



UNIVERSITÀ DEGLI STUDI DI NAPOLI
FEDERICO II

Facoltà Di Scienze MM.FF.N

Corso Di Laurea In Informatica

2019/2020

Progettazione di un'applicativo software:
Gestione Prenotazione Voli

Gruppo composto:

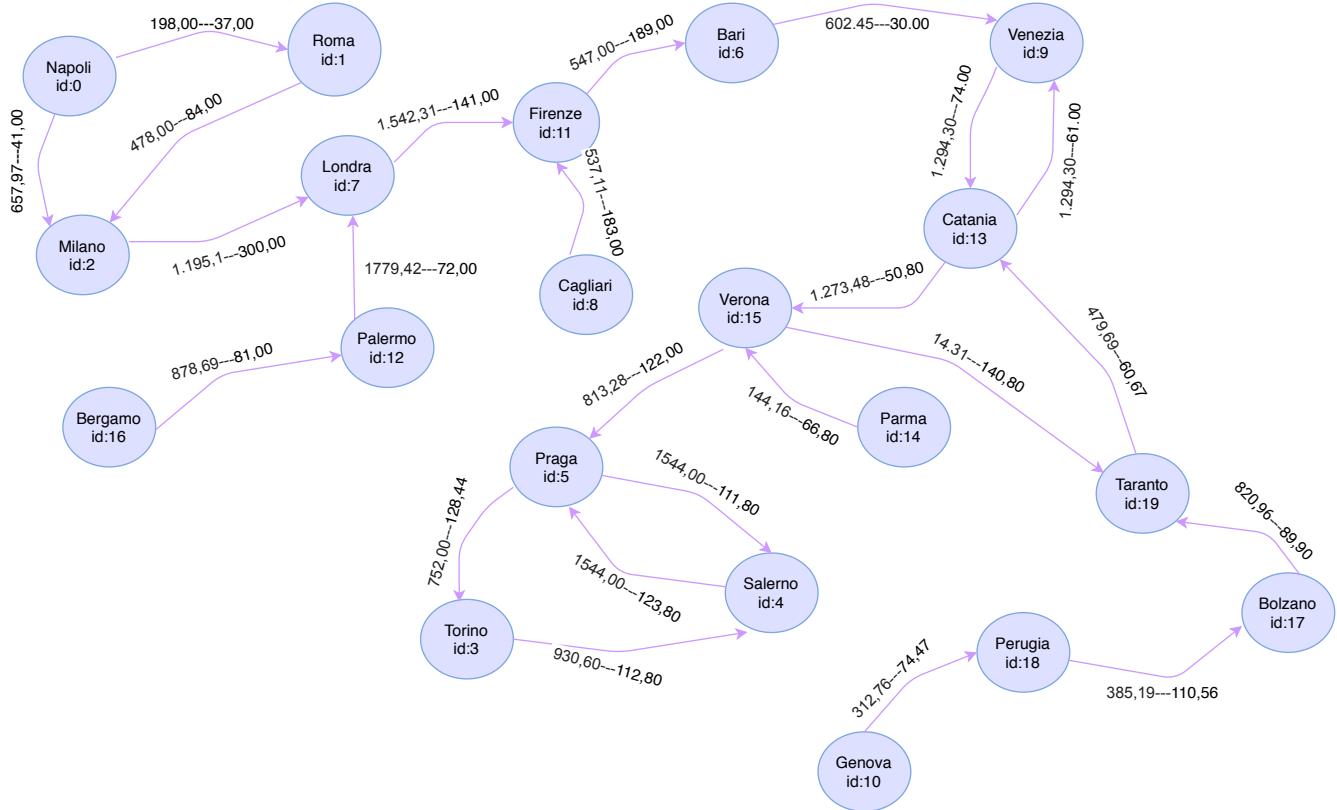
Antonio Petrillo → N86002818

Nunzia Trancredi → N86002947

Salomone Fabiola → N886002870

• Disegno grafo

Per l'implementazione del progetto abbiamo considerato tale rappresentazione grafica del grafo:



Dove:

1. Ogni vertice del grafo rappresenta la città identificata mediante un codice id;
2. Le città sono collegate tramite archi;
3. Ogni arco è costituito da due valori il primo indica la distanza tra due città ed il secondo valore indica il costo.

• Suddivisione del lavoro tra i componenti del gruppo

Il lavoro è stato suddiviso nel seguente modo:

1. Salomone Fabiola si è dedicata dall'admin ed ha realizzato anche la funzione per determinare la metà più popolare.
2. Antonio Petrillo si è dedicato alla creazione delle funzioni per determinare la tratta più economica ed la tratta più breve considerando la città di partenza e la città di destinazione che il sistema gli propone ed la questione relativa agli sconti/punti.
3. Nancy Tancredi si è dedicata alla parte utente ed inoltre ha effettuato anche all'aggiunta degli archi nella parte dell'amministratore

Ciò però è una suddivisione del lavoro parziale ma non totale perché tutti abbiamo lavorato su tutto migliorando le varie parti implementate man mano.

- **Motivazioni per le scelte implementative**

L'obiettivo del progetto è quello di realizzare un applicativo software impiegato per la gestione delle prenotazioni di voli. Abbiamo creato un menu principale ,chiamato “LoginMenu” , in cui viene data la possibilità di accedere come:

1. Admin→Accesso da amministratore

2. User→Accesso da utente

3. Exit→Uscita

Scegliendo l'opzione **Admin**, e possibile inserire l' **email** e la **password**, con le credenziali indicate nella funzione “`populateListWithAdmin`” ed osserviamo che se l'admin è presente all'interno della funzione per popolare la lista degli Admin allora gli è permesso di accedere al “`AdminMenu`” ed effettuare, a sua scelta, le seguenti operazioni.

0. Logout→permette di ritornare al menu “`LoginMenu`”

1. Print destinations→stampa destinazioni

2. Insert destinations→inserisce destinazioni

3. Insert new edge/new connection between two city→ collega città

4. Delete destinations→elimina destinazioni

In caso contrario, cioè quando l'Admin non è presente nella funzione “`populateListWithAdmin`” ciò da origine ad un messaggio di errore “login failed!”.

