

POLITECNICO DI TORINO

Dipartimento di Ingegneria Gestionale e della Produzione

Corso di Laurea in Ingegneria Gestionale
Classe L-8

Tesi di Laurea

Applicazione per gestione itinerario e tempo in una giornata di sci



Relatore

Prof. Corno Fulvio

Candidato

Delleani Mattia

AA. 2019/2020

1	Proposta di progetto	2
1.1	Titolo della proposta.....	2
1.2	Descrizione del problema proposto.....	2
1.3	Descrizione della rilevanza gestionale del problema	3
1.4	Descrizione dei data-set per la valutazione	3
1.5	Descrizione preliminare degli algoritmi coinvolti	3
1.6	Descrizione preliminare delle funzionalità previste per l'applicazione software	5
2	Introduzione.....	6
3	Descrizione del problema affrontato.....	7
4	Dataset	8
4.1	Struttura database.....	8
4.2	Tabella <i>impianti</i>	8
4.3	Tabella <i>piste</i>	9
4.4	Tabella <i>stazioni</i>	9
5	Strutture dati e algoritmi.....	10
5.1	Descrizione delle strutture dati	10
5.2	Algoritmi coinvolti	11
5.2.1	Cammino minimo	13
5.2.2	Algoritmo ricorsivo	16
6	Videate dell'applicazione e dimostrazione	20
7	Conclusione e valutazione dei risultati	23

1 Proposta di progetto

1.1 Titolo della proposta

Tool per gestione itinerario e tempo in una giornata di sci

1.2 Descrizione del problema proposto

L'applicazione mediante alcuni parametri di input, quali numero di partecipanti, il rispettivo livello di esperienza, una località di partenza e di arrivo, durata dell'attività, permetterà di calcolare il cammino minimo ed un itinerario sciistico il più variegato possibile al fine di massimizzare il divertimento dei partecipanti, rispettando i vincoli da essi imposti e gestendone al meglio il tempo.

1.3 Descrizione della rilevanza gestionale del problema

La rilevanza del problema in ambito gestionale è data dal fatto che l'applicazione tratta la scelta di un determinato itinerario sciistico cercando di rendere una giornata di sci il meno monotona possibile, scegliendo piste diverse da quelle già effettuate, e gestendone il tempo in funzione degli orari introdotti in input, in modo da evitare ritardi o eccessivi anticipi nel termine dell'attività, che in grandi comprensori sciistici è un problema molto ricorrente e che comporta situazioni poco piacevoli ad esempio la chiusura di impianti di collegamento, oppure per i maestri sci il ritardo nel termine dell'attività comporta la perdita del cliente nell'ora successiva e il troppo anticipo comporta uno scontento al cliente. Inoltre, mediante la possibilità di calcolare il cammino più veloce ottimizza lo spostamento dei professionisti all'interno del comprensorio.

1.4 Descrizione dei data-set per la valutazione

Non riuscendo a trovare data-set relativi a questo tipo di tematica, in alternativa sono disposto a realizzarli basandomi su dati reali (sia cartacei sia reperibili online). Nel caso avrei deciso di realizzarlo sul comprensorio sciistico ViaLattea (<https://www.vialattea.it/Ski-Area/Ski-Map>) in quanto è composto da più stazioni di sci con collegamenti tra le diverse località. Il dataset sarebbe composto da diverse tabelle, quali:

- La tabella *piste* avente le informazioni relative a tutte le piste del comprensorio tra cui nome, località, stazione di monte, stazione di valle, difficoltà della pista.
- La tabella *impianti* avente le informazioni di tutti gli impianti a fune del comprensorio tra cui stazione di monte, stazione di valle, località, tipologia, tempo di risalita, posti.
- La tabella *stazioni* avente le informazioni relative alle singole stazioni, sia di valle sia di monte, di tutti gli impianti del comprensorio, tra cui l'identificativo assegnato e il nome della stazione stessa.

Sperando di riuscire a reperire sufficiente materiale saranno introdotte altre informazioni al fine di arricchire il data-set e migliorare le funzionalità dell'applicazione.

1.5 Descrizione preliminare degli algoritmi coinvolti

Il problema verrà affrontato mediante la teoria dei grafi, in cui i vertici sono le stazioni, gli archi sono orientati e pesati, dove il peso corrisponde al tempo. Pensavo di utilizzare un multigrafo orientato e pesato, in quanto supponiamo una stazione di valle è collegata ad un'altra stazione di monte mediante un impianto per risalire, quindi con un determinato verso ed impiega un determinato tempo, dato del tempo di risalita dell'impianto. Dalla stazione di monte le piste collegano a loro volta le stazioni a valle rispetto a quella da cui partono e una stazione di monte può essere collegata alla stessa stazione di valle mediante piste diverse, per questo opterei per un multigrafo orientato e pesato. Il peso (tempo) di un arco costituito da una pista viene calcolato come lunghezza della pista moltiplicata per la velocità media di percorrenza, dove la velocità media è un parametro fisso che varia a

seconda del livello dell'utente.

Come operazioni sul grafo avrei intenzione di effettuare:

Cammino minimo tra due vertici, scelti in input dall'utente, per calcolare il percorso più veloce in termini di tempo che li collega, se questo fosse possibile in funzione del livello del gruppo di partecipanti (il colore delle piste può essere blu, rosso, nero e rappresentano il minimo livello per percorrerle rispettivamente principiante, intermedio ed esperto).

Cammino massimo tra due vertici inseriti in input mediante un algoritmo ricorsivo, che permetta di raggiungere la destinazione cercando di utilizzare il maggior numero di piste diverse, al fine di rendere il meno monotono possibile il percorso, nel rispetto dei vincoli di tempo selezionati di input. Sarà quindi definita una funzione obbiettivo che calcolerà un punteggio in funzione del numero di piste utilizzate, della loro ripetizione e del tempo di impiego dell'attività, cercando dunque di evitare di terminare l'attività troppo presto. Dunque, nella realizzazione si avrà una distinzione degli archi del grafo in due tipologie: *pista* o *impianto*. Il punteggio di una singola pista verrebbe calcolato rispetto alla sua difficoltà e a quella dello sciatore, ad esempio per uno sciatore esperto fare piste troppo facili porterebbe ad annoiarsi quindi si potrebbero avere questi punteggi: pista NERA= 3pt, pista ROSSA = 2pt, pista BLU=1pt, mentre per uno sciatore di livello intermedio si potrebbe avere: pista ROSSA=3pt, pista BLU= 2pt; in quanto le piste ROSSE per uno sciatore di livello intermedio risultano difficili.

