Università degli Studi di Napoli - Federico II

Facoltà di
Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica

Laboratorio di Algoritmi e Strutture Dati
A.A. 2005 / 2006

Prof. Aniello Murano

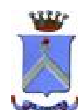
SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE

Realizzato da

Marco Sommella
50 / 482

Giovanni Di Cecca 50 / 887

Virginia Bellino 408 / 466



http://www.dicecca.net

Marco Sommella, Giovanni Di Cecca, Virginia Bellino

Indice

- **Sezione 1 – Schema Grafo**
  Schema del Programma

- **Sezione 2 – Documentazione esterna**

- **Sezione 3 – Codice Sorgente**

  1. **programma main.c**
  2. **libreria airport.h**
  3. **libreria airport.c**
  4. **libreria grafo.h**
  5. **libreria grafo.c**

- **Sezione 4 – Appendice**

  Versione Algoritmo di Dijkstra

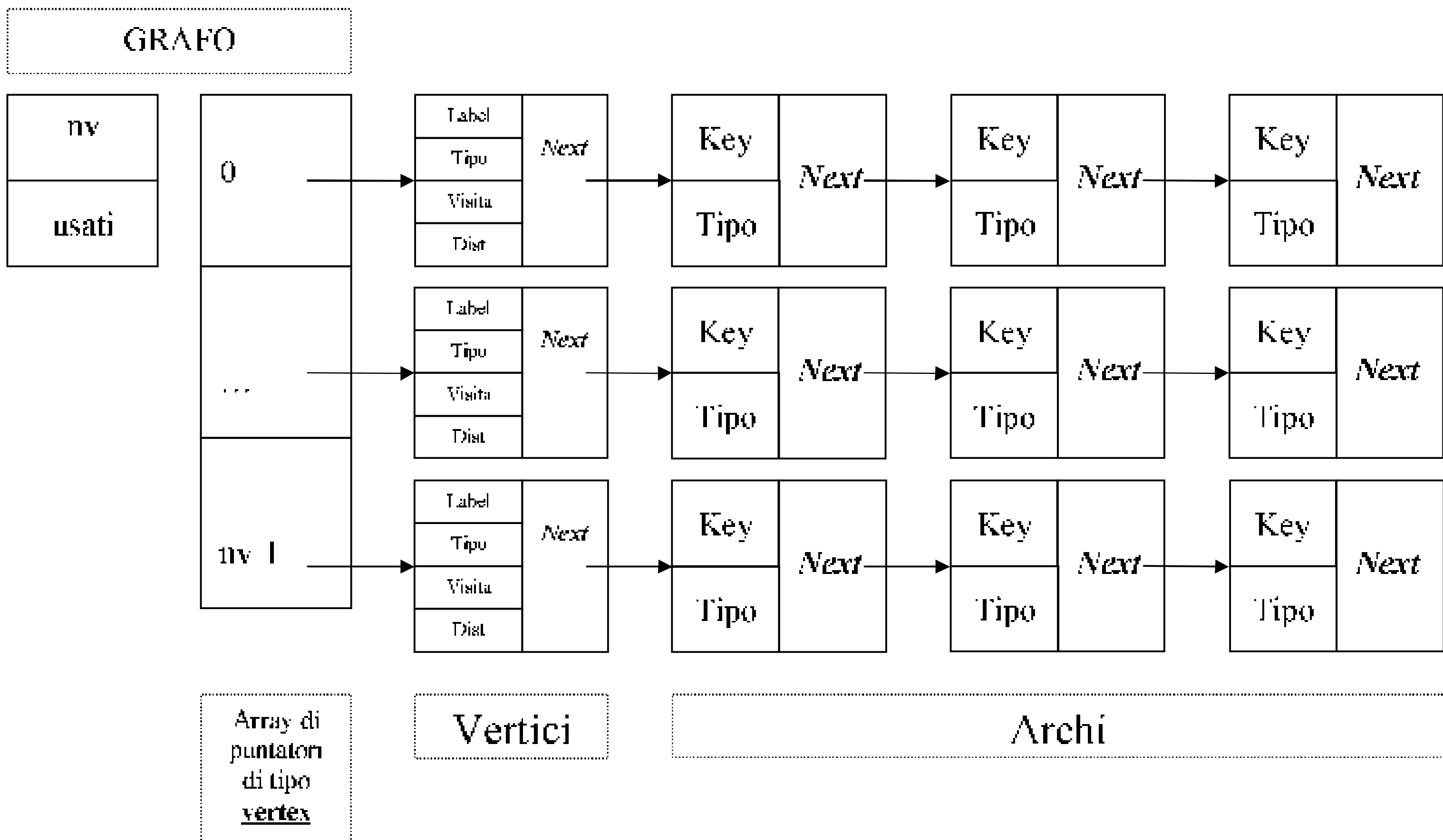
- **Sezione 5 – Codice Sorgente**

  1. **programma main.c**
  2. **libreria airport.h**
  3. **libreria airport.c**
  4. **libreria grafo.h**
  5. **libreria grafo.c**
  6. **libreria heap.h**
  7. **libreria heap.c**

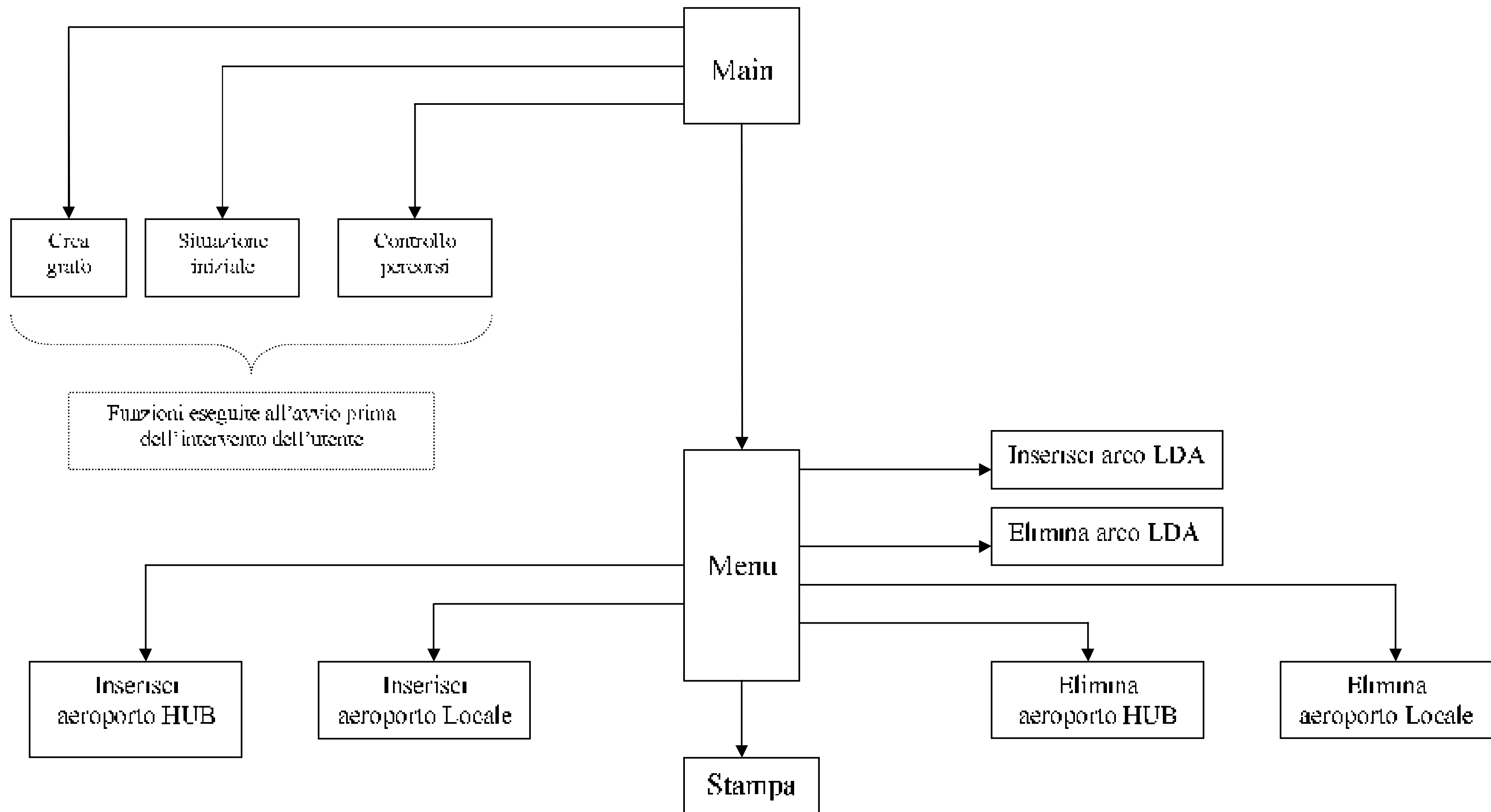
Marco Sommella, Giovanni Di Cecca, Virginia Bellino

Sezione 1
Schema grafico

Struttura del Grafo



Struttura del Programma



Ogni modifica al grafo (tipo inserimento di un vertice) comporta l'esecuzione della funzione `controllapercorsi`

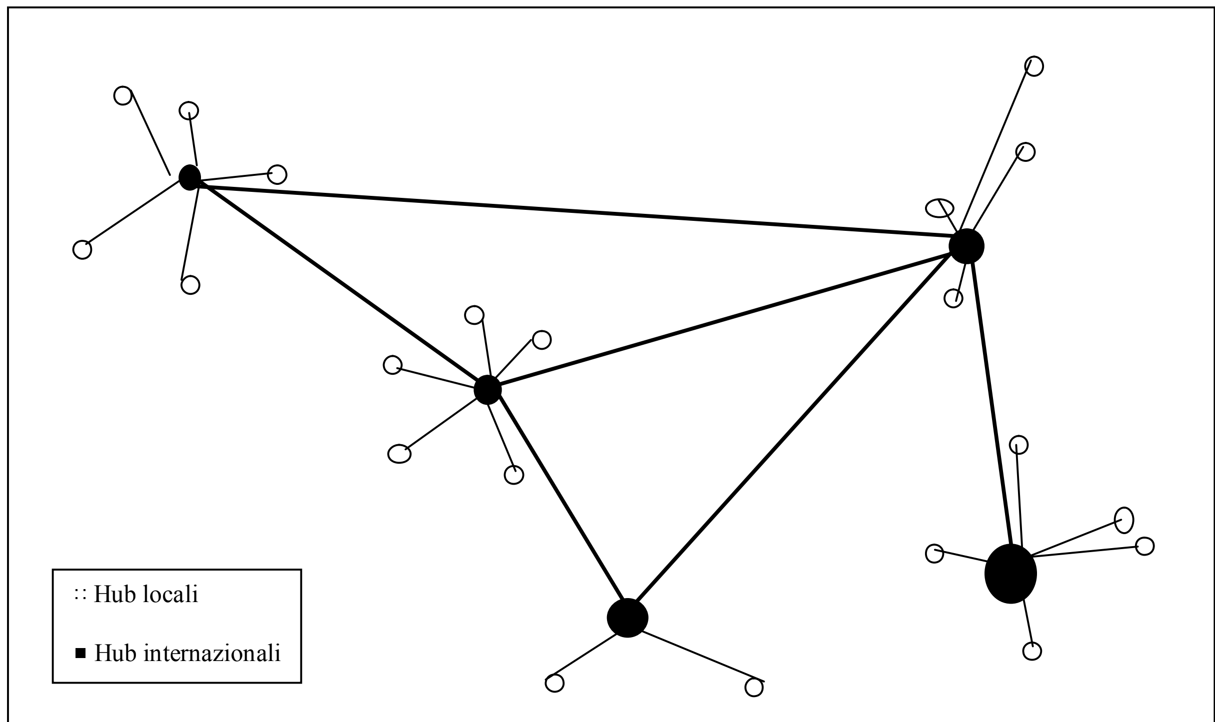
Sezione 2

Documentazione Esterna

Documentazione esterna

- **Scopo:** il progetto si propone di realizzare la simulazione di una compagnia di voli internazionali che propone di poter andare da un punto all'altro del Mondo in soli 4 passaggi, potendo creare e togliere collegamenti tra gli Hub locali e gli Hub internazionali.