Scegliendo l'opzione User del menù “`LoginMenu`” , e possibile effettuare tre opzioni mediante un nuovo menu chiamato “`UserMenuLogin`”:

1. Quit→Permette di ritornare al menu “`LoginMenu`”

2. SingUp→Effettuo iscrizione inserendo email e password

3. SingIn→Effettuo accesso inserendo email e password

L' implementazione di tale menù la si è realizzata perchè solo l'utente che è in possesso delle credenziali (**email** e **password**) può effettuare una prenotazione. Mentre colui che non è in possesso delle credenziali non può effettuare nessuna operazione relativa alla prenotazione. L'iscrizione e l'accesso da parte dell'User e dell'Admin così come anche la questione relativa alle prenotazioni è stato implementato tramite una **linked list** in quanto il progetto necessita di operazioni semplici ed siccome in una lista semplice le operazioni vengono effettuato mediante un tempo costante **O(n)** abbiamo pensato che la linked list tra le varie struttura

dati ,esaminate durante il corso, risulta la più semplice ed la più adatta a risolvere operazioni di inserimento, stampa.. In particolare abbiamo implementato una funzione “user_exist” che controlla se l’utente già registrato è presente all’interno della Data structure allora viene stampato un messaggio di benvenuto con nome e cognome dell’utente ed quest’ultimo accede liberamente al menu che gli permetteva di effettuare una prenotazione, ma se l’utente sbaglia a inserire email o password oppure entrambe allora l’accesso non va a buon fine si interrompe il ramo dello switch case relativo all’accesso dell’utente e viene stampato un messaggio di errore “Unregistered user”. Una volta che l’utente effettua l’accesso con successo accede ad un’ulteriore menu, chiamato “[UserMenu](#)” che gli consente di effettuare una prenotazione in base a tale scelte:

0. LogOut

- 1. Booking based on the cheapest route by entering the city and the departure destination**
- 2. Booking based on the shortest routes by entering the city of departure and destination**
- 3. Booking based on the fewer stopover by entering the city of departure and destination**
- 4. Booking based on the most popular destination by entering only the city of departure**
- 5. View prenotations**
- 6. View City list**
- 7. View entire graph**

Per le implementazioni delle scelte del menu “[AdminMenu](#)” abbiamo utilizzato come struttura dati i grafi con liste di adiacenza. in quanto è una struttura dati che non necessita di relazioni gerarchiche, a differenza degli alberi, tra i nodi ed non viene imposta alcuna restrizione sul numero di vertici o sul numero di connessioni che un vertice può avere verso altri vertici, tale scelta è dovuta anche al fatto che i grafi sono delle strutture dati molto versatili che possono rappresentare un vasto numero di diverse situazioni.

Inizialmente abbiamo inizializzato una matrice contenente le 20 città ed le relative tratte ed le abbiamo memorizzate all’interno di un grafo, immaginando che le città sono i nodi, in tal caso abbiamo utilizzato anche la seguente struttura dati: **Queue**, utilizzata per contenere valori interi che rappresentano i vertici del grafo mentre il percorso tra una città all’altra l’abbiamo immaginato come un arco. Consecutivamente abbiamo anche associato ad ogni arco un indice uno che indica il costo ed l’altro che indica la distanza. Per la prima scelta del menù “[AdminMenu](#)” abbiamo utilizzato una funzione ricorsiva di stampa, tale funzione controlla che il grafo non è vuoto dopo di che vado ad effettuare un ciclo for che va ad iterare su tutti i nodi del grafo partendo da i=0 a G->count_nodes ed associa ad ogni id una città, poi viene effettuato un altro ciclo for che itera sulla lista di adiacenza ed a ogni id associato alla città, ottenuto dal ciclo for precedente, gli siamo andati ad associare la città corrispondente con id di tale città ed anche il relativo costo e la relativa distanza in tal caso per questi due indici è stata utilizzata un’enumerazione, che ci ha permesso di capire se utilizziamo l’indice del prezzo oppure l’indice della distanza.

```

void PrintGraph(Graph G){
    assert(G!=NULL);
    for(int i=0; i<G->nodes_count; i++){
        printf("(id:%d, city: %s) ", i, G->cityNames[i]);
        for(List iter = G->adj[i]; iter !=NULL;iter = iter->next){
            printf("-> (id: %d, city: %s, costo:%lf, distanza:%lf)", iter->target, G->cityNames[iter->target], iter->peso[PRICE], iter->peso[DISTANCE]);
        }
        puts("");
    }
    return;
}

```

Per la seconda scelta abbiamo realizzato tre funzioni: "addNode" che permette di aggiungere un una città nella struttura del grafo , "getCityName" che permette permette di ottenere il nome della città.

```

void addNode(Graph G, char* newCity) {
    if (G == NULL) {
        List * old=G->adj;
        char** old_matrix = G->cityNames;
        int i=0;
        G->adj = (List *)calloc(G->nodes_count+1, sizeof(List));
        assert(G->adj != NULL);
        G->cityNames = (char**)calloc(G->nodes_count+1, sizeof(char**));
        assert(G->cityNames != NULL);
        for(i=0;i<G->nodes_count;i++)
            G->adj[i]=old[i];
        G->cityNames[i] = old_matrix[i];
    }
    G->nodes_count += 1;
    G->adj[G->nodes_count-1] = NULL;
    G->cityNames[G->nodes_count-1] = (char*) calloc(strlen(newCity)+1, sizeof(char));
    assert(G->cityNames[G->nodes_count-1] != NULL);
    strcpy(G->cityNames[G->nodes_count-1], newCity);
}

```

```

char* getCityName(char* message){
    char buffer[BUFFER_SIZE];
    char* new_string;
    printf("Insert the %s:", message);
    scanf("%s", buffer);
    new_string = (char*) calloc( strlen(buffer)+1, sizeof(char));
    assert(new_string != NULL);
    strcpy(new_string, buffer);
    return new_string;
}

```

Per la terza scelta abbiamo utilizzato quattro funzioni:

```

void removeNodeByIndex(Graph G, int node_to_remove) {
    if (G == NULL) {
        if(node_to_remove >= G->nodes_count || node_to_remove < 0){
            return;
        }
        int i = 0;
        int x = 0;
        List *tmp = G->adj;
        char** old_matrix = G->cityNames;
        G->adj = (List *)realloc(G->adj, G->nodes_count-1, sizeof(List));
        assert(G->adj != NULL);
        G->cityNames = (char**)calloc(G->nodes_count-1, sizeof(char**));
        assert(G->cityNames != NULL);
        for (i = 0; i < G->nodes_count; i++) {
            if (i != node_to_remove) {
                G->adj[i] = checklistRemoval(tmp[i], node_to_remove);
                G->cityNames[x] = old_matrix[i];
                x++;
            } else {
                free(old_matrix[i]);
                G->adj[i] = freeList(tmp[i]);
            }
        }
        free(tmp);
        free(old_matrix);
        G->nodes_count -= 1;
    }
}

```

```

void removeNodeByString(Graph G, char* name){
    removeNodeByIndex(G, getCityIndexByName(G, name));
    return;
}

```

```

char* getCityName(char* message){
    char buffer[BUFFER_SIZE];
    char* new_string;
    printf("insert the %s:", message);
    scanf("%s", buffer);
    new_string = (char*) calloc( strlen(buffer)+1, sizeof(char));
    assert(new_string != NULL);
    strcpy(new_string, buffer);
    return new_string;
}

```

```

void addEdge(Graph G, int source, int target, double prezzo, double distanza) {
    assert(G != NULL);
    assert(source < G->nodes_count);
    assert(target < G->nodes_count);
    if (source != target) {
        G->adj[source] = appendNodeList(G->adj[source], target, prezzo, distanza);
    }
}

```

La prima,"removeNodeByIndex" ci ha permesso di rimuovere una città, vista come nodo dal grafo, sistemando poi gli indici e riallocando la memoria. Mentre la seconda "removeNodeByString" ci ha permesso di rimuovere una città tramite il nome ed infine la funzione "getCityName" ed infine "addEdge"che ci ha permesso di collegare città tramite un arco.

