UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA IN INFORMATICA
INSEGNAMENTO DI LABORATORIO DI ALGORITMI E STRUTTURE DATI
ANNO ACCADEMICO 2021/2022

CREAZIONE DI UN PROGRAMMA PER LA PRENOTAZIONE DI VIAGGI IN AEREO O TRENO

Lorenzo Sepe N86003622
Antonio Todisco N86003642
Vincenzo Papale N86003599
— 25/05/2022

Questa pagina è stata lasciata intenzionalmente bianca.

Indice

Capitolo 1

Report del Progetto

1.1 Requisiti del sistema

Il sistema simula una piattaforma per la prenotazione di viaggi dove un utente, dopo essersi registrato, può accedere e prenotare un viaggio specificando la partenza e la destinazione da una lista di mete.
Inoltre il sistema offre diverse opzioni di viaggio, se viaggiare in aereo o in treno. Per ciascuna opzione di viaggio, l'utente può visualizzare la scelta più economica e la più veloce. Dopo aver effettuato l'opzione di viaggio il sistema propone una lista di alberghi disponibili nella città di destinazione, se l'utente ne selezionerà uno gli verrà mostrato il modo più veloce per arrivare dall'aeroporto , o dalla stazione se il viaggio è in treno, all'albergo.
Il programma ha anche una sezione dedicata all'amministratore. Qualora la destinazione di viaggio selezionata dall'utente non fosse raggiungibile in alcun modo, il sistema notificherà l'admin al prossimo accesso se vuole rendere raggiungibile quella meta o eliminarla definitivamente.

1.2 Strutture dati usate

Il programma usa tre diverse strutture dati: un grafo (MappaCollegamenti) usato per memorizzare i collegamenti tra città e treni, città e aerei ed alberghi e città; un albero binario di ricerca per memorizzare gli Utenti registrati (chiamato AlberoUtenti); un heap (HeapMinimo) per definire l'algoritmo di Dijkstra che cerca il percorso minimo in un grafo.
Nel resto del documento si useranno questi nomi per le strutture per semplicità di spiegazione.
Le struttura ad albero binario di ricerca per gli utenti è stata scelta per utilità e costo computazionale, visto che dobbiamo eseguire molte ricerche sugli utenti, la velocità di queste operazioni nel contesto di un ABR ($\log(numero_n odi)$) e la loro struttura ricorsiva efficiente. Per quanto riguarda la scelta della struttura dati dei grafi ci siamo attenuti da quanto richiesto nella traccia.
Si è scelto di scelto di utilizzare un Heap che ci permetta di definire l'algoritmo di Dijkstra per la ricerca della scelta più economica e più veloce.

1.2.1 Struttura MappaCollegamenti

```c
#define MAX_STRINGHE 1000
//Definizione struttura grafo
struct GrafoCollegamenti {
    int NumeroNodi;
    struct AdjList *ListaAdiacenza;
};

struct AdjList
{
    char NomeTappa[MAX_STRINGHE];
    struct Tappa *head; // Puntatore al primo elemento di lista
};

struct Tappa{
    int key; //rapresenta la  destinazione dell'arco
    char NomeTappa[MAX_STRINGHE];
    int distanza; //valori associati all'arco per differenziare gli archi
    int costo; //
    int visibilita; //se il nodo è visibile/esistente. 1 = vero, 0 = falso
    struct Tappa *next;
};
typedef struct Tappa* ArchiGrafo; // lista di collegamenti
typedef struct GrafoCollegamenti* MappaCollegamenti;

```

1.2.2 Struttura AlberoUtenti

```c
#define MAX_STRINGHE 1000

//Definizione struttura albero di ricerca per gli Utenti
struct Utente {
   char Mail[MAX_STRINGHE];
   char Password[MAX_STRINGHE];
   int Saldo;
   struct NodoListaDesideri *ListaDesideri;
   struct Utente *DX; //puntatore al sottoalbero destro
   struct Utente *SX; //puntatore al sottoalbero sinistro
};

typedef struct Utente* AlberoUtenti;