  Graficamente, il problema può essere rappresentato nel seguente modo:



- **Specifiche della libreria `airport.c`:**

  graph *SituazioneIniziale (graph *G, int hub, int locali);

  void StampaHUB (graph *G);

  void StampaLocali (graph *G);

  void StampaHUBAdiacenti (graph *G, int i);

  void ControllaPercorsi (graph *G);

♦ Specifiche della libreria `grafi.c`:

graph *CreaGrafo ();

vertex *CreaVertice (int label, int tipo);

int GrafoVuoto (graph *G);

int Adiacenti (graph *G, int u, int v);

int VerticeValido (graph *G, int u);

int TrovaHUB (graph *G, int u);

graph *InserisciVertice (graph *G, vertex *vertice);

void InserisciArco (graph *G, int u, int v);

void InserisciArcoN_O (graph *G, int u, int v);

void EliminaArco (graph *G, int u, int v);

void EliminaArcoN_O (graph *G, int u, int v);

void EliminaVertice (graph *G, int u);

void StampaGrafo (graph *G);

void Ampiezza (graph *G, int start);

Descrizione:

Aspetti generali

All'avvio del programma, l'utente, si trova di fronte ad un menu di selezione che prevede la possibilità di scegliere alcune opzioni.

```
------------------------------
a: Inserisci Aereoporto HUB
b: Inserisci Aereoporto Locale
------------------------------
c: Elimina Aereoporto HUB
d: Elimina Aereoporto Locale
------------------------------
e: Inserisci Arco LDA
f: Elimina Arco LDA
------------------------------
s: Stampa Situazione
q: Quit
------------------------------
```

Le lettere `a` et `b`, consentono l'inserimento di degli Hub internazionali e locali.

Le lettere `c` et `d`, consentono l'eliminazione degli Hub internazionali e locali. Naturalmente eliminando un Hub internazionale, gli Hub locali, vengono automaticamente eliminati.

Le lettere `e` et `f`, consentono l'inserimento e l'eliminazione degli archi Long Distance. Non è stata implementata la funzione di eliminazione di archi tra aeroporti locali ed Hub internazionali, in quanto la funzione elimina aeroporto locale cancellando un aeroporto locale elimina anche l'arco con l'Hub internazionale.

Ogni modifica al grafo (tipo inserimento di un vertice) comporta l'esecuzione della funzione `controllapercorsi`

La lettera `s` Stampa a monitor la situazione della compagnia:

```
101 ->   118-[1]   110-[1]   160-[2]
```

Hub dal quale parte l'arco — Vertici di arrivo 1 = LDA, 2 = LNA

Aspetti Implementativi

Abbiamo strutturato il programma su tre livelli:
una funzione main, che stampa il menu di scelta e richiama le funzioni;
l'insieme di funzioni che gestisce l'aereoporto (airport.c), una libreria di funzioni che gestisce il grafo su cui poggia il programma che gestisce le tratte aeroportuali della compagnia

- **Indicazioni di utilizzo:**

Il programma è stato compilato utilizzando il compilatore DevC++ versione 4.9.9.2.
Segnaliamo inoltre che, essendovi chiamate alla funzione system che comprendono comandi MS-DOS ("pause" et "cls"), se il programma viene compilato in ambiente Linux, il mancato riconoscimento di tali comandi da parte del sistema operativo comporta una visualizzazione dei risultati corretta ma poco ordinata.

- **Complessità computazionale**

complessità di tempo
La Funzione di inserimento di un vertice Hub e Locale ha una complessità di $O(|V|)$

La Funzione Controllo Percorsi esegue V volte la BFS, quindi è un $O(|V|^2)$

La Funzione di Inserimento archi, è a tempo costante $O(1)$

La Funzione di Eliminazione di archi è lineare sulla grandezza della lista di adiacenza, quindi un $O(|E|)$

La Funzione di Eliminazione dei Vertici è un $O(|E|)$

complessità di spazio
in entrambe le implementazioni adottate nel progetto, la complessità di spazio è $O(|V|+|E|)$.

Esempio d'uso

Main all'avvio

```
************************************************
* SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
*             con Algoritmo BFS                *
* REALIZZATO DA:                               *
* Marco Sommella 50/482                        *
* Giovanni Di Cecca 50/887                     *
* Virginia Bellino 408/466                     *
************************************************



------------------------------
a: Inserisci Aereoporto HUB
b: Inserisci Aereoporto Locale
------------------------------
c: Elimina Aereoporto HUB
d: Elimina Aereoporto Locale
------------------------------
e: Inserisci Arco LDA
f: Elimina Arco LDA
------------------------------
s: Stampa Situazione
q: Quit
------------------------------
Scegli:
```

Inserisci Aeroporto HUB

```
Inserisci la label ( numerica ) dell'Aereoporto da inserire: 1001

Inserimento Terminato.
Utilizzare la funzione di Stampa
Premere un tasto per continuare . . .
```

Stampa Situazione

```
Sono stampati solo gli Aereoporti HUB.
Tra parentesi quadre si indica il tipo di tratta: 1 se LDA, 2 se LNA.

Ci sono 61 Aereoporti ( HUB e Locali )

  100 ->  1001-[1]   103-[1]   117-[1]   114-[1]   111-[1]   107-[1]   101-[1]
 140-[2]   120-[2]
  101 ->   115-[1]   113-[1]   112-[1]   109-[1]   108-[1]   119-[1]   102-[1]
 100-[1]   141-[2]   121-[2]
  102 ->   107-[1]   116-[1]   103-[1]   101-[1]   142-[2]   122-[2]   105-[1]
 106-[1]
  103 ->   115-[1]   112-[1]   118-[1]   108-[1]   100-[1]   113-[1]   104-[1]
 102-[1]   143-[2]   123-[2]
  104 ->   117-[1]   116-[1]   109-[1]   119-[1]   107-[1]   103-[1]   144-[2]
 124-[2]
  105 ->   115-[1]   112-[1]   110-[1]   117-[1]   109-[1]   145-[2]   125-[2]
 118-[1]   102-[1]   108-[1]
  106 ->   117-[1]   115-[1]   119-[1]   114-[1]   112-[1]   146-[2]   126-[2]
 110-[1]   102-[1]
  107 ->   118-[1]   112-[1]   102-[1]   110-[1]   108-[1]   100-[1]   104-[1]
 147-[2]   127-[2]   115-[1]
  108 ->   111-[1]   114-[1]   101-[1]   103-[1]   107-[1]   148-[2]   128-[2]
 105-[1]
  109 ->   111-[1]   101-[1]   104-[1]   114-[1]   105-[1]   149-[2]   129-[2]
  110 ->   116-[1]   105-[1]   113-[1]   107-[1]   150-[2]   130-[2]   106-[1]
  111 ->   109-[1]   108-[1]   116-[1]   112-[1]   100-[1]   151-[2]   131-[2]
 113-[1]   111-[1]   118-[1]
  112 ->   105-[1]   103-[1]   101-[1]   107-[1]   111-[1]   106-[1]   152-[2]
 132-[2]
  113 ->   117-[1]   114-[1]   101-[1]   110-[1]   103-[1]   153-[2]   133-[2]
 111-[1]
  114 ->   118-[1]   116-[1]   113-[1]   108-[1]   106-[1]   100-[1]   109-[1]
 154-[2]   134-[2]
  115 ->   105-[1]   103-[1]   106-[1]   101-[1]   116-[1]   155-[2]   135-[2]
 107-[1]
  116 ->   114-[1]   110-[1]   104-[1]   111-[1]   102-[1]   115-[1]   156-[2]
 136-[2]
  117 ->   113-[1]   106-[1]   104-[1]   105-[1]   100-[1]   157-[2]   137-[2]
  118 ->   114-[1]   107-[1]   103-[1]   119-[1]   158-[2]   138-[2]   105-[1]
 111-[1]
  119 ->   106-[1]   101-[1]   104-[1]   118-[1]   159-[2]   139-[2]
 1001 ->   100-[1]

Premere un tasto per continuare . . .
```

Inserisci aeroporto locale

```
Inserisci la label ( numerica ) dell'Aereoporto da inserire: 1010

Scegli tra i seguenti Aereoporti HUB a quale collegare il Locale appena creato:
  100     101     102     103     104     105     106     107     108     109
  110     111     112     113     114     115     116     117     118     119
 1001
Scegli: 1001

Inserimento Terminato.
Utilizzare la funzione di Stampa
Premere un tasto per continuare . . .
```

Inserisci arco LDA

```
Scegli tra i seguenti Aereoporti HUB quali collegare:
  100     101     102     103     104     105     106     107     108     109
  110     111     112     113     114     115     116     117     118     119
 1001
Scegli il primo: 112
Scegli il secondo: 1001

Collegamento Terminato.
Utilizzare la funzione di Stampa
Premere un tasto per continuare . . .
```

Elimina arco LDA

```
Scegli tra i seguenti Aereoporti HUB quali scollegare:
  100     101     102     103     104     105     106     107     108     109
  110     111     112     113     114     115     116     117     118     119
 1001
Scegli il primo: 1001
    118    112    110    106    104    103    100
Scegli il secondo: 100

Scollegamento Terminato.
Utilizzare la funzione di Stampa
Premere un tasto per continuare . . .
```

Elimina aeroporto locale

```
Scegli tra i seguenti Aereoporti HUB quali scollegare:
  100     101     102     103     104     105     106     107     108     109
  110     111     112     113     114     115     116     117     118     119
 1001
Scegli il primo: 105
    112    118    100    107    103    110    106
Scegli il secondo: 100

Scollegamento Terminato.
Utilizzare la funzione di Stampa
Premere un tasto per continuare . . .
```

Elimina aeroporto Locale

```
Scegli tra i seguenti Aereoporti Locali quale eliminare:
  120     121     122     123     124     125     126     127     128     129
  130     131     132     133     134     135     136     137     138     139
  140     141     142     143     144     145     146     147     148     149
  150     151     152     153     154     155     156     157     158     159
 1010
Scegli: 151

Eliminazione Terminata.
Utilizzare la funzione di Stampa
Premere un tasto per continuare . . .
```

