Problem solving avanzato

Esercizi risolti

1 Esercizio: “Cruci-Puzzle”

Si realizzi un programma C in grado di risolvere il gioco enigmistico “*cruci-puzzle”* in base alle specifiche di seguito riportate:

- un primo file contiene una matrice di caratteri quadrata le cui dimensioni sono date dal numero di caratteri presenti nella prima riga
- un secondo file contiene un elenco di parole (una parola su ciascuna riga)
- si ricerchino all’interno della matrice tutte le parole nelle quattro possibili direzioni (orizzontale, verticale, diagonale e diagonale inversa) e nei due versi possibili lungo ciascuna direzione (da sinistra a destra e viceversa, dall’alto in basso e viceversa e così via)
- si memorizzi in un terzo file l’insieme e la “posizione” delle parole rintracciate.

Esempio

Il file contenente la matrice di caratteri sia il seguente:

```
xxxxxxtxxx
xxpippoxxx
xxxxxxpxxx
xxxxxxoxxx
xxxxxxlxxx
pxxxxxiuqx
xlxxxxnxxx
xxuxxxoxxx
xxxtxxxxxx
xxxxoxxxxx
```

e la seconda matrice è quella indicata nel seguito:

1	27
36	1

In essa sono state graficamente evidenziate le seguenti parole contenute nel secondo file:

```
pippo
topolino
pluto
qui
```

Il file in uscita deve avere il seguente contenuto:

```
      t
  pippo
      p
      o
      l
p     iuq
 l    n
  u   o
   t
    o
```

in cui tutti i caratteri “inutili” (solo nel nostro esempio e per comodità sono state inserite delle x) sono stati sostituiti da spazi.

Soluzione

Il problema consta di un sotto-problema di ricerca/confronto e di uno di gestione di matrici. In particolare, il problema di ricerca e confronto va realizzato da funzioni specifiche, per renderlo idoneo a confronti (carattere per carattere) su 4 direzioni nella matrice.

La struttura dati richiede

- due matrici di caratteri, per immagazzinare la matrice acquisita in input e quella elaborata per l'output.
- Una struttura dati per le stringhe acquisite dal secondo file. L'algoritmo proposto eleborerà ogni stringa non appena acquisita da file. Non è quindi necessaria una struttura dati in grado di immagazzinare tutte le stringhe, ma è sufficiente un solo vettore di caratteri.

L'algoritmo proposto per risolvere il problema può essere riassunto come segue:

- si leggono i nomi dei file e si aprono i file rispettivamente in lettura e in scrittura
- si legge il file contenente la matrice di caratteri memorizzandone il contenuto in una matrice
- si effettua un ciclo di lettura del secondo file, contenente la lista delle parole, e si ricerca contestualmente ciascuna parola nella matrice secondo procedimento seguente:
  - si confronta il primo carattere della parola con i vari caratteri della matrice (muovendosi lungo quest'ultima su righe e colonne)
  - se i due caratteri sono
    - diversi: si passa a confrontare il primo carattere della parola con il successivo carattere della matrice sino a "percorrere" tutta la matrice
    - uguali: esiste la possibilità di rintracciare la parola a partire da quella posizione negli otto versi possibili. Si incomincia quindi una ricerca lungo tali versi (verso destra, sinistra, alto, basso, alto a destra, alto a sinistra, basso a sinistra e basso a destra). Il sotto-problema di ricerca si basa sull'utilizzo di 8 versori, ciascuno a due componenti, individuanti gli 8 versi e nel muoversi poi in tali direzioni di una quantità crescente e pari, al massimo, al numero di caratteri della parola che si sta ricercando; se la parola viene rintracciata la si ricopia in una seconda matrice (di output)
- si ripete con la parola successiva sino alla terminazione del file contenente le parole
- si copia la matrice di output nel file di output
- si chiudono i file.

