Aufgabenblatt 5

Kompetenzstufe 1 & Kompetenzstufe 2

Allgemeine Informationen zum Aufgabenblatt:

- Die Abgabe erfolgt in TUWEL. Bitte laden Sie Ihr IntelliJ-Projekt bis spätestens **Freitag, 29.05.2020 13:00 Uhr** in TUWEL hoch.
- Zusätzlich müssen Sie in TUWEL ankreuzen, welche Aufgaben Sie gelöst haben.
- Ihre Programme müssen kompilierbar und ausführbar sein.
- Ändern Sie bitte **nicht** die **Dateinamen** und die **vorhandene Ordnerstruktur**.
- Verwenden Sie, falls nicht anders angegeben, für alle Ausgaben `System.out.println()` bzw. `System.out.print()`.
- Verwenden Sie für die Lösung der Aufgaben keine Aufrufe (Klassen) aus der Java-API, außer diese sind ausdrücklich erlaubt.
- Erlaubt sind die Klassen `String`, `Math`, `Integer` und `StdDraw` oder Klassen, die in den Hinweisen zu den einzelnen Aufgaben aufscheinen.

In diesem Aufgabenblatt werden folgende Themen behandelt:

- Ein- und Zweidimensionale Arrays
- Rekursion
- Grafische Ausgabe

Aufgabe 1

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `genFilledArray`:

  ```
  int[][] genFilledArray(int n)
  ```

  Die Methode erzeugt ein zweidimensionales Array der Größe n×n und befüllt dieses mit Zahlen, wie in den nachfolgenden Beispielen gezeigt. Es wird links oben mit `n` begonnen und in jeder weiteren Gegendiagonalen die Zahl um 1 verkleinert. Nach der Zahl 1 wird mit jeder weiteren Gegendiagonalen die Zahl wieder um 1 erhöht, sodass es rechts unten wieder mit der Zahl `n` endet.

  Vorbedingung: `n > 1`.

  Beispiele:

  `genFilledArray(2)` erzeugt →

  ```
  2 1
  1 2
  ```

  `genFilledArray(4)` erzeugt →

  ```
  4 3 2 1
  3 2 1 2
  2 1 2 3
  1 2 3 4
  ```

  `genFilledArray(7)` erzeugt →

  ```
  7 6 5 4 3 2 1
  6 5 4 3 2 1 2
  5 4 3 2 1 2 3
  4 3 2 1 2 3 4
  3 2 1 2 3 4 5
  2 1 2 3 4 5 6
  1 2 3 4 5 6 7
  ```

- Implementieren Sie eine Methode `extendArray`:

  `int[][] extendArray(int[][] inputArray)`

  Diese Methode erstellt ein ganzzahliges zweidimensionales Array, bei dem jede Zeile die gleiche Länge aufweist. Die Länge der Zeilen wird durch die längste Zeile von `inputArray` bestimmt, da das Array `inputArray` unterschiedliche Zeilenlängen aufweisen kann. Die einzelnen Zeilen von `inputArray` werden dabei abwechselnd links und rechts mit Nullen aufgefüllt, sodass alle Zeilen des neuen Arrays gleich viele Einträge haben. Die erste Zeile beginnt mit dem Auffüllen der Nullen von links.

  Vorbedingungen: `inputArray != null` und `inputArray.length > 0`, dann gilt auch für alle gültigen `i`, dass `inputArray[i].length > 0`.

  Beispiele:

  `extendArray(new int[][]{{1, 2, 3}, {4}, {5, 6}, {7, 8, 9, 1}})` erzeugt →

  ```
  0 1 2 3
  4 0 0 0
  0 0 5 6
  7 8 9 1
  ```

  ```
  extendArray(new int[][]
  {{0, 1, 1, 1, 1, 1},
  {1, 1},
  {1, 0, 0, 0},
  {1, 1, 0, 1},
  {1},
  {1, 0, 1, 1, 0, 0, 1, 1}})
  ```
  erzeugt →

  ```
  0 0 0 1 1 1 1 1
  1 1 0 0 0 0 0 0
  0 0 0 0 1 0 0 0
  1 1 0 1 0 0 0 0
  0 0 0 0 0 0 0 1
  1 0 1 1 0 0 1 1
  ```

  ```
  extendArray(new int[][]
  {{1, 3, 2},
  {5, 1},
  {6, 8, 5, 4},
  {9, 4, 1, 9, 2},
  {3},
  {1, 8, 7, 5, 3, 2, 5}})
  ```
  erzeugt →

  ```
  0 0 0 0 1 3 2
  5 1 0 0 0 0 0
  0 0 0 6 8 5 4
  9 4 1 9 2 0 0
  0 0 0 0 0 0 3
  1 8 7 5 3 2 5
  ```