Per la quarta scelta abbiamo utilizzato una funzioni "removeNodeByIndex" che ci ha permesso di rimuovere un nodo dal grafo, sistemando gli indici e riallocando la memoria

```

void removeNodeByString(Graph G, char* name){
    removeNodeByIndex(G, getCityIndexByName(G, name));
    return;
}

```

```

void removeNodeByIndex(Graph G, int node_to_remove) {
    if (G == NULL) {
        if (node_to_remove >= G->nodes_count || node_to_remove < 0) {
            // il nodo da rimuovere non appartiene al grafo
            return;
        }
        int i = 0;
        int x = 0;
        List *tmp = G->adj;
        char** old_matrix = G->cityNames;
        G->adj = (List*)calloc(G->nodes_count-1, sizeof(List));
        assert(G->adj != NULL);
        G->cityNames = (char**)calloc(G->nodes_count-1, sizeof(char));
        assert(G->cityNames != NULL);
        for (i = 1; i < G->nodes_count; i++) {
            if (i != node_to_remove) {
                G->adj[i] = checkListRemoval(tmp[i], node_to_remove);
                G->cityNames[i] = old_matrix[i];
                x++;
            } else {
                free(old_matrix[i]);
                G->adj[i] = freeList(tmp[i]);
            }
        }
        free(tmp);
        free(old_matrix);
        G->nodes_count -= 1;
    }
}

```

Per le scelte del menu utente abbiamo implementato due algoritmi che vengono analizzati di seguito:

- **BFS: Breadth-First Search**

La **BFS** nota anche come **visita in ampiezza sui grafi** è stata implementata sfruttando una semplice coda **FIFO**, ha come parametri, oltre al grafo in se, un vertice di partenza e un vertice di arrivo.

L'algoritmo inizia estraendo un vertice dalla testa della coda (inizialmente vi è solo il nodo sorgente) e successivamente procede inserendo tutti gli adiacenti che non sono stati già visitati sempre nella coda, durante questa operazione salva le informazioni su come esplora il grafo (in pratica genera il sotto grafo, o sotto albero se si vuole essere più precisi, dei percorsi da un'unica sorgente), calcola i pesi (sia in base al prezzo che la distanza), segna il nodo corrente come visitato ed infine ripete questo procedimento da capo.

L'algoritmo si interrompe nel momento in cui non vi sono più nodi da estrarre dalla coda e di seguito viene chiamata una funzione ausiliaria per generare un percorso, sotto forma di lista, tra i due vertici di input, se il vertice di destinazioni non è raggiungibile viene restituita una lista vuota (la lista restituita è equivalente, come tipo, parametri, ecc.., ad una delle liste di adiacenza del grafo, il che è anche corretto semanticamente). Il costo computazionale di questa **BFS** e' **$O(|V| + |E|)$** , quindi lineare sulla dimensione del grafo. Prima di passare alla descrizione di Dijkstra e' meglio spiegare perché è stato inserito questo algoritmo nel progetto, dato che se utilizzato su di un grafo pesato esso non calcola i percorsi minimi, anzi su di un grafo pesato generalmente la BFS standard non raccoglie alcuna informazione utile, ma immerso nel contesto/semantica di questo progetto in particolare questo algoritmo minimizza il numero di scali/ cambi di aereo per un passeggero, il che può non corrispondere né alla metà più breve, né a quella più economica, inoltre è buon modo per vedere la differenza, a fini didattici, tra questi due algoritmi qui trattati.

```

List bfs(Graph G, int source, int target) {
    assert(G != NULL); // controllo che il grafo non sia nullo
    assert(G->nodes_count > 0); // controllo che i vertici inseriti appartengano al grafo
    Queue Q = initQueue();
    int visit_path = (int*)calloc(G->nodes_count, sizeof(int));
    assert(visit_path != NULL);
    double* priceInfo = (double*)calloc(G->nodes_count, sizeof(double));
    assert(priceInfo != NULL);
    double* distanceInfo = (double*)calloc(G->nodes_count, sizeof(double));
    assert(distanceInfo != NULL);
    double* priceInfo_v = (double*)calloc(G->nodes_count, sizeof(double));
    assert(priceInfo_v != NULL);
    // calli initialize gli interi a zero, che nella enum corrisponde a WHITE
    GraphColor* graphColor = (GraphColor*)calloc(G->nodes_count, sizeof(GraphColor));
    assert(graphColor != NULL);
    queueEnqueue(Q, source);
    visited[source] = BLACK;
    // il modo di partenza e' raggiungibile da se stesso con un percorso nullo
    visitPath[source] = source;
    distanceInfo[source] = 0.0;
    priceInfo[source] = 0.0;
    int targetFound = 0;
    int v;
    while (!isQueueEmpty(Q) && !targetFound) {
        // non occorre controllare che la coda abbiamo terminato l'operazione correttamente
        // data che in questo punto del codice sicuramente non e' vuota;
        dequeque(Q, &v);
        for(List iter = G->adj[v]; iter != NULL; iter = iter->next) {
            if(visited[iter->target] == WHITE) {
                enqueue(Q, iter->target);
                emplace(visited, iter->target, BLACK);
                // se non e' stato visitato il vertice in cui termina l'arco
                visited[iter->target] = BLACK;
                // segnalo che il vertice in questione e' raggiungibile con un arco partendo dal vertice v
                visitPath[iter->target] = v;
                // calcolo il peso dell'arco
                distanceInfo[iter->target] = distanceInfo[v] + iter->peso[DISTANCE];
                priceInfo[iter->target] = priceInfo[v] + iter->peso[PRICE];
                if(iter->target == target) {
                    targetFound = 1;
                    break;
                }
            }
        }
    }
    freeQueue(Q);
    List path = NULL;
    if(targetFound == 1) {
        path = generatePath(visit_path, priceInfo, distanceInfo, source, target);
    }
    free(visited);
    free(visitPath);
    free(distanceInfo);
    free(priceInfo);
    return path;
}

```

- DIJKSTRA:

L'algoritmo di Dijkstra non è altro che una versione raffinata della BFS, informalmente si potrebbe dire che è più astuto, ed ora vediamo il perchè.