Sezione 3

Codice Sorgente

```
 1: /*
 2:     **********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     *              con Algoritmo BFS               *
 5:     * REALIZZATO DA:                               *
 6:     * Marco Sommella 50/482                        *
 7:     * Giovanni Di Cecca 50/887                     *
 8:     * Virginia Bellino 408/466                     *
 9:     **********************************************
10: */
11:
12: #include <stdio.h>
13: #include "grafi.c"
14: #include "airport.c"
15:
16: main()
17: {
18:   /* per il menu */
19:   int in, in2;
20:   char scelta;
21:
22:                           /* INIZIALIZZAZIONE */
23:   graph *G = CreaGrafo ();
24:   G = SituazioneIniziale ( G, 20, 2 );
25:   ControllaPercorsi ( G );
26:
27:                         /* MENU DI SELEZIONE */
28:   do
29:   {
30:     system ( "CLS" );
31:     printf ( "************************************************\n"
    );
32:     printf ( "* SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *\n"
    );
33:     printf ( "*              con Algoritmo BFS               *\n"
    );
34:     printf ( "* REALIZZATO DA:                               *\n"
    );
35:     printf ( "* Marco Sommella 50/482                        *\n"
    );
36:     printf ( "* Giovanni Di Cecca 50/887                     *\n"
    );
37:     printf ( "* Virginia Bellino 408/466                     *\n"
    );
38:     printf (
    "************************************************\n\n" );
39:
40:     printf ( "-------------------------------\n" );
41:     printf ( "a: Inserisci Aereoporto HUB\nb: Inserisci
    Aereoporto Locale\n" );
42:     printf ( "-------------------------------\n" );
```

```
43:     printf ( "c: Elimina Aereoporto HUB\nd: Elimina Aereoporto
    Locale\n" );
44:     printf ( "-------------------------------\n" );
45:     printf ( "e: Inserisci Arco LDA\nf: Elimina Arco LDA\n" );
46:     printf ( "-------------------------------\n" );
47:     printf ( "s: Stampa Situazione\nq: Quit\n" );
48:     printf ( "-------------------------------\n" );
49:     printf ( "Scegli: " );
50:     fflush ( stdin );
51:     scanf( "%c", &scelta );
52:     switch( scelta ) /* l'utente può inserire lettere maiuscole o
53:                         minuscole indifferentemente */
54:     {
55:       case 'A':
56:       case 'a': system ( "CLS");
57:                 printf ( "Inserisci la label ( numerica )
    dell'Aereoporto da inserire: ");
58:                 scanf ( "%d", &in );
59:                 G = InserisciVertice ( G, CreaVertice ( in, 1 ) );
60:                 ControllaPercorsi ( G );
61:                 printf ( "\nInserimento Terminato.\n" );
62:                 printf ( "Utilizzare la funzione di Stampa\n" );
63:                 system ( "PAUSE" );
64:                 break;
65:       case 'B':
66:       case 'b': system ( "CLS");
67:                 printf ( "Inserisci la label ( numerica )
    dell'Aereoporto da inserire: ");
68:                 scanf ( "%d", &in );
69:                 G = InserisciVertice ( G, CreaVertice ( in, 2 ) );
70:                 printf ( "\nScegli tra i seguenti Aereoporti HUB
    a quale collegare il Locale appena creato:\n" );
71:                 StampaHUB ( G );
72:                 printf ( "Scegli: " );
73:                 scanf( "%d", &in2 );
74:                 InserisciArcoN_O ( G, TrovaLabel ( G, in ),
    TrovaLabel ( G, in2 ) );
75:                 ControllaPercorsi ( G );
76:                 printf ( "\nInserimento Terminato.\n" );
77:                 printf ( "Utilizzare la funzione di Stampa\n" );
78:                 system ( "PAUSE" );
79:                 break;
80:       case 'C':
81:       case 'c': system ( "CLS" );
82:                 printf ( "Scegli tra i seguenti Aereoporti HUB
    quale eliminare:\n" );
83:                 StampaHUB ( G );
84:                 printf ( "Scegli: " );
85:                 scanf( "%d", &in );
86:                 EliminaVertice ( G, TrovaLabel ( G, in ) );
87:                 ControllaPercorsi ( G );
```

```
 88:                 printf ( "\nEliminazione Terminata.\n" );
 89:                 printf ( "Utilizzare la funzione di Stampa\n" );
 90:                 system ( "PAUSE" );
 91:                 break;
 92:         case 'D':
 93:         case 'd': system ( "CLS" );
 94:                 printf ( "Scegli tra i seguenti Aereoporti Locali
    quale eliminare:\n" );
 95:                 StampaLocali ( G );
 96:                 printf ( "Scegli: " );
 97:                 scanf( "%d", &in );
 98:                 EliminaVertice ( G, TrovaLabel ( G, in ) );
 99:                 ControllaPercorsi ( G );
100:                 printf ( "\nEliminazione Terminata.\n" );
101:                 printf ( "Utilizzare la funzione di Stampa\n" );
102:                 system ( "PAUSE" );
103:                 break;
104:         case 'E':
105:         case 'e': system ( "CLS" );
106:                 printf ( "Scegli tra i seguenti Aereoporti HUB
    quali collegare:\n" );
107:                 StampaHUB ( G );
108:                 printf ( "Scegli il primo: " );
109:                 scanf( "%d", &in );
110:                 printf ( "Scegli il secondo: " );
111:                 scanf( "%d", &in2 );
112:                 InserisciArcoN_O ( G, TrovaLabel ( G, in ),
    TrovaLabel ( G, in2 ) );
113:                 ControllaPercorsi ( G );
114:                 printf ( "\nCollegamento Terminato.\n" );
115:                 printf ( "Utilizzare la funzione di Stampa\n" );
116:                 system ( "PAUSE" );
117:                 break;
118:         case 'F':
119:         case 'f': system ( "CLS" );
120:                 printf ( "Scegli tra i seguenti Aereoporti HUB
    quali scollegare:\n" );
121:                 StampaHUB ( G );
122:                 printf ( "Scegli il primo: " );
123:                 scanf( "%d", &in );
124:                 StampaHUBAdiacenti ( G, TrovaLabel ( G, in ) );
125:                 printf ( "Scegli il secondo: " );
126:                 scanf( "%d", &in2 );
127:                 EliminaArcoN_O ( G, TrovaLabel ( G, in ),
    TrovaLabel ( G, in2 ) );
128:                 ControllaPercorsi ( G );
129:                 printf ( "\nScollegamento Terminato.\n" );
130:                 printf ( "Utilizzare la funzione di Stampa\n" );
131:                 system ( "PAUSE" );
132:                 break;
133:         case 'S':
```

```
134:        case 's': StampaGrafo ( G );
135:                  break;
136:        default:  break;
137:      }
138:    }
139:    while ( scelta != 'q' && scelta != 'Q' );
140: }
141:
```

```
 1: /*
 2:     *********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     * REALIZZATO DA:                               *
 5:     * Marco Sommella 50/482                        *
 6:     * Giovanni Di Cecca 50/887                     *
 7:     * Virginia Bellino 408/466                     *
 8:     *********************************************
 9: */
10:
11:
12: /* Prototipi delle funzioni */
13: graph *SituazioneIniziale ( graph *G, int hub, int  locali );
14: void StampaHUB ( graph *G );
15: void StampaLocali ( graph *G );
16: void StampaHUBAdiacenti ( graph *G, int i );
17: void ControllaPercorsi ( graph *G );
18:
```

```
 1: /*
 2:     *********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     *              con Algoritmo BFS              *
 5:     * REALIZZATO DA:                              *
 6:     * Marco Sommella 50/482                       *
 7:     * Giovanni Di Cecca 50/887                    *
 8:     * Virginia Bellino 408/466                    *
 9:     *********************************************
10: */
11:
12:
13: #include "airport.h"
14:
15: /*  Crea una situazione iniziale dove ci sono "hub" vertici di
    tipo hub,
16:     "locali" vertici locali per ogni hub. Le label sono impostate
    uguali
17:     all'indice del for +100 solo per praticità.
18:
19:     INPUT:  *G - puntatore al grafo
20:             hub - numero di hub che si desidera avere
21:             locali - numeri di aereoporti locali che si desidera
    avere
22:     OUTPUT: ritorna il grafo modificato
23:
24: */
25: graph *SituazioneIniziale ( graph *G, int hub, int  locali )
26: {
27:   int i; /* indice per il for */
28:
29:   srand ( time ( NULL ) );
30:
31:   for ( i = 0; i < hub; i++ ) /* crea gli HUB */
32:   {
33:     G = InserisciVertice ( G, CreaVertice ( i+100, 1 ) );
34:   }
35:
36:   for ( i = 0; i <  hub/locali; i++ ) /* crea gli archi LNA */
37:   {
38:     InserisciArcoN_O ( G, rand() % hub,  rand() % hub );
39:   }
40:
41:   for ( i = hub; i < ( ( hub * locali ) + hub ); i++ ) /* crea i
    locali e li collega */
42:   {
43:     G = InserisciVertice ( G, CreaVertice ( i+100, 2 ) );
44:     InserisciArcoN_O ( G, i, (i % hub) );
45:   }
46:
47:   return G;
```

```
48: }
49:
50:
51: /*  Stampa tutti i vertici HUB del grafo G
52:
53:     INPUT:  *G - puntatore al grafo
54: */
55: void StampaHUB ( graph *G )
56: {
57:   int i; /* indice per il for */
58:
59:   if( !GrafoVuoto ( G ) )
60:   {
61:     for (i = 0; i < G->nv; i++ )
62:     {
63:       if( G->ver[i] != NULL && G->ver[i]->tipo == 1 )
64:       {
65:         printf ( "%5d\t", G->ver[i]->label );
66:       }
67:     }
68:     printf ( "\n" );
69:   }
70:   else
71:   {
72:     printf("Grafo Vuoto\n");
73:   }
74: }
75:
76:
77: /*  Stampa tutti i vertici locali del grafo G
78:
79:     INPUT:  *G - puntatore al grafo
80: */
81: void StampaLocali ( graph *G )
82: {
83:   int i; /* indice per il for */
84:
85:   if( !GrafoVuoto ( G ) )
86:   {
87:     for (i = 0; i < G->nv; i++ )
88:     {
89:       if( G->ver[i] != NULL && G->ver[i]->tipo == 2 )
90:       {
91:         printf ( "%5d\t", G->ver[i]->label );
92:       }
93:     }
94:     printf ( "\n" );
95:   }
96:   else
97:   {
98:     printf("Grafo Vuoto\n");
```

```
 99:   }
100: }
101:
102:
103: /*  Stampa tutti i vertici HUB adiacenti al vertice i del grafo G
104:
105:     INPUT:  *G - puntatore al grafo
106:             i - posizione del vertice del quale si vogliono gli
    adiacenti HUB
107: */
108: void StampaHUBAdiacenti ( graph *G, int i )
109: {
110:   edge *tmp; /* puntatore per scorrere la lista di adiacenza */
111:
112:   if( !GrafoVuoto ( G ) && G->ver[i] != NULL )
113:   {
114:     tmp = G->ver[i]->next;
115:     while ( tmp != NULL )
116:     {
117:       if ( tmp->tipo == 1 )
118:       {
119:         printf ( "  %5d", G->ver[tmp->key]->label );
120:       }
121:       tmp = tmp->next;
122:     }
123:     printf ( "\n" );
124:   }
125:   else
126:   {
127:     printf("Grafo Vuoto\n");
128:   }
129: }
130:
131:
132: /*  Verifica la condizione di distanza tra 2 vertici usando BFS
133:
134:     INPUT:  *G - puntatore al grafo
135:     OUTPUT: modifica il grafo G se necessario
136: */
137: void ControllaPercorsi ( graph *G )
138: {
139:   int i, j; /* indici per i for */
140:
141:   int w, x; /* memorizzano gli indici nei vertici tra i quali
    inserire un arco */
142:
143:   for ( i = 0; i < G->nv; i++ )
144:   {
145:     if ( G->ver[i] != NULL )
146:     {
147:       Ampiezza ( G, i ); /* esegue la visita in ampiezza partendo
    da i */
```

```
148:       for ( j = 0; j < G->nv; j++ )
149:       {
150:         if ( G->ver[j] != NULL )
151:         {
152:           if ( G->ver[j]->dist > 4 )
153:           {
154:             if ( G->ver[i]->tipo == 1 )
155:             {
156:               w = i;
157:             }
158:             else
159:             {
160:               w = TrovaHUB ( G, i );
161:             }
162:             if ( G->ver[j]->tipo == 1 )
163:             {
164:               x = j;
165:             }
166:             else
167:             {
168:               x = TrovaHUB ( G, j );
169:             }
170:             if ( w != x )
171:             {
172:               InserisciArcoN_O ( G, w, x );
173:               break; /* necessario per bilaciare i collegamenti
     tra gli HUB */
174:             }
175:           }
176:         }
177:       }
178:     }
179:   }
180: }
181:
```

```
 1: /*
 2:     ***********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     * REALIZZATO DA:                               *
 5:     * Marco Sommella 50/482                        *
 6:     * Giovanni Di Cecca 50/887                     *
 7:     * Virginia Bellino 408/466                     *
 8:     ***********************************************
 9: */
10:
11:
12: #include <malloc.h>
13: #include <limits.h>
14:
15:
16: /* Struttura che rappresenta un arco */
17: typedef struct edge
18: {
19:   int key; /* contiene l'indice del vertice entrate */
20:   int tipo; /* indica il tipo di arco: 1 - LDA, 2 - LNA */
21:   struct edge *next; /* puntatore all'adiacente successivo */
22: } edge;
23:
24:
25: /* Struttura che rappresenta un vertice */
26: typedef struct vertex
27: {
28:   int label; /* nome del vertice */
29:   int tipo; /* indica il tipo di vertice: 1 - HUB, 2 - LOCALE */
30:   int visita; /* indica se già visitato */
31:   int dist; /* distanza dall'ultimo punto considerato */
32:   edge *next; /* puntatore a lista di adiacenza */
33: } vertex;
34:
35: /* Rappresentazione del grafo con lista di adiacenza */
36: typedef struct graph
37: {
38:   int nv; /* grandezza dall'array dei vertici */
39:   int usati; /* numero di celle dell'array usate */
40:   vertex **ver; /* array di puntatori a vertici */
41: } graph;
42:
43:
44: /* Prototipi delle funzioni */
45: graph *CreaGrafo ();
46: vertex *CreaVertice ( int label, int tipo );
47: int GrafoVuoto ( graph *G );
48: int Adiacenti ( graph *G, int u, int v );
49: int VerticeValido ( graph *G, int u );
50: int TrovaHUB ( graph *G, int u );
51: graph *InserisciVertice ( graph *G, vertex *vertice );
```

```
52: void InserisciArco ( graph *G, int u, int v );
53: void InserisciArcoN_O ( graph *G, int u, int v );
54: void EliminaArco ( graph *G, int u, int v );
55: void EliminaArcoN_O ( graph *G, int u, int v );
56: void EliminaVertice ( graph *G, int u );
57: void StampaGrafo ( graph *G );
58: void Ampiezza ( graph *G, int start );
59:
```

```
1: /*
2:      ***********************************************
3:      * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
4:      *               con Algoritmo BFS              *
5:      * REALIZZATO DA:                               *
6:      * Marco Sommella 50/482                        *
7:      * Giovanni Di Cecca 50/887                     *
8:      * Virginia Bellino 408/466                     *
9:      ***********************************************
10: */
11:
12:
13: #include "grafi.h"
14:
15: /*
16: ===========================
17:    FUNZIONI DI CREAZIONE
18: ===========================
19: */
20:
21: /*  funzione per la creazione di un grafo con lista di adiacenza