- Implementieren Sie eine Methode `reformatArray`:

  `int[] reformatArray(int[][] inputArray)`

  Diese Methode interpretiert jede Zeile von `inputArray` als Binärzahl und erstellt ein neues eindimensionales Array, das jede dieser Binärzahlen als Dezimalzahl beinhalten soll. Das Array mit den Dezimalzahlen wird anschließend zurückgegeben. Jede Zeile von `inputArray` kann von hinten nach vorne gelesen werden und dabei kann die Wertigkeit jeder Stelle ermittelt werden. Das letzte Element in jeder Zeile von `inputArray` hat die Wertigkeit 2^0, das vorletzte Element jeder Zeile die Wertigkeit 2^1, usw. Nach diesem Schema wird jede Zeile in eine Dezimalzahl umgewandelt. Die so entstandenen Dezimalzahlen werden im neuen Array der Reihe nach, beginnend beim Index 0, abgelegt.

  Vorbedingungen: `inputArray != null`, `inputArray.length > 0`, dann gilt für alle gültigen `i`, dass `inputArray[i].length > 0` $\wedge$ `inputArray[i].length < 32` ist. Alle Zahlen in `inputArray` sind Nullen oder Einsen.

  Beispiele:

  `reformatArray(new int[][]`
  `{{1,0,1},`
  `{0,1,1}})` erzeugt $\rightarrow$

  Zeile 1: `{1,0,1}` $\rightarrow 2^2 * 1 + 2^1 * 0 + 2^0 * 1 = 5$
  Zeile 2: `{0,1,1}` $\rightarrow 2^2 * 0 + 2^1 * 1 + 2^0 * 1 = 3$

  ```
  5 3
  ```

  ```
  reformatArray(new int[][]
  {{1, 0, 1, 1},
  {0, 1, 1},
  {1, 1, 0, 0, 0},
  {1, 0, 1, 0, 1},
  {1, 0},
  {1, 1, 1, 1, 1}})
  ```
  erzeugt $\rightarrow$

  ```
  11 3 24 21 2 31
  ```

  ```
  reformatArray(new int[]
  {{0,0,0,1,1,1,1,1},
  {1,1,0,0,0,0,0,0},
  {0,0,0,0,1,0,0,0},
  {1,1,0,1,0,0,0,0},
  {0,0,0,0,0,0,0,1},
  {1,0,1,1,0,0,1,1}})
  ```
  erzeugt $\rightarrow$

  ```
  31 192 8 208 1 179
  ```

Aufgabe 2

Implementieren Sie folgende Aufgabenstellung:

- Implementieren Sie eine Methode `shiftLinesInArray`:

  ```
  void shiftLinesInArray(int[][] workArray)
  ```

  Diese Methode baut ein ganzzahliges zweidimensionales Array `workArray` so um, sodass alle Zeilen innerhalb des Arrays um eine Zeile nach oben verschoben werden. Die oberste Zeile wird ganz unten im Array wieder eingefügt.

  Vorbedingungen: `workArray != null` und `workArray.length > 0`, dann gilt auch für alle gültigen `i`, dass `workArray[i].length > 0`.