Al fine di rendere più entusiasmante e meno monotono l'itinerario, pensavo che nel calcolo del punteggio totale venga considerato come fattore negativo lo svolgimento multiplo della stessa pista (monotonia dell'itinerario), che comporta ad una diminuzione del 50% del punteggio unitario della pista per ogni discesa della stessa (es. pista NERA per sciatore esperto: prima discesa ->3pt, seconda discesa ->1,5pt, terza discesa -> 0,75pt).

Nel calcolo del punteggio si potrebbe introdurre una componente che tenga conto della lunghezza della pista mediante l'uso di un parametro K che dipende sia dalla difficoltà della pista sia dal livello dello sciatore e che ha come unità di misura [pt/km]. Ad esempio, supponiamo un valore di $K=0,6[pt/km]$ per una pista ROSSA eseguita da uno sciatore ESPERTO, mentre per uno sciatore INTERMEDIO si avrebbe $K=0,9[pt/km]$ per una pista rossa. Un esempio della scala da utilizzare potrebbe essere:

- SCIATORE ESPERTO: NERA->0,8[pt/km], ROSSA->0,5[pt/km], BLU->0,2[pt/km];
- SCIATORE INTERMEDIO: ROSSA->0,8[pt/km], BLU->0,4[pt/km];
- SCIATORE PRINCIPIANTE: 0,8[pt/km] pista blu;

Infine, viene poi sommato al punteggio una componente legata al tempo impiegato nell'attività che verrà calcolato come $\mu * tempoAttività$, dove μ sarà una costante.

Riassumendo per un dato livello dello sciatore il punteggio di una data pista viene calcolato come somma del suo valore unitario in funzione del numero di ripetizioni e della lunghezza moltiplicata per il fattore di scala K.

1.6 Descrizione preliminare delle funzionalità previste per l'applicazione software

L'applicazione permette all'utente di selezionare un numero di sciatori e il rispettivo livello (principiante, intermedio, esperto) una stazione di partenza e di arrivo di una data località che possono anche differire tra loro. Come località è intesa il comune di partenza e parametro *iniziale* per la tabella *impianti* viene inteso che esso sia raggiungibile a piedi dalla strada.

Saranno realizzate due interfacce una relativa al cammino minimo e uno a quello massimo. Entrambe saranno dotate di apposito bottone per il calcolo del percorso in funzione del numero di utenti introdotto da una TextBox. Livello di esperienza degli utenti, durata attività, località, impianto di partenza e arrivo saranno selezionati mediante un menù a scelta multipla.

2 Introduzione

L'idea di realizzare un'applicazione per il calcolo degli itinerari in un comprensorio sciistico come tesi di laurea nasce dalla mia passione per lo sci alpino, o più in generale per la montagna, che legate ad un forte interesse per la programmazione hanno scaturito in me i presupposti per formulare questo progetto. Sin da bambino ho avuto l'opportunità di praticare come sport lo sci alpino ed in seguito ad un percorso agonistico di diventare maestro di sci, un traguardo importante a farmi maturare come persona e professionista. Proprio questo mestiere mi ha fatto vedere lo sport che ho sempre praticato sin da bambino in un'ottica completamente diversa, presentandomi problemi a cui non avevo minimamente pensato precedentemente e che al giorno d'oggi sono ancora presenti.

Uno tra i tanti problemi, e piuttosto ricorrente quando si pratica lo sci alpino, è la gestione del tempo, che se non opportunamente controllato potrebbe portare a spiacevoli inconvenienti tra cui la chiusura degli impianti oppure per i professionisti, nel caso di ore di lezione, il troppo anticipo nel termine dell'attività comporta uno scontento al cliente e viceversa finire in ritardo l'attività comporta la perdita del cliente successivo.

Un altro problema che si presenta spesso è invece legato al divertimento dell'attività sciistica, dovuto alla monotonia nella scelta delle piste causata dalla paura di scegliere piste non ancora effettuate che potrebbero essere non adeguate al livello dell'utente oppure troppo dispendiose in termini di tempo.

Dunque, pensando alle diverse problematiche relative a questa tematica e grazie alle competenze di programmazione informatica acquisite principalmente nei corsi *Programmazione ad oggetti* e *Tecniche di Programmazione* ho potuto realizzare e rendere concreta quest'idea che spero possa tornare utile a chiunque decida di usufruirne.

3 Descrizione del problema affrontato

L'applicazione realizzata permette di calcolare itinerari sciistici e gestirne il tempo all'interno del comprensorio sciistico della Via Lattea in funzione dei vincoli imposti dall'utente, un problema piuttosto ricorrente quando si pratica l'attività di sci alpino in comprensori di grandi dimensioni o poco conosciuti da chi si cimenta.

Un comprensorio sciistico nel suo complesso può essere paragonato ad una vera e propria azienda che opera nel settore del turismo, in cui lavorano numerosi dipendenti e liberi professionisti, che ogni anno contribuiscono all'economia in maniera consistente, basti pensare a regioni come Valle d'Aosta e Trentino-Alto-Adige.

Fornire strumenti utili per ridurre eventuali problematiche e massimizzare l'esperienza e il divertimento dei turisti è un requisito fondamentale per alimentarne il flusso annuale e, quindi, far crescere l'economia locale.

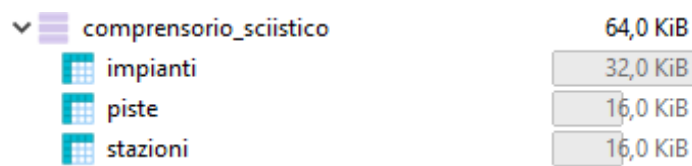
Un'applicazione che calcoli itinerari sciistici permette di risolvere alcuni dei problemi che si possono presentare ai clienti, che magari essendo la prima volta che visitano il comprensorio potrebbero sentirsi spaesati ed avere difficoltà nella scelta delle piste e negli impianti di risalita da prendere, che se erroneamente selezionati potrebbero portare a situazioni poco piacevoli, ad esempio l'utente si potrebbe trovare di fronte ad una pista di livello superiore al proprio oppure potrebbe comportare un cattiva gestione del tempo con conseguente ritardo nel fine dell'attività, entrambe situazioni che portano ad uno scontento del cliente che, spesso, associa come causa la disinformazione fornita dal comprensorio. Questo strumento per il calcolo dei percorsi potrebbe sostituire le vecchie mappe cartacee, spesso disordinate e confusionarie, aggiungendo a queste la possibilità di non dover calcolare manualmente il percorso, che porterebbe via tempo all'utente, ma di calcolarlo velocemente mediante pochi passaggi.

