Aufgaben zum Übungskomplex
Algorithmen und Programme
Teil II: Felder, Arrays, Matrizen

Aufgabe 1.

Seien *A* und *B* untere *n x n* Dreiecksmatrizen, d.h., alle Elemente oberhalb der Hauptdiagonalen sind Null.

$$A_{i,j} = B_{i,j} = 0 \Leftrightarrow i < j,\ \forall i,j \in \{1...n\}$$

Beschreiben Sie, wie Sie die Matrizen *A* und *B* in einem Array *C* minimaler Größe speichern können. Welche Größe hat *C*?

Schreiben Sie zwei Funktionen

```
Function A (C:MinArrayTyp; i,j:integer) : integer;
Function B (C:MinArrayTyp; i,j:integer) : integer;
```

die die entsprechenden in *C* gespeicherten Matrixelemente zurückgeben.

Aufgabe 2.

Geben Sie analog zu Aufgabe 1 einen Algorithmus an, um die belegten Elemente *einer n x n* Dreiecksmatrix *M* (alle Elemente unterhalb der Hauptdiagonale sind Null, alle anderen Elemente sind positive ganze Zahlen) eindeutig in einem eindimensionalen Feld *A* zu speichern. Wie lang muss das Feld *A* sein? Entwickeln Sie auch eine Funktion GetElement(A:EindimFeld; i,j:integer) : integer, die den Wert des ursprünglichen Elementes M[i,j] aus dem Feld A wieder Zurückliefert.

Aufgabe 3.

Gegeben sei eine quadratische Matrix. Die Summe der Elemente einer Zeile (Spalte) sei z_i (s_j). Ermitteln Sie die Zeile *m* mit maximalem z_i. Tauschen Sie in der Matrix die Zeile *m* mit der Spalte *m* aus. Beschreiben Sie Ihre Lösung mit einem Algorithmus.

Aufgabe 4.

Ein Matrixelement *s* heißt *Senke*, wenn alle (maximal 8) angrenzenden Elemente g_k größer als *s* sind. Ermitteln Sie alle Senken einer gegebenen beliebigen *n x m* Matrix, mit $n,m \geq 2$. Geben Sie die tiefste Senke (maximaler Abstand zu einem Nachbarelement g_k) an.

Aufgabe 5.

Das Spiel "Selektion" soll das Prinzip der natürlichen Auslese demonstrieren. Es werden folgende Hilfsmittel benötigt:

1. Ein quadratisches Spielbrett mit *6 x 6* Feldern,
2. jeweils ein Würfel zur Ermittlung einer zufälligen Zeile und Spalte auf dem Spielbrett und
3. jeweils 36 Steine in einer der vier verschiedenen Farben (rot, blau, gelb, schwarz), insgesamt also 144 Steine.

Zu Beginn des Spieles werden auf dem Spielfeld völlig willkürlich jeweils 9 Steine aller 4 Farben angeordnet, so dass das Spielfeld vollständig mit Steinen bedeckt ist. Danach folgt der erste Schritt, der, wie alle folgenden, aus zweimaligem Würfeln mit den beiden Würfeln besteht. So werden beim ersten Wurf die Koordinaten eines Spielfeldes ermittelt. Der auf diesem Feld befindliche Stein wird entfernt, die Farbe an dieser Stelle also vernichtet. Im zweiten Wurf werden die Koordinaten eines weiteren Feldes ermittelt. Sollte sich zufällig das leere Spielfeld ergeben, wird der 2. Wurf wiederholt. Auf dem erwürfelten Spielfeld befindet sich eine bestimmte Farbe. Sie wird auf dem Feld belassen, die gleiche Farbe aber auf das leergewordene Spielfeld gesetzt. Dies wird solange wiederholt, bis nur noch Spielsteine genau einer Farbe auf dem Spielbrett sind.

Versuchen Sie dieses Spiel zuerst auf einem Blatt Papier zu spielen. Wann haben sie aufgehört, weiter zu würfeln? Schreiben Sie danach ein Computerprogramm, das dieses Spiel alleine durchführen kann und den Spielverlauf grafisch darstellt. Erweitern Sie Ihr Programm derart, dass es für beliebige *n x m* Spielbretter verwendbar ist.

Aufgabe 6.

Die Spalten und Zeilen eines *n x m* Arrays *a* seien jeweils aufsteigend sortiert, es gelte also

$$a[i,j] \leq a[i,j+1] \Leftrightarrow i=1,...,n \ \wedge \ j=1,...,m-1$$

$$a[i,j] \leq a[i+1,j] \Leftrightarrow i=1,...,n-1 \ \wedge \ j=1,...,m$$

Entwerfen Sie einen Algorithmus, der feststellt, ob eine gegebene Zahl *x* in *a* abgespeichert ist. Der Algorithmus soll mit möglichst wenigen Vergleichsoperationen auskommen.

Aufgabe 7.

In einem Text *T* aus *t* Zeichen sollen Muster der Form "xxxxxxxx" gesucht werden, also *n*-fache Wiederholungen desselben Zeichens *Z* in *t* (hier: *Z*="x" und *n*=8). Schreiben Sie einen Algorithmus, der für gegebene *T*, *t*, *n*, *Z* als Resultat die Position ($1,...,t$) des ersten Auftretens des Musters liefert. Wenn das Muster nicht in *T* enthalten ist, soll der Algorithmus als Ergebnis liefern. Auch dieser Algorithmus soll möglichst wenig Vergleichsoperationen ausführen.

Aufgabe 8.

Simulieren Sie eine an den Banden eines Billardtisches abprallende Billardkugel. Die Bewegung soll ohne Reibung ablaufen, die Kugel habe also zu allen Zeiten die gleiche Geschwindigkeit.

Auf den Tisch sei eine *m x n* Matrix projiziert. Die Kugel soll sich nur auf Geraden bewegen, die parallel zu den beiden Hauptdiagonalen der Matrix sind. Die Kugel wird also an den Banden im rechten Winkel reflektiert. Stellen Sie zu allen Zeitpunkten, an denen die Kugel genau in der Mitte eines Feldes der Matrix liegt, den Billardtisch (symbolisiert durch die *m x n* Matrix) grafisch dar. Der Punkt an dem die Kugel zu Beginn der Simulation ist, soll frei wählbar sein. Zu diesem Zeitpunkt soll sich die Kugel in nord-östlicher Richtung bewegen.

Geben Sie einen Algorithmus an, der eine solche Simulation realisiert.

Die Simulation soll beendet werden, wenn die Kugel sich wieder am Startpunkt befindet. Wieviele Simulationsschritte wurden ausgeführt?