22:
23:    OUTPUT: ritorna il grafo creato
24: */
25: graph *CreaGrafo ()
26: {
27:   graph *G = ( graph* ) malloc ( sizeof ( graph ) ); /* alloca il
grafo */
28:
29:     if ( G == NULL )
30:   {
31:     printf ( "\nERRORE: impossibile allocare memoria per il
grafo\n" );
32:     system ( "PAUSE" );
33:   }
34:   else
35:     {
36:     G->ver = NULL;
37:     G->nv = 0;
38:     G->usati = 0;
39:   }
40:
41:   return G;
42: }
43:
44:
45: /*  Funzione per la creazione di un vertice
46:
47:     INPUT:  label - nome vertice
48:             tipo - tipo di vertice
49:     OUTPUT: ritorna un nuovo vertice
```

```
50: */
51: vertex *CreaVertice ( int label, int tipo )
52: {
53:   vertex *e = ( vertex * ) malloc ( sizeof ( vertex ) ); /*
    alloca un vertice */
54:
55:   if ( e != NULL )
56:   {
57:     e->label = label;
58:     e->tipo = tipo;
59:     e->next = NULL;
60:   }
61:
62:   return e;
63: }
64:
65:
66: /*
67: ======================
68:    FUNZIONI DI TEST
69: ======================
70: */
71:
72:
73: /*  Controlla se il grafo è vuoto
74:
75:     INPUT:  *G - puntatore al grafo
76:     OUTPUT: ritorna 1 (vero) - il grafo è vuoto
77:             ritorna 0 (falso) - il grafo non è vuoto
78: */
79: int GrafoVuoto ( graph *G )
80: {
81:   return ( G == NULL );
82: }
83:
84:
85: /*  Trova una label nel grafo ritornando la posizione all'interno
    dell'array
86:
87:     INPUT:  *G - puntatore al grafo
88:             label - il nome cercato
89:     OUTPUT: ritorna -1 - label non trovata
90:             ritorna l'indice altrimenti
91: */
92: int TrovaLabel ( graph *G, int label )
93: {
94:   int i=0, /* indice per il while */
95:       trovato=-1; /* variabile di ritorno */
96:   if ( !GrafoVuoto ( G ) )
97:   {
98:     while ( i < G->nv && trovato == -1 )
```

```
 99:     {
100:       if ( G->ver[i] != NULL )
101:       {
102:         if ( G->ver[i]->label == label )
103:         {
104:           trovato = i;
105:         }
106:       }
107:       i++;
108:     }
109:   }
110:   return trovato;
111: }
112:
113:
114: /*  Controllo se esiste un arco tra u e v
115:
116: INPUT: *G - puntatore al grafo
117:        u,v - indici dei vertici tra i quali si vuole verificare
     l'esistanza
118:                   di un arco
119: OUTPUT: ritorna 1 (vero) - adiacenti
120:         ritorna 0 (falso) - non adiacenti
121: */
122: int Adiacenti ( graph *G, int u, int v )
123: {
124:   int trovato=0; /* variabile di ritorno */
125:
126:   edge *e = G->ver[u]->next; /* puntatore per scorrere la lista
     di adiacenza */
127:
128:   while ( e != NULL )
129:   {
130:     if ( e->key == v )
131:     {
132:       trovato = 1;
133:     }
134:     e = e->next;
135:   }
136:
137:   return trovato;
138: }
139:
140:
141: /*  Verifica se u è vertice valido
142:
143:     INPUT:  *G - puntatore al grafo
144:             u - indici del vertice che si vuole verificare
145:     OUTPUT: ritorna 1 (vero) - valido
146:       ritorna 0 (falso) - non valido
147: */
```

```
148: int VerticeValido ( graph *G, int u )
149: {
150:   if ( u != -1 )
151:   {
152:     if ( G->ver[u] != NULL )
153:     {
154:       return 1;
155:     }
156:   }
157:   else
158:   {
159:     return 0;
160:   }
161: }
162:
163:
164: /*  Trova l'HUB di un vertice locale u
165:
166:     INPUT:   *G - puntatore al grafo
167:              u - indice del vertice locale
168:     OUTPUT: ritorna -1 se non trovato
169:              ritorna l'indice dell'HUB altrimenti
170: */
171: int TrovaHUB ( graph *G, int u )
172: {
173:   int trovato=-1; /* variabile di ritorno */
174:
175:   edge *e = G->ver[u]->next; /* puntatore per scorrere la lista
     di adiacenza */
176:
177:   while ( e != NULL && trovato == -1 )
178:   {
179:     if ( e->tipo == 2 )
180:     {
181:       trovato = e->key;
182:     }
183:     e = e->next;
184:   }
185:
186:   return trovato;
187: }
188:
189:
190: /*
191: =============================
192:    FUNZIONI DI INSERIMENTO
193: =============================
194: */
195:
196: /*  Inserisce un nuovo vertice nel grafo
197:
```

```
198:     INPUT:   *G - puntatore al grafo
199:              *vertice - puntatore al nuovo vertice
200:     OUTPUT: *G - puntatore al grafo modificato
201: */
202: graph *InserisciVertice ( graph *G, vertex *vertice )
203: {
204:   vertex **e; /* puntatore a vertice */
205:
206:   int i=0; /* indice per il while */
207:
208:   if ( G->nv == 0 ) /* se non vi sono vertici si creano */
209:   {
210:     e = ( vertex ** ) realloc ( G->ver, ( G->nv + 1 ) * sizeof (
     vertex * ) );
211:   }
212:   else
213:   { /* si cerca una posizione dell'array non utilizzata */
214:     while (G->ver[i] != NULL && i < G->nv )
215:     {
216:       i++;
217:     }
218:   }
219:
220:   if ( i < G->nv ) /* se la posizione è trovata, la si usa*/
221:   {
222:     G->usati++;
223:     G->ver[i] = vertice;
224:   }
225:   else /* altrimenti si crea una nuova posizione */
226:   {
227:     e = ( vertex ** ) realloc ( G->ver, ( G->nv + 1 ) * sizeof (
     vertex * ) );
228:     if ( e == NULL )
229:     {
230:       printf ( "\nERRORE: impossibile allocare memoria\n");
231:       system ( "PAUSE" );
232:     }
233:     else
234:     {
235:       G->ver = e;
236:       G->ver[G->nv] = vertice;
237:       G->nv++;
238:       G->usati++;
239:     }
240:   }
241:
242:   return G;
243: }
244:
245:
246: /*  Inserisce un nuovo arco ( orientato ) nel grafo da u a v
```

```
247:
248:     INPUT:  *G - puntatore al grafo
249:             u,v - indici dei vertici tra i quali creare l'arco
250: */
251: void InserisciArco ( graph *G, int u, int v )
252: {
253:   edge *e = ( edge * ) malloc ( sizeof ( edge ) ); /* alloca un
     nuovo arco */
254:   if ( e == NULL )
255:   {
256:     printf ( "\nERRORE: impossibile allocare memoria\n");
257:     system ( "PAUSE" );
258:   }
259:   else
260:   {
261:     if ( !GrafoVuoto ( G ) )
262:     {
263:       if ( VerticeValido ( G, u ) && VerticeValido ( G, v ) )
264:       {
265:         if ( !Adiacenti ( G, u, v ) )
266:         {
267:           if ( G->ver[u]->tipo == 1 && G->ver[v]->tipo == 1 )
268:           {
269:             e->tipo = 1; /* si sta inserendo un LDA */
270:           }
271:           else
272:           {
273:             e->tipo = 2; /* si sta inserendo un LNA */
274:           }
275:           e->key = v;
276:           e->next = G->ver[u]->next;
277:           G->ver[u]->next = e;
278:         }
279:       }
280:       else
281:       {
282:         printf ( "\nERRORE: impossibile creare un arco se uno dei
     2 vertici non esiste\n");
283:         system ( "PAUSE" );
284:       }
285:     }
286:     else
287:     {
288:       printf ( "\nERRORE: impossibile creare un arco in un grafo
     vuoto\n");
289:       system ( "PAUSE" );
290:     }
291:   }
292: }
293:
294:
```

```
295: /*  Inserisce un nuovo arco ( non orientato )
296:
297:     INPUT:  *G - puntatore al grafo
298:             u,v - indici dei vertici tra i quali creare l'arco
299: */
300: void InserisciArcoN_O ( graph *G, int u, int v )
301: {
302:   InserisciArco ( G, u, v);
303:   InserisciArco ( G, v, u);
304: }
305:
306:
307: /*
308: ===============================
309:    FUNZIONI DI CANCELLAZIONE
310: ===============================
311: */
312:
313: /*  Elimina un arco ( orientato ) dal grafo da u a v
314:
315:     INPUT:  *G - puntatore al grafo
316:             u,v - indici dei vertici tra i quali eliminare l'arco
317: */
318: void EliminaArco ( graph *G, int u, int v )
319: {
320:   edge *prev, /* l'arco precedente a quello da togliere nella
     lista */
321:         *e; /* l'arco da togliere dalla lista */
322:
323:   if ( !GrafoVuoto ( G ) )
324:   {
325:     if ( VerticeValido ( G, u ) && VerticeValido ( G, v ) )
326:     {
327:       if ( Adiacenti ( G, u, v ) )
328:       {
329:         e = G->ver[u]->next;
330:         if ( e->key == v ) /* se è l'elemento in testa */
331:         {
332:           G->ver[u]->next = e->next;
333:         }
334:         else
335:         {
336:           prev = e;
337:           while ( prev->next->key != v ) /* altrimenti lo cerco */
338:           {
339:             prev = prev->next;
340:           }
341:           e = prev->next;
342:           prev->next = e->next;
343:         }
344:       }
```

```
345:         else
346:         {
347:           printf ( "\nERRORE: impossibile eliminare un arco se non
     esiste\n");
348:           system ( "PAUSE" );
349:         }
350:       }
351:       else
352:       {
353:         printf ( "\nERRORE: impossibile eliminare un arco se uno
     dei 2 vertici non esiste\n");
354:         system ( "PAUSE" );
355:       }
356:     }
357:     else
358:     {
359:       printf ( "\nERRORE: impossibile eliminare un arco in un grafo
     vuoto\n");
360:       system ( "PAUSE" );
361:     }
362:
363:     free ( e );
364: }
365:
366:
367: /*  Elimina un arco ( non orientato ) dal grafo da u a v
368:
369:       INPUT:   *G - puntatore al grafo
370:                u,v - indici dei vertici tra i quali eliminare l'arco
371: */
372: void EliminaArcoN_O ( graph *G, int u, int v )
373: {
374:     EliminaArco ( G, u, v );
375:
376:     EliminaArco ( G, v, u );
377: }
378:
379:
380: /*  Elimina un un vertice dal grafo ( lasciando la posizione
     dell'array a NULL )
381:
382:       INPUT:   *G - puntatore al grafo
383:                u - indice del vertice da eliminare
384: */
385: void EliminaVertice ( graph *G, int u )
386: {
387:     vertex *e; /* puntatore per memorizzare l'indirizzo del vertice
     da eliminare */
388:
389:     edge *a; /* puntatore per scorrere la lista di adiacenza */
390:
```

```
391:   if ( !GrafoVuoto ( G ) )
392:   {
393:     if ( VerticeValido ( G, u ) )
394:     {
395:       e = G->ver[u];
396:       a = G->ver[u]->next;
397:       if ( G->ver[u]->tipo == 2 ) /* u è un locale devo solo
    eliminare
398:                                       il rispettivo LNA */
399:       {
400:         EliminaArcoN_O ( G, u, TrovaHUB ( G, u ) );
401:         G->ver[u] = NULL;
402:       }
403:       else /* u è un HUB */
404:       {
405:         while ( a != NULL )
406:         {
407:           if ( a->tipo == 1 ) /* devo eliminare tutti gli LDA
    relativi */
408:           {
409:             EliminaArcoN_O ( G, u, a->key );
410:           }
411:           else /* e tutti i locali ad esso collegati */
412:           {
413:             EliminaVertice ( G, a->key );
414:           }
415:           a = G->ver[u]->next;
416:         }
417:       }
418:       free ( e );
419:       G->ver[u] = NULL;
420:       --G->usati;
421:     }
422:     else
423:     {
424:       printf ( "\nERRORE: impossibile eliminare un vertice se non
    esiste\n");
425:       system ( "PAUSE" );
426:     }
427:   }
428:   else
429:   {
430:     printf ( "\nERRORE: impossibile eliminare un vertice in un
    grafo vuoto\n");
431:     system ( "PAUSE" );
432:   }
433: }
434:
435:
436: /*
437: ========================
```

```
438:    FUNZIONI DI STAMPA
439: ========================
440: */
441:
442: /* Stampa informazioni sul grafo
443:
444:     INPUT: *G - puntatore al grafo
445: */
446: void StampaGrafo ( graph *G )
447: {
448:   edge *tmp; /* puntatore per scorrere la lista di adiacenza */
449:
450:   int i; /* indice del for */
451:
452:   system ( "CLS" );
453:
454:   if( !GrafoVuoto ( G ) )
455:   {
456:     printf ( "Sono stampati solo gli Aereoporti HUB.\n" );
457:     printf ( "Tra parentesi quadre si indica il tipo di tratta: 1
     se LDA, 2 se LNA.\n" );
458:     printf ( "\nCi sono %d Aereoporti ( HUB e Locali )\n\n", G-
     >usati );
459:     for (i = 0; i < G->nv; i++ )
460:     {
461:       if( G->ver[i] != NULL && G->ver[i]->tipo == 1 )
462:       {
463:         printf ( "%5d ->", G->ver[i]->label );
464:         tmp = G->ver[i]->next;
465:         while ( tmp != NULL )
466:         {
467:           printf ( " %5d-[%d]", G->ver[tmp->key]->label, tmp-
     >tipo );
468:           tmp = tmp->next;
469:         }
470:         printf ( "\n" );
471:       }
472:     }
473:   }
474:   else
475:   {
476:     printf("Grafo Vuoto");
477:   }
478:   printf ( "\n" );
479:
480:   system("PAUSE");
481: }
482:
483:
484: /*
485: ========================
```

```
486:    FUNZIONI DI VISITA
487: ========================
488: */
489:
490: /*  Visita in ampiezza
491:
492:     INPUT: *G - puntatore al grafo
493:            start - punto di partenza della visita
494:     OUTPUT: modifica il campo dist dei vertici del grafo con la
    distanza da start
495: */
496: void Ampiezza ( graph *G, int start )
497: {
498:   edge *Succ;   /* puntatore di ricerca tra gli archi orientati */
499:
500:   int IndNodo,  /* posizione nell'array del nodo di partenza */
501:       IndEsame, /* posizione nell'array del nodo in esame */
502:       Livello,  /* distanza dei nodi dal nodo base */
503:       *Fifo,    /* coda per gli indici dei figli da visitare */
504:       Front,    /* prossimo elemento da estrarre in Fifo[] */
505:       Rear,     /* casella libera per prossimo inserimento in
    coda */
506:       *Colori, *Pred, *Dist;  /* vettore dei colori e dei
    predecessori */
507:
508:   Colori = ( int * ) malloc ( G->nv * sizeof ( int ) );
509:
510:   Pred = ( int * ) malloc ( G->nv * sizeof ( int ) );
511:
512:   Dist = ( int * ) malloc ( G->nv * sizeof ( int ) );
513:
514:   /*  Azzera la flag 'Nodo Visitato' in ogni nodo */
515:   for (IndEsame = 0; IndEsame < G->nv; IndEsame ++)
516:   {
517:     Dist[IndEsame] = INT_MAX;
518:
519:     Colori[IndEsame] = 0;
520:
521:     Pred[IndEsame] = -1;
522:   }
523:   IndNodo = start;
524:
525:   Fifo = ( int * ) malloc ( sizeof ( int ) * G->nv );
526:
527:   Front = Rear = 0;
528:
529:   Fifo [Rear ++] = IndNodo; /* il nodo base in coda */
530:
531:   Colori[IndNodo] = 1;  /* marca 'visitato' */
532:
533:   Dist[IndNodo] = 0;
```

```
534:
535:   while (Front < Rear)
536:   {
537:     IndEsame = Fifo[Front ++]; /* estrae un nodo dalla coda */
538:
539:     /* Stampa informazioni sul Nodo */
540:
541:     /* inserisce in coda tutti i figli non visitati */
542:     Succ = G->ver[IndEsame]->next;
543:     while ( Succ != NULL )
544:     {
545:       if ( Colori[Succ->key] == 0 )
546:       {
547:         Fifo [Rear ++] =  Succ->key;
548:         Colori[Succ->key] = 1;
549:         Pred[Succ->key] = IndEsame;
550:         Dist[Succ->key] = Dist[Pred[Succ->key]] + 1;
551:       }
552:       Succ = Succ->next;
553:     }
554:   }
555:   free (Fifo);
556:   free (Colori);
557:   free (Pred);
558:   for (IndEsame = 0; IndEsame < G->nv; IndEsame ++)
559:   {
560:     if ( G->ver[IndEsame] != NULL )
561:     {
562:       G->ver[IndEsame]->dist = Dist[IndEsame];
563:     }
564:   }
565:   free (Dist);
566: }
567:
```