4 Dataset

Per funzionare in maniera corretta, l'applicazione necessita di un database con una struttura specifica, infatti è stato necessario creare opportunamente il database al fine di realizzare l'applicazione ed ottenere risultati corretti.

4.1 Struttura database

Il database che prende il nome di *comprensorio_sciistico* è costituito da tre tabelle: *piste*, *impianti* e *stazioni*.



comprensorio_sciistico	64,0 KiB
impianti	32,0 KiB
piste	16,0 KiB
stazioni	16,0 KiB

4.2 Tabella *impianti*

La tabella *impianti* avente le caratteristiche di tutti gli impianti a fune di risalita presenti nel comprensorio. È costituita dai seguenti attributi:

Nome	Tipo di dati
id	INT
nome	VARCHAR
stazione_monte	VARCHAR
stazione_valle	VARCHAR
localita	VARCHAR
tipo	VARCHAR
iniziale	INT
posti	TINYINT
tempo_risalita	DOUBLE
intervallo	DOUBLE
h_apertura	TIME
h_chiusura	TIME

- id identificativo univoco dell'impianto (chiave primaria)
- nome nome dell'impianto di risalita
- stazione_valle identificativo della stazione di valle
- stazione_monte identificativo della stazione di monte
- localita località dell'impianto di risalita
- tipo tipologia dell'impianto (seggiovia, cabinovia, ...)
- iniziale se raggiungibile dalla strada
- posti numero di posti per singolo mezzo di risalita
- tempo_risalita tempo di risalita espresso in secondi
- intervallo intervallo di tempo in minuti tra i singoli mezzi di risalita
- h_apertura orario apertura impianto
- h_chiusura orario chiusura impianto

4.3 Tabella *piste*

La tabella *piste* che rappresenta le piste di sci presenti nel comprensorio ed è costituita dai seguenti attributi:

Nome	Tipo di dati
id	INT
nome	VARCHAR
stazione_monte	VARCHAR
stazione_valle	VARCHAR
colore	VARCHAR
lunghezza	SMALLINT
localita	VARCHAR

- id identificativo univoco della pista (chiave primaria)
- nome nome della pista
- stazione_monte partenza a monte della pista
- stazione_valle arrivo a valle della pista
- colore difficoltà della pista {BLU, ROSSA, NERA}
- lunghezza lunghezza della pista espressa in metri
- localita località della pista

4.4 Tabella *stazioni*

La tabella *stazioni* che rappresenta le stazioni degli impianti a fune. La tabella è costituita dai seguenti attributi:

Nome	Tipo di dati
id	VARCHAR
quota	INT

- id identificativo univoco della stazione (chiave primaria)
- quota altitudine espressa in metri sul livello del mare

5 Strutture dati e algoritmi

5.1 Descrizione delle strutture dati

L'applicazione è stata realizzata utilizzando il linguaggio di programmazione Java per la gestione della logica applicativa e JavaFX per gestire la componente grafica dell'applicazione. Il tutto è stato realizzato mediante il pattern MVC (Model-View-Controller) che permette di separare all'interno del software la logica applicativa, l'interfaccia grafica e la struttura dati. Infatti, l'applicazione è suddivisa in tre package distinti:

- Il package *compensorio_sciistico* che si comporta da controller e contiene tutte le classi che si occupano di realizzare e interagire con l'interfaccia grafica con la quale l'utente inserire i parametri di input e visualizza i risultati ottenuti.

```
▼ [Icona] > it.polito.tdp.compensorio_sciistico
  > [Icona] EntryPoint.java
  > [Icona] FXMLController.java
  > [Icona] Main.java
  > [Icona] > MassimoFXMLController.java
  > [Icona] > MinimoFXMLController.java
```

- Il package *compensorio_sciistico.db* che si comporta da DAO (Data Access Object), ossia gestisce l'iterazione con il database, dalla connessione all'estrapolazione dei dati utili per il funzionamento del software.

```
▼ [Icona] it.polito.tdp.compensorio_sciistico.db
  > [Icona] CompensorioDAO.java
  > [Icona] DBConnect.java
  > [Icona] TestDAO.java
```

- Il package *compensorio_sciistico.model* che si comporta da model e contiene tutte le classi legate alla componente algoritmica e di elaborazione dei dati.

```
▼ [Icona] > it.polito.tdp.compensorio_sciistico.model
  > [Icona] Esperto.java
  > [Icona] Impianto.java
  > [Icona] Intermedio.java
  > [Icona] Livello.java
  > [Icona] > Model.java
  > [Icona] Pista.java
  > [Icona] Principiante.java
  > [Icona] Stazione.java
  > [Icona] > TestModel.java
  > [Icona] Tratta.java
  > [Icona] TrattaTabella.java
```

5.2 Algoritmi coinvolti

L'applicazione permette di effettuare due ricerche differenti, quella del cammino minimo e quella di un cammino massimo mediante un algoritmo ricorsivo.

Per svolgere gli opportuni calcoli è stata utilizzata la teoria dei grafi con l'adozione di un multigrafo orientato e pesato, realizzato grazie alla libreria *jGraphT* che fornisce classi e metodi per operare sui grafi.

Nel grafo utilizzato i vertici sono rappresentati da oggetti della classe *Stazione* mentre gli archi, orientati e pesati, sono rappresentati da oggetti della classe *DefaultWeightedEdge*.

Gli archi possono essere classificati in due tipi, possono rappresentare oggetti della classe *Pista* oppure oggetti della classe *Impianto*. Questa distinzione viene effettuata mediante l'implementazione di due *HashMap*, rispettivamente *mappaPiste* e *mappaImpianti*. Gli archi di tipo *Pista* collegano due stazioni, in cui quella di partenza ha una quota maggiore di quella di arrivo, mentre per gli archi di tipo *Impianto* collegano una stazione avente quota minore verso una stazione con quota maggiore. Il peso degli archi è determinato dal tempo, nel caso delle piste viene determinato attraverso la moltiplicazione dell'attributo lunghezza della classe con la velocità media di percorrenza, un attributo che dipende dal livello dell'utente, opportunamente convertita in metri al secondo.