  Beispiele:

  ```
  shiftLinesInArray(new int[][]
  {{1,3,5},
  {6,2,1},
  {0,7,9}}) erzeugt →

  6 2 1
  0 7 9
  1 3 5

  shiftLinesInArray(new int[][]
  {{1,5,6,7},
  {1,9,3},
  {4},
  {6,3,0,6,2},
  {6,3,0}, erzeugt →

  1 9 3
  4
  6 3 0 6 2
  6 3 0
  1 5 6 7
  ```

- Implementieren Sie eine Methode `reshapeArray`:

  ```
  void reshapeArray(int[][] workArray)
  ```

  Diese Methode baut ein ganzzahliges zweidimensionales Array `workArray` so um, sodass jede Zeile danach die gleiche Länge aufweist. Die Länge der Zeilen wird durch die längste Zeile von `workArray` bestimmt. Bei kürzeren Zeilen bleibt der Originalinhalt der Zeile erhalten und steht linksbündig. Die neu hinzugekommenen Arrayeinträge rechts innerhalb einer Zeile werden mit der Anzahl an neu hinzugekommenen Arrayeinträgen befüllt (z.B. hat die ursprüngliche Zeile eine Länge von 3 und wird jetzt auf die im Array längste Zeile mit der Länge 7 erweitert, dann werden die 4 neuen Arrayeinträge mit der Zahl 4 befüllt).

  Vorbedingungen: `workArray != null` und `workArray.length > 0`, dann gilt auch für alle gültigen `i`, dass `workArray[i].length > 0`.

  Beispiele:

  `reshapeArray(new int[][]{{4, 7}, {5}, {6, 1, 3, 8, 9}})` erzeugt →

  ```
  4 7 3 3 3
  5 4 4 4 4
  6 1 3 8 9
  ```

  ```
  reshapeArray(new int[][]
  {{1,3,2},
  {7,1},
  {0,11,7,5,3,2,5},
  {6,8,5,10},
  {9,4,1,5,8},
  {3}})
  ```
  erzeugt →

  ```
  1 3 2 4 4 4 4
  7 1 5 5 5 5 5
  0 11 7 5 3 2 5
  6 8 5 10 3 3 3
  9 4 1 5 8 2 2
  3 6 6 6 6 6 6
  ```

- Implementieren Sie eine Methode `reduceArray`:

  ```
  void reduceArray(int[][] workArray)
  ```

  Diese Methode baut ein ganzzahliges zweidimensionales Array `workArray` so um, sodass alle Einträge, die kleiner gleich 0 sind, aus dem Array entfernt werden. Jede Zeile, bei der Einträge entfernt werden müssen, muss neu erstellt werden und darf danach nur noch Einträge enthalten, die größer 0 sind. Sollte eine Zeile nur Werte kleiner gleich 0 enthalten, dann wird die Länge dieser Zeile nach der Bearbeitung die Länge 0 aufweisen.

  Vorbedingungen: `workArray != null` und `workArray.length > 0`.

  Beispiele:

  `reduceArray(new int[][]{{-1, 3, 2}, {-5}, {6, -8, 5, 10},{},{3}})` erzeugt →

  ```
  3 2

  6 5 10

  3
  ```

  ```
  reduceArray(new int[][]
  {{-1, 4, 9},
  {0, 7, -3, 1},
  {6, -8, 5, 10},
  {9, 0, 1, 5, 8},
  {3},
  {0, 11, -7, 5, 3, 2, -5}})
  ```
  erzeugt →

  ```
  4 9
  7 1
  6 5 10
  9 1 5 8
  3
  11 5 3 2
  ```

Aufgabe 3

Erweitern Sie die Aufgabe um folgende Funktionalität:

- Implementieren Sie eine Methode `genArray`:

  ```
  int[][] genArray(int numLines)
  ```

  Diese Methode erzeugt ein zweidimensionales ganzzahliges Array mit `numLines` Zeilen. Für jede Zeile soll ein Array mit unterschiedlicher Länge erzeugt werden. Die Länge des Arrays jeder Zeile soll mittels Zufallszahlengenerator erzeugt werden und eine Länge zwischen 1 (inklusive) und 10 (inklusive) haben. Zusätzlich soll jeder Arrayeintrag des zweidimensionalen Arrays mit Zufallswerten zwischen 1 (inklusive) und 40 (inklusive) befüllt werden.

