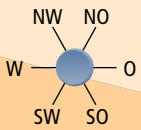
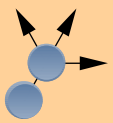
■ BEISPIELAUFGABE - ZAHLEN-HÜPFEN

Auf dem unten abgebildeten sechseckigen Spielbrett soll ein Spielstein so viele Felder in gerader Linie weiterhüpfen, wie die Zahl auf dem Feld, auf dem er steht, angibt. Begonnen wird auf dem Feld in der Mitte.
Der Anfangszug darf in eine beliebige der sechs möglichen Richtungen erfolgen. Bei jedem weiteren Zug darf jeweils nur in derselben Richtung weitergehüpft werden – oder in eine rechts oder links benachbarte Richtung. Die Hüpferei ist erfolgreich zu Ende, wenn das Ausgangsfeld wieder erreicht wird.

Richtungen:



Erlaubt:



Aufgabe

Schreibe ein Programm, welches einen erfolgreichen Hüpfweg sucht und ein Protokoll des Weges ausgibt (pro Hüpfer: die Richtung und Hüpfweite).
Falls das Ausgangsfeld nicht erreicht werden kann, soll dies an Stelle des Protokolls ausgegeben werden.

Lösungsidee

Der erste Schritt zur Lösung der Aufgabe besteht darin, sich eine geeignete Datenstruktur für das sechseckige Spielbrett zu überlegen. Das Brett lässt sich in eine quadratische Matrix schreiben, deren Größe sich an der „Mittelzeile" orientiert, die also 9x9 Einträge hat. Der obere Teil des Spielbrettes wird dann am linken Rand der Matrix orientiert, der untere Teil am rechten Rand. Alle verbleibenden Stellen der Matrix können z. B. mit 0 aufgefüllt werden. Das sieht dann so aus:

```
2 4 1 5 4 0 0 0 0
1 4 5 2 3 2 0 0 0
2 1 3 3 2 5 2 0 0
2 1 3 2 1 2 2 4 0
4 3 4 3 3 4 2 3 2
0 2 2 1 3 2 2 1 2
0 0 2 2 2 3 4 3 2
0 0 0 1 3 1 2 1 4
0 0 0 0 2 1 6 3 2
```

Für jede Richtung wird festgelegt, wie eine Bewegung auf dem Brett durch die Wahl eines neuen Matrix-Elements simuliert wird.
Zum Finden eines Weges wird das Spielbrett systematisch durchsucht, und zwar rekursiv. Dabei wird jedoch eine Einschränkung beachtet: Ein Feld darf in einem Hüpfweg nicht zweimal von der gleichen Richtung kommend betreten werden. Damit werden möglicherweise endlose Rundläufe vermieden.

Programm-Dokumentation

Die oben beschriebene Matrix für das Spielbrett wird in dem 9x9 großen Integer-Array `brett` gespeichert.
Die Richtungen werden mit Hilfe des Enumerationstyps `richtungen` mit Zahlen von 0 bis 5 identifiziert. Die Richtungsnamen sind im Array `bezeichner` passend abgelegt.
Das Array `bewegung` legt für jede Richtung fest, wie sich die Matrixkoordinaten bei einem Schritt in diese Richtung verändern; bei mehreren Schritten muss multipliziert werden. Im Array `weg` werden die Koordinaten aller im aktuellen Weg besuchten Felder für die Ausgabe gespeichert.
Das Integer-Array `besucht` ist genau so groß wie das Array `brett`. Es dient dazu, für jedes Feld markieren zu können, in welcher Richtung es beim aktuellen Weg schon besucht wurde. Für jede Richtung wird dabei ein eigenes Bit benutzt. Das Bit für eine Richtung z kann mit einer XOR-Zuweisung und einer Shift-Operation (`^= 1 << z`) gesetzt und wieder gelöscht werden.
Die Funktion `suche` gibt einen gefundenen Weg aus, wenn das Ausgangsfeld wieder erreicht ist. Ansonsten wird für alle drei möglichen Fortsetzungsrichtungen (die im Kopf der for-Schleife mit Hilfe des Modulo-Operators % berechnet werden) das nächste Feld bestimmt und von dort aus weitergesucht. Dabei wird die Rekursionstiefe als Parameter mitgeführt, da sie gleichzeitig angibt, an wievielter Stelle eines Weges das aktuelle Feld besucht wurde.
Das Hauptprogramm ruft die Funktion `suche` mit den einander gegenüberliegenden Richtungen O und W auf, um alle sechs vom Startfeld aus möglichen Richtungen abzuarbeiten.