Di seguito la tabella contenente le velocità per livello e difficoltà pista:

LIVELLO UTENTE→ DIFFICOLTÀ PISTA ↓	PRINCIPIANTE	INTERMEDIO	ESPERTO
BLU	10 km/h	20 km/h	50 km/h
ROSSA	NON ESEGUIBILE	10 km/h	30 km/h
NERA	NON ESEGUIBILE	NON ESEGUIBILE	15 km/h

La scelta di utilizzare un multigrafo è dovuta dal fatto che due vertici possono essere collegati da più archi aventi stesso verso, in quanto una stazione di monte ed una di valle possono essere collegate da più piste distinte.

Il grafo viene realizzato mediante il metodo *creaGrafo()* nel package *model*. I vertici vengono caricati indipendentemente dal livello dell'utente mentre gli archi vengono caricati mediante il metodo *caricaPiste()*, che carica sul grafo solo le piste in grado di essere praticate a seconda del livello introdotto dall'utente.

Alcuni screenshot relativi al popolamento del grafo.

```
public void creaGrafo(Livello livello) {

    this.grafo = new DirectedWeightedMultigraph<Stazione, DefaultWeightedEdge>(DefaultWeightedEdge.class);
    dao.caricaStazioni(idMap);
    Graphs.addAllVertices(this.grafo, idMap.values());
    //aggiunta impianti
    for (Impianto i: dao.caricaImpianti()) {

        if (idMap.containsKey(i.getIdMonte()) && idMap.containsKey(i.getIdValle())) {
            //aggiungo alla mappa degli impianti
            String chiave = ""+i.getIdValle()+"-"+i.getIdMonte();
            mappaImpianti.put(chiave, i);

            Graphs.addEdgeWithVertices(this.grafo, idMap.get(i.getIdValle()), idMap.get(i.getIdMonte()), i.getTempoRisalita());
        }
    }
    impiantiDiscesa = dao.listaImpiantiDiscesa(mappaImpianti);
    caricaPiste(livello);
}

private void caricaPiste(Livello livello) {

    switch (livello.getLivello()) {

        case "Esperto":

            for (Pista p: dao.caricaPiste()) {
                if (idMap.containsKey(p.getIdMonte()) && idMap.containsKey(p.getIdValle())) {
                    //aggiungo alla mappa delle piste
                    String chiave = ""+p.getIdMonte()+"-"+p.getIdValle();
                    if (!mappaPiste.containsKey(chiave)) {
                        List<Pista> lista = new ArrayList<>();
                        lista.add(p);
                        mappaPiste.put(chiave, lista);
                    } else {
                        mappaPiste.get(chiave).add(p);
                    }

                    double tempo = 0;
                    if (p.getColore().equals("ROSSA")) {
                        tempo = p.getLunghezza() / (Esperto.getVelocitaRossa() / 3.6);
                    } else if (p.getColore().equals("BLU")) {
                        tempo = p.getLunghezza() / (Esperto.getVelocitaBlu() / 3.6);
                    } else {
                        tempo = p.getLunghezza() / (Esperto.getVelocitaNera() / 3.6);
                    }
                    tempo = Math.ceil(tempo);
                    p.setTempoPercorrenza(tempo);
                    Graphs.addEdgeWithVertices(this.grafo, idMap.get(p.getIdMonte()), idMap.get(p.getIdValle()), tempo);
                }
            }
            break;
    }
}
```

```

case "Intermedio":
    for(Pista p: dao.caricaPiste()) {
        if(!p.getColore().equals("NERA")) {
            if(idMap.containsKey(p.getIdMonte()) && idMap.containsKey(p.getIdValle())) {
                //aggiungo alla mappa delle piste
                String chiave = ""+p.getIdMonte()+"-"+p.getIdValle();
                if(!mappaPiste.containsKey(chiave)) {
                    List<Pista> lista = new ArrayList<>();
                    lista.add(p);
                    mappaPiste.put(chiave, lista);
                }else {
                    mappaPiste.get(chiave).add(p);
                }

                double tempo = 0.0;
                if(p.getColore().equals("ROSSA")) {
                    tempo = p.getLunghezza()/(Intermedio.getVelocitaRossa()/3.6);
                }else {
                    tempo = p.getLunghezza()/(Intermedio.getVelocitaBlu()/3.6);
                }
                tempo= Math.ceil(tempo);
                p.setTempoPercorrenza(tempo);
                Graphs.addEdgeWithVertices(this.grafo, idMap.get(p.getIdMonte()),idMap.get(p.getIdValle()), tempo);
            }
        }
    }
    break;

case "Principiante":
    for(Pista p: dao.caricaPiste()) {
        if(p.getColore().equals("BLU")) {
            if(idMap.containsKey(p.getIdMonte()) && idMap.containsKey(p.getIdValle())) {
                //aggiungo alla mappa delle piste
                String chiave = ""+p.getIdMonte()+"-"+p.getIdValle();
                if(!mappaPiste.containsKey(chiave)) {
                    List<Pista> lista = new ArrayList<>();
                    lista.add(p);
                    mappaPiste.put(chiave, lista);
                }else {
                    mappaPiste.get(chiave).add(p);
                }

                double tempo = p.getLunghezza()/(Principiante.getVelocitaBlu()/3.6);
                tempo= Math.ceil(tempo);
                p.setTempoPercorrenza(tempo);
                Graphs.addEdgeWithVertices(this.grafo, idMap.get(p.getIdMonte()),idMap.get(p.getIdValle()), tempo);
            }
        }
    }
    break;
}

```

5.2.1 Cammino minimo

Il metodo *camminoMinimo()* del package *model* permette di calcolare il percorso più veloce in termini di tempo tra due stazioni. È stato utilizzato l'algoritmo *Dijkstra's Shortest Path* implementato nella libreria *jGraphT*, che mediante il metodo *getPath()* permette di ottenere un oggetto di tipo *GraphPath* contenente una lista di archi da percorrere. Il tempo minimo calcolato viene poi maggiorato per ogni impianto percorso di una componente tempo associata all'intervallo di tempo tra i mezzi di risalita, al numero di utenti introdotti in input e al numero di posti dell'impianto secondo la seguente formula

$$\text{Math.ceil}\left(\frac{\text{numero utenti}}{\text{numero posti}}\right) \times \text{intervallo}$$

Il risultato di ritorno del metodo risulta essere una lista di oggetti di tipo *Tratta*, classe padre delle classi *Pista* e *Impianto*.

