POLITECNICO DI TORINO

DIPARTIMENTO DI INGEGNERIA GESTIONALE E DELLA PRODUZIONE

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

Applicazione per cammino minimo azienda di trasporti AMAT

RELATORE
Prof. Fulvio Corno

CANDIDATO
Marcello Zampella
s246314

A.A. 2019/2020

INDICE

PROPOSTA DI PROGETTO

Descrizione del problema proposto

L'applicazione si pone l'obiettivo di trovare il cammino minimo tra due punti della città di Taranto, muovendosi solo attraverso le linee dell'azienda di trasporti AMAT. L'idea, quindi, è quella di trovare tutti i percorsi effettuabili dalla fermata iniziale a quella finale attraverso un algoritmo ricorsivo e trovare quello a costo minore.
La scelta del costo sarà effettuata dall'utente, che potrà scegliere tra il percorso a tempo minore e quello con meno cambi.

Descrizione della rilevanza gestionale del problema

L'azienda di trasporti per essere il più efficiente possibile deve ridurre al minimo il tempo di attesa affinché il servizio venga fornito al cliente (raggiungere un determinato punto).
A tal fine, non è solo necessario che questa impieghi le sue risorse il meglio possibile, ma anche che il cliente sia pienamente cosciente dei percorsi possibili.

Descrizione dei data-set per la valutazione

I data-set non sono disponibili in rete, ma mi sono stati gentilmente offerti dall'azienda. Questi sono suddivisi in 4 tabelle, a seconda che l'orario di interesse sia estivo o invernale, feriale o festivo, ognuna delle quali composta da 8 colonne:

1) idCors: identificativo univoco della corsa
2) linea: il numero di linea della corsa
3) identificativo: indica il tipo di corsa all'interno della linea.
4) numeroFermata: numero progressivo delle fermate a cui l'autobus si ferma
5) codiceLocale: codice univoco della località
6) Desc_Staz: nome della località
7) tempoPartenza: orario di partenza della corsa
8) tempoPassato: tempo di passaggio alla fermata

Descrizione preliminare degli algoritmi coinvolti

- Costruzione del multiGrafo, i cui nodi rappresentano le fermate e gli archi, pesati e orientati, le varie corse che collegano le fermate.
- Algoritmo ricorsivo per visitare il grafo, al fine di trovare tra tutte le combinazione possibili, quella a costo minore.

Descrizione preliminare delle funzionalità previste per l’applicazione software

L'applicazione prevederà un'interfaccia attraverso la quale l'utente potrà selezionare l'orario di partenza o di arrivo e il periodo dell’anno in cui si effettua il viaggio.
In seguito l'utente sceglierà la via di partenza a la via di destinazione, quindi la tipologia di viaggio tra quello più comodo (minor cambio di mezzi, a discapito del tempo impiegato) e quello più veloce. Nel caso fosse scelta la prima opzione, l’utente dovrà inserire anche un numero massimo di cambi da poter effettuare.
In ogni caso se ci dovessero essere percorsi ugualmente comodi verrebbe scelto quello più veloce e viceversa.
In seguito sarà riportato il percorso scelto, indicando all'utente a che ora recarsi alla fermata, quale bus prendere, eventualmente dove scendere e quale linea aspettare e il tempo di percorrenza totale (quindi l'ora di arrivo). Tutte le informazioni potranno essere successivamente salvate in un documento di testo.

Descrizione dettagliata del problema affrontato

Amat è l’azienda di trasporto pubblico operante su tutto il territorio del comune di Taranto. Essendo il territorio molto grande, l’azienda fornisce circa 100 corse diverse (il numero varia a seconda della stagione) e 900 fermate. Nasce quindi la necessità da parte degli utenti di un'applicazione attraverso la quale poter notevolmente facilitare la ricerca di percorsi all’interno della città.
Generalmente i cittadini conoscono soluzioni buone per poche tratte/orari, scoperte nel corso del tempo con metodi deduttivi, ma questa è un’esperienza sicuramente molto limitata rispetto all’utilizzo di un applicativo, a causa della maggiore capacità di calcolo e conoscenza.
Il problema consiste, quindi, nel trovare la giusta combinazione di tratte di autobus, ricordando che linee diverse seguono percorsi diversi, ma possono intrecciarsi per una o più fermate. É proprio in queste fermate che l’utente potrebbe fermarsi e aspettare il passaggio di una linea diversa, arrivando a nuove fermate cui la linea precedente non sarebbe potuta arrivare (o magari in meno tempo).