Programm-Ablaufprotokoll

Das Programm erwartet keine Eingaben, das Spielfeld ist ja festgelegt. Nach weniger als einer Sekunde werden 46 gefundene Lösungen ausgegeben, darunter die folgende mit der kürzesten Weglänge:

```
18 Züge: 3 SO, 1 SW, 3 W, 2 NW, 2 NW, 4 NO, 2
O, 1 SO, 2 SW, 2 SO, 2 O, 1 NO, 3 NW, 3 W, 4
SW, 2 SO, 1 O, 3 NO
```

Programm-Text

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

int brett[9][9] = {  // Werte des Spielbretts
  2, 4, 1, 5, 4, 0, 0, 0, 0,
  1, 4, 5, 2, 3, 2, 0, 0, 0,
  2, 1, 3, 3, 2, 5, 2, 0, 0,
  2, 1, 3, 2, 1, 2, 2, 4, 0,
  4, 3, 4, 3, 3, 4, 2, 3, 2,
  0, 2, 2, 1, 3, 2, 2, 1, 2,
  0, 0, 2, 2, 2, 3, 4, 3, 2,
  0, 0, 0, 1, 3, 1, 2, 1, 4,
  0, 0, 0, 0, 2, 1, 6, 3, 2,
};

enum richtungen { O, SO, SW, W, NW, NO };
char* bezeichner[6] = { "O", "SO", "SW", "W", "NW", "NO" };

int bewegung[6][2] =  // dx und dy der sechs Bewegungsrichtungen
  { { 1, 0 }, { 1, 1 }, { 0, 1 }, { -1, 0 }, { -1, -1 }, { 0, -1
} };

int weg[100][2];     // speichert den gegangenen Weg
int besucht[9][9];   // benötigt zur Markierung besuchter Felder

void suche(int x, int y, int ausrichtung, int tiefe)
{
  int z;
  int newx, newy;

  if (x == 4 && y == 4 && tiefe != 0)  // Ausgangsposition wieder
  erreicht
  {
    printf("%d Züge: ", tiefe);
    for (z = 0; z < tiefe; z++)
    // Ausgabe des gegangenen Weges
      printf("%d %s%s", weg[z][1], bezeichner[weg[z][0]],
       (z < tiefe - 1 ? ", " : "\n"));
    return;
  }
  // Fortsetzung der Suche in alle drei möglichen Richtungen:
  for (z = (ausrichtung + 5) % 6;
       z != (ausrichtung + 2) % 6;
       z = (z + 1) % 6)
  {
   weg[tiefe][0] = z;           // Speichern von Ausrichtung und
   weg[tiefe][1] = brett[y][x]; // Sprungweite für spätere Ausgabe

    newx = x + brett[y][x] * bewegung[z][0];  // neue x-Koordinate
    newy = y + brett[y][x] * bewegung[z][1];  // neue y-Koordinate

    // Prüfung, ob die neue Position innerhalb des Bretts liegt
    // und noch nicht in gleicher Richtung begangen worden ist:

    if (newx >= 0 && newy >= 0 && newx <= 8 && newy <= 8 &&
      brett[newy][newx] && !(besucht[newy][newx] & 1 << z))
    {
      besucht[newy][newx] ^= 1 << z;      // Markierung setzen
      suche(newx, newy, z, tiefe + 1);    // rekursiver Aufruf
      besucht[newy][newx] ^= 1 << z ;     // Markierung entfernen
    }
  }
}

int main()
{
  memset(besucht, 0, sizeof(besucht));
  suche(4, 4, O, 0);  // Aufruf für die Startrichtungen NO, O, SO
  suche(4, 4, W, 0);  // und zweiter Aufruf für NW, W, SW
  return 0;
}
```

Lösung nach Alexander Hullmann und Alexander Kreuzer

Design: Kuhn, Kammann & Kuhn, Köln Illustration: Chris Janata, Köln Druck: DFS Druck und Verlag, Köln