Sezione 4

Appendice

Descrizione:

Aspetti generali

Questa versione si differenza dalla precedente perché usa l'Algoritmo di Dijkstra per il calcolo del cammino minimo e riorganizzazione delle distanze, utilizzando come struttura dati di appoggio una coda a priorità implementata con un heap.

♦ Specifiche della libreria `grafi.c`:

graph *CreaGrafo ();

vertex *CreaVertice (int label, int tipo);

int GrafoVuoto (graph *G);

int TrovaLabel (graph *G, int label);

int Adiacenti (graph *G, int u, int v);

int VerticeValido (graph *G, int u);

int TrovaHUB (graph *G, int u);

graph *InserisciVertice (graph *G, vertex *vertice);

void InserisciArco (graph *G, int u, int v);

void InserisciArcoN_O (graph *G, int u, int v);

void EliminaArco (graph *G, int u, int v);

void EliminaArcoN_O (graph *G, int u, int v);

void EliminaVertice (graph *G, int u);

void StampaGrafo (graph *G);

- **Specifiche della libreria `airport.c`:**

  graph *SituazioneIniziale (graph *G, int hub, int locali);

  void StampaHUB (graph *G);

  void StampaLocali (graph *G);

  void StampaHUBAdiacenti (graph *G, int i);

- **Specifiche della libreria `heap.c`:**

  int left(int i);

  int right(int i);

  int p(int i);

  void swap (heap *q, int i, int j);

  void Heapify (heap *q, int i);

  int EmptyHeap (heap *q);

  int GetMin (heap *q);

  int TrovaCorr (graph *G, heap *q, int start);

  int *Dijkstra (graph *G, int start);

  int left(int i);

  int right(int i);

  int p(int i);

  void swap (heap *q, int i, int j);

  void Heapify (heap *q, int i);

  int EmptyHeap (heap *q);

int GetMin (heap *q);

int TrovaCorr (graph *G, heap *q, int start);

int *Dijkstra (graph *G, int start);

void ControllaPercorsi (graph *G);void ControllaPercorsi (graph *G);