Il tutto, a livello di codice, può essere semplificato mediante l'utilizzo di funzioni che garantiscano una sufficiente modularità.

```
#include <stdio.h>
#include <string.h>

#define   L_NOME     20
#define   DIM_MAT    10
#define   MAX_LINE   80

/* prototipi funzioni */
void rintraccia1 (char nome[], char mat1[][DIM_MAT], char mat2[][DIM_MAT]);
void rintraccia2 (int row,int col, char nome[],
                  char mat1[][DIM_MAT], mat2[][DIM_MAT]);
FILE *apriFile (char *nome, char *modo);

/* programma principale */
void main (void)
{
  char c;
```

```
  char nome1[L_NOME],           /* nome file 1 */
       nome2[L_NOME],           /* nome file 2 */
       nome3[L_NOME];           /* nome file 3 */
  char nome[DIM_MAT];           /* parole da ricercare */
  char mat1[DIM_MAT][DIM_MAT],  /* matrice letta */
       mat2[DIM_MAT][DIM_MAT];  /* matrice generata e scritta */
  int i, j;
  FILE *fp1, *fp2, *fp3;

  printf ("Nome file matrice : ");
  scanf ("%s", nome1);
  printf ("Nome file nomi    : ");
  scanf ("%s", nome2);
  printf ("Nome file output  : ");
  scanf ("%s", nome3);

  /* Lettura della matrice */
  fp1 = apriFile (nome1, "r");
  i = j = 0;
  while (fscanf (fp1, "%c", &c) != EOF) {
    if (c != '\n') {
      mat1[i][j] = c;
      j++;
      if (j>=DIM_MAT) {
         j=0; i++;
      }
    }
  }
  fclose (fp1);

  /* Elaborazione */
  fp2 = apriFile (nome2, "r");

  /* inizializzazione matrice da "generare" */
  for (i=0; i<DIM_MAT; i++)
    for (j=0; j<DIM_MAT; j++)
      mat2[i][j] = ' ';

  /* lettura di una parola e immediata ricerca */
  while (fgets (nome, MAX_LINE, fp2) != NULL)
    rintraccia1 (nome, mat1, mat2);

  /* scrittura file finale */
  fp3 = apriFile (nome3, "w");

  for (i=0; i<DIM_MAT; i++) {
    for (j=0; j<DIM_MAT; j++)
      fprintf (fp3, "%c", mat2[i][j]);
    fprintf (fp3, "\n");
  }

  fclose (fp2);
  fclose (fp3);
}
```

```
/* Ricerca il primo carattere della parola all'interno della
   matrice mat1; ogni volta che il confronto ha un esito
   positivo esiste la possibilità rintracciare la parola
   e si richiama la funzione rintraccia2                        */
void rintraccia1 (char nome[], char mat1[][DIM_MAT],
                  char mat2[][DIM_MAT]);
{
  int i, j;

  for (i=0; i<DIM_MAT; i++)
    for (j=0; j<DIM_MAT; j++)
      if (mat1[i][j] == nome[0])
        rintraccia2 (i, j, nome, mat1, mat2);
}

/* Ricerca negli 8 versi possibili la parola nome; il vettore
   vetoff definisce gli 8 versori, di componenti:
   (vetoff[0][0], vetoff[1][0]), (vetoff[0][1], vetoff[1][1]),
   (vetoff[0][2], vetoff[1][2]), ...                            */
void rintraccia2 (int row, int col, char nome[],
                  char mat1[][DIM_MAT],
                  char mat2[][DIM_MAT])
{
  char flag;
  int r, c, i, j;
  /* Definizione dei versori:
     (0, 1) = in alto, (-1, 1) = sinitra in alto,
     (-1, 0) = sinistra, (-1, -1) = sinistra in basso,
     (0, -1) = basso, (1, -1) = destra in basso,
     (1, 0) = destra, (1, 1) = destra in alto          */
  int vetoff[2][8] = { 0, -1, -1, -1,  0,  1,  1,  1,
                       1,  1,  0, -1, -1, -1,  0,  1 };

  for (i=0; i<=7; i++) {
    flag = 1;
    for (j=1; (j<(strlen(nome)-1)) && flag; j++) {
      /* calcolo dello spostamento lungo righe, r, e colonne, c,
         dato dalla posizione attuale, row e col, dal modulo del
         vettore, j, e dai versori */
      r = row + j * vetoff[0][i];
      c = col + j * vetoff[1][i];

      if (r>=0 && r<DIM_MAT && c>=0 && c<DIM_MAT) {
         if (mat1[r][c] != nome[j])
           flag = 0;
      }
      else
        flag = 0;
    }

    if (flag) {
      for (j=0; j<(strlen(nome)-1) ; j++)
        mat2[row+j*vetoff[0][i]][col+j*vetoff[1][i]] =
        mat1[row+j*vetoff[0][i]][col+j*vetoff[1][i]];
    }
  }
}
```

```
/* funzione per aprire file,
   in caso di errore stampa messaggio e interrompe esecuzione */
FILE *apriFile (char *nome, char *modo) {
  FILE *fp = fopen (nome, modo);
  if (fp == NULL) {
    printf (" FILE %s NON APRIBILE !!! \n");
    /* termina l'esecuzione del programma ... */
    exit;
  }
  return fp;
}
```

Per concludere si osservi quanto segue:

- il meccanismo di lettura proposto è piuttosto particolare, nel senso che la lunghezza della prima riga del file, contenente la matrice, fissa il numero di colonne e di righe della matrice stessa. Tale meccanismo è robusto nel senso che gli "a capo" vengono saltati e, ad esempio, un file del tipo:

  ```
  xxxxxxtxxx
  xxp
  ippoxxx
  xxx
  xxxpxxx
  xxxxxxoxxx
  x
  xxxxxlxxx
  pxxxxxixxx
  xlxxx
  xnxxx
  xxuxxxoxxx
  xxxtxxxxxx
  x
  xxxoxxxxx
  ```

  fornisce una matrice identica a quello analizzata nell'esempio iniziale.

