**Documentazione**

**Progetto 3**

**Gruppo 6**

*Antonino Durazzo*

*Francesco de Pertis*

*Valentino Vastola*

**Timetable**

E’ stata progettata una timetable per la struttura di supporto agli esercizi della prova, che rappresenta un ipergrafo.

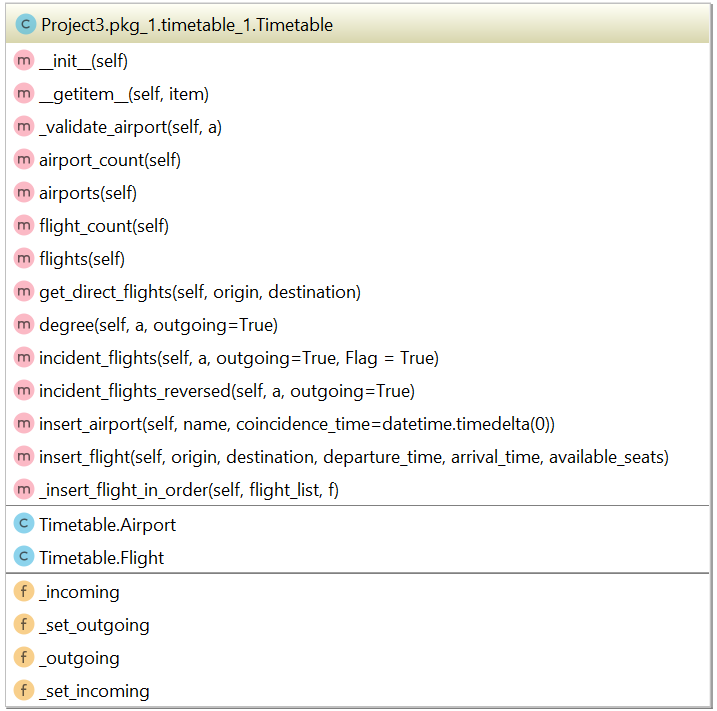
I nodi di questo grafo sono definiti come classe Airport e gli archi del grafo sono definiti come classe Flights con i relativi attributi e metodi richiesti dalle specifiche.

Questa timetable è stata sviluppata associando un lista di voli a ogni airport del grafo, tale lista di voli è ordinata in ordine crescente considerando l’orario di partenza di ciascun volo in modo tale da ottimizzare gli algoritmi richiesti dalla prova.

**Esercizio 1-Timetable modifiche apportate**

Per questo punto viene apportata una modifica alla timetable per tenersi traccia di un set degli archi incidenti su un nodo, ciò è stato fatto per non aggravare la complessità dell’algoritmo list\_routes.

**Uml timetable**

****

**Timetable**

timetable : timetable ipergrafo in input

**list\_routes(timetable, a, b, t, T)**

:param: timetable : timetable ipergrafo in input

a: airport aeroporto di partenza

b: airport aeroporto di destinazione

t: datetime tempo dal quale cercare tutte le rotte tra a e b

T: timedelta intervallo temporale

:return: None se non esiste nessun percorso tra a e b altrimenti ritorna la lista di percorsi da a to b

Complessità: O(m+n\*m)

L’algoritmo ritorna None se il primo volo uscente dal nodo di partenza parte dopo l’orario t + T, poichè abbiamo ordinato gli archi uscenti su un nodo in ordine crescente per l’orario di partenza, diversamente viene invocata la funzione dfs\_paths(), che genererà tutti i percorsi validi dal nodo di partenza al nodo di destinazione.

dfs\_paths() sfrutta la logica dell’algoritmo “depth first path” per grafi normali e opera ricorsivamente nel seguente modo:

-Per ogni volo che parte dall’aeroporto di partenza verrà inizializzata una lista vuota, nella quale ricorsivamente verrà aggiunto un volo verso un altro aeroporto in direzione di b solo se esso è valido, lo sarà nel caso in cui l’orario di partenza del volo è minore di t+T perché potrebbe essere una rotta valida verso b, in caso contrario la ricorsione corrente terminerà.

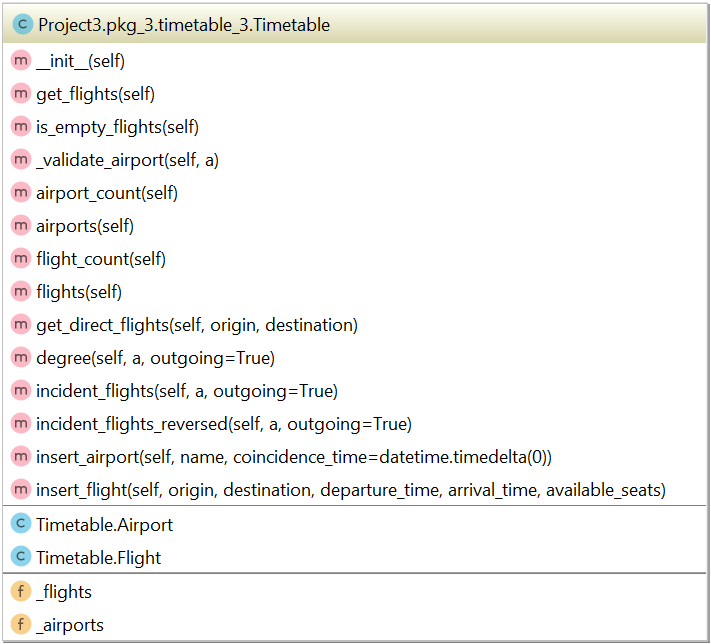
Inoltre la ricorsione non verrà lanciata verso un altro aeroporto in direzione di b, se l’orario di arrivo dell’ultimo volo scansionato, sommato al tempo di coincidenza dell’aeroporto non permette di prendere il volo che si sta scansionando.