Descrizione del data-set utilizzato per l'analisi

Come accennato nella proposta, il database è composto da 4 tabelle, corrispondenti ai diversi tipi di orari nell'anno e, quindi, in alcun modo legate tra loro. Tutte le tabelle hanno 8 colonne diverse e 2 chiavi primarie (IdCors, ossia l'id univoco della corsa e numeroFermata, che è un valore intero che incrementa con il numero di fermate visitate dalla corsa).
La riga di una tabella descrive per una corsa la fermata visitata e il momento della giornata in cui la visita.

Descrizione ad alto livello delle strutture dati e degli algoritmi utilizzati

Il metodo **creaGrafo** genera un multi grafo pesato e orientato attraverso le informazioni

- *String scelta:* comunica quale tabella utilizzare
- *String tipo:* comunica se l'orario è di arrivo o di partenza
- *LocalTime ltorario*: indica l'orario scelto dall'utente

Il multigrafo ha come nodi tutte le fermate presenti nel database (classe Collegamento), come archi tutte le possibile corse che collegano i nodi (classe Arco) e il peso è dato dal tempo in minuti impiegato dalla corsa per arrivare dalla fermata di origine a quella di destinazione.
A seconda che l'orario sia di partenza o di arrivo, le informazioni prese dal database riguardano esclusivamente le corse che lavorano, rispettivamente, dopo o prima l'orario indicato. É stato opportuno inserire un limite di ore dall'orario selezionato pari a 3, oltre al quale le corse sono escluse, questo perchè un utente non aspetterebbe mai più di 3 ore per arrivare al punto di arrivo e inoltre in questo modo si diminuisce la grandezza del grafo e il tempo impiegato dall'algoritmo ricorsivo.
Il metodo cercaPercorso restituisce il percorso minimo attraverso le informazioni

- *Collegamento partenza*: fermata di partenza scelta dall'utente
- *Collegamento arrivo*: fermata di destinazione scelta dall'utente
- *LocalTime orario*: orario scelto dall'utente
- *String scelta*: comunica se l'orario è di arrivo o di partenza
- *String sceltaRicerca*: comunica la tipologia di ricerca da effettuare (tempo minimo o cambi minimi)
- *int numeroMassimo*: nel caso si sia scelto il tempo minimo, indica il massimo numero di cambi consentiti dall'utente

Questo metodo sostanzialmente inizializza e salva le variabili, quindi richiama l'algoritmo ricorsivo che, a seconda degli input dell'utente, può essere espandiPartenza o espandiArrivo.
I due metodi sono molto simili tra di loro: il primo parte dalla fermata di partenza e trova tutte le migliori combinazioni per arrivare alla fermata di arrivo, il secondo lavora in maniera opposta.
Data la similitudine tra i 2 eventi, li descriverò prendendo in considerazione **espandiPartenza**. Le variabili richieste dal metodo sono:

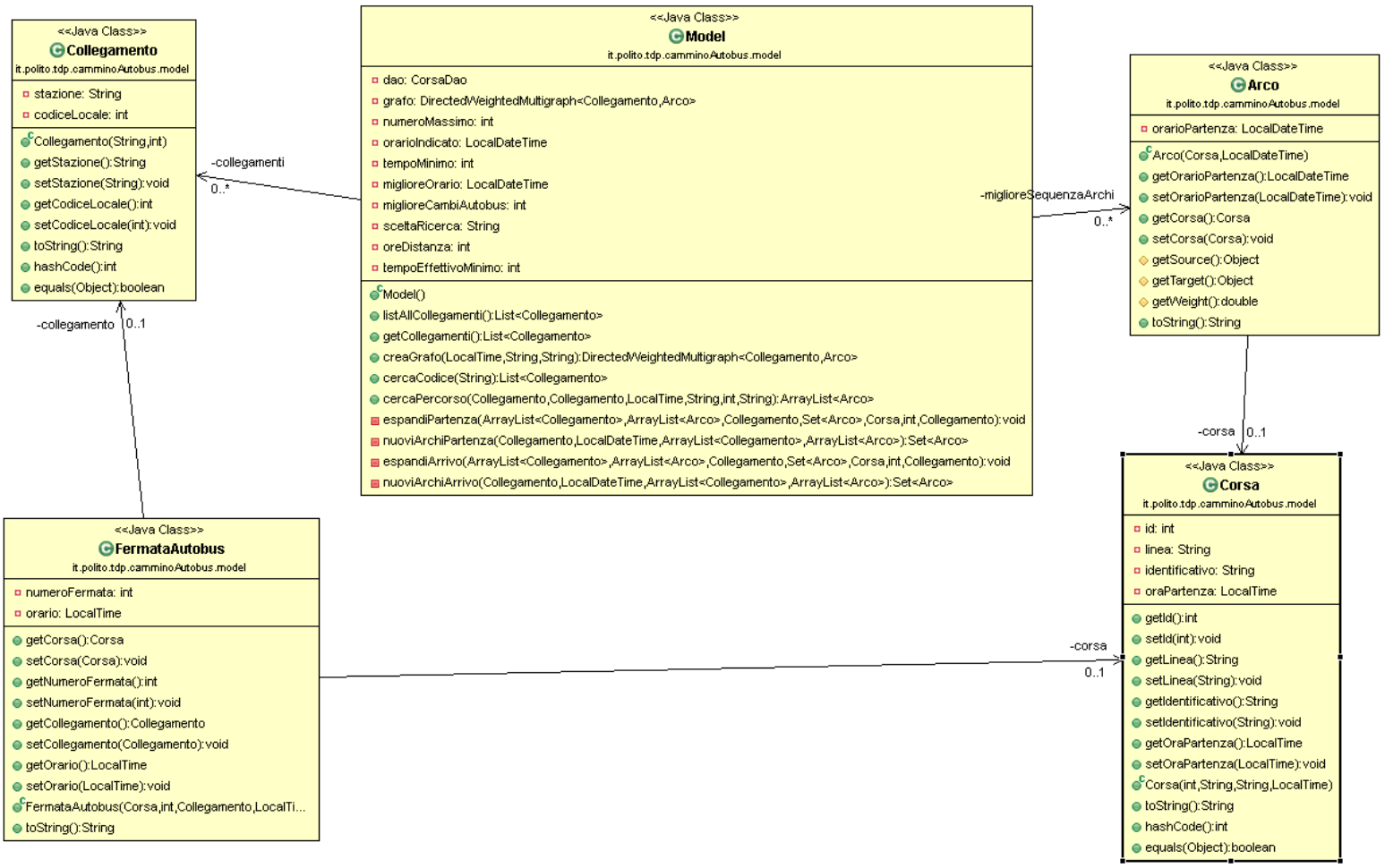
- *ArrayList<Collegamento> parziale*: lista dei nodi già visitati
- *ArrayList<Arco> parzialeArchi*: lista degli archi già utilizzati
- *Collegamento codiceLocaleAttuale*: fermata attulmente in visita
- *Set<Arco> successivi*: set degli archi visitabili
- *Corsa corsaAttuale*: corsa attualmente visitata
- *int cambiAutobus*: numero di autobus già visitati

- *Collegamento arrivo*: fermata di arrivo

La ricorsione consiste nell'analizzare il set di archi *successivi*, studiando se ha senso prendere in considerazione l'arco o meno. Quindi, con l'obiettivo di minimizzare il tempo necessario dell'algoritmo ricorsivo, si escludono quegli archi che porterebbero ad avere un tempo di viaggio superiore al migliore già studiato, oppure un numero di cambi superiore al numero consentito. Nel caso l'arco dovesse andare bene, si aggiornerebbero le variabili utilizzate, passando ad una fermata successiva, richiamando la funzione espandiPartenza e passando un nuovo set di archi analizzabili, trovati dalla funzione nuoviArchiPartenza. La condizione di terminazione dell'algoritmo è data dall'uguaglianza tra la fermata attuale e la fermata di arrivo, in questo caso si studia se i parametri della combinazione trovata siano migliori della combinazione salvata precedentemente, ricordando che i parametri hanno importanza diversa in base alla tipologia di ricerca effettuata.