- Piccole modifiche sono sufficienti per ottenere formati diversi; si potrebbe ad esempio supporre che la prima riga del file riporti numero di righe e di colonne della matrice.

- La dimensione del file e la lunghezza delle parole è limitata dalla definizione della costante `DIM_MAT`.

2 Esercizio: "Voli aerei"

Si scriva un programma C per la gestione di un sistema di informazione automatizzato relativo a voli nazionali. La rete di voli è memorizzata in un file contenente, su ciascuna riga, le informazioni relative a ogni singolo volo organizzate nel seguente modo:

```
<sigla aeroporto_di_partenza>
<sigla aeroporto_di_arrivo>
<sigla_del_volo>
<ora_di_partenza>
<ora_di_arrivo>
```

Le sigle degli aeroporti di partenza e di arrivo sono ciascuna di 3 lettere. La sigla del volo è memorizzata su 5 caratteri (2 lettere e 3 numeri). Le ore di partenza e di arrivo sono memorizzate nel formato `hh:mm`.

Il programma deve eseguire le seguenti operazioni:

- leggere da tastiera richieste di volo, espresse mediante aeroporto di partenza, aeroporto di arrivo e ora desiderata di partenza, in formato analogo a quello del file.

- ricercare, per ognuna delle richieste, tutti i voli che partono dall'aeroporto di partenza indicato dopo l'ora indicata e arrivano all'aeroporto selezionato
- visualizzare tutti i voli che soddisfano alla condizione precedente.

Deve essere tenuta in conto la possibilità di (al più) uno scalo in un aeroporto intermedio.

Soluzione

Si tratta di un problema di ricerca, cui si aggiunge la complicazione di accettare eventuali voli con scalo. Siccome non si hanno dati sulla dimensione del file in ingresso, la struttura dati per i voli disponibili può essere realizzata mediante vettore dinamico (con doppia lettura del file, una prima per contare i dati e una seconda per acquisirli) oppure una lista.

Verrà proposto nel seguito un programma particolarmente semplice, facente uso di una lista di elementi, ciascuno dei quali contiene le informazioni relative a un singolo volo. Il programma risulta composto dalle seguenti fasi:

1. lettura del file e creazione della lista; ogni riga letta coincide con la creazione di un nuovo elemento; la lista viene gestita nella maniera più semplice possibile: inserzione in testa dell'elemento letto
2. vengono quindi acquisiti i dati relativi a città di partenza, di arrivo e orario del volo richiesto; l'introduzione, come città di partenza, della parola "fine", provoca la terminazione del programma
3. si ricercano nella lista i voli soddisfacenti alla richiesta con al più uno scalo intermedio
4. si ripete il procedimento a partire dal passo 2.

```
#include <stdio.h>
#include <string.h>

#define   MAX_NOME   20
#define   MAX_LINE   50

struct volo_s
{
  char ap[4];       /* aereoporto di partenza */
  char aa[4];       /* aereoporto di arrivo */
  char si[6];       /* sigla del volo */
  char op[6];       /* ora di partenza */
  char oa[6];       /* ora di arrivo */
  struct volo_s *p; /* puntatore al successivo elemento
                       della lista */
};

/* dichiarazione funzione per la stampa di un volo */
void stampo (struct volo_s *p);

void main (void)
{
  char linea[MAX_LINE];
  char nome[MAX_NOME], vap[4], vaa[4], scalo[4], orascalo[6], vop[6];
  struct volo_s *ptesta, *pe1, *pe2;
  FILE *fp;

  /* Lettura Nome file */
  printf ("Nome file : ");
  scanf ("%s", nome);

  /* Apertura del file */
  if ((fp = fopen(nome, "r")) == NULL) {
    printf ("FILE NON APRIBILE !!! \n");
    return; /* terminazione del programma */
  }

  /* Lettura del file creando una lista */
  ptesta = NULL;
  while (fgets (linea, MAX_LINE, fp) != NULL) {
```

```
    /* creazione di un nuovo elemento della lista */
    pe1 = malloc (sizeof (struct volo_s));

    /* assegnazione dei campi del nuovo elemento */
    sscanf (“%s %s %s %s %s”, pe1->ap, pe1->aa, pe1->si, pe1->op, pe1->oa);

    pe1->p = ptesta;
    ptesta = pe1;
  }

  /* Chiusura file */
  fclose (fp);

  while (!fine) {
    /* Lettura del volo richiesto */
    printf ("\n\n########################################\n");

    printf ("Citta` di partenza : "); scanf ("%s", vap);
    printf ("Citta` di arrivo   : "); scanf ("%s", vaa);
    printf ("Orario             : "); scanf ("%s", vop);

    fine = strcmp(va,”fine”)==0;

    /* Ricerca volo */
    if (!fine)
    {
      pe1 = ptesta;
      while ( pe1!=NULL) {
        if (strcmp(pe1->ap, vap)==0 && strcmp(pe1->op, vop)>=0) {
          /* trovato volo da città di partenza con orario adatto */
          if (strcmp(pe1->aa, vaa)==0) {
            /* Trovato volo diretto: stampo informazioni */
            stampo (pe1); printf ("\n");
          }
          else {
            /* Puo` esistere un volo NON diretto: ricerca annidata */
            strcpy (scalo, pe1->aa);
            strcpy (orascalo, pe1->oa);
            pe2 = ptesta;
            while (pe2!=NULL) {
              if (strcmp(pe2->ap, scalo)==0
               && strcmp (pe2->aa, vaa)==0
               && strcmp (pe2->op, orascalo)>0 ) {
                /* Volo NON diretto trovato */
                stampo (pe1);
                printf ("   --- SCALO --- \n");
                stampo (pe2);
                printf ("\n");
              }
              pe2 = pe2->p;
            }
          }
        }
        pe1 = pe1->p;
      }
    }
  }
}

/* Stampa per le informazioni di un volo */
void stampo (struct volo_s *p)
{
  printf ("%s %s %s %s %s", p->ap, p->aa, p->si, p->op, p->oa);
}
```

Supponendo che il formato iniziale del file sia il seguente:

```
tor mil tm100 07:00 07:30
tor mil tm200 13:00 13:30
tor mil tm300 19:00 19:30
tor rom tr101 07:00 08:00
tor rom tr201 13:00 14:00
tor rom tr301 19:00 20:00
tor rom tr401 22:00 23:00
mil rom mr102 07:30 08:25
mil rom mr202 11:30 12:25
mil rom mr302 13:30 14:25
mil rom mr402 15:30 16:25
mil rom mr502 17:30 17:25
mil rom mr602 19:30 20:25
mil rom mr702 21:30 22:25
mil rom mr802 23:30 24:25
```

si presenta un esempio di esecuzione del programma stesso in cui si effettuano diverse ricerche:

```
########################################
Citta` di partenza : tor
Citta` di arrivo   : mil
Orario             : 07:00
tor mil tm300 19:00 19:30
tor mil tm200 13:00 13:30
tor mil tm100 07:00 07:30

########################################
Citta` di partenza : tor
Citta` di arrivo   : mil
Orario             : 15:00
tor mil tm300 19:00 19:30

########################################
Citta` di partenza : mil
Citta` di arrivo   : rom
Orario             : 16:00
mil rom mr802 23:30 24:25
mil rom mr702 21:30 22:25
mil rom mr602 19:30 20:25
mil rom mr502 17:30 17:25

########################################
Citta` di partenza : tor
Citta` di arrivo   : rom
Orario             : 21:00
tor rom tr401 22:00 23:00

########################################
Citta` di partenza : tor
Citta` di arrivo   : rom
Orario             : 13:00
tor rom tr401 22:00 23:00
tor rom tr301 19:00 20:00
tor rom tr201 13:00 14:00
tor mil tm300 19:00 19:30    --- SCALO ---
mil rom mr802 23:30 24:25
tor mil tm300 19:00 19:30    --- SCALO ---
mil rom mr702 21:30 22:25
tor mil tm200 13:00 13:30    --- SCALO ---
mil rom mr802 23:30 24:25
tor mil tm200 13:00 13:30    --- SCALO ---
mil rom mr702 21:30 22:25
tor mil tm200 13:00 13:30    --- SCALO ---
mil rom mr602 19:30 20:25
tor mil tm200 13:00 13:30    --- SCALO ---
mil rom mr502 17:30 17:25
tor mil tm200 13:00 13:30    --- SCALO ---
mil rom mr402 15:30 16:25
```

```
########################################
Citta` di partenza : fi
Citta` di arrivo   : ne
Orario             : pprpr
```