Di base questi due algoritmi hanno lo stesso modus operandi, partono da un nodo, esplorano gli adiacenti e li mettono in una coda, raccolgono informazioni sui pesi e ripetono il procedimento prendendo un nuovo nodo dalla coda.

La differenza sostanziale è nell'uso della coda, infatti Dijkstra invece che usare una coda FIFO usa una **coda di priorità** basata sul peso del percorso con il quale si raggiunge un nodo. Osserviamo che vi è un'ulteriore differenza tra i 2 algoritmi, un semplice controllo “**IF**” dove nel caso in cui viene trovato un percorso “più’ leggero” per raggiungere un vertice aggiorna appunto i pesi ed i rami dell'albero dei percorsi minimi.

Utilizzando la coda di priorità si ha la certezza che il vertice che viene estratto è l'ultimo nodo del percorso minimo fino ad ora conosciuto nel grafo e dato che questa proprietà viene conservata come invariate ad ogni iterazione, la banale conseguenza di ciò è che l'algoritmo calcola i percorsi minimi anche in un grafo pesato.

Va specificato che questo algoritmo funziona solo su archi di peso non negativo, infatti nel caso in cui gli archi possono avere un peso negativo l'ordine della coda di priorità non sarebbe corretto e di conseguenza I percorsi non sarebbero minimi.

Il costo computazionale, una volta noto quello della BFS e' piuttosto scontato, infatti essendo praticamente identico al primo algoritmo illustrato si ha che il **costo** e' un $O(|V| + |E|)$ più' un ulteriore overhead ricavato dalla gestione della coda di priorità. Se prima sono state spese un paio di parole sul perche' della BFS ora e' opportuna parlare un po' dell'overhead della coda di priorità. Infatti una coda del genere può essere implementata in vari modi differenti, fra i più efficienti vi sono le strutture ad **heap**, per il quale il costo di inserimento e' **$O(\log V)$** e l'estrazione e' **$O(1)$** , con alcuni accorgimenti è possibile usare anche un'albero binario di ricerca bilanciato, come possono essere gli **AVL** o i **RedBlack**, basta conservare un puntatore al minimo nell'albero e ottenere anche un costo per l'estrazione ed inserimento di **$O(\log V)$** .

L'implementazione più' semplice e' con un lista o un vettore, solo che in un caso del genere i costi di gestione salgono a $O(V)$. Quindi considerando queste informazione possiamo dire che il **costo finale per l'algoritmo di Dijkstra** può' essere:

- $O(|V \log V| + |E|)$ con un miHeap o un albero bilanciato
- $O(|V^2| + |E|)$ con liste e array .

In questa implementazione si è preferito usare gli array per semplicità, infatti implementare un heap in C può essere complicato dato che non è conveniente usare la funzione buildheap ed heapify viste durante il corso dato che ad ogni inserimento occorre costruire l'heap da zero e ciò ha un costo di $O(V)$ senza ottenere alcun vantaggio rispetto ad un implementazione con gli array.

Potevamo utilizzare un albero binario bilanciato, ma per lasciare il codice un po' più' asciutto si è preferito evitare.

Un ultima osservazione su Dijkstra e' che stato implementato in modo tale che può essere utilizzato sia per calcolare il percorso minimo considerando come peso la distanza o il prezzo, che rispettivamente dà informazioni sulla meta più vicina e la meta più economica.

```

List dijkstra(Graph G, int source, int target, weightSelector selector){
    assert(G != NULL);
    int* visit_path = (int*) calloc(G->nodes_count, sizeof(int));
    assert(visit_path != NULL);
    double* primaryWeight = (double*) calloc(G->nodes_count, sizeof(double));
    assert(primaryWeight != NULL);
    double* secondaryWeight = (double*) calloc(G->nodes_count, sizeof(double));
    assert(secondaryWeight != NULL);
    assert(selector == secondarySelector = (selector == PRICE) ? DISTANCE : PRICE);
    List path = NULL;
    GraphColor* visited = (GraphColor*) calloc(G->nodes_count, sizeof(GraphColor));
    assert(visited != NULL);
    for(int i=0; i<G->nodes_count; i++){
        primaryWeight[i] = (source == i) ? 0.0 : INFINITY;
        secondaryWeight[i] = (source == i) ? 0.0 : INFINITY;
    }
    int targetFound = 0;
    int currentVertex = -1;
    while(1){
        // estrarremo il vertice di distanza minima
        currentVertex = findMinIndexInVect(primaryWeight, visited, G->nodes_count);
        // esploriamo gli adiacenti del vertice estratto
        // printf("minvertex %d\n", currentVertex);
        if (currentVertex == -1){
            // in questo caso non ci sono piu' percorsi da esplorare, possiamo uscire dal while
            break;
        }
        visited[currentVertex] = BLACK;
        for(List iter = G->adj[currentVertex]; iter != NULL; iter = iter->next){
            // se il vertice in cui si arriva attraversando l'arco ha un peso superiore a quello attualmente calcolato si aggiornano le informazioni
            if(iter->target == target){
                // in questo caso si e' certi che esiste almeno un percorso
                targetFound = 1;
            }
            double tmp = primaryWeight[currentVertex] + iter->peso[selector];
            if(primaryWeight[iter->target] > tmp){
                primaryWeight[iter->target] = tmp;
                visit_path[iter->target] = currentVertex;
                secondaryWeight[iter->target] = secondaryWeight[currentVertex] + iter->peso[secondarySelector];
            }
        }
    }
    if(targetFound == 1){
        if(selector == BY_PRICE){
            path = generatePath(visit_path, primaryWeight, secondaryWeight, source, target);
        }else{
            path = generatePath(visit_path, secondaryWeight, primaryWeight, source, target);
        }
    }
    free(visit_path);
    free(primaryWeight);
    free(secondaryWeight);
    free(visited);
    return path;
}

```

• Mete più gettonate

Per la realizzazione della metà più gettonata abbiamo pensato di utilizzare la strategia usata dai network , grafi, dove la pagina più visitata viene scelta percorrendo in modo casuale il grafo ed il nodo in cui ci si ferma più spesso è il più visitato. Quindi abbiamo utilizzato una semplice funzione detta **randomize** implementata nella libreria standard del linguaggio C ovvero la <**stdlib.h**>

```

mostPopularDestination = random(g->nodes_count);
path = dijkstra(g, getCityIndexByName(g, cityPartenza), mostPopularDestination, BY_PRICE);
if(path != NULL){ // la città' destinazione e' raggiungibile

```

Restituisce in modo del tutto casuale una destinazione contenuta nel grafo inizializzato all'inizio del progetto. Inizialmente avevamo provato ad cercare la destinazione, più gettonata , in modo casuale all'interno della matrice che conteneva le destinazioni ma nel momento in cui l'amministratore aggiungeva una nuova destinazione questa non veniva presa in considerazione per cui abbiamo adottato il metodo di cercare in modo casuale la città più popolare contenuta all'interno del grafo.