- Implementieren Sie eine Methode `getMaxValue`:

  ```
  int getMaxValue(int[] workArray))
  ```

  Diese Methode bekommt ein eindimensionales ganzzahliges Array übergeben und retourniert den größten Wert innerhalb des Arrays.

- Implementieren Sie eine Methode `getMaxValue`:

  ```
  int getMaxValue(int[][] workArray)
  ```

  Diese Methode bekommt ein zweidimensionales ganzzahliges Array übergeben und retourniert den größten Wert innerhalb des Arrays.

- Implementieren Sie eine Methode `genHistogram`:

  ```
  int[] genHistogram(int[][] workArray)
  ```

  Diese Methode bekommt ein zweidimensionales ganzzahliges Array übergeben und soll ein Histogramm [1] erstellen. Dazu wird ein eindimensionales Array angelegt, dessen Länge durch den höchsten im Array vorkommenden Eintrag bestimmt wird. Es muss das übergebene Array durchlaufen werden und für jeden Arrayeintrag (entspricht einer Zahl) wird der entsprechende Eintrag im Zielarray um eins erhöht. Am Ende erhält man ein Array, bei dem jeder Eintrag das Vorkommen einer gewissen Zahl repräsentiert.

  Exkurs: Abbildung 1 zeigt in einem kleinen Beispiel den Aufbau eines Histogramms. Punkt A stellt ein kleines zweidimensionales Array dar, das als Input dienen soll. Es beinhaltet zufällige Werte zwischen 1 und 9. Danach wird unter Punkt B ein Array erzeugt, das so lange ist, wie die höchste vorkommende Zahl im Array unter Punkt A. Damit hat das Array die Länge 9 und jeder Eintrag im Array steht für eine Zahl zwischen 1 und 9. Das heißt, dass an Indexstelle 0 das Vorkommen der Ziffer 1 gezählt wird, an Indexstelle 1 das Vorkommen der Ziffer 2 usw. Nachdem das Array unter Punkt A komplett durchlaufen wurde, beinhaltet das Array unter Punkt B alle Vorkommen der Ziffern 1 bis 9. Für die Erstellung des Diagramms unter Punkt C wird jene Ziffer, die am häufigsten vorkommt, für die Dimensionierung der Diagrammhöhe herangezogen (in diesem Fall die Ziffer 1, die 3 mal vorkommt). Die Breite der Balken ergibt sich aus der Länge des Arrays unter Punkt B.

[1] Histogramm: `https://de.wikipedia.org/wiki/Histogramm`

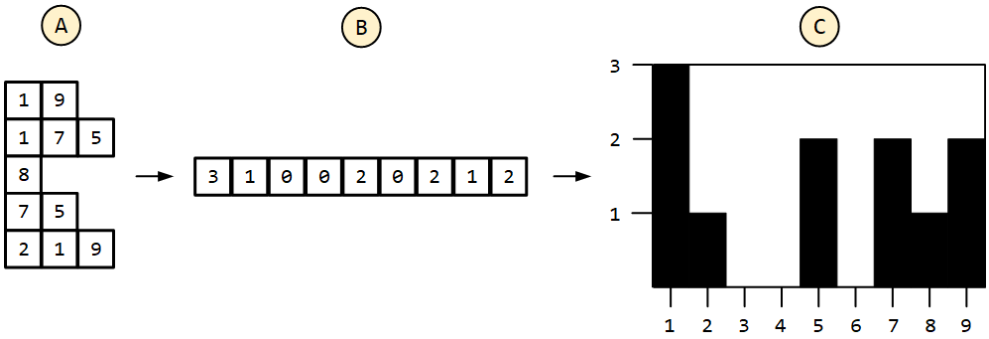


Abbildung 1: Beispiel eines Histogramms, erstellt aus Zufallszahlen

- Implementieren Sie eine Methode `drawHistogram`:

```
void drawHistogram(int[] workArray, int width, int height)
```

Diese Methode bekommt ein eindimensionales ganzzahliges Array übergeben, in dem sich ein Histogramm befindet, welches visualisiert werden soll. Zusätzlich werden die Breite (`width`) und Höhe (`height`) des Ausgabefensters als Parameter übergeben. In Abbildung 2 sehen Sie Beispiele für Histogramme mit verschiedenen Werten für `numLines` (Parameter in der Methode `genArray(int numLines)`, welcher die Anzahl der Zeilen und somit die Menge an Werten steuert). Es ist zu berücksichtigen, dass der höchste Wert mit dem höchsten Balken