Immagini relative al metodo *camminoMinimo* e alle classi *Tratta*, *Pista*, *Impianto*

```
public List<Tratta> camminoMinimo(String inizio, String fine, int numeroUtenti) {
    this.tempoMinimo = 0.0;
    List<Tratta> soluzione = new ArrayList<>();

    Stazione partenza = idMap.get(inizio);
    Stazione arrivo = idMap.get(fine);
    DijkstraShortestPath<Stazione, DefaultWeightedEdge> dij = new DijkstraShortestPath<Stazione, DefaultWeightedEdge>(this.grafo);

    GraphPath<Stazione, DefaultWeightedEdge> cammino = dij.getPath(partenza, arrivo);
    if(cammino!=null) {
        for(int i = 0; i< cammino.getVertexList().size()-1; i++) {
            String chiave = cammino.getVertexList().get(i).getCodice()+"-"+cammino.getVertexList().get(i+1).getCodice();

            if(mappaPiste.containsKey(chiave)) {
                //mi da una lista, devo scegliere la prima pista in ordine decrescente, perche è la piu breve in termini di tempo
                List<Pista> lista = new ArrayList<>( mappaPiste.get(chiave));
                Collections.sort(lista, new Comparator<Pista>() {
                    @Override
                    public int compare(Pista o1, Pista o2) {
                        return -(o1.getTempoPercorrenza()-o2.getTempoPercorrenza());
                    }
                });
                tempoMinimo+= lista.get(0).getTempoPercorrenza();
                soluzione.add(lista.get(0));
            } else {
                Impianto impianto = mappaImpianti.get(chiave);

                tempoMinimo+= impianto.getTempoRisalita() + Math.ceil(numeroUtenti/impianto.getPosti())*(impianto.getIntervallo()*60);
                soluzione.add(mappaImpianti.get(chiave));
            }
        }
    }

    return soluzione;
}
```

```

public class Tratta {
    private String tipo;

    /**
     * @param tipo
     */
    public Tratta(String tipo) {
        super();
        this.tipo = tipo;
    }

    public String getTipo() {
        return tipo;
    }
}

public class Pista extends Tratta {
    private int id;
    private String nome;
    private String idMonte;
    private String idValle;
    private String colore;
    private int lunghezza;
    private String localita;
    private double tempoPerCorrenza;

    public Pista(int id, String nome, String idMonte, String idValle, String colore, int lunghezza, String localita, String tipo) {
        super(tipo);
        this.id = id;
        this.nome = nome;
        this.idMonte = idMonte;
        this.idValle = idValle;
        this.colore = colore;
        this.lunghezza = lunghezza;
        this.localita = localita;
        this.tempoPerCorrenza=0.0;
    }

    public int getId() {return id;}
    public String getNome() {return nome;}

    public String getIdMonte() {return idMonte;}
    public String getIdValle() {return idValle;}
    public String getColore() {return colore;}
    public int getLunghezza() {return lunghezza;}
    public String getLocalita() {return localita;}
    public double getTempoPerCorrenza() {return tempoPerCorrenza;}
    public void setTempoPerCorrenza(double tempoPerCorrenza) {this.tempoPerCorrenza=tempoPerCorrenza;}
    public String toString() {return "Pista: " + id + " " + nome + " " + idMonte + " " + idValle + " " + colore + " " + lunghezza + " " + localita + " " + tempoPerCorrenza + "\n";}
}

```

```

public class Impianto extends Tratta{

    private int id;
    private String nome;
    private String idMonte;
    private String idValle;
    private String localita;
    private String tipologia;
    private boolean iniziale;
    private int posti;
    private double tempoRisolita;
    private double intervallo;
    private LocalTime oraApertura;
    private LocalTime oraChiusura;

    public Impianto(int id, String nome, String idMonte, String idValle, String localita,
        String tipologia, boolean iniziale, int posti, double tempoRisolita, double intervallo,
        LocalTime oraApertura, LocalTime oraChiusura, String tipo) {
        super(tipo);
        this.id = id;
        this.nome = nome;
        this.idMonte = idMonte;
        this.idValle = idValle;
        this.localita = localita;
        this.tipologia = tipologia;
        this.iniziale = iniziale;
        this.posti = posti;
        this.tempoRisolita = tempoRisolita;
        this.intervallo = intervallo;
        this.oraApertura = oraApertura;
        this.oraChiusura = oraChiusura;
    }

    public int getId() {return id;}
    public String getNome() {return nome;}
    public String getIdMonte() {return idMonte;}
    public String getIdValle() {return idValle;}
    public String getLocalita() {return localita;}
    public String getTipologia() {return tipologia;}
    public boolean isIniziale() {return iniziale;}
    public int getPosti() {return posti;}
    public double getTempoRisolita() {return tempoRisolita;}
    public double getIntervallo() {return intervallo;}
    public LocalTime getOraApertura() {return oraApertura;}
    public LocalTime getOraChiusura() {return oraChiusura;}
}

```

5.2.2 Algoritmo ricorsivo

Per calcolare il cammino più variegato, ossia un cammino volto a minimizzare la monotonia nella scelta delle piste al fine di massimizzare il divertimento sempre rispettando i vincoli di tempo imposti dall'utente, è stato utilizzato un algoritmo ricorsivo il cui scopo è quello di massimizzare la seguente funzione:

$$punteggio = \sum_{i=1}^n \left(\frac{K}{\alpha} + \mu \times \lambda \right) + \varepsilon \times \tau$$

dove:

- la sommatoria viene effettuata dalle 1 alle n piste diverse presenti nella soluzione
- α è il numero di ripetizioni della stessa pista
- ε è una costante per valorizzare l'impiego del tempo
- τ è il tempo necessario a svolgere l'attività
- K è il valore di punti costante associata ad una pista in base alla sua difficoltà e all'esperienza dell'utente.