```

1.2.3 Struttura HeapMinimo

```c
// Structure to represent a min heap node
struct MinHeapNode
{
    int  v;
    int dist;
};

typedef struct MinHeapNode* NodoHeap;

// Structure to represent a min heap with an array
struct MinHeap
{

    int size;   // Number of heap nodes present currently
    int capacity;  // Capacity of min heap
    int *pos;  // This is needed for decreaseKey()
    struct MinHeapNode **array;
};

typedef struct MinHeap* HeapMinimo;

// Structure to represent a min heap node using float type
struct MinHeapNodeFloat
```

```
24 {
25     int  v;
26     float cost;
27 };
28
29 typedef struct MinHeapNodeFloat* NodoHeapFloat;
30
31 // Structure to represent a min heap with an array
32 struct MinHeapFloat
33 {
34     // Number of heap nodes present currently
35     int size;
36
37     // Capacity of min heap
38     int capacity;
39
40     // This is needed for decreaseKey()
41     int *pos;
42     NodoHeapFloat *array;
43 };
44
45 typedef struct MinHeapFloat* HeapMinimoFloat;
46
```

1.3 Funzionamento Generale

Le funzioni sono divise in multipli file .c in base al loro tipo di ritorno e le strutture che prendono come input. Le strutture che lavorano sugli alberi, grafi ed heap sono strutturate in modo ricorsivo per una maggiore economia di codice.
Alla fine dell'esecuzione: l'AlberoUtenti sarà salvato in file di testo nella cartella del codice sorgente, ottentendo così un database permanente; stessa cosa per ogni grafo, che però sarà salvato su due file di testo: un file che servirà per salvare tutti collegamenti ed un file che conterrà il numero di nodi. Se i file che contengono i dati sui file non esistono, il sistema crea automaticamente un grafo vuoto con un singolo nodo.
Essendo questo un semplice programma senza funzioni di bilanciamento dell'albero, per evitare di avere un ABR degere dopo aver memorizzato l'albero in database, l'operazione di salvataggio su file non è eseguita con una visita InOrder, ma con un PreOrder, mescolando i nodi cosicché la lettura ed inserimento ordinato di essi non producano risultati imprevisti. Durante l'esecuzione, verrà richiesta una mail e password per l'accesso (permettendo al contempo di registrare un nuovo utente) e in base all'input ci saranno due costrutti switch-case che controllano le azioni.
Si possono eseguire molteplici azioni una volta selezionata la modalità di funzionamento, ma l'esecuzione termina se si inserisce un valore fuori dal range del case-switch, e non si può cambiare utente, e quindi modalità di esecuzione, una volta eseguito il login.

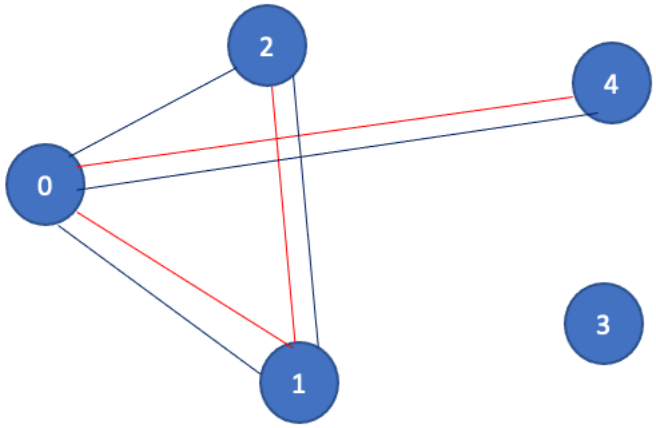
1.4 Funzioni Particolari

Per l'admin abbiamo varie funzionalità particolari quali: aggiungere o rimuovere una tappa ai viaggi in aereo; aggiungere o rimuovere un collegamento tra gli aeroporti; aggiungere o rimuovere una tappa ai viaggi in treno; aggiungere o rimuovere un collegamento tra le stazioni ferroviarie; aggiungere o rimuovere un albergo dalla lista associata ad una tappa.
Per l'utente, oltre la registrazione e login iniziale, abbiamo varie funzionalità particolari quali: selezionare il punto di partenza; selezionare la meta di destinazione; selezionare viaggio in aereo (se disponibile); selezionare viaggio in treno (se disponibile); visualizzare scelta più economica o più veloce; selezionare un albergo disponibile nella lista alberghi delle città di destinazione.
Quando una Tappa viene selezionata ma non è raggiungibile in un dato Grafo, il programma salva tale informazione in uno di due file chiamati CheckTreno e CheckAereo. Se al'inizio dell'esecuzione del programma in modalità amministratore esso legge che il file contiene un 1, l'applicativo controlla con la funzione GrafoSconnesso quale nodo non possiede archi e stampa il nome della tappa a video. Quando il programma esce da modalità amministratore, le funzioni updateCheckAereo e updateCheckTreno controllano che il grafo è connesso e pongono il bit nel file a 0 se lo è.