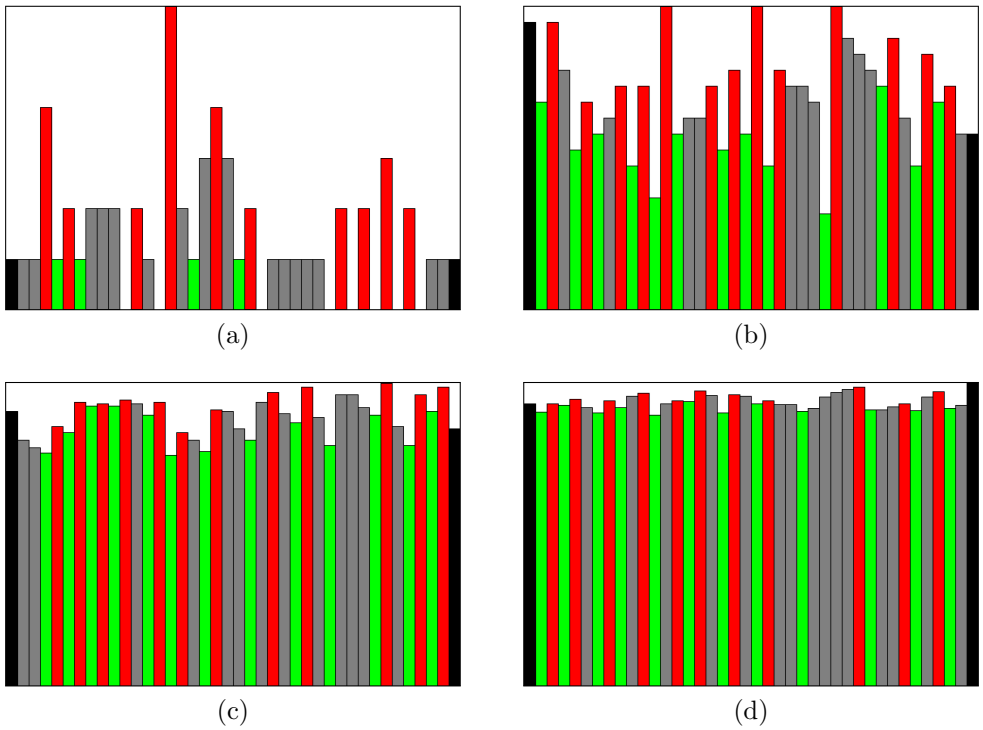


Abbildung 2: Histogramme mit a) 10, b) 100, c) 1000 und d) 10000 `numlines`

repräsentiert wird. Dieser füllt die komplette Höhe des Ausgabefensters aus. Alle anderen Werte (Balken) werden darauf angepasst. Die Breite der Balken ergibt sich aus der Länge

des Arrays. Zusätzlich sollen die Balken nach vorgegebenen Regeln eingefärbt werden. Der erste und der letzte Balken werden schwarz gezeichnet. Bei allen anderen Balken gilt:

- Der Balken wird rot eingefärbt, wenn sein Vorgänger und Nachfolger kleiner ist als der Balken selbst.
- Der Balken wird grün eingefärbt, wenn sein Vorgänger und Nachfolger größer ist als der Balken selbst.
- Der Balken wird grau in allen anderen Fällen eingefärbt.

Aufgabe 4

Erweitern Sie die Aufgabe um folgende Funktionalität:

- Implementieren Sie eine Methode `genLandscape`:

  ```
  Color[][] genLandscape(int size)
  ```

  Diese Methode erzeugt eine Landschaft in einem zweidimensionalen Array vom Typ `Color` mit der Größe `size`×`size` und retourniert dieses Array. Dazu soll jedem Element im Array zufallsgeneriert entweder `Color.GRAY` (Felsen) oder `Color.GREEN` (Wiese) zugewiesen werden. Felsen sollen mit einer Wahrscheinlichkeit von 10% vorkommen (Bestimmung mittels `Math.random()`). Es entsteht dann eine Landschaft, wie in Abbildung 3a gezeigt.