Tabella dei valori di K:

LIVELLO UTENTE→ DIFFICOLTÀ PISTA ↓	PRINCIPIANTE	INTERMEDIO	ESPERTO
BLU	3pt	2pt	1pt
ROSSA	NON DETERMINATO	3pt	2pt
NERA	NON DETERMINATO	NON DETERMINATO	3pt

- μ è un valore costante misurato in $\frac{pt}{km}$ associato con valori diversi rispetto alla difficoltà della pista e l'esperienza dell'utente, in modo tale da valorizzare maggiormente la pista più lunga a parità di difficoltà.

Tabella relativa ai valori:

LIVELLO UTENTE→ DIFFICOLTÀ PISTA ↓	PRINCIPIANTE	INTERMEDIO	ESPERTO
BLU	0.8 pt/km	0.4 pt/km	0.2 pt/km
ROSSA	NON DETERMINATO	0.8 pt/km	0.5 pt/km
NERA	NON DETERMINATO	NON DETERMINATO	0.8 pt/km

Questi valori sono presenti nelle classi *Principiante*, *Intermedio* ed *Esperto*, sottoclassi della classe *Livello* e ottenibili mediante opportuni metodi getters.

Alcune immagini relative ai metodi per il calcolo del percorso più variegato.

Metodo trovaMassimoPercorso()

```
public List<Tratta> trovaMassimoPercorso(Livello livello, int numeroutenti, int tempoMax, Impianto partenza, Impianto arrivo){
    //se esiste un cammino minimo ci proco
    List<Tratta> camminoMinimo = this.camminoMinimo(partenza.getIdValle(), arrivo.getIdValle(), numeroutenti);
    System.out.println("TEMPOMINIMO: " + this.tempoMinimo + " - " + camminoMinimo);
    if(this.tempoMinimo <= tempoMax && camminoMinimo.size() != 0) {
        List<Tratta> parziale = new ArrayList<>();
        partenzaRicorsivo = idMap.get(partenza.getIdValle());
        arrivoRicorsivo = idMap.get(arrivo.getIdValle());
        this.numeroutenti = numeroutenti;

        parziale.add(partenza);

        bestSoluzione = new ArrayList<>();
        punteggioMax = 0.0;
        this.tempoMax = tempoMax;
        double tempoIniziale = partenza.getTempoRisolita() + (partenza.getIntervallo() * (Math.ceil(numeroutenti / partenza.getPosti()))) * 60;
        cercaRicorsivo(parziale, tempoMax, tempoIniziale, idMap.get(partenza.getIdMonte()), livello);

        return bestSoluzione;
    } else {
        return null;
    }
}
```

Metodo cercaRicorsivo()

```
private void cercaRicorsivo(List<Tratta> parziale, int tempoMax, double tempo, Stazione ultimaInserita, Livello livello) {

    if(tempo>this.tempoMax) {

        List<Tratta> soluzione = new ArrayList<>(parziale);
        boolean impianto = false;

        double tempoPercorrenza = 0.0;

        Tratta ultimaTratta = (soluzione.get(soluzione.size()-1));
        if(ultimaTratta.getTipo().equals("Pista")) {
            tempoPercorrenza = tempo - ((Pista)ultimaTratta).getTempoPercorrenza();
        }
        else {
            Impianto i = ((Impianto)ultimaTratta);
            tempoPercorrenza = tempo - i.getTempoRisalita() - (i.getIntervallo()*(Math.ceil(this.numeroUtenti/i.getPosti())))*60;
        }
        //ultima tratta valida nel tempo
        Tratta ultimaTrattaValida = (parziale.get(parziale.size()-2));

        Stazione arrivo = null;
        if(ultimaTrattaValida.getTipo().equals("Pista")) {
            arrivo = idMap.get(((Pista)ultimaTrattaValida).getIdValle());
        }
        else {
            Impianto i = ((Impianto)ultimaTrattaValida);
            arrivo = idMap.get(i.getIdValle());
            impianto = true;
            tempoPercorrenza -= i.getTempoRisalita() - (i.getIntervallo()*(Math.ceil(this.numeroUtenti/i.getPosti())))*60;
        }
        if(arrivo.equals(arrivoRicorsivo)) {
            //parametro per valorizzare il tempo nel calcolo del punteggio
            double k = 0.05;

            double punteggio = calcolaPunteggio(soluzione, livello) + k*tempoPercorrenza;

            if(punteggio>punteggioMax) {

                //rimuovo quella non valida
                soluzione.remove(soluzione.size()-1);

                //se la penultima è impianto tolgo
                if(impianto) {
                    soluzione.remove(soluzione.size()-1);
                }

                tempoBest = tempoPercorrenza;
                punteggioMax = punteggio;

                bestSoluzione = new ArrayList<>(soluzione);
                return;
            }
        }
        return;
    }
    else {
        for(Stazione prossimo: Graphs.successorListOf(this.grafo, ultimaInserita)) {
            String chiave = ""+ultimaInserita.getCodice()+"-"+prossimo.getCodice();

            if(mappaImpianti.containsKey(chiave)) {
                //aggiungo impianto

                Impianto i = mappaImpianti.get(chiave);
                if(!parziale.contains(i)) {
                    parziale.add(i);

                    double tempoImpianto = i.getTempoRisalita() + (i.getIntervallo()*(Math.ceil(this.numeroUtenti/i.getPosti())))*60;
                    tempo+=tempoImpianto;
                    //ricorsivo
                    cercaRicorsivo(parziale, tempoMax, tempo, prossimo, livello);
                    //backtracking
                    tempo-=tempoImpianto;
                    parziale.remove(parziale.size()-1);
                }
            }
            else if(mappaPiste.containsKey(chiave)) {
```

```

        List<Pista> lista = mappaPiste.get(chiave);
        for(Pista p: lista) {
            if(!parziale.contains(p)) {
                parziale.add(p);
                double tempoPista= p.getTempoPercorrenza();
                tempo+=tempoPista;

                cercaRicorsivo(parziale, tempoMax, tempo, prossimo, livello);

                //backtracking
                tempo-=tempoPista;
                parziale.remove(parziale.size()-1);
            }
        }
    }
}
return;
}