La funzione **nuoviArchiPartenza** (analoga alla funzione nuoviArchiArrivo per espandiArrivo) restituisce un Set di archi da analizzare. Questa è una funzione molto importante per ottimizzare il tempo impiegato dall'algoritmo ricorsivo, in quanto non restituisce tutti gli archi che partono da una fermata (molto problematico in quanto si tratta di un multigrafo), ma solo un solo arco per identificativo di corsa, eliminando quegli archi la cui corsa ha un identificativo già considerato e orario di partenza inferiore alla corsa considerata.

Diagramma delle classi



L'applicazione, scritta in linguaggio Java, segue il pattern MVC (Model-View-Controller), quindi dividendo la struttura software in 3 parti principali:

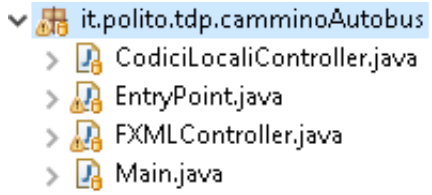
1) Model: parte algoritmica del programma
2) View: interagisce con l'utente
3) Controller: Raccoglie le informazioni di input dell'utente, trasmette e riceve informazioni dal model, genera output.

Inoltre l'applicazione sfrutta il pattern DAO (Data Access Object), che permette di accedere ai dati sul database, basandosi sugli input dell'utente (ad esempio l'orario) e ricavando informazioni poi processate dal Model

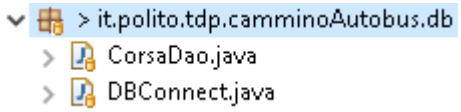
Il progetto è quindi suddiviso in 3 packages:

1) *it.polito.tdp.camminoAutobus*: contenente le classi Main da lanciare per avviare l'applicazione, EntryPoint che gestisce l'interfaccia iniziale, FXMLContreller contenente i metodi per gestire l'interazione con l'utente nell'interfaccia

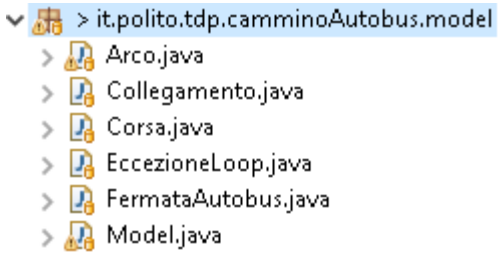
principale e CodiciLocaliController per un'interfaccia secondaria.



2) *it.polito.tdp.camminoAutobus.db*: contiene le classi CorsaDao che attraverso query SQL interagisce con il database e DBConnect utilizzata per la creazione della connessione al database.