1.5 Rappresentazione Grafi Utilizzati

1.5.1 Grafo Collegamenti Città

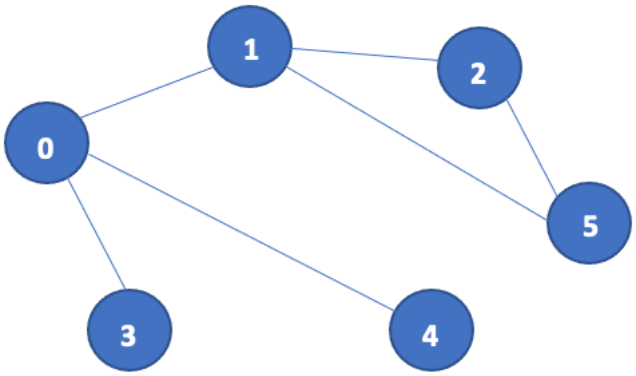
Per i collegamenti tra le varie città abbiamo utilizzato il seguente grafo, dove in rosso sono rappresentati i collegamenti in treno mentre in blu sono rappresentati i collegamenti in aereo. I nodi rappresentano le seguenti città: Milano=0 , Napoli=1, Roma=2, Venezia=3, Torino=4 e sono stati pesati usando numeri di fantasia che non rispecchiano la realtà.



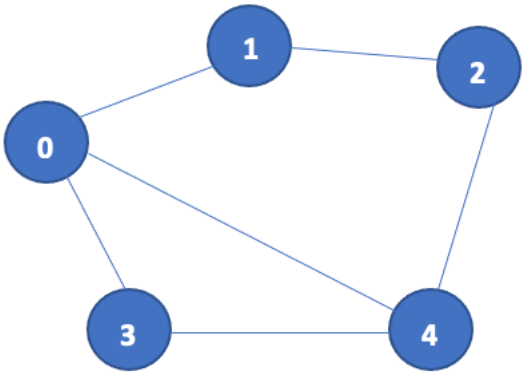
1.5.2 Grafo Collegamenti Stazione/Aeroporto-Alberghi

Per i collegamenti tra le stazioni e gli aeroporti agli alberghi, abbiamo utilizzato i seguenti grafi. Dove l'aeroporto è il nodo 0 e la stazione è il nodo 1, mentre il resto dei nodi sono gli alberghi disponibili per ogni città. Gli archi blu rappresentano i collegamenti dall'aeroporto all'albergo, mentre quelli rossi rappresentano i collegamenti dalla stazione all'albergo.

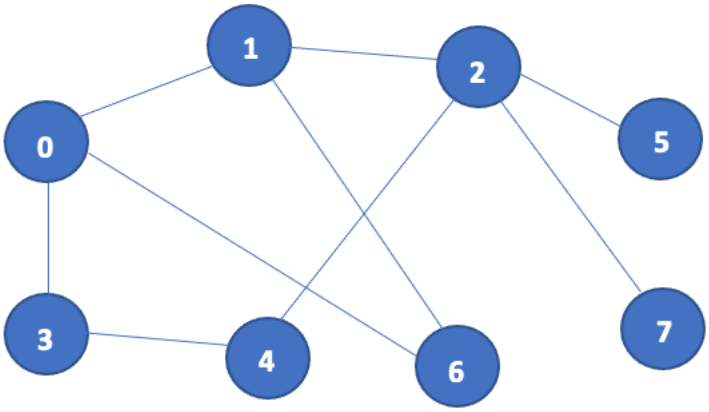
Milano:



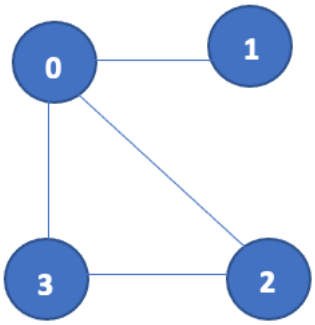
Napoli:



Roma:



Venezia:



Torino: