```
/*
Corso di Fondamenti di Informatica
Esercizio: tema d'esame (modificato).
*/

/*
Si vuole scrivere un programma di pianificazione di percorsi da un punto di partenza a un punto di arrivo in una mappa
bidimensionale quadrata contenente ostacoli. Il punto di partenza e il punto di arrivo sono scelti dall'utente e non possono
trovarsi in elementi della mappa occupati da ostacoli.

La mappa Ã¨ rappresentata da un array di 100 x 100 elementi di tipo intero, i cui elementi corrispondono alle possibili
posizioni nella mappa. Il valore 100 Ã¨ rappresentato da un parametro chiamato DIM. Il numero intero contenuto in ciascun
elemento dell'array rappresenta una misura di distanza dal punto di arrivo. Alcune posizioni nella mappa sono occupate da
ostacoli: tali posizioni saranno caratterizzate convenzionalmente da un valore negativo pari a -1, identificato da un
parametro chiamato OSTACOLO.
Due posizioni nella mappa si dicono vicine se hanno lo stesso indice di riga e indici di colonna consecutivi, oppure lo
stesso indice di colonna e gli indici di riga consecutivi.

La tecnica che permette la pianificazione di un percorso da un punto di partenza a un punto di arrivo consiste in tre fasi:
1. Lâ€™inizializzazione degli elementi della mappa, 2. la preparazione degli elementi della mappa in base al punto di
arrivo, e 3. la costruzione del percorso dal punto di partenza.

1.  Durante la prima fase, per ciascuno degli elementi che corrispondono a posizioni occupate da ostacoli il valore intero
viene posto a OSTACOLO, mentre per tutti gli altri elementi il valore viene inizializzato a DIM*DIM. Questo valore Ã¨
identificato da un parametro chiamato MAX.

2.  La seconda fase Ã¨ costituita da una serie di passi, durante il primo dei quali il valore associato al punto di arrivo
del percorso viene posto pari a 0. Nel successivo passo si considerano tutti gli elementi vicini allâ€™elemento di valore 0
che non siano occupati da ostacoli, e per ciascuno degli elementi considerati si modifica il valore ponendolo a 1. Nel terzo
passo si selezionano tutti gli elementi che soddisfano entrambe le condizioni seguenti: (i) che abbiano un vicino di valore
1; (ii) il cui valore sia maggiore di 1 (in questo modo si scartano sia gli elementi occupati da ostacoli sia gli elementi
giÃ  modificati al passo precedente). Per gli elementi cosÃ¬ selezionati il valore viene modificato ponendolo a 2.
In generale, si procede per passi successivi nel modo appena descritto. Detto k il valore assegnato agli elementi nel passo
precedente, il passo corrente considera tutti gli elementi (i) che abbiano un vicino di valore k, e (ii) il cui valore sia
maggiore di k: agli elementi cosÃ¬ considerati viene assegnato un nuovo valore pari a k+1.
La seconda fase termina quando non vi sono piÃ¹ nuovi elementi che soddisfino entrambe le condizioni.

3. Nella terza fase si considera il punto di partenza e il valore contenuto nell'elemento corrispondente della mappa. Si
costruisce un percorso esaminando di volta in volta i vicini del punto corrente, e selezionando come prossimo punto uno
qualsiasi dei suoi vicini il cui valore sia inferiore al valore dellâ€™elemento corrente. Il percorso termina quando si
giunge al punto di valore 0. Per costruzione, se il punto di partenza ha un valore diverso da OSTACOLO e da MAX al termine
```

della seconda fase, esiste almeno un vicino del punto corrente il cui valore Ã¨ inferiore a quello del punto corrente (a meno che non sia stato giÃ raggiunto il punto di arrivo).
Per indicare nella mappa il percorso dal punto di partenza al punto di arrivo, si sovrascrivono i valori delle celle appartenenti al percorso con un valore negativo pari a -88, rappresentato da un parametro chiamato PERCORSO.

Si scriva un programma C implementi la seconda e la terza fase della pianificazione su una mappa predefinita. Le coordinate dei punti di arrivo e di partenza vanno richieste all'utente.
*/

```
#include <stdio.h>
#define DIM 5
/* lunghezza del lato della mappa, in numero di elementi */

const int MAX = DIM*DIM;
/* valore iniziale dato agli elementi non elaborati della mappa */
const int OSTACOLO = -1;
/* valore che indica la presenza di un ostacolo in una cella della mappa. PerchÃ© l'algoritmo funzioni correttamente, deve
essere negativo e diverso da PERCORSO. */
const int PERCORSO = -88;
/* valore usato per indicare che un elemento della mappa e' parte del percorso dal punto di partenza al punto di arrivo.
PerchÃ© l'algoritmo funzioni correttamente, deve essere negativo e diverso da OSTACOLO. */

void StampaMappa(int MappaDaStamp[DIM][DIM]);
/* stampa a schermo il contenuto di una mappa (operazione non richiesta all'esame) */

int main()
{
  typedef enum {false, true} boolean;

  int Mappa[DIM][DIM];
  /* la mappa */
  int Col, Riga;
  /* indici della riga e colonna della mappa attualmente in esame */
  int ColArrivo, RigaArrivo;
  /* indici della riga e colonna del punto di arrivo */
  int ValoreCorrente;
  /* il valore di distanza dal punto di arrivo attualmente considerato, durante
     la fase di preparazione della mappa */
  boolean FinePreparazione;
  /* durante la fase di preparazione della mappa, il valore false di questa
     variabile indica che la fase non e' ancora terminata */
```

```
int DistanzaCorrente;
/* il valore di distanza dal punto di arrivo attualmente considerato, durante
   la fase di determinazione del percorso */

/*** Fase 1: inizializzazione della mappa (non richiesta all'esame) ***/
for (Riga = 0; Riga < DIM; ++Riga)
{
  for (Col = 0; Col < DIM; ++Col)
  {
    Mappa[Riga][Col] = MAX;
  }
}

/* inserimento di alcuni ostacoli a titolo di prova */
for (Riga = 0; Riga < DIM; ++Riga)
{
  for (Col = 0; Col < DIM; ++Col)
  {
    if ( (0 == (Col+1)%(2*(Riga+1))) || (0 == (Riga+1)%(2*(Col+1))) )
    /* nota: aggiungere 1 evita le divisioni per 0 */
    {
      Mappa[Riga][Col] = OSTACOLO;
    }
  }
}

printf("\nMappa originale con ostacoli:");
StampaMappa(Mappa);

/*** Fase 2: inserimento punto di arrivo e preparazione mappa ***/

printf("\nRiga [0...%d] del punto di arrivo sulla mappa? ", DIM-1);
scanf("%d", &RigaArrivo);
printf("Colonna [0...%d] del punto di arrivo sulla mappa? ", DIM-1);
scanf("%d", &ColArrivo);

Mappa[RigaArrivo][ColArrivo] = 0;
ValoreCorrente = 0;

FinePreparazione = false;
while (false == FinePreparazione)
```

```
/* esplora la mappa alla ricerca di elementi di valore superiore a
   ValoreCorrente che siano adiacenti ad un elemento di valore ValoreCorrente;
   quando ne trova uno, lo pone a ValoreCorrente+1. Il processo termina
   quando viene eseguita una iterazione senza modificare alcun elemento
   della mappa */
{
  FinePreparazione = true;

  for (Riga = 0; Riga < DIM; ++Riga)
  {
    for (Col = 0; Col < DIM; ++Col)
    {
      if ( Mappa[Riga][Col] > ValoreCorrente )
      {
        /* se la cella va considerata */
        if ( ( (Riga > 0) && (Mappa[Riga-1][Col] == ValoreCorrente) ) ||
             ( (Riga < DIM-1) && (Mappa[Riga+1][Col] == ValoreCorrente) ) ||
             ( (Col > 0) && (Mappa[Riga][Col-1] == ValoreCorrente) ) ||
             ( (Col < DIM-1) && (Mappa[Riga][Col+1] == ValoreCorrente) ) )
        /* se una delle celle della mappa adiacenti a quella di posizione
           [Riga, Col] contiene il valore ValoreCorrente; notare l'ordine,
           che non puo' essere invertito, delle due condizioni in && che
           compaiono in ciascuna delle 4 linee dell'istruzione if  */
        {
          Mappa[Riga][Col] = ValoreCorrente + 1;
          FinePreparazione = false;
        }
      }
    }
  }
  ValoreCorrente = ValoreCorrente+1;
}

printf("\nMappa con punto di arrivo e distanze:");
StampaMappa(Mappa);


/*** Fase 3: costruzione del percorso dal punto di partenza ***/
printf("\nRiga [0...%d] del punto di partenza sulla mappa? ", DIM-1);
scanf("%d", &Riga);
printf("Colonna [0...%d] del punto di partenza sulla mappa? ", DIM-1);
```

```
scanf("%d", &Col);

while ( (Riga != RigaArrivo) || (Col != ColArrivo) )
{
  DistanzaCorrente = Mappa[Riga][Col];
  Mappa[Riga][Col] = PERCORSO;

  if ( (Riga > 0) && (Mappa[Riga-1][Col] >= 0) &&
       (Mappa[Riga-1][Col] < DistanzaCorrente ) )
  /* se esiste un elemento della mappa di valore inferiore a DistanzaCorrente
     immediatamente al di sotto di quello considerato; notare che la
     condizione componente (Riga > 0) deve comparire necessariamente per prima
     nella condizione composta dell'istruzione if */
  {
    Riga = Riga-1;
  }
  else if ( (Riga < DIM-1) && (Mappa[Riga+1][Col] >= 0) &&
            (Mappa[Riga+1][Col] < DistanzaCorrente ) )
  /* se esiste un elemento della mappa di valore inferiore a DistanzaCorrente
     immediatamente al di sopra di quello considerato; notare che la
     condizione componente (Riga < DIM-1) deve comparire necessariamente per
     prima nella condizione composta dell'istruzione if */
  {
    Riga = Riga+1;
  }
  else if ( (Col > 0) && (Mappa[Riga][Col-1] >= 0) &&
            (Mappa[Riga][Col-1] < DistanzaCorrente ) )
  /* se esiste un elemento della mappa di valore inferiore a DistanzaCorrente
     immediatamente a sinistra di quello considerato; notare che la
     condizione componente (Col > 0) deve comparire necessariamente per
     prima nella condizione composta dell'istruzione if */
  {
    Col = Col-1;
  }
  else if ( (Col < DIM-1) && (Mappa[Riga][Col+1] >= 0) &&
            (Mappa[Riga][Col+1] < DistanzaCorrente ) )
  /* se esiste un elemento della mappa di valore inferiore a DistanzaCorrente
     immediatamente a destra di quello considerato; notare che la
     condizione componente (Col < DIM-1) deve comparire necessariamente per
     prima nella condizione composta dell'istruzione if */
  {
```

```
      Col = Col+1;
    }
  }

  printf("\nMappa con percorso da punto di partenza a punto di arrivo:");
  StampaMappa(Mappa);

  printf("\n\n");

  return 0;
}

void StampaMappa(int MappaDaStamp[][DIM])
{
  const char OST = '-';
  /* carattere stampato a schermo per indicare la presenza di un ostacolo */
  const char PERC = '*';
  /* carattere stampato a schermo per indicare un elemento del percorso */
  int Riga, Colonna;
  /* identificano l'elemento correntemente in stampa */

  for (Riga = 0; Riga < DIM; ++Riga)
  {
    printf("\n");
    for (Colonna = 0; Colonna < DIM; ++Colonna)
    {
      if (OSTACOLO == MappaDaStamp[Riga][Colonna])
      {
        printf("\t%c", OST);
      }
      else if (PERCORSO == MappaDaStamp[Riga][Colonna])
      {
        printf("\t%c", PERC);
      }
      else
      {
        printf("\t%d", MappaDaStamp[Riga][Colonna]);
      }

    }
  }
```

```
  printf("\n");
}

/* Esempio di esecuzione del programma con mappa di dimensioni 5x5:

Mappa originale con ostacoli:
    25    -    25    -    25
    -    25    25    -    25
    25    25    25    25    25
    -    -    25    25    25
    25    25    25    25    25

Riga [0...4] del punto di arrivo sulla mappa? 0
Colonna [0...4] del punto di arrivo sulla mappa? 2

Mappa con punto di arrivo e distanze:
    25    -    0    -    6
    -    2    1    -    5
    4    3    2    3    4
    -    -    3    4    5
    6    5    4    5    6

Riga [0...4] del punto di arrivo sulla mappa? 0
Colonna [0...4] del punto di arrivo sulla mappa? 4

Mappa con percorso da punto di partenza a punto di arrivo:
    25    -    0    -    *
    -    2    *    -    *
    4    3    *    *    *
    -    -    3    4    5
    6    5    4    5    6

E' interessante notare che la cella di coordinate [0][0] rimane al suo valore iniziale pari a 25, dal momento che le uniche
celle ad essa adiacenti sono ostacoli.

Si noti che l'algoritmo trova sempre un percorso tra i punti di partenza e di arrivo, ma non necessariamente trova il
percorso piÃ¹ breve. Infatti esso sceglie, tra le celle vicine a quella in esame, una qualsiasi di quelle con valore
positivo inferiore a quello della cella corrente, e non la cella tra esse che contiene il valore minimo. */
```