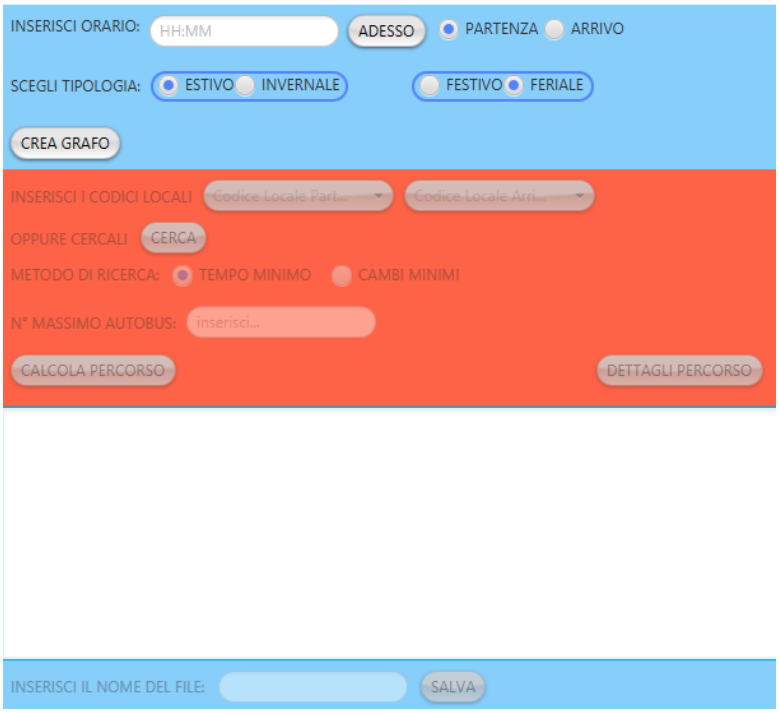


3) *it.polito.tdp.camminoAutobus.model*: contiene le classi Arco, estensione della classe DefaultWeightedEdge per avere archi personalizzati, Collegamento, per ricavare la relazione esistente tra codice locale e descrizione stazione all'interno del database,utilizzata poi come nodo del grafo, EccezioneLoop, eccezione personalizzata nel caso di errori (trovati frequentemente, poi risolti) nel database, FermataAutobus, utilizzato per ricavare una serie di informazioni dal database e Model a cui è affidata tutta la parte algoritmica del programma.

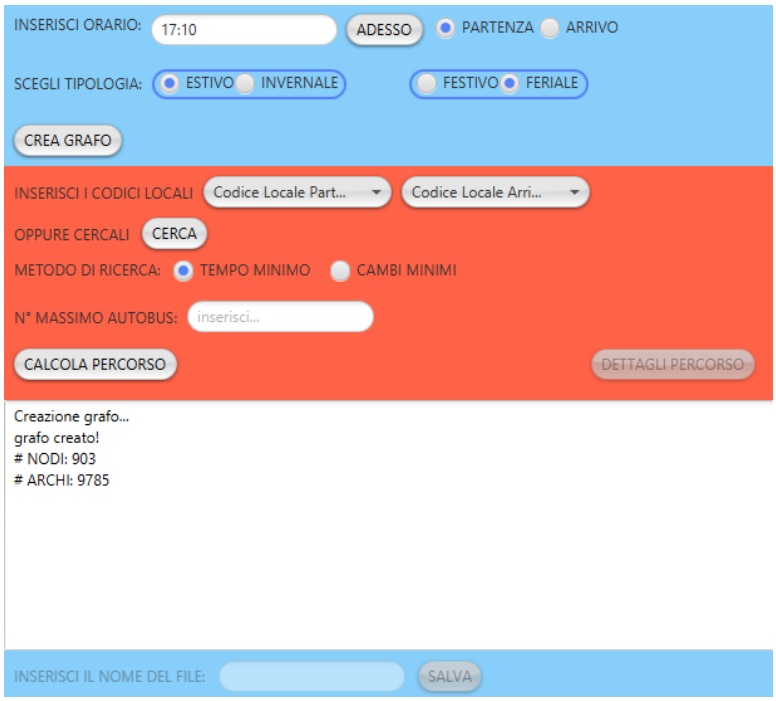


Videate dell’applicazione realizzata e link al video dimostrativo del software

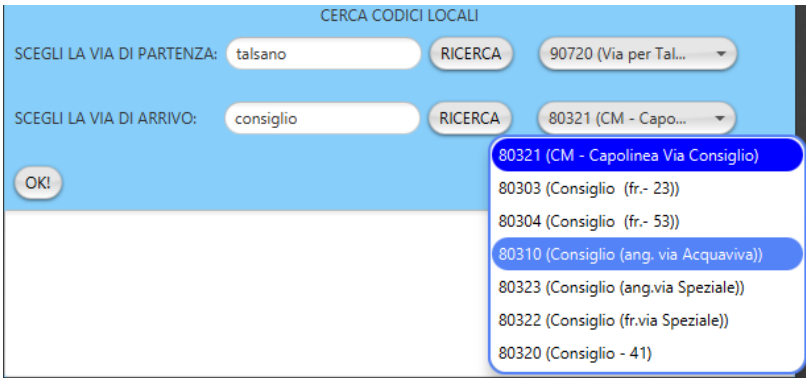
Link al video dimostrativo: https://youtu.be/x3bLajvsfnk
I colori dell’applicazione distinguono la parte dell’interfaccia addetta alla creazione del grafo e all’algoritmo ricorsivo.



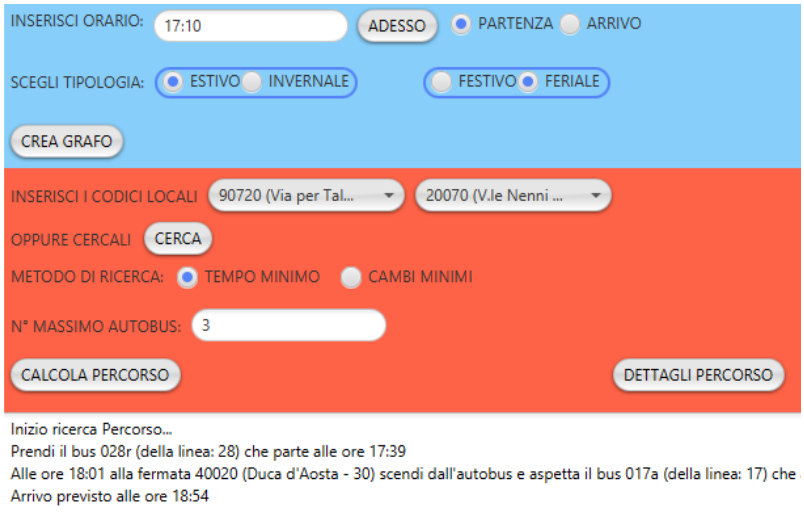
La schermata di avvio permette la sola costruzione del grafo.



Una volta costruito il grafo, sono stampate a schermo le informazioni che lo riguardano. Inoltre si sblocca l’interfaccia per la ricorsione



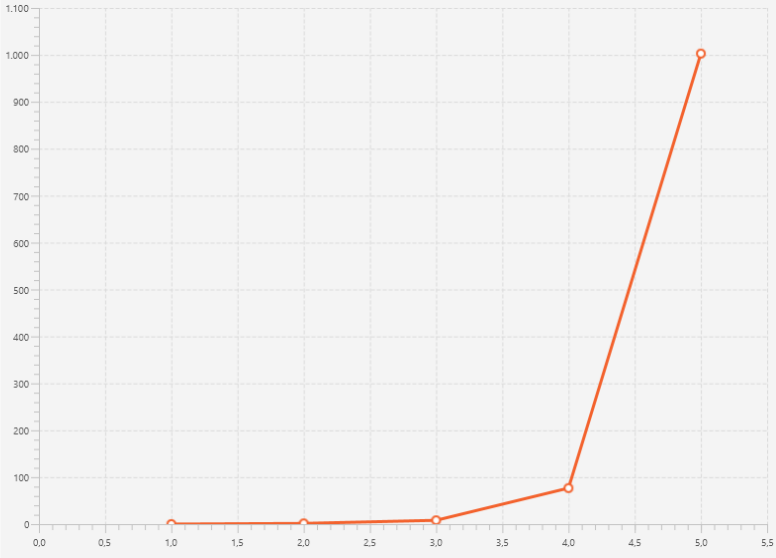
Premendo sul bottone “CERCA” si passa ad un’altra interfaccia in cui si possono trovare le fermate di interesse attraverso parole chiave.