- Implementieren Sie eine Methode `drawLandscape`:

  ```
  void drawLandscape(Color[][] landscape)
  ```

  Diese Methode zeichnet die Landschaft in einem Ausgabefenster von `canvasSize`×`canvasSize` Pixel. Jeder Eintrag des Arrays wird als gefülltes Quadrat gezeichnet.

- Implementieren Sie eine **rekursive** Methode `simLiquidFlow`:

  ```
  void simLiquidFlow(Color[][] landscape, int x, int y)
  ```

  Diese Methode generiert bzw. simuliert eine durch die Landschaft fließende Flüssigkeit. Die Flüssigkeit soll die Landschaft von unten nach oben mit der Breite eines Array-Elements orange einfärben (`Color.ORANGE`). Immer wenn die Flüssigkeit auf einen grauen Felsen trifft (`Color.GRAY`), spaltet sich die Flüssigkeit links und rechts (neben dem Felsen auf gleicher Höhe y) auf. Zusätzlich wird das Pixel unter dem Felsen immer orange eingefärbt. Der Felsen selbst wird mit `Color.BLACK` dunkler eingefärbt. Ansonsten fließt die Flüssigkeit mit einer Chance von 50% entweder nach links oder rechts (aber immer zusätzlich hinauf). Das Ergebnis nach einem Aufruf von `simLiquidFlow` ist in Abbildung 3b dargestellt.

  Beachten Sie, dass durch die rekursiven Aufrufe auch folgende Situationen entstehen können: Trifft die Flüssigkeit auf einen schwarzen Felsen, dann passiert nichts, d.h. der Aufruf wird beendet. Trifft die Flüssigkeit auf einen orangen Array-Eintrag, dann bleibt dieser orange und der Aufruf wird weiter fortgesetzt, d.h. zwei Flüssigkeitsströme können sich an einer Stelle überschneiden und dann auch wieder trennen.

- Implementieren Sie eine **rekursive** Methode `simSpreadingFire`:

  ```
  void simSpreadingFire(Color[][] landscape, int x, int y)
  ```

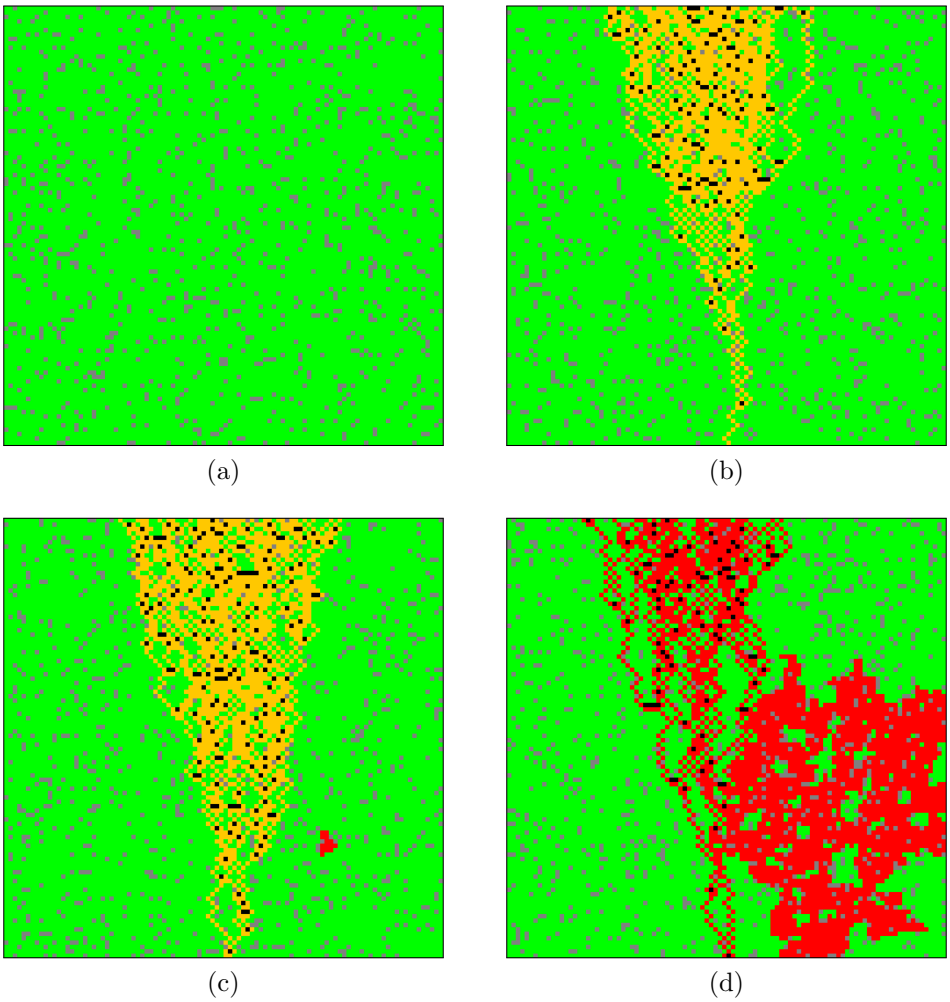
  Diese Methode simuliert ein Feuer und dessen Ausbreitung in einer gegebenen Landschaft. Dazu wird ein Array-Eintrag, der eine Wiese darstellt, entzündet. Danach soll sich das Feuer per Zufall in alle vier Himmelsrichtungen (Norden, Osten, Süden, Westen) ausbreiten und rot (`Color.RED`) eingefärbt werden. Für jede dieser 4 Richtungen wird sich das Feuer mit 60%