Sezione 5

Codice Sorgente

```
 1: /*
 2:     **********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     *           con Algoritmo di Dijkstra         *
 5:     * REALIZZATO DA:                              *
 6:     * Marco Sommella 50/482                       *
 7:     * Giovanni Di Cecca 50/887                    *
 8:     * Virginia Bellino 408/466                    *
 9:     **********************************************
10: */
11:
12: #include <stdio.h>
13: #include "grafi.c"
14: #include "airport.c"
15: #include "heap.c"
16:
17: main()
18: {
19:   /* per il menu */
20:   int in, in2;
21:   char scelta;
22:
23:                           /* INIZIALIZZAZIONE */
24:   graph *G = CreaGrafo ();
25:   G = SituazioneIniziale ( G, 20, 2 );
26:   ControllaPercorsi ( G );
27:
28:                         /* MENU DI SELEZIONE */
29:   do
30:   {
31:     system ( "CLS" );
32:     printf ( "**********************************************\n"
   );
33:     printf ( "* SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *\n"
   );
34:     printf ( "*         con Algoritmo di  Dijkstra          *\n"
   );
35:     printf ( "* REALIZZATO DA:                              *\n"
   );
36:     printf ( "* Marco Sommella 50/482                       *\n"
   );
37:     printf ( "* Giovanni Di Cecca 50/887                    *\n"
   );
38:     printf ( "* Virginia Bellino 408/466                    *\n"
   );
39:     printf (
   "**********************************************\n\n" );
40:
41:     printf ( "------------------------------\n" );
42:     printf ( "a: Inserisci Aereoporto HUB\nb: Inserisci
   Aereoporto Locale\n" );
```

```
43:     printf ( "------------------------------\n" );
44:     printf ( "c: Elimina Aereoporto HUB\nd: Elimina Aereoporto
   Locale\n" );
45:     printf ( "------------------------------\n" );
46:     printf ( "e: Inserisci Arco LDA\nf: Elimina Arco LDA\n" );
47:     printf ( "------------------------------\n" );
48:     printf ( "s: Stampa Situazione\nq: Quit\n" );
49:     printf ( "------------------------------\n" );
50:     printf ( "Scegli: " );
51:     fflush ( stdin );
52:     scanf( "%c", &scelta );
53:     switch( scelta ) /* l'utente può inserire lettere maiuscole o
54:                         minuscole indifferentemente */
55:     {
56:       case 'A':
57:       case 'a': system ( "CLS");
58:                 printf ( "Inserisci la label ( numerica )
   dell'Aereoporto da inserire: ");
59:                 scanf ( "%d", &in );
60:                 G = InserisciVertice ( G, CreaVertice ( in, 1 ) );
61:                 ControllaPercorsi ( G );
62:                 printf ( "\nInserimento Terminato.\n" );
63:                 printf ( "Utilizzare la funzione di Stampa\n" );
64:                 system ( "PAUSE" );
65:                 break;
66:       case 'B':
67:       case 'b': system ( "CLS");
68:                 printf ( "Inserisci la label ( numerica )
   dell'Aereoporto da inserire: ");
69:                 scanf ( "%d", &in );
70:                 G = InserisciVertice ( G, CreaVertice ( in, 2 ) );
71:                 printf ( "\nScegli tra i seguenti Aereoporti HUB
   a quale collegare il Locale appena creato:\n" );
72:                 StampaHUB ( G );
73:                 printf ( "Scegli: " );
74:                 scanf( "%d", &in2 );
75:                 InserisciArcoN_O ( G, TrovaLabel ( G, in ),
   TrovaLabel ( G, in2 ) );
76:                 ControllaPercorsi ( G );
77:                 printf ( "\nInserimento Terminato.\n" );
78:                 printf ( "Utilizzare la funzione di Stampa\n" );
79:                 system ( "PAUSE" );
80:                 break;
81:       case 'C':
82:       case 'c': system ( "CLS" );
83:                 printf ( "Scegli tra i seguenti Aereoporti HUB
   quale eliminare:\n" );
84:                 StampaHUB ( G );
85:                 printf ( "Scegli: " );
86:                 scanf( "%d", &in );
87:                 EliminaVertice ( G, TrovaLabel ( G, in ) );
```

```
 88:                    ControllaPercorsi ( G );
 89:                    printf ( "\nEliminazione Terminata.\n" );
 90:                    printf ( "Utilizzare la funzione di Stampa\n" );
 91:                    system ( "PAUSE" );
 92:                    break;
 93:          case 'D':
 94:          case 'd': system ( "CLS" );
 95:                    printf ( "Scegli tra i seguenti Aereoporti Locali
    quale eliminare:\n" );
 96:                    StampaLocali ( G );
 97:                    printf ( "Scegli: " );
 98:                    scanf( "%d", &in );
 99:                    EliminaVertice ( G, TrovaLabel ( G, in ) );
100:                    ControllaPercorsi ( G );
101:                    printf ( "\nEliminazione Terminata.\n" );
102:                    printf ( "Utilizzare la funzione di Stampa\n" );
103:                    system ( "PAUSE" );
104:                    break;
105:          case 'E':
106:          case 'e': system ( "CLS" );
107:                    printf ( "Scegli tra i seguenti Aereoporti HUB
    quali collegare:\n" );
108:                    StampaHUB ( G );
109:                    printf ( "Scegli il primo: " );
110:                    scanf( "%d", &in );
111:                    printf ( "Scegli il secondo: " );
112:                    scanf( "%d", &in2 );
113:                    InserisciArcoN_O ( G, TrovaLabel ( G, in ),
    TrovaLabel ( G, in2 ) );
114:                    ControllaPercorsi ( G );
115:                    printf ( "\nCollegamento Terminato.\n" );
116:                    printf ( "Utilizzare la funzione di Stampa\n" );
117:                    system ( "PAUSE" );
118:                    break;
119:          case 'F':
120:          case 'f': system ( "CLS" );
121:                    printf ( "Scegli tra i seguenti Aereoporti HUB
    quali scollegare:\n" );
122:                    StampaHUB ( G );
123:                    printf ( "Scegli il primo: " );
124:                    scanf( "%d", &in );
125:                    StampaHUBAdiacenti ( G, TrovaLabel ( G, in ) );
126:                    printf ( "Scegli il secondo: " );
127:                    scanf( "%d", &in2 );
128:                    EliminaArcoN_O ( G, TrovaLabel ( G, in ),
    TrovaLabel ( G, in2 ) );
129:                    ControllaPercorsi ( G );
130:                    printf ( "\nScollegamento Terminato.\n" );
131:                    printf ( "Utilizzare la funzione di Stampa\n" );
132:                    system ( "PAUSE" );
133:                    break;
```

```
134:        case 'S':
135:        case 's': StampaGrafo ( G );
136:                  break;
137:        default:  break;
138:      }
139:    }
140:   while ( scelta != 'q' && scelta != 'Q' );
141: }
142:
```

```
 1: /*
 2:     *********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     *            con Algoritmo di Dijkstra          *
 5:     * REALIZZATO DA:                                *
 6:     * Marco Sommella 50/482                         *
 7:     * Giovanni Di Cecca 50/887                      *
 8:     * Virginia Bellino 408/466                      *
 9:     *********************************************
10: */
11:
12: /* Prototipi delle funzioni */
13: graph *SituazioneIniziale ( graph *G, int hub, int  locali);
14: void StampaHUB (graph *G);
15: void StampaLocali (graph *G);
16: void StampaHUBAdiacenti (graph *G, int i);
17:
18:
```

```
 1: /*
 2:     *********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     *           con Algoritmo di Dijkstra          *
 5:     * REALIZZATO DA:                               *
 6:     * Marco Sommella 50/482                        *
 7:     * Giovanni Di Cecca 50/887                     *
 8:     * Virginia Bellino 408/466                     *
 9:     *********************************************
10: */
11:
12: #include "airport.h"
13:
14: /*  Crea una situazione iniziale dove ci sono "hub" vertici di
    tipo hub,
15:     "locali" vertici locali per ogni hub. Le label sono impostate
    uguali
16:     all'indice del for +100 solo per praticità.
17:
18:     INPUT:  *G - puntatore al grafo
19:             hub - numero di hub che si desidera avere
20:             locali - numeri di aereoporti locali che si desidera
    avere
21:     OUTPUT: ritorna il grafo modificato
22:
23: */
24: graph *SituazioneIniziale ( graph *G, int hub, int  locali )
25: {
26:   int i; /* indice per il for */
27:
28:   srand ( time ( NULL ) );
29:
30:   for ( i = 0; i < hub; i++ ) /* crea gli HUB */
31:   {
32:     G = InserisciVertice ( G, CreaVertice ( i+100, 1 ) );
33:   }
34:
35:   for ( i = 0; i <  hub/locali; i++ ) /* crea gli archi LNA */
36:   {
37:     InserisciArcoN_O ( G, rand() % hub,  rand() % hub );
38:   }
39:
40:   for ( i = hub; i < ( ( hub * locali ) + hub ); i++ ) /* crea i
    locali e li collega */
41:   {
42:     G = InserisciVertice ( G, CreaVertice ( i+100, 2 ) );
43:     InserisciArcoN_O ( G, i, (i % hub) );
44:   }
45:
46:   return G;
47: }
```

```
48:
49:
50: /*  Stampa tutti i vertici HUB del grafo G
51:
52:     INPUT:  *G - puntatore al grafo
53: */
54: void StampaHUB ( graph *G )
55: {
56:   int i; /* indice per il for */
57:
58:   if( !GrafoVuoto ( G ) )
59:   {
60:     for (i = 0; i < G->nv; i++ )
61:     {
62:       if( G->ver[i] != NULL && G->ver[i]->tipo == 1 )
63:       {
64:         printf ( "%5d\t", G->ver[i]->label );
65:       }
66:     }
67:     printf ( "\n" );
68:   }
69:   else
70:   {
71:     printf("Grafo Vuoto\n");
72:   }
73: }
74:
75:
76: /*  Stampa tutti i vertici locali del grafo G
77:
78:     INPUT:  *G - puntatore al grafo
79: */
80: void StampaLocali ( graph *G )
81: {
82:   int i; /* indice per il for */
83:
84:   if( !GrafoVuoto ( G ) )
85:   {
86:     for (i = 0; i < G->nv; i++ )
87:     {
88:       if( G->ver[i] != NULL && G->ver[i]->tipo == 2 )
89:       {
90:         printf ( "%5d\t", G->ver[i]->label );
91:       }
92:     }
93:     printf ( "\n" );
94:   }
95:   else
96:   {
97:     printf("Grafo Vuoto\n");
98:   }
```

```
 99: }
100:
101:
102: /*  Stampa tutti i vertici HUB adiacenti al vertice i del grafo G
103:
104:     INPUT:  *G - puntatore al grafo
105:             i - posizione del vertice del quale si vogliono gli
    adiacenti HUB
106: */
107: void StampaHUBAdiacenti ( graph *G, int i )
108: {
109:   edge *tmp; /* puntatore per scorrere la lista di adiacenza */
110:
111:   if( !GrafoVuoto ( G ) && G->ver[i] != NULL )
112:   {
113:     tmp = G->ver[i]->next;
114:     while ( tmp != NULL )
115:     {
116:       if ( tmp->tipo == 1 )
117:       {
118:         printf ( "  %5d", G->ver[tmp->key]->label );
119:       }
120:       tmp = tmp->next;
121:     }
122:     printf ( "\n" );
123:   }
124:   else
125:   {
126:     printf("Grafo Vuoto\n");
127:   }
128: }
129:
```

```
 1: /*
 2:     ***********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     *           con Algoritmo di Dijkstra          *
 5:     * REALIZZATO DA:                               *
 6:     * Marco Sommella 50/482                        *
 7:     * Giovanni Di Cecca 50/887                     *
 8:     * Virginia Bellino 408/466                     *