Queste, finora trattate sono le funzioni principali, non che le altre non siano importanti ma considerando il vincolo del realizzare una documentazione racchiusa in sei pagine abbiamo fatto una accurata selezione delle funzioni da analizzare. Ma di sicuro un'altra situazione da analizzare è il problema che si rifà alla situazione in cui l'utente ogni qualvolta effettua un acquisto accumula dei punti, proporzionalmente al costo della tratta prenotata. Accumulare punti consente di maturare degli sconti per la prenotazione, quindi l'acquisto, dei viaggi successivi. Tale situazione è stata implementata facendo in modo che ad ogni prenotazione si ottiene un punteggio in base ai soldi spesi, ovviamente questi punti corrispondono all'utente che ha fatto un pagamento, quindi si è aggiunto un campo **"discountPoint"** alla lista utente. Nel momento in cui si fa una prenotazione dopo aver stampato la tratta, con il costo ,distanza viene chiesto all'utente se desidera usare i suoi punti e in base a tale scelta vengono sottratti al conto del cliente inoltre se l'utente fa un acquisto ed decide di applicare uno sconto su quel determinato biglietto, che sta acquistando, ovviamente gli sconti associati all'utente

diminuiscono ed infine va osservato che quando l'utente effettua un acquisto usufruendo degli sconti che ha acquisito non gli vengono concessi ulteriori sconti.

- **Esempio di esecuzione**

```
*****  
Welcome to flight booking management  
*****  
Who are you?  
0. Admin  
1. User  
2. Exit  
Insert your choice:
```



Menu principale: ci permette di accedere come **Admin**, come **User** oppure **Exit**.

```
*****  
Welcome to flight booking management  
*****  
Who are you?  
0. Admin  
1. User  
2. Exit  
Insert your choice:  
Thanks for choosing us, come back to visit us!  
Program ended with exit code: 0
```



Se si seleziona nel menu principale la scelta **Exit** si esce definitivamente dal programma.

```
*****  
Welcome to flight booking management  
*****  
Who are you?  
0. Admin  
1. User  
2. Exit  
Insert your choice:0  
Insert e-mail: email_stranieri  
Insert password(max 7 characters): password  
admin access Silvia Stranieri carried out successfully  
*****  
ADMINISTRATOR  
*****  
0. Logout  
1. Print destinations  
2. Insert destinations  
3. Insert new edge/ new connection between two city  
4. Delete destinations
```



Se accedo come **Admin** devo inserire le credenziali indicate nella funzione **“populateListWithAdmin”**. Se l'accesso è valido accedo all'AdminMunù.

```
0. Admin  
1. User  
2. Exit  
Insert your choice:0  
Insert e-mail: email_stranieri  
Insert password(max 7 characters): password  
admin access Silvia Stranieri carried out successfully  
*****  
ADMINISTRATOR  
*****  
0. Logout  
1. Print destinations  
2. Insert destinations  
3. Insert new edge/ new connection between two city  
4. Delete destinations  
Insert your choice:  
*****  
Welcome to flight booking management  
*****  
Who are you?  
0. Admin  
1. User  
2. Exit  
Insert your choice:
```



Se seleziono la scelta di **Logout(0)** ritorno al **menù principale**.

```
*****
ADMINISTATOR
*****
0. Logout
1. Print destinations
2. Insert destinations
3. Insert new edge/ new connection between two city
4. Delete destinations
Insert your choice:1

The destinations are as follows:
[id:0, city: Napoli] --> (id: 2, city: Milano, costo:41.000000, distanza:657.000000)--> (id: 1, city: Roma, costo:37.000000, distanza:198.000000)
[id:1, city: Roma] --> (id: 2, city: Milano, costo:84.000000, distanza:478.000000)
[id:2, city: Milano] --> (id: 7, city: Londra, costo:300.500000, distanza:1195.100000)
[id:3, city: Torino] --> (id: 4, city: Salerno, costo:112.000000, distanza:938.000000)
[id:4, city: Salerno] --> (id: 5, city: Praga, costo:123.000000, distanza:1544.000000)
[id:5, city: Praga] --> (id: 4, city: Salerno, costo:111.000000, distanza:1554.000000)--> (id: 3, city: Torino, costo:123.000000, distanza:1554.000000)
[id:6, city: Barl] --> (id: 9, city: Venezia, costo:38.000000, distanza:162.450000)
[id:7, city: Londra] --> (id: 11, city: Firenze, costo:1842.310000, distanza:151.000000)
[id:8, city: Cagliari] --> (id: 1, city: Firenze, costo:537.110000, distanza:183.000000)
[id:9, city: Venezia] --> (id: 13, city: Catania, costo:74.000000, distanza:1294.300000)
[id:10, city: Genova] --> (id: 11, city: Firenze, costo:111.000000, distanza:1180.000000)
[id:11, city: Firenze] --> (id: 6, city: Barl, costo:189.000000, distanza:547.000000)
[id:12, city: Palermo] --> (id: 7, city: Londra, costo:72.000000, distanza:1779.420000)
[id:13, city: Catania] --> (id: 9, city: Venezia, costo:122.000000, distanza:1294.300000)
[id:14, city: Verona] --> (id: 5, city: Praga, costo:122.000000, distanza:938.000000)
[id:15, city: Bergamo] --> (id: 12, city: Palermo, costo:81.000000, distanza:878.590000)
[id:16, city: Bolzano] --> (id: 17, city: Taranto, costo:89.000000, distanza:820.760000)
[id:17, city: Perugia] --> (id: 18, city: Bolzano, costo:118.560000, distanza:385.100000)
[id:18, city: Taranto] --> (id: 15, city: Catania, costo:106.570000, distanza:479.390000)
```

Se seleziono la scelta **Print destination** è possibile stampare tutte le 20 città ed le relative tratta

```
*****
ADMINISTATOR
*****
0. Logout
1. Print destinations
2. Insert destinations
3. Insert new edge/ new connection between two city
4. Delete destinations
Insert your choice:2
Insert the new destination(first uppercase character):California
```

Se seleziono la scelta di **Insert destinations** è possibile inserire una destinazione in tal caso supponiamo di voler inserire come tratta: **California** con relativo **costo** ed **distanza**. Notiamo che questa è stata aggiunta con successo

```
*****
ADMINISTATOR
*****
0. Logout
1. Print destinations
2. Insert destinations
3. Insert new edge/ new connection between two city
4. Delete destinations
Insert your choice:3
Insert price of new edge(example 23.78): 234.22
Insert distance of new edge(example 345.98):1234.55
Insert the new destination(first uppercase character):California
Insert the new destination(first uppercase character):Napoli
~Operation successfully performed~
```

```
*****
ADMINISTATOR
*****
0. Logout
1. Print destinations
2. Insert destinations
3. Insert new edge/ new connection between two city
4. Delete destinations
Insert your choice:1

The destinations are as follows:
[id:0, city: Napoli] --> (id: 2, city: Milano, costo:41.000000, distanza:657.000000)--> (id: 1, city: Roma, costo:37.000000, distanza:198.000000)
[id:1, city: Roma] --> (id: 2, city: Milano, costo:84.000000, distanza:478.000000)
[id:2, city: Milano] --> (id: 7, city: Londra, costo:300.500000, distanza:1195.100000)
[id:3, city: Torino] --> (id: 4, city: Salerno, costo:112.000000, distanza:938.000000)
[id:4, city: Salerno] --> (id: 5, city: Praga, costo:123.000000, distanza:1544.000000)
[id:5, city: Praga] --> (id: 4, city: Salerno, costo:111.000000, distanza:1554.000000)--> (id: 3, city: Torino, costo:123.000000, distanza:1554.000000)
[id:6, city: Barl] --> (id: 9, city: Venezia, costo:38.000000, distanza:162.450000)
[id:7, city: Londra] --> (id: 11, city: Firenze, costo:1842.310000, distanza:151.000000)
[id:8, city: Cagliari] --> (id: 1, city: Firenze, costo:537.110000, distanza:183.000000)
[id:9, city: Venezia] --> (id: 13, city: Catania, costo:74.000000, distanza:1294.300000)
[id:10, city: Genova] --> (id: 11, city: Firenze, costo:111.000000, distanza:1180.000000)
[id:11, city: Firenze] --> (id: 6, city: Barl, costo:189.000000, distanza:547.000000)
[id:12, city: Palermo] --> (id: 7, city: Londra, costo:72.000000, distanza:1779.420000)
[id:13, city: Catania] --> (id: 9, city: Venezia, costo:122.000000, distanza:1294.300000)
[id:14, city: Verona] --> (id: 5, city: Praga, costo:66.000000, distanza:144.160000)
[id:15, city: Bergamo] --> (id: 12, city: Palermo, costo:81.000000, distanza:878.590000)
[id:16, city: Bolzano] --> (id: 17, city: Taranto, costo:89.000000, distanza:820.760000)
[id:17, city: Perugia] --> (id: 18, city: Bolzano, costo:118.560000, distanza:385.100000)
[id:18, city: Taranto] --> (id: 15, city: Catania, costo:106.570000, distanza:479.390000)
[id:20, city: California]
```

Se seleziono la scelta di **Insert new edge** è possibile collegare due città inserendo il relativo costo e la relativa distanza.

```
The destinations are as follows:
[id:0, city: Napoli] --> (id: 2, city: Milano, costo:41.000000, distanza:657.000000)--> (id: 1, city: Roma, costo:37.000000, distanza:198.000000)
[id:1, city: Roma] --> (id: 2, city: Milano, costo:84.000000, distanza:478.000000)
[id:2, city: Milano] --> (id: 7, city: Londra, costo:300.500000, distanza:1195.100000)
[id:3, city: Torino] --> (id: 4, city: Salerno, costo:112.000000, distanza:938.000000)
[id:4, city: Salerno] --> (id: 5, city: Praga, costo:123.000000, distanza:1544.000000)
[id:5, city: Praga] --> (id: 4, city: Salerno, costo:111.000000, distanza:1554.000000)--> (id: 3, city: Torino, costo:123.000000, distanza:1554.000000)
[id:6, city: Barl] --> (id: 9, city: Venezia, costo:38.000000, distanza:162.450000)
[id:7, city: Londra] --> (id: 11, city: Firenze, costo:1842.310000, distanza:151.000000)
[id:8, city: Cagliari] --> (id: 1, city: Firenze, costo:537.110000, distanza:183.000000)
[id:9, city: Venezia] --> (id: 13, city: Catania, costo:74.000000, distanza:1294.300000)
[id:10, city: Genova] --> (id: 11, city: Firenze, costo:111.000000, distanza:1180.000000)
[id:11, city: Firenze] --> (id: 6, city: Barl, costo:189.000000, distanza:547.000000)
[id:12, city: Palermo] --> (id: 7, city: Londra, costo:72.000000, distanza:1779.420000)
[id:13, city: Catania] --> (id: 9, city: Venezia, costo:122.000000, distanza:1294.300000)
[id:14, city: Verona] --> (id: 5, city: Praga, costo:66.000000, distanza:144.160000)
[id:15, city: Bergamo] --> (id: 12, city: Palermo, costo:81.000000, distanza:878.590000)
[id:16, city: Bolzano] --> (id: 17, city: Taranto, costo:89.000000, distanza:820.760000)
[id:17, city: Perugia] --> (id: 18, city: Bolzano, costo:118.560000, distanza:385.100000)
[id:18, city: Taranto] --> (id: 15, city: Catania, costo:106.570000, distanza:479.390000)
[id:20, city: California]
```

```
*****
ADMINISTATOR
*****  

0. Logout  

1. Print destinations  

2. Insert destinations  

3. Insert new edge/ new connection between two city  

4. Delete destinations  

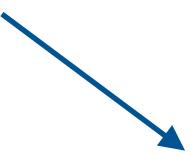
Insert your choice:4  

Insert the city to be deleted:  

Napoli  

~  

Operation successfully performed~
```



Se seleziono la scelta di **Delete destinations** è possibile inserire una destinazione da eliminare, in tal caso supponiamo di voler eliminare:Napoli. Notiamo che l'operazione è avvenuta con successo.

```
The destinations are as follows:  

[id:0, city: Roma] --> (id: 1, city: Milano, costo:84.000000, distanza:478.000000)  

[id:1, city: Milano] --> (id: 6, city: Londra, costo:300.000000, distanza:1195.100000)  

[id:2, city: Torino] --> (id: 3, city: Salerno, costo:112.000000, distanza:930.000000)  

[id:3, city: Salerno] --> (id: 4, city: Praga, costo:123.000000, distanza:1544.000000)  

[id:4, city: Praga] --> (id: 3, city: Salerno, costo:111.000000, distanza:1554.000000)--> (id: 2, city: Torino, costo:120.000000, distanza:720.000000)  

[id:5, city: Barri] --> (id: 8, city: Venezia, costo:38.000000, distanza:402.450000)  

[id:6, city: Londra] --> (id: 10, city: Firenze, costo:1542.310000, distanza:141.000000)  

[id:7, city: Cagliari] --> (id: 10, city: Firenze, costo:537.110000, distanza:183.000000)  

[id:8, city: Venezia] --> (id: 12, city: Catania, costo:74.000000, distanza:1294.300000)  

[id:9, city: Roma] --> (id: 10, city: Firenze, costo:1542.310000, distanza:141.000000)  

[id:10, city: Firenze] --> (id: 5, city: Barri, costo:150.000000, distanza:157.400000)  

[id:11, city: Palermo] --> (id: 6, city: Londra, costo:72.000000, distanza:1779.420000)  

[id:12, city: Catania] --> (id: 8, city: Venezia, costo:1.000000, distanza:1294.300000)  

[id:13, city: Parma] --> (id: 14, city: Verona, costo:16.800000, distanza:144.160000)  

[id:14, city: Verona] --> (id: 10, city: Firenze, costo:1542.310000, distanza:141.000000)  

[id:15, city: Bolzano] --> (id: 18, city: Taranto, costo:89.000000, distanza:829.760000)  

[id:16, city: Bolzano] --> (id: 16, city: Bolzano, costo:110.560000, distanza:385.190000)  

[id:17, city: Perugia] --> (id: 16, city: Bolzano, costo:68.670000, distanza:479.690000)  

[id:18, city: Taranto] --> (id: 12, city: Catania, costo:46.000000, distanza:479.690000)
```

```
*****  

Welcome to flight booking management  

*****  

Who are you?  

0. Admin  

1. User  

2. Exit  

Insert your choice:1  

*****  

USER  

*****  

0. Quit  

1. SignUp  

2. SignIn  

Insert your choice:|
```



Se si selezione la scelta **User** nel menu principale si accede al menu degli utenti in cui l'utente può effettuare l'iscrizione, può effettuare l'accesso oppure scegliere QUIT ed ritornare al menu principale.

```
*****  

USER  

*****  

0. Quit  

1. SignUp  

2. SignIn  

Insert your choice:0  

*****  

Welcome to flight booking management  

*****  

Who are you?  

0. Admin  

1. User  

2. Exit  

Insert your choice:
```



Se seleziono la scelta di **Logout(0)** ritorno al **menu principale**.

```

*****
USER
*****

0. Quit
1. SignUp
2. SignIn
Insert your choice:1

Insert name:
Sara

Insert surname:
Rossi

Insert e-mail:
saraRossi@live.it

Insert password(max 7 characters):
774

```

La scelta **SignUp** mi permette di effettuare l'iscrizione

```

tto(voli)
*****
USER
*****

0. Quit
1. SignUp
2. SignIn
Insert your choice:2

Insert e-mail:
saraRossi@live.it

Insert password(max 7 characters):
774
Welcome sara rossi

*****
USER
***** 

0. LogOut
1. Booking based on the cheapest route by entering the city and the departure destination
2. Booking based on the shortest routes by entering the city of departure and destination
3. Booking based on the fewer stopover by entering the city of departure and destination
4. Booking based on the most popular destination by entering only the city of departure
5. View prenotations
6. View city list
7. View entire graph
Insert your choice:|
All Output ◊ Filter | ⌛ | □□

```

La scelta **SingIn** mi permette di effettuare l'accesso ed di accedere al menù con le operazioni dedicate alla prenotazione.

```

*****
USER
*****

0. LogOut
1. Booking based on the cheapest route by entering the city and the departure destination
2. Booking based on the shortest routes by entering the city of departure and destination
3. Booking based on the fewer stopover by entering the city of departure and destination
4. Booking based on the most popular destination by entering only the city of departure
5. View prenotations
6. View city list
7. View entire graph
Insert your choice:1
Insert city of start: Napoli
Insert city coming: Milano
PATH: id: 0, name: Napoli --> id: 2, name: Milano --> (prezzo: 41.000000, distanza: 657.000000)

```

La **prima scelta** permette di determinare la **tratta più economica in termini di prezzo** date due città(partenza ed arrivo).

```
*****  
USER  
*****  
0. LogOut  
1. Booking based on the cheapest route by entering the city and the departure destination  
2. Booking based on the shortest routes by entering the city of departure and destination  
3. Booking based on the fewer stopover by entering the city of departure and destination  
4. Booking based on the most popular destination by entering only the city of departure  
5. View prenotations  
6. View city list  
7. View entire graph  
Insert your choice:2  
Insert city of start: Torino  
Insert city coming: Salerno  
PATH: id: 3, name: Torino --> id: 4, name: Salerno --> (prezzo: 112.800000,  
distanza: 930.600000)
```

La **seconda scelta** permette di determinare la **tratta più economica in termini di distanza** date due città(partenza ed arrivo).

```
*****  
USER  
*****  
0. LogOut  
1. Booking based on the cheapest route by entering the city and the departure destination  
2. Booking based on the shortest routes by entering the city of departure and destination  
3. Booking based on the fewer stopover by entering the city of departure and destination  
4. Booking based on the most popular destination by entering only the city of departure  
5. View prenotations  
6. View city list  
7. View entire graph  
Insert your choice:3  
Insert city of start: Torino  
Insert city coming: Salerno  
PATH: id: 3, name: Torino --> id: 4, name: Salerno --> (prezzo: 112.800000,  
distanza: 930.600000)
```

La **terza scelta** permette di determinare la **tratta che minimizza il numero di scali** date due città(partenza ed arrivo).

```
*****  
USER  
*****  
0. LogOut  
1. Booking based on the cheapest route by entering the city and the departure destination  
2. Booking based on the shortest routes by entering the city of departure and destination  
3. Booking based on the fewer stopover by entering the city of departure and destination  
4. Booking based on the most popular destination by entering only the city of departure  
5. View prenotations  
6. View city list  
7. View entire graph  
Insert your choice:4  
Insert city of start: Milano  
PATH: id: 2, name: Milano --> id: 7, name: Londra --> id: 11, name: Firenze -->  
id: 6, name: Bari --> (prezzo: 2031.810000, distanza: 1883.100000)
```

La **quarta scelta** permette di determinare la **metà più gettonata**, qualora sia possibile.

```
*****  
USER  
*****  
0. LogOut  
1. Booking based on the cheapest route by entering the city and the departure destination  
2. Booking based on the shortest routes by entering the city of departure and destination  
3. Booking based on the fewer stopover by entering the city of departure and destination  
4. Booking based on the most popular destination by entering only the city of departure  
5. View prenotations  
6. View city list  
7. View entire graph  
Insert your choice:5  
  
Booking by: sara rossi:  
prenotazione 1:: Start:Milano - Coming:Londra, Bill:300.500000  
prenotazione 2:: Start:Napoli - Coming:Milano, Bill:41.000000  
prenotazione 3:: Start:Salerno - Coming:Praga, Bill:43.800000
```