Wahrscheinlichkeit ausbreiten, sofern es sich um eine Wiese handelt (Felsen werden nicht entzündet und stoppen das Feuer). Trifft das Feuer auf die zuvor bereits in die Landschaft eingetragene orange Flüssigkeit, wird diese entzündet und färbt alle orangen Array-Einträge ebenfalls rot ein. Dazu soll die Methode `spreadFireInLiquid` (nachfolgend beschrieben) aufgerufen werden. Abbildung 3c zeigt ein Ergebnis, wo das Feuer die brennbare Flüssigkeit nicht erreicht hat. Abbildung 3d hingegen zeigt ein Endergebnis, wo das Feuer auch die Flüssigkeit erreicht und entzündet hat.

- Implementieren Sie eine **rekursive** Methode `spreadFireInLiquid`:

```
void spreadFireInLiquid(Color[][] landscape, int x, int y)
```

Diese Methode dient als Hilfsmethode, um beim Entzünden der Flüssigkeit alle orangen Array-Einträge rot einzufärben. Dazu muss, ausgehend von einem Eintrag, in alle 8 Richtungen gesucht werden, ob es noch einen weiteren orangen Eintrag gibt, der rot eingefärbt werden muss. Achten Sie bei der Implementierung darauf, dass diese Methode keinerlei Schleifen enthält.



(a) (b) (c) (d)

Abbildung 3: Landschaftsdarstellung nach a) Generierung, b) simulierter Flüssigkeit, c) simulierter Flüssigkeit mit Feuer und d) simulierter Flüssigkeit, die ebenfalls durch das Feuer entzündet wurde.

Aufgabe 5

Implementieren Sie folgende Aufgabenstellung:

(!) In dieser Aufgabe sollen bei allen Methoden die angegebenen Bedingungen überprüft werden. Sie dürfen zusätzliche Überprüfungen durchführen, wenn diese dienlich sind. Bei Verletzung einer Bedingung ist `null` zu retournieren. Es werden bei dieser Aufgabe, abweichend zur Java-Konvention, Großbuchstaben als Variablennamen für die Matrizen verwendet.

- Implementieren Sie eine Methode `addMatrices`:

  ```
  int[][] addMatrices(int[][] A, int[][] B)
  ```

  Diese Methode berechnet die Summe[2] der beiden Matrizen `A` und `B` und gibt die so neu entstandene Matrix zurück.