```

Metodo calcolaPunteggio()

```

private double calcolaPunteggio(List<Tratta> parziale, Livello livello) {
    Map<Pista, Integer> ripetizioni = new HashMap<>();
    double punteggio = 0.0;
    final int K = 3;
    //fino a parziale.size()-1 perchè ce ancora quella non valida
    for(int i = 0; i< parziale.size()-1; i++) {
        if(parziale.get(i).getTipo().equals("Pista")) {
            Pista pista = (Pista)parziale.get(i);
            if(!ripetizioni.containsKey(pista))
                ripetizioni.put(pista, 1);
            else {
                int rip = ripetizioni.get(pista) +1;
                ripetizioni.put(pista, rip);
            }
        }
    }

    switch(livello.getLivello()) {
        case "Principiante":
            for(Pista pista: ripetizioni.keySet()) {
                punteggio+= (K/ripetizioni.get(pista))+(Principiante.getkBlu()*(pista.getLunghezza()/1000));
            }
            break;

        case "Intermedio":
            for(Pista pista: ripetizioni.keySet()) {
                if(pista.getColore().equals("BLU")) {
                    punteggio+= ((K-1)/ripetizioni.get(pista))+(Intermedio.getkBlu()*(pista.getLunghezza()/1000));
                }else{
                    punteggio+= (K/ripetizioni.get(pista))+(Intermedio.getkRossa()*(pista.getLunghezza()/1000));
                }
            }
            break;
    }
}

```

6 Videate dell'applicazione e dimostrazione

L'applicazione è costituita da due interfacce grafiche, una per il calcolo del cammino più veloce in termini di tempo che collega due impianti di risalita e una per il calcolo del cammino più variegato che si possa effettuare tenendo conto dei vincoli di tempo imposti dall'utente.

In ogni interfaccia di calcolo è presente una input box in cui va introdotto il numero di utenti da gestire per il calcolo, una combo box per selezionare uno dei tre livelli possibili degli utenti. Vi sono poi alcune combo box per selezionare le località sciistiche e i rispettivi impianti di partenza ed arrivo.

Nell'interfaccia per il calcolo del cammino minimo vi sono poi due radio button, quello relativo alla colonna *Solo iniziali* che permette il popolamento della combo box relativa agli impianti solo con quelli raggiungibili dalla strada e quello nominato *Stazione valle*, che permette se cliccato di indicare che ci si trova alla stazione di valle dell'impianto, altrimenti si il percorso viene calcolato dalla stazione di monte.

Nell'interfaccia per il calcolo dell'itinerario più variegato le combo box vengono popolate solo con gli impianti iniziali della località selezionata e sono presenti due combo box per specificare la durata massima dell'attività.



CalcolaPercorsi

Percorso più entusiasmante

Numero partecipanti: Seleziona livello: ▼

	Località	Impianto	Durata attività	
Partenza	▼	▼	Ore	▼
Arrivo	▼	▼	Minuti	▼

Calcola percorso

Tratta	Descrizione	Nome	Località	Tempo
--------	-------------	------	----------	-------

Nessun contenuto nella tabella

Durata attività: Indietro

Esempio di risultato

CalcolaPercorsi

Percorso più entusiasmante

Numero partecipanti: Seleziona livello: ESPERTO ▼

	Località	Impianto	Durata attività	
Partenza	Sestriere ▼	Cit-Roc (Seggiovìa) ▼	Ore	0 ▼
Arrivo	Sauze D'Oulx ▼	Clotes (Seggiovìa) ▼	Minuti	43 ▼

Calcola percorso

Tratta	Descrizione	Nome	Località	Tempo
Impianto	Seggiovìa	Cit-Roc	Sestriere	5' 2"
Pista	ROSSA	3	Sestriere	5'
Pista	BLU	Baby SX	Sestriere	26"
Impianto	Tappeto	Jolly	Sestriere	3'
Pista	BLU	Jolly	Sestriere	26"
Pista	BLU	Golf	Sestriere	11"
Pista	BLU	Boulevard Golf	Sestriere	36"
Impianto	Cabinovia	Sestriere-Fraiteve	Sestriere	5'
Impianto	Cabinovia	Sestriere-Fraiteve 2	Sestriere	5'
Pista	ROSSA	81 alta	Sansicario	18"
Pista	NERA	25	Sauze D'Oulx	5' 12"
Impianto	Seggiovìa	Chamonier	Sauze D'Oulx	6' 32"

Durata attività: Indietro

Nessun contenuto nella tabella

Esempio di risultato

22

7 Conclusione e valutazione dei risultati

Calcolando il cammino minimo mediante i metodi implementati nella libreria jGraphT i risultati si ottengono rapidamente, quindi potrebbe essere di utile applicazione ai problemi citati precedentemente. Tale considerazione non è conforme invece all'algoritmo ricorsivo utilizzato per il calcolo del cammino massimo, questo è dovuto al fatto che nella procedura di backtracking la rimozione attraverso il metodo *remove()* implementato nelle array list ha una complessità computazionale $O(n)$, dove n è la dimensione della lista stessa. Questa criticità temporale nel fornire risultati dipende dal livello introdotto e dalla massima durata dell'attività, in quanto minore è il livello introdotto minore sono gli archi presenti nel grafo perché minore sono le piste che l'utente del livello selezionato può praticare e minore è la massima durata minore sarà il tempo che l'algoritmo impiega a rispondere. Infatti, effettuando diverse prove, l'algoritmo tende a rispondere in pochi secondi introducendo come livello *Principiante* e come durata massima un'ora circa. Mantenendo la stessa durata, ciò non accade per il livello *Esperto*, che impiega mediamente circa un minuto a fornire risultati.

Per durate oltre le due ore l'applicazione non fornisce risultati in tempi 'umani'.

Di seguito vengono riportati alcuni risultati.

