsante calcola, una tabella appare nella parte interiore dell'applicazione.

L'applicazione è in grado di capire se manca un parametro, o non è realistico (per esempio un budget inferiore a 100) e non esegue i calcoli, ma stampa nella casella di testo il problema riscontrato.

Alla pressione del pulsante *Sono un commerciante* i pulsanti in basso diventano cliccabili, viceversa quelli di sopra cliccando *Sono un cliente*.

Un esempio con entrambe le possibilità, dopo avere cliccato mostra tutti.

La casella di testo presente tra la tabella e i pulsanti mostra informazioni utili riguardanti la scelta effettuata. Ad esempio con *mostra tutti* indica quante combinazioni sono presenti.

7. Link del video YouTube

https://youtu.be/PrjANCcYfCM

8. Valutazione risultati ottenuti e conclusioni

L'applicazione ha come scopo principale risolvere il problema dello zaino che si presenta ad una famiglia in attesa di un bambino.
La stagione di nascita e il budget sono i soli vincoli da inserire. Il sesso del nascituro, per la tipologia di prodotti utilizzati nell'analisi, è superfluo. Con le funzionalità implementate il software si rende però utile anche al commerciante, che utilizzando il proprio listino prezzi è in grado di capire quale combinazione conviene proporre per avere un maggiore ricavo.
Analizzando i risultati evince che anche mantenendo una tolleranza delle proporzioni sulle categorie presenti sul database molto bassa le combinazioni trovate con budget superiori ai 100 euro sono diverse.
Per questo le funzioni che permettono di selezionare un solo corredino in base alle preferenze si sono dimostrate indispensabili.
Utilizzando un dataset dalle dimensioni rilevanti purtroppo non è possibile arrivare ad un risultato in tempi ragionevoli. Per questo motivo nel repository della prova finale è presente anche un listino ridotto, che ci permette di avere i corredini in pochi secondi con budget discreti, il tempo ovviamente aumenta all'aumentare del budget inserito.
L'algoritmo ricorsivo di per sé non è complesso da implementare, il problema principale è saper bloccare il più possibile le soluzioni già in partenza non coerenti con lo scopo.
Per alleggerire il lavoro alla macchina la ricorsione non è svolta sugli oggetti Product ma su una lista ordinata di interi che corrispondono direttamente con i prodotti per indice comune. Stesso accorgimento è stato preso per le categorie ed i controlli su di esse. Queste ultime in più sono salvate in ordine di prezzo medio decrescente. Non è stato possibile ridurre ulteriormente i tempi di esecuzione poiché un algoritmo ricorsivo che per ogni prodotto presente sul database permuta max-min volte (con max e min che sono i dati presenti nella tabella categorie del db) con i restanti prodotti ha bisogno dei sui tempi di esecuzione. Ad esempio, già con un budget di 500 euro e utilizzano il listino ridotto l'applicazione ha trovato 1820 combinazioni che rispettano i vincoli, senza contare tutte quelle scartate durante i calcoli.
Sono presenti vincoli sulle categorie e sulle quantità da inserire per poter ridurre al minimo le operazioni svolte.

Nella realizzazione dell'applicazione è emerso che con i dati presenti non è possibile ottenere un corredino perfetto con meno di 100 euro, soglia poi inserita nell'interfaccia per evitare di far svolgere operazioni inutili al software.
L'applicazione è facilmente adattabile per ogni negozio che si occupa di corredini semplicemente utilizzando il proprio listino. Ma non solo, poiché il codice è astratto e non fa riferimento a prodotti specifici, con un dataset ad hoc è possibile impiegarla per qualsiasi problema dello zaino con proporzioni da rispettare. Anche le sezioni specifiche dei corredini (massimizzare il numero di prodotti, o di categoria, massimizzare i ricavi o minimizzare il costo) sono funzioni utili in qualsiasi ambito.
Con piccoli accorgimenti le funzionalità di tale applicazione potrebbero essere numerose e personalizzabili.