# Gruppo 20

# progetto 3

## Esercizio 1

## Esercizio 2

La funzione *find\_route()* prende in input l’orario della compagnia, gli aeroporti *a* e *b*, ed un orario di partenza *t*, e trova la rotta che permette di arrivare da *a* a *b* nel minor tempo possibile, partendo ad un orario non precedente a t.

L’algoritmo consiste in una versione modificata dell’algoritmo di Dijkstra in cui il peso di un arco è dato dal costo in termini di tempo della scelta di quel volo. In particolare, il peso è dato dal tempo di coincidenza dell’aeroporto sommato alla durata del volo e al tempo di attesa di quest’ultimo.

Le differenze principali introdotte rispetto all’algoritmo di Dijkstra sono:

* La fase di inizializzazione non prevede l’inserimento di tutti gli aeroporti nella coda ma solo quelli di cui viene scoperta l’esistenza tramite i voli analizzati.
* Si è reso necessario memorizzare per ogni aeroporto il tempo di arrivo per poter calcolare il costo di un volo.
* Per ogni aeroporto incontrato viene inoltre memorizzato il volo che ho preso per arrivarci in modo da semplificare la fase di ricostruzione della soluzione.

### Complessità computazionale

La complessità coincide con quella dell’algoritmo di Dijkstra nel caso di rappresentazione mediante matrici di adiacenza.

O(|E|\*log(|V|)

## Esercizio 3

## Esercizio 4

Data la seguente dimostrazione:

Date le tre affermazioni

1. G e bipartito;
2. G e 2-colorabile e
3. G non contiene cicli di lunghezza dispari,

dimostriamo (i) ⇒ (ii), (ii) ⇒ (iii), (iii) ⇒ (i). Questo dimostra che (i) ⇔ (ii) e (i) ⇔ (iii).

1. Se G e bipartito, ` e 2-colorabile. Semplicemente, diamo colore ` 1 a tutti i nodi in una partizione, diamo colore 2 a tutti i nodi nell’altra. Non essendoci archi fra i nodi di una partizione, la colorazione è valida.

2. Se G e 2-colorabile, non contiene cicli di lunghezza dispari. Supponiamo per assurdo che esista un ` ciclo (v1, v2),(v2, v3). . . ,(vk−1, vk),(vk, v1), con k dispari. Se il nodo v1 ha colore 1, il nodo v2 deve avere colore 2; il nodo v3 deve avere colore 1, e così via fino al nodo vk, che deve avere colore 1. Poiché` v1 e successore di ` vk, v1 deve avere colore 2, assurdo.

3. Se non esistono cicli di lunghezza dispari, il grafo e bipartito. Dimostriamo questa affermazione ` costruttivamente. Si prenda un nodo x lo si assegna alla partizione S1. Si prendono poi tutti i nodi adiacenti a nodi in S1 e li si assegna alla partizione S2. Si prendono tutti i nodi adiacenti a nodi in S2 e li si assegna alla partizione S1. Questo processo termina quando tutti i nodi appartengono ad una o all’altra partizione. Un nodo può essere assegnato più di una volta se e solo se fa parte di un ciclo. Ma ` affinchè venga assegnato a due colori diversi, deve far parte di un ciclo di lunghezza dispari, e questo non è possibile.

Utilizziamo l’implicazione ii => i per trovare soluzione all’esercizio.

In particolare, viene utilizzata una versione modificata della BFS per verificare la 2-colorabilità del grafo. L’algoritmo utilizzato è il seguente:  
1. Assegna colore ROSSO (o “0”) al vertice sorgente (inserendolo nella partizione X).   
2. Colora tutti i vicini con colore BLU (o “1”)(inserendoli nella partizione Y).   
3. Colora il vicino di tutti i vicini con il colore ROSSO (inserendoli nella partizione X).   
4. Procedendo in questo modo, assegnare il colore a tutti i vertici in modo che vengano soddisfatti tutti i vincoli del problema di colorazione a m dove m = 2.  
5. Assegnando i colori, se troviamo un vicino che è colorato con lo stesso colore del vertice corrente, allora il grafo non può essere 2-colorato e di conseguenza non è bipartito.

### Complessità computazionale

La complessità computazionale rispecchia quella della BFS con grafo implementato mediante liste di adiacenza.

O(|V| + |E|)