La **quinta scelta** permette all'utente di **visualizzare le proprie prenotazioni attive**.

```
*****
USER
*****
0. LogOut
1. Booking based on the cheapest route by entering the city and the departure destination
2. Booking based on the shortest routes by entering the city of departure and destination
3. Booking based on the fewer stopover by entering the city of departure and destination
4. Booking based on the most popular destination by entering only the city of departure
5. View prenotations
6. View city list
7. View entire graph
Insert your choice:6
id: 0, city name: Napoli
id: 1, city name: Roma
id: 2, city name: Milano
id: 3, city name: Torino
id: 4, city name: Salerno
id: 5, city name: Praga
id: 6, city name: Bari
id: 7, city name: Londra
id: 8, city name: Cagliari
id: 9, city name: Venezia
id: 10, city name: Genova
id: 11, city name: Firenze
id: 12, city name: Palermo
id: 13, city name: Catania
id: 14, city name: Parma
id: 15, city name: Verona
id: 16, city name: Bergamo
id: 17, city name: Bolzano
id: 18, city name: Perugia
id: 19, city name: Taranto
```

La **sesta scelta** permette all'utente di **visualizzare elenco città**.



```
*****
USER
*****
0. LogOut
1. Booking based on the cheapest route by entering the city and the departure destination
2. Booking based on the shortest routes by entering the city of departure and destination
3. Booking based on the fewer stopover by entering the city of departure and destination
4. Booking based on the most popular destination by entering only the city of departure
5. View prenotations
6. View city list
7. View entire graph
Insert your choice:7
(id: 0, city: Napoli) --> (id: 2, city: Milano, costo:41.00, distanza:657.00)
(id: 1, city: Roma) --> (id: 2, city: Milano, costo:96.00, distanza:476.00)
(id: 2, city: Milano) --> (id: 7, city: Londra, costo:396.56, distanza:1196.10)
(id: 3, city: Torino) --> (id: 4, city: Salerno, costo:112.00, distanza:1939.60)
(id: 4, city: Salerno) --> (id: 6, city: Bari, costo:111.00, distanza:119.00)
(id: 5, city: Praga) --> (id: 4, city: Salerno, costo:111.00, distanza:1564.00) --> (id: 3, city: Torino, costo:128.44, distanza:752.00)
(id: 6, city: Bari) --> (id: 9, city: Venezia, costo:98.00, distanza:482.45)
(id: 7, city: Londra) --> (id: 11, city: Firenze, costo:542.31, distanza:151.00)
(id: 8, city: Cagliari) --> (id: 15, city: Firenze, costo:537.11, distanza:183.00)
(id: 9, city: Venezia) --> (id: 13, city: Catania, costo:74.00, distanza:1294.20)
(id: 10, city: Genova) --> (id: 12, city: Catania, costo:112.00, distanza:171.00)
(id: 11, city: Firenze) --> (id: 6, city: Bari, costo:189.00, distanza:57.00)
(id: 12, city: Palermo) --> (id: 7, city: Londra, costo:72.00, distanza:1779.42)
(id: 13, city: Catania) --> (id: 14, city: Praga, costo:112.00, distanza:119.00)
(id: 14, city: Praga) --> (id: 15, city: Venezia, costo:164.00, distanza:144.10)
(id: 15, city: Venezia) --> (id: 8, city: Praga, costo:122.00, distanza:183.28)
(id: 16, city: Bergamo) --> (id: 12, city: Palermo, costo:81.00, distanza:878.49)
(id: 17, city: Bolzano) --> (id: 12, city: Palermo, costo:119.00, distanza:119.00)
(id: 18, city: Perugia) --> (id: 17, city: Bolzano, costo:119.00, distanza:188.19)
[id:19, city: Taranto] --> (id: 13, city: Catania, costo:68.67, distanza:1479.69)
```

La **settima scelta** permette all'utente di **visualizzare elenco contenente le relative tratte**.



Va osservato che per ogni operazione nel menù delle prenotazioni viene chiesto all'utente se desidera usufruire degli sconti in possesso, indicando 1, oppure se desidera non utilizzarli, indicando 0. Mostriamo alcuni esempio, ma teniamo a sottolineare che tale richiesta viene fatta per ogni tipo di scelta nel menu delle prenotazioni :

```
*****
USER
*****
0. LogOut
1. Booking based on the cheapest route by entering the city and the departure destination
2. Booking based on the shortest routes by entering the city of departure and destination
3. Booking based on the fewer stopover by entering the city of departure and destination
4. Booking based on the most popular destination by entering only the city of departure
5. View prenotations
6. View city list
7. View entire graph
Insert your choice:1
Insert city of start: Napoli
Insert city coming: Milano
PATH: id: 0, name: Napoli --> id: 2, name: Milano -->
(pieza: 41.000000, distane: 657.000000)
do you want to use your discount point? [Y:n:0]:0
```

Come possiamo vedere in basso viene mostrata la **domanda** per **usufruire** degli **sconti**



```

*****
USER
*****
0. LogOut
1. Booking based on the cheapest routes by entering the
   city and the departure destination
2. Booking based on the shortest routes by entering the
   city of departure and destination
3. Booking based on the fewer stopover by entering the
   city of departure and destination
4. Booking based on the most popular destination by
   entering only the city of departure
5. View prenotations
6. View city list
7. View entire graph
Insert your choice:2
Insert city of start: Napoli
Insert city coming: Milano
PATH: id: 0, name: Napoli --> id: 2, name: Milano -->
(pieces: 41.000000, distance: 657.000000)
do you want to use your discount point? [Y:1/n:0]:1
You have 0 point
With 5 point you can get a discount of 8 euros!
At most you can use 5 points
Insert the amount of point you want to use:
All Output ◊ Filter | □□

```

Se voglio **usufruire** dello sconto indico **1**.



```

*****
USER
*****
0. LogOut
1. Booking based on the cheapest route by entering the city
   and the departure destination
2. Booking based on the shortest routes by entering the city
   of departure and destination
3. Booking based on the fewer stopover by entering the city of
   departure and destination
4. Booking based on the most popular destination by entering
   only the city of departure
5. View prenotations
6. View city list
7. View entire graph
Insert your choice:1
Insert city of start: Milano
Insert city coming: Londra
PATH: id: 2, name: Milano --> id: 7, name: Londra --> (pieces:
300.500000, distance: 1195.100000)
do you want to use your discount point? [Y:1/n:0]:0
*****
All Output ◊ Filter | □□

```

Se voglio **non usufruire** dello sconto indico **0**.