7.1 Cammino minimo:

CalcolaPercorsi

Percorso più veloce

Numero partecipanti: Seleziona livello: ESPERTO ▾

	Località	Solo iniziali	Impianto	Stazione valle
Partenza	Sestriere ▾	<input type="radio"/>	Combetta (Skilift) ▾	<input type="radio"/>
Arrivo	Sauze D'Oulx ▾	<input checked="" type="radio"/>	Sportinia (Seggiovvia) ▾	<input type="radio"/>

Calcola percorso

Tratta	Descrizione	Nome	Località	Tempo
Pista	BLU	87 bis	Sestriere	24"
Impianto	Seggiovvia	Trebiats	Sestriere	5' 38"
Pista	ROSSA	Baby DX	Sestriere	42"
Pista	BLU	Golf	Sestriere	11"
Pista	BLU	Boulevard Golf	Sestriere	36"
Impianto	Cabinovia	Sestriere-Fraiteve	Sestriere	5'
Impianto	Cabinovia	Sestriere-Fraiteve 2	Sestriere	5'
Pista	BLU	Broussailles	Sauze D'Oulx	2' 32"
Pista	BLU	Mini	Sauze D'Oulx	11"

Tempo totale: 21' 2" Indietro

Diminuendo il numero di partecipanti a due il cammino rimane lo stesso ma diminuisce il tempo totale:

CalcolaPercorsi

Percorso più veloce

Numero partecipanti: 2 Seleziona livello: ESPERTO

	Località	Solo iniziali	Impianto	Stazione valle
Partenza	Sestriere	<input type="radio"/>	Combetta (Skilift)	<input type="radio"/>
Arrivo	Sauze D'Oulx	<input checked="" type="radio"/>	Sportinia (Seggiovio)	<input type="radio"/>

Calcola percorso

Tratta	Descrizione	Nome	Località	Tempo
Pista	BLU	87 bis	Sestriere	24"
Impianto	Seggiovio	Trebials	Sestriere	5' 38"
Pista	ROSSA	Baby DX	Sestriere	42"
Pista	BLU	Golf	Sestriere	11"
Pista	BLU	Boulevard Golf	Sestriere	36"
Impianto	Cabinovia	Sestriere-Fraiteve	Sestriere	5'
Impianto	Cabinovia	Sestriere-Fraiteve 2	Sestriere	5'
Pista	BLU	Broussailles	Sauze D'Oulx	2' 32"
Pista	BLU	Mini	Sauze D'Oulx	11"

Tempo totale: 20' 14" Indietro

Cambiando il livello dei partecipanti:

CalcolaPercorsi

Percorso più veloce

Numero partecipanti: 2 Seleziona livello: PRINCIPIANTE

	Località	Solo iniziali	Impianto	Stazione valle
Partenza	Sestriere	<input type="radio"/>	Combetta (Skilift)	<input type="radio"/>
Arrivo	Sauze D'Oulx	<input checked="" type="radio"/>	Sportinia (Seggiovio)	<input type="radio"/>

Calcola percorso

Tratta	Descrizione	Nome	Località	Tempo
Pista	BLU	87 bis	Sestriere	1' 56"
Impianto	Seggiovio	Trebials	Sestriere	5' 38"
Pista	BLU	Baby SX	Sestriere	2' 6"
Pista	BLU	Golf	Sestriere	54"
Pista	BLU	Boulevard Golf	Sestriere	3'
Impianto	Cabinovia	Sestriere-Fraiteve	Sestriere	5'
Impianto	Cabinovia	Sestriere-Fraiteve 2	Sestriere	5'
Pista	BLU	Broussailles	Sauze D'Oulx	12' 36"
Pista	BLU	Mini	Sauze D'Oulx	54"

Tempo totale: 37' 4" Indietro

7.2 Cammino massimo:

CalcolaPercorsi

Percorso più entusiasmante

Numero partecipanti:
Seleziona livello: ESPERTO

Località
Impianto
Durata attività

Partenza
Sestriere
Cit-Roc (Seggiovvia)
Ore
0

Arrivo
Sauze D'Oulx
Clotes (Seggiovvia)
Minuti
43

Calcola percorso

Tratta	Descrizione	Nome	Località	Tempo
Impianto	Seggiovvia	Cit-Roc	Sestriere	5' 2"
Pista	ROSSA	3	Sestriere	5'
Pista	BLU	Baby SX	Sestriere	26"
Impianto	Tappeto	Jolly	Sestriere	3'
Pista	BLU	Jolly	Sestriere	26"
Pista	BLU	Golf	Sestriere	11"
Pista	BLU	Boulevard Golf	Sestriere	36"
Impianto	Cabinovia	Sestriere-Fraiteve	Sestriere	5'
Impianto	Cabinovia	Sestriere-Fraiteve 2	Sestriere	5'
Pista	ROSSA	81 alta	Sansicario	18"
Pista	NERA	25	Sauze D'Oulx	5' 12"
Impianto	Seggiovvia	Chamonier	Sauze D'Oulx	6' 32"

Durata attività:
Indietro

Percorso più entusiasmante

Numero partecipanti:
Seleziona livello: PRINCIPIANTE

Località
Impianto
Durata attività

Partenza
Sestriere
Capret (Seggiovvia)
Ore
1

Arrivo
Sestriere
Baby Sestriere (Skilifit)
Minuti
3

Calcola percorso

Tratta	Descrizione	Nome	Località	Tempo
Impianto	Seggiovvia	Capret	Sestriere	9' 1"
Pista	BLU	Jolly	Sestriere	2' 6"
Pista	BLU	Golf	Sestriere	54"
Pista	BLU	Boulevard Golf	Sestriere	3'
Impianto	Cabinovia	Sestriere-Fraiteve	Sestriere	5'
Impianto	Cabinovia	Sestriere-Fraiteve 2	Sestriere	5'
Pista	BLU	46	Sestriere	21'
Impianto	Skilift	Combetta	Sestriere	3' 35"
Pista	BLU	87	Sestriere	1' 48"
Impianto	Seggiovvia	Trebials	Sestriere	5' 38"
Pista	BLU	Baby SX	Sestriere	2' 6"

Durata attività:
Indietro

In conclusione, sono soddisfatto per la realizzazione di questo progetto come prova finale della laurea triennale convinto che potrebbe essere migliorato e ampliato in più direzioni, sicuramente se si riuscissero ad ottenere maggiori dati relativi al comprensorio e al flusso di utenti si potrebbero effettuare diversi studi, quali gestione delle code agli impianti di risalita mediante simulazioni ad eventi discreti e, di conseguenza, migliorare l'accuratezza del calcolo dei cammini introducendo al tempo anche la componente relativa alle code degli impianti di risalita.

Link al video dimostrativo: <https://youtu.be/AFToEzgNM9k>

Link al progetto GitHub: <https://github.com/TdP-prove-finali/DelleaniMattia>



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/)