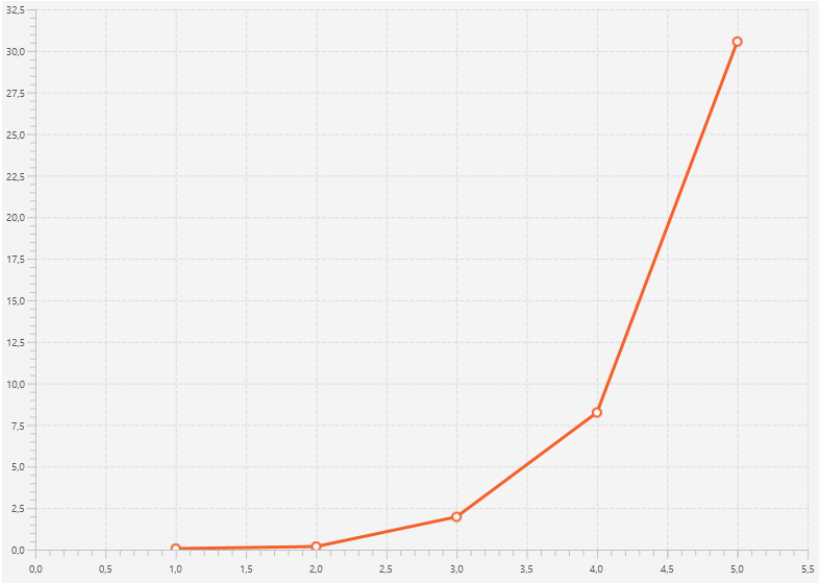
Premendo sul bottone “CALCOLA PERCORSO”, si attiva l’algoritmo ricorsivo, al cui termine viene stampato le informazioni salienti sul percorso ottimo trovato. Premendo poi su “DETTAGLI PERCORSO” è possibile visualizzare ulteriori dettagli e salvarli in un file di testo.

Risultati Sperimentali

Il grafo rappresenta il tempo in secondi (asse Y) dovuto alla ricorsione dell'esempio delle videate, variando il numero limite di autobus da cambiare da 1 a 5.



Come si può vedere la curva ha un andamento caratteristico delle funzioni esponenziali, perchè il numero delle combinazioni cresce in modo esponenziale con il numero dei cambi effettuabili, nonostante gli accorgimenti inseriti per eliminare le soluzioni non buone. Infatti, il tempo necessario alla ricerca del percorso migliore in modalità "Cambi Minimi", a causa della mancanza di un basso limite di cambi, è notevolmente maggiore. Il tempo impiegato dall'algoritmo ricorsivo dipende anche dalla combinazione di input selezionata: vediamo, nell'esempio successivo, che nonostante la curva simile, i tempi sono drasticamente ridotti



(esempio con input: partenza 100080, arrivo:80321, orario 17:10, "PARTENZA", orari_estivo_feriale)

Valutazioni sui risultati ottenuti e conclusioni

Il programma, nonostante sia stato pensato per l'azienda AMAT, può facilmente essere utilizzato da altre aziende di trasporto, impostando il proprio database, che deve rispettare la struttura del database 'amat'.
L'applicativo ha il difetto di non poter utilizzare la geolocalizzazione di fermate e autobus, queste permetterebbero infatti di migliorare la ricerca dei percorsi (ad esempio si può arrivare a 300 metri dalla fermata di arrivo) e adattarsi a situazioni reali. A quel punto, però, per un'azienda sarebbe molto più utile usare applicazioni come Google Maps, che lavorano in modo sicuramente più comodo e veloce, ma questa applicazione è pensata per tutte quelle realtà che non possono permettersi gli elevati costi della geolocalizzazione.
Reputo, quindi, che con una buona conoscenza della città si possa sfruttare molto bene l'applicazione, ad esempio cambiando fermate di partenza e destinazione con fermate a loro vicine.
L'applicazione, pensata nella sua forma più completa, potrebbe inviare dati riguardo le ricerche più frequenti, al fine di aiutare l'azienda in indagini statistiche e poter impiegare meglio le risorse disponibili, o si potrebbe anche inserire una sezione di acquisti, per incentivare l'uso del biglietto.
Un'applicazione capace di rendere il viaggio più comodo e veloce è un forte strumento di persuasione da parte dell'azienda all'utilizzo dei propri mezzi, questo non comporta solo un aumento dei profitti, ma anche un minore traffico cittadino e un minor inquinamento ambientale dovuta all'eccessivo numero di automobili.