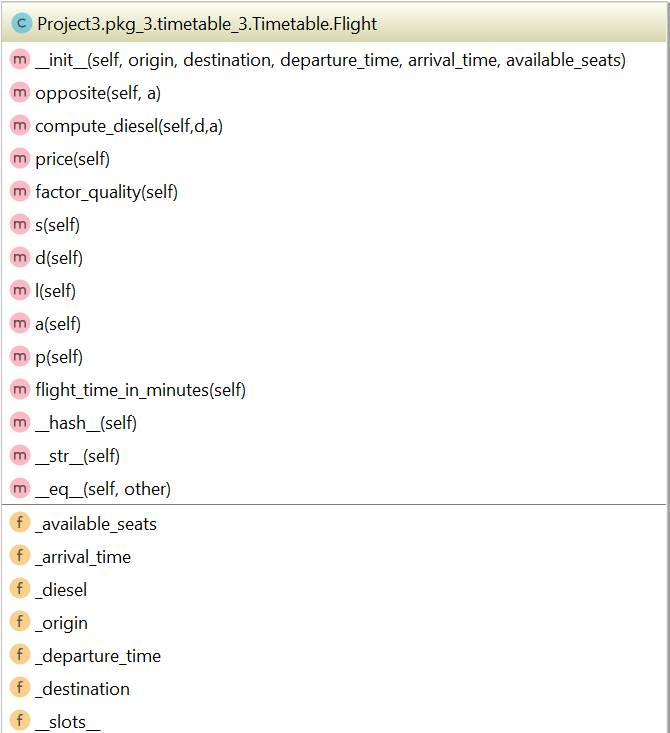
Nel caso si arrivi all’aeroporto di destinazione, viene controllato se il primo volo della sequenza e l’ultimo aggiunto, che arriva in b ha un tempo di percorrenza pari a T. Nel caso in cui questa condizione si verifichi, il percorso da a verso b ottenuto sarà valido e verrà generato dalla funzione.

**Esercizio 3-Timetable modifiche apportate**

Rispetto alle timetable viste finora viene implementato un ipergrafo i cui attributi, outcoming e incoming, sono due liste in cui verranno inseriti tutti gli archi dei nodi del grafo, in ordine decresente e in base al numero di posti. Inoltre, è stato aggiunto alla classe flights un campo self.\_diesel, che rappresenta il carburante necessario per far decollare quel volo.

Nel seguito sono riportati i diagrammi UML delle classi in cui sono state apportate le modifche:

**Uml timetable**

**Uml Flights**

**select\_flights(timetable,B)**

:param: timetable : timetable ipergrafo in modificato

B: int Budget in input

:return: ritorna None nel caso in cui non è possibile selezionare dei voli, altrimenti ritorna una tupla contenente la sequenza di voli, per la quale si massimizza il numero di posti complessivo e una lista dei budget associati a ogni aeroporto da cui partono i voli della sequenza.

Complessità: O(m\*B)

Viene fornita una soluzione di programmazione dinamica per poter calcolare la sequenza di voli per la quale si massimizza il numero di posti, rientrando nel budget B in input.

Questa soluzione viene implementata attraverso il meteodo knapsackRec(), ricorsivamente calcola il massimo valore del numero di posti per l’i-esima combinazione e lo inserisce nella posizione i-esima della matrice “DP” di grandezza m\*B, in modo tale da ottenere alla fine della ricorsione in posizione DP[n][B] il massimo numero di posti ottenibile da una sottosequenza dell’insieme di voli in input.

Per poter tenere traccia della sequenza, dove è massimo il numero di posti, si memorizza il massimo tra i massimi valori ricorsivamente calcolati in modo tale da capire quando il massimo di cui si tiene traccia cambia rispetto al massimo calcolato nella ricorsione e cosi facendo si ottiene un volo della sequenza ottimale, che infine massimizza il numero di posti complessivo.

Oltre a tenere traccia della sequenza di voli si deve mantenere memorizzato anche il budget da assegnare all’aeroporto dal quale parte il volo della sequenza ottimale.

**Bipartite(G)**

:param: G : G grafo

:return: None se il grafo non è bipartito e le due partizioni X e Y se il grafo è bipartito

Complessità: O(n+m)

Questa funzione si basa sull’algoritmo BFS che và a scorrere tutti i vertici di un grafo, in particolare prende un vertice di partenza e lo colora di rosso controlla i vertici ad esso collegati e li colora del coloro opposto in questo caso blu. Nel caso in cui due vertici adiacenti sono stati colorati dello stesso colore il grafo non è bipartito e la funzione ritorna None , se questo non avviene il grafo è bipartito e la funzione ritorna la partizione (X,Y) tale che tutti gli archi del grafo collegano un vertice di X a un vertice di Y.