 9:     ***********************************************
10: */
11:
12: #include <malloc.h>
13:
14: /* Struttura che rappresenta un arco */
15: typedef struct edge
16: {
17:   int key; /* contiene l'indice del vertice entrate */
18:   int tipo; /* indica il tipo di arco: 1 - LDA, 2 - LNA */
19:   struct edge *next; /* puntatore all'adiacente successivo */
20: } edge;
21:
22:
23: /* Struttura che rappresenta un vertice */
24: typedef struct vertex
25: {
26:   int label; /* nome del vertice */
27:   int tipo; /* indica il tipo di vertice: 1 - HUB, 2 - LOCALE */
28:   edge *next; /* puntatore a lista di adiacenza */
29: } vertex;
30:
31: /* Rappresentazione del grafo con lista di adiacenza */
32: typedef struct graph
33: {
34:   int nv; /* grandezza dall'array dei vertici */
35:   int usati; /* numero di celle dell'array usate */
36:   vertex **ver; /* array di puntatori a vertici */
37: } graph;
38:
39:
40: /* Prototipi delle funzioni */
41: graph *CreaGrafo ();
42: vertex *CreaVertice ( int label, int tipo );
43: int GrafoVuoto ( graph *G );
44: int TrovaLabel ( graph *G, int label );
45: int Adiacenti ( graph *G, int u, int v );
46: int VerticeValido ( graph *G, int u );
47: int TrovaHUB ( graph *G, int u );
48: graph *InserisciVertice ( graph *G, vertex *vertice );
49: void InserisciArco ( graph *G, int u, int v);
50: void InserisciArcoN_O ( graph *G, int u, int v);
51: void EliminaArco ( graph *G, int u, int v );
```

```
52: void EliminaArcoN_O ( graph *G, int u, int v );
53: void EliminaVertice ( graph *G, int u );
54: void StampaGrafo (graph *G);
55:
```

```
1: /*
2:     *************************************************
3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
4:     *           con Algoritmo di Dijkstra          *
5:     * REALIZZATO DA:                               *
6:     * Marco Sommella 50/482                        *
7:     * Giovanni Di Cecca 50/887                     *
8:     * Virginia Bellino 408/466                     *
9:     *************************************************
10: */
11:
12: #include "grafi.h"
13:
14: /*
15: ===========================
16:    FUNZIONI DI CREAZIONE
17: ===========================
18: */
19:
20: /*  funzione per la creazione di un grafo con lista di adiacenza
21:
22:    OUTPUT: ritorna il grafo creato
23: */
24: graph *CreaGrafo ()
25: {
26:   graph *G = ( graph* ) malloc ( sizeof ( graph ) ); /* alloca il
    grafo */
27:
28:     if ( G == NULL )
29:   {
30:     printf ( "\nERRORE: impossibile allocare memoria per il
    grafo\n" );
31:     system ( "PAUSE" );
32:   }
33:   else
34:     {
35:     G->ver = NULL;
36:     G->nv = 0;
37:     G->usati = 0;
38:   }
39:
40:   return G;
41: }
42:
43:
44: /*  Funzione per la creazione di un vertice
45:
46:     INPUT:  label - nome vertice
47:             tipo - tipo di vertice
48:     OUTPUT: ritorna un nuovo vertice
49: */
```

```
50: vertex *CreaVertice ( int label, int tipo )
51: {
52:   vertex *e = ( vertex * ) malloc ( sizeof ( vertex ) ); /*
    alloca un vertice */
53:
54:   if ( e != NULL )
55:   {
56:     e->label = label;
57:     e->tipo = tipo;
58:     e->next = NULL;
59:   }
60:
61:   return e;
62: }
63:
64:
65: /*
66: ======================
67:    FUNZIONI DI TEST
68: ======================
69: */
70:
71:
72: /*  Controlla se il grafo è vuoto
73:
74:     INPUT:  *G - puntatore al grafo
75:     OUTPUT: ritorna 1 (vero) - il grafo è vuoto
76:             ritorna 0 (falso) - il grafo non è vuoto
77: */
78: int GrafoVuoto ( graph *G )
79: {
80:   return ( G == NULL );
81: }
82:
83:
84: /*  Trova una label nel grafo ritornando la posizione all'interno
    dell'array
85:
86:     INPUT:  *G - puntatore al grafo
87:             label - il nome cercato
88:     OUTPUT: ritorna -1 - label non trovata
89:             ritorna l'indice altrimenti
90: */
91: int TrovaLabel ( graph *G, int label )
92: {
93:   int i=0, /* indice per il while */
94:       trovato=-1; /* variabile di ritorno */
95:   if ( !GrafoVuoto ( G ) )
96:   {
97:     while ( i < G->nv && trovato == -1 )
98:     {
```

```
 99:       if ( G->ver[i] != NULL )
100:       {
101:         if ( G->ver[i]->label == label )
102:         {
103:           trovato = i;
104:         }
105:       }
106:       i++;
107:     }
108:   }
109:   return trovato;
110: }
111:
112:
113: /*  Controllo se esiste un arco tra u e v
114:
115: INPUT: *G - puntatore al grafo
116:        u,v - indici dei vertici tra i quali si vuole verificare
     l'esistanza
117:                   di un arco
118: OUTPUT: ritorna 1 (vero) - adiacenti
119:         ritorna 0 (falso) - non adiacenti
120: */
121: int Adiacenti ( graph *G, int u, int v )
122: {
123:   int trovato=0; /* variabile di ritorno */
124:
125:   edge *e = G->ver[u]->next; /* puntatore per scorrere la lista
     di adiacenza */
126:
127:   while ( e != NULL )
128:   {
129:     if ( e->key == v )
130:     {
131:       trovato = 1;
132:     }
133:     e = e->next;
134:   }
135:
136:   return trovato;
137: }
138:
139:
140: /*  Verifica se u è vertice valido
141:
142:     INPUT:  *G - puntatore al grafo
143:             u - indici del vertice che si vuole verificare
144:     OUTPUT: ritorna 1 (vero) - valido
145:       ritorna 0 (falso) - non valido
146: */
147: int VerticeValido ( graph *G, int u )
```

```
148: {
149:   if ( u != -1 )
150:   {
151:     if ( G->ver[u] != NULL )
152:     {
153:       return 1;
154:     }
155:   }
156:   else
157:   {
158:     return 0;
159:   }
160: }
161:
162:
163: /*  Trova l'HUB di un vertice locale u
164:
165:     INPUT:  *G - puntatore al grafo
166:             u - indice del vertice locale
167:     OUTPUT: ritorna -1 se non trovato
168:             ritorna l'indice dell'HUB altrimenti
169: */
170: int TrovaHUB ( graph *G, int u )
171: {
172:   int trovato=-1; /* variabile di ritorno */
173:
174:   edge *e = G->ver[u]->next; /* puntatore per scorrere la lista
     di adiacenza */
175:
176:   while ( e != NULL && trovato == -1 )
177:   {
178:     if ( e->tipo == 2 )
179:     {
180:       trovato = e->key;
181:     }
182:     e = e->next;
183:   }
184:
185:   return trovato;
186: }
187:
188:
189: /*
190: =============================
191:    FUNZIONI DI INSERIMENTO
192: =============================
193: */
194:
195: /*  Inserisce un nuovo vertice nel grafo
196:
197:     INPUT:  *G - puntatore al grafo
```

```
198:                *vertice - puntatore al nuovo vertice
199:        OUTPUT: *G - puntatore al grafo modificato
200: */
201: graph *InserisciVertice ( graph *G, vertex *vertice )
202: {
203:   vertex **e; /* puntatore a vertice */
204:
205:   int i=0; /* indice per il while */
206:
207:   if ( G->nv == 0 ) /* se non vi sono vertici si creano */
208:   {
209:     e = ( vertex ** ) realloc ( G->ver, ( G->nv + 1 ) * sizeof (
     vertex * ) );
210:   }
211:   else
212:   { /* si cerca una posizione dell'array non utilizzata */
213:     while (G->ver[i] != NULL && i < G->nv )
214:     {
215:       i++;
216:     }
217:   }
218:
219:   if ( i < G->nv ) /* se la posizione è trovata, la si usa*/
220:   {
221:     G->usati++;
222:     G->ver[i] = vertice;
223:   }
224:   else /* altrimenti si crea una nuova posizione */
225:   {
226:     e = ( vertex ** ) realloc ( G->ver, ( G->nv + 1 ) * sizeof (
     vertex * ) );
227:     if ( e == NULL )
228:     {
229:       printf ( "\nERRORE: impossibile allocare memoria\n");
230:       system ( "PAUSE" );
231:     }
232:     else
233:     {
234:       G->ver = e;
235:       G->ver[G->nv] = vertice;
236:       G->nv++;
237:       G->usati++;
238:     }
239:   }
240:
241:   return G;
242: }
243:
244:
245: /*  Inserisce un nuovo arco ( orientato ) nel grafo da u a v
246:
```

```
247:     INPUT:  *G - puntatore al grafo
248:             u,v - indici dei vertici tra i quali creare l'arco
249: */
250: void InserisciArco ( graph *G, int u, int v )
251: {
252:   edge *e = ( edge * ) malloc ( sizeof ( edge ) ); /* alloca un
     nuovo arco */
253:   if ( e == NULL )
254:   {
255:     printf ( "\nERRORE: impossibile allocare memoria\n");
256:     system ( "PAUSE" );
257:   }
258:   else
259:   {
260:     if ( !GrafoVuoto ( G ) )
261:     {
262:       if ( VerticeValido ( G, u ) && VerticeValido ( G, v ) )
263:       {
264:         if ( !Adiacenti ( G, u, v ) )
265:         {
266:           if ( G->ver[u]->tipo == 1 && G->ver[v]->tipo == 1 )
267:           {
268:             e->tipo = 1; /* si sta inserendo un LDA */
269:           }
270:           else
271:           {
272:             e->tipo = 2; /* si sta inserendo un LNA */
273:           }
274:           e->key = v;
275:           e->next = G->ver[u]->next;
276:           G->ver[u]->next = e;
277:         }
278:       }
279:       else
280:       {
281:         printf ( "\nERRORE: impossibile creare un arco se uno dei
     2 vertici non esiste\n");
282:         system ( "PAUSE" );
283:       }
284:     }
285:     else
286:     {
287:       printf ( "\nERRORE: impossibile creare un arco in un grafo
     vuoto\n");
288:       system ( "PAUSE" );
289:     }
290:   }
291: }
292:
293:
294: /*  Inserisce un nuovo arco ( non orientato )
```

```
295:
296:     INPUT:  *G - puntatore al grafo
297:             u,v - indici dei vertici tra i quali creare l'arco
298: */
299: void InserisciArcoN_O ( graph *G, int u, int v )
300: {
301:   InserisciArco ( G, u, v);
302:   InserisciArco ( G, v, u);
303: }
304:
305:
306: /*
307: ===============================
308:    FUNZIONI DI CANCELLAZIONE
309: ===============================
310: */
311:
312: /*  Elimina un arco ( orientato ) dal grafo da u a v
313:
314:     INPUT:  *G - puntatore al grafo
315:             u,v - indici dei vertici tra i quali eliminare l'arco
316: */
317: void EliminaArco ( graph *G, int u, int v )
318: {
319:   edge *prev, /* l'arco precedente a quello da togliere nella
     lista */
320:             *e; /* l'arco da togliere dalla lista */
321:
322:   if ( !GrafoVuoto ( G ) )
323:   {
324:     if ( VerticeValido ( G, u ) && VerticeValido ( G, v ) )
325:     {
326:       if ( Adiacenti ( G, u, v ) )
327:       {
328:         e = G->ver[u]->next;
329:         if ( e->key == v ) /* se è l'elemento in testa */
330:         {
331:           G->ver[u]->next = e->next;
332:         }
333:         else
334:         {
335:           prev = e;
336:           while ( prev->next->key != v ) /* altrimenti lo cerco */
337:           {
338:             prev = prev->next;
339:           }
340:           e = prev->next;
341:           prev->next = e->next;
342:         }
343:       }
344:       else
```

```
345:       {
346:         printf ( "\nERRORE: impossibile eliminare un arco se non
    esiste\n");
347:         system ( "PAUSE" );
348:       }
349:     }
350:     else
351:     {
352:       printf ( "\nERRORE: impossibile eliminare un arco se uno
    dei 2 vertici non esiste\n");
353:       system ( "PAUSE" );
354:     }
355:   }
356:   else
357:   {
358:     printf ( "\nERRORE: impossibile eliminare un arco in un grafo
    vuoto\n");
359:     system ( "PAUSE" );
360:   }
361:
362:   free ( e );
363: }
364:
365:
366: /*  Elimina un arco ( non orientato ) dal grafo da u a v
367:
368:     INPUT:  *G - puntatore al grafo
369:             u,v - indici dei vertici tra i quali eliminare l'arco
370: */
371: void EliminaArcoN_O ( graph *G, int u, int v )
372: {
373:   EliminaArco ( G, u, v );
374:
375:   EliminaArco ( G, v, u );
376: }
377:
378:
379: /*  Elimina un un vertice dal grafo ( lasciando la posizione
    dell'array a NULL )
380:
381:     INPUT:  *G - puntatore al grafo
382:             u - indice del vertice da eliminare
383: */
384: void EliminaVertice ( graph *G, int u )
385: {
386:   vertex *e; /* puntatore per memorizzare l'indirizzo del vertice
    da eliminare */
387:
388:   edge *a; /* puntatore per scorrere la lista di adiacenza */
389:
390:   if ( !GrafoVuoto ( G ) )
```

```
391:   {
392:     if ( VerticeValido ( G, u ) )
393:     {
394:       e = G->ver[u];
395:       a = G->ver[u]->next;
396:       if ( G->ver[u]->tipo == 2 ) /* u è un locale devo solo
eliminare
397:                                      il rispettivo LNA */
398:       {
399:         EliminaArcoN_O ( G, u, TrovaHUB ( G, u ) );
400:         G->ver[u] = NULL;
401:       }
402:       else /* u è un HUB */
403:       {
404:         while ( a != NULL )
405:         {
406:           if ( a->tipo == 1 ) /* devo eliminare tutti gli LDA
relativi */
407:           {
408:             EliminaArcoN_O ( G, u, a->key );
409:           }
410:           else /* e tutti i locali ad esso collegati */
411:           {
412:             EliminaVertice ( G, a->key );
413:           }
414:           a = G->ver[u]->next;
415:         }
416:       }
417:       free ( e );
418:       G->ver[u] = NULL;
419:       --G->usati;
420:     }
421:     else
422:     {
423:       printf ( "\nERRORE: impossibile eliminare un vertice se non
esiste\n");
424:       system ( "PAUSE" );
425:     }
426:   }
427:   else
428:   {
429:     printf ( "\nERRORE: impossibile eliminare un vertice in un
grafo vuoto\n");
430:     system ( "PAUSE" );
431:   }
432: }
433:
434:
435: /*
436: ========================
437:    FUNZIONI DI STAMPA
```

```
438: ========================
439: */
440:
441: /* Stampa informazioni sul grafo
442:
443:     INPUT: *G - puntatore al grafo
444: */
445: void StampaGrafo ( graph *G )
446: {
447:   edge *tmp; /* puntatore per scorrere la lista di adiacenza */
448:
449:   int i; /* indice del for */
450:
451:   system ( "CLS" );
452:
453:   if( !GrafoVuoto ( G ) )
454:   {
455:     printf ( "Sono stampati solo gli Aereoporti HUB.\n" );
456:     printf ( "Tra parentesi quadre si indica il tipo di tratta: 1
     se LDA, 2 se LNA.\n" );
457:     printf ( "\nCi sono %d Aereoporti ( HUB e Locali )\n\n", G-
     >usati );
458:     for (i = 0; i < G->nv; i++ )
459:     {
460:       if( G->ver[i] != NULL && G->ver[i]->tipo == 1 )
461:       {
462:         printf ( "%5d ->", G->ver[i]->label );
463:         tmp = G->ver[i]->next;
464:         while ( tmp != NULL )
465:         {
466:           printf ( " %5d-[%d]", G->ver[tmp->key]->label, tmp-
     >tipo );
467:           tmp = tmp->next;
468:         }
469:         printf ( "\n" );
470:       }
471:     }
472:   }
473:   else
474:   {
475:     printf("Grafo Vuoto");
476:   }
477:   printf ( "\n" );
478:
479:   system("PAUSE");
480: }
481:
```

```
 1: /*
 2:     *********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     *           con Algoritmo di Dijkstra          *
 5:     * REALIZZATO DA:                               *
 6:     * Marco Sommella 50/482                        *
 7:     * Giovanni Di Cecca 50/887                     *
 8:     * Virginia Bellino 408/466                     *
 9:     *********************************************
10: */
11:
12: #include <limits.h>
13:
14:
15: /* Elementi dell'heap */
16: typedef struct heap_el
17: {
18:   int pos; /* indice posizione nell'array di vertici del grafo */
19:   int dist; /* distanza */
20:   int s; /* flag per Dijkstra */
21: } heap_el;
22:
23:
24: /* Rappresentazione di un heap */
25: typedef struct heap
26: {
27:   int size; /* dimensione dell'heap */
28:   heap_el *data; /* array di elementi dell'heap */
29: } heap;
30:
31:
32: /* Prototipi delle funzioni */
33: int left(int i);
34: int right(int i);
35: int p(int i);
36: void swap ( heap *q, int i, int j );
37: void Heapify ( heap *q, int i );
38: int EmptyHeap ( heap *q );
39: int GetMin ( heap *q );
40: int TrovaCorr (graph *G, heap *q, int start );
41: int *Dijkstra ( graph *G, int start );
42: void ControllaPercorsi ( graph *G );
43:
```

```
 1: /*
 2:     ***********************************************
 3:     * SIMULAZIONE COLLEGAMENTI AEREI RETE MONDIALE *
 4:     *           con Algoritmo di Dijkstra          *
 5:     * REALIZZATO DA:                               *
 6:     * Marco Sommella 50/482                        *
 7:     * Giovanni Di Cecca 50/887                     *
 8:     * Virginia Bellino 408/466                     *
 9:     ***********************************************
10: */
11:
12: /* heap con la regole che il padre è minore dei figli */
13: #include "heap.h"
14:
15:
16: /*
17: =========================
18:    Funzioni dello HEAP
19: =========================
20: */
21:
22: /*  Figlio sinistro
23:
24:     INPUT:  i - indice della radice
25:     OUTPUT: ritorna il figlio sinistro di i
26: */
27: int left(int i)
28: {
29:   return 2*i+1;
30: }
31:
32:
33: /*  Figlio destro
34:
35:     INPUT:  i - indice della radice
36:     OUTPUT: ritorna il figlio destro di i
37: */
38: int right(int i)
39: {
40:   return 2*i+2;
41: }
42:
43:
44: /* Padre
45:
46:     INPUT:  i - indice della radice
47:     OUTPUT: ritorna il padre di i
48: */
49: int p(int i)
50: {
51:   return (i-1)/2;
```

```
 52: }
 53:
 54:
 55: /* Scambia la posizione nell'heap di 2 elementi
 56:
 57:     INPUT:  *q - puntatore all'heap
 58:             i,j - indici degli elementi da scambiare
 59:     OUTPUT: modifica l'heap
 60: */
 61: void swap ( heap *q, int i, int j )
 62: {
 63:   heap_el tmp = q->data[i];
 64:   q->data[i] = q->data[j];
 65:   q->data[j] = tmp;
 66: }
 67:
 68:
 69: /* Ripristina la condizione di heap
 70:
 71:     INPUT:  *q - puntatore all'heap
 72:             i - radice del sotto-albero dal quale far partire il
    controllo
 73:     OUTPUT: modifica l'heap
 74: */
 75: void Heapify ( heap *q, int i )
 76: {
 77:   int l,r,min; /* varibili per memorizzare i figli e il minimo */
 78:   l = left(i);
 79:   r = right(i);
 80:   if ( l < q->size && q->data[l].dist < q->data[i].dist )
 81:     min = l;
 82:   else
 83:     min = i;
 84:   if ( r < q->size && q->data[r].dist < q->data[min].dist )
 85:     min = r;
 86:   if ( min != i )
 87:   {
 88:     swap ( q, i, min );
 89:     Heapify ( q, min );
 90:   }
 91: }
 92:
 93:
 94: /*  Costruisce l'heap
 95:
 96:     INPUT:  *q - puntatore all'heap
 97:     OUTPUT: modifica l'heap
 98: */
 99: void BuildHeap ( heap *q )
100: {
101:   int i; /* indice per il for */
```

```
102:   for (i=q->size/2; i>=0; i--)
103:     Heapify ( q, i);
104: }
105:
106:
107: /* Controlla se l'heap è vuoto
108:
109:     INPUT:   *q - puntatore all'heap
110:     OUTPUT: ritorna vero se vuoto
111: */
112: int EmptyHeap ( heap *q )
113: {
114:   return !( q->size >= 1 );
115: }
116:
117:
118: /*  Estrae il minimo dall'heap
119:
120:     INPUT:   *q - puntatore all'heap
121:     OUTPUT: ritorna il minimo estratto
122: */
123: int GetMin ( heap *q )
124: {
125:   int tmp = q->data[0].pos;
126:   q->data[0].s = 1;
127:   swap ( q, 0, --q->size );
128:   Heapify( q, 0 );
129:   return tmp;
130: }
131:
132:
133: /*  Trova la posizione corrispondente ad un elemento del heap nel
     grafo
134:
135:     INPUT:   *G - puntatore al grafo
136:              *q - puntatore all'heap
137:              start - valore cercato
138:     OUTPUT: ritorna la posizione
139: */
140: int TrovaCorr (graph *G, heap *q, int start )
141: {
142:   int i; /* indice per il for */
143:   for ( i = 0; i < G->usati; i++ )
144:   {
145:     if ( q->data[i].pos == start )
146:     {
147:       return i;
148:     }
149:   }
150: }
151:
```

```
152:
153: /*
154: ============================
155:    Algoritmo di Dijkstra
156: ============================
157: */
158: /*  Algoritmo di Dijkstra per il percorso minimo
159:
160:     INPUT:  *G - puntatore al grafo
161:             start - punto di partenza
162:     OUTPUT: puntatore ad array con le distanze
163: */
164: int *Dijkstra ( graph *G, int start )
165: {
166:   int i, j = 0, /* indici */
167:       u; /* memorizza il minimo */
168:   edge *a; /* puntatore per scorrere la lista di adiacenza */
169:
170:                    /* init */
171:   heap *q = ( heap * ) malloc ( sizeof ( heap ) );
172:
173:   q->data = ( heap_el * ) calloc ( G->usati, sizeof ( heap_el ) );
174:
175:   q->size = G->usati;
176:
177:   for ( i = 0; i < G->nv; i++ ) /* scrive i valori da considerare
     */
178:   {
179:     if ( G->ver[i] != NULL )
180:     {
181:       q->data[j].pos = i;
182:       if ( i == start )
183:       {
184:         q->data[j].dist = 0;
185:         q->data[j].s = 1;
186:       }
187:       else
188:       {
189:         q->data[j].dist = INT_MAX;
190:         q->data[j].s = 0;
191:       }
192:       j++;
193:     }
194:   }
195:
196:   BuildHeap ( q );
197:
198:   while ( EmptyHeap ( q ) );
199:   {
200:     u = GetMin ( q );
201:     a = G->ver[u]->next;
```

```
202:     i = TrovaCorr ( G, q, u );
203:
204:     while ( a != NULL ) /* RELAX */
205:     {
206:       j = TrovaCorr ( G, q, a->key );
207:       if ( q->data[j].s == 0)
208:       {
209:         if ( q->data[i].dist + 1 < q->data[j].dist )
210:         {
211:           q->data[j].dist = q->data[i].dist + 1;
212:         }
213:       }
214:       a = a->next;
215:     }
216:   }
217:
218:   /* trascrive le distanze nel array di ritorno */
219:   int *dist = ( int * ) calloc ( G->nv , sizeof ( int ) );
220:
221:   for ( i = 0; i < G->usati; i++ )
222:   {
223:     dist[q->data[i].pos] = q->data[i].dist;
224:   }
225:
226:   free (q->data);
227:
228:   free ( q );
229:
230:   return dist;
231: }
232:
233:
234: /*  Verifica la condizione di distanza tra 2 vertici usando
    Dijkstra
235:
236:     INPUT:   *G - puntatore al grafo
237:     OUTPUT: modifica il grafo G se necessario
238: */
239: void ControllaPercorsi ( graph *G )
240: {
241:   int i, j, k, z=-1;
242:
243:   int *dist;
244:
245:   for ( i = 0; i < G->nv; i++ )
246:   {
247:     if ( G->ver[i] != NULL )
248:     {
249:       dist = Dijkstra ( G, i );
250:       if ( z == -1 )
251:         z = i;
```

```
252:        for ( j = 0; j < G->nv; j++ )
253:        {
254:          if ( dist[j] > 4 && G->ver[j] != NULL )
255:          {
256:            if ( G->ver[z]->tipo == 2 )
257:            {
258:              z = TrovaHUB ( G, z );
259:            }
260:            if ( G->ver[j]->tipo == 2 )
261:            {
262:              k = TrovaHUB ( G, j );
263:            }
264:            else
265:            {
266:              k = j;
267:            }
268:            if ( z != k )
269:            {
270:              InserisciArcoN_O ( G, z, k );
271:            }
272:          }
273:        }
274:      }
275:    }
276: }
277:
```