  Zu überprüfende Bedingungen: `A != null`, `B != null`, `A` und `B` haben die Größe `n`×`n` und `n` $\in \{1, 2, ..., 30\}$

  Beispiele:

  ```
  int[][] A = new int[][]{{2, 3}, {4, 1}};
  int[][] B = new int[][]{{1, 4}, {2, 5}};
  ```

  `addMatrices(A, B)` erzeugt →

  ```
  3 7
  6 6
  ```

  ```
  int[][] A = new int[][]{{2, 1, 3, 4}, {1, 2, 0, 1},
                          {2, 1, -6, 1}, {-3, 1, -6, 5}};
  int[][] B = new int[][]{{1, -2, 3, 2}, {3, -2, 0, 2},
                          {1, 0, -6, -3}, {-3, -5, -6, -3}};
  ```

  `addMatrices(A, B)` erzeugt →

  ```
  3 -1 6 6
  4 0 0 3
  3 1 -12 -2
  -6 -4 -12 2
  ```

- Implementieren Sie eine Methode `subMatrices`:

  ```
  int[][] subMatrices(int[][] A, int[][] B)
  ```

  Diese Methode berechnet die Differenz[1] der beiden Matrizen `A` und `B` und gibt die so neu entstandene Matrix zurück.

  Zu überprüfende Bedingungen: `A != null`, `B != null`, `A` und `B` haben die Größe `n`×`n` und `n` $\in \{1, 2, ..., 30\}$

[2] https://www.abi-mathe.de/buch/matrizen/addition-subtraktion-und-multiplikation/

Beispiele:

```
int[][] A = new int[][]{{2, 3}, {4, 1}};
int[][] B = new int[][]{{1, 4}, {2, 5}};
```

`subMatrices(A, B)` erzeugt →

```
1 -1
2 -4
```

```
int[][] A = new int[][]{{2, 1, 3, 4}, {1, 2, 0, 1},
                        {2, 1, -6, 1}, {-3, 1, -6, 5}};
int[][] B = new int[][]{{1, -2, 3, 2}, {3, -2, 0, 2},
                        {1, 0, -6, -3}, {-3, -5, -6, -3}};
```

`subMatrices(A, B)` erzeugt →

```
1 3 0 2
-2 4 0 -1
1 1 0 4
0 6 0 8
```

- Implementieren Sie eine Methode `multMatrices`:

  ```
  int[][] multMatrices(int[][] A, int[][] B)
  ```

  Diese Methode berechnet das Produkt[3] der beiden Matrizen `A` und `B` ($A * B$) und gibt die so neu entstandene Matrix zurück.

  Zu überprüfende Bedingungen: `A != null`, `B != null`, `A` hat die Größe `m`×`n`, `B` die Größe `n`×`k` und `m,n,k` $\in \{1, 2, ..., 30\}$

  Beispiele:

  ```
  int[][] A = new int[][]{{2, 3}, {4, 1}};
  int[][] B = new int[][]{{1, 4}, {2, 5}};
  ```

  `multMatrices(A, B)` erzeugt →

  ```
  8 23
  6 21
  ```

  ```
  int[][] A = new int[][]{{2, 1, 3, 4}, {1, 2, 1, 1}, {2, 1, 1, 1}};
  int[][] B = new int[][]{{2, 3}, {1, 4}, {1, 5}, {1, 6}};
  ```

  `multMatrices(A, B)` erzeugt →

  ```
  12 49
  6 22
  7 21
  ```

[3] `https://de.wikipedia.org/wiki/Matrizenmultiplikation`

```
int[][] A = new int[][]{{2, 1, 3, 4}, {1, 2, 0, 1},
                        {2, 1, -6, 1}, {-3, 1, -6, 5}};
int[][] B = new int[][]{{1, -2, 3, 2}, {3, -2, 0, 2},
                        {1, 0, -6, -3}, {-3, -5, -6, -3}};
```

`multMatrices(A, B)` erzeugt →

```
-4 -26 -36 -15
4 -11 -3 3
-4 -11 36 21
-21 -21 -3 -1
```

```
A = new int[][]{{2, 3, 1, 4, 6}, {2, 0, 3, 4, 1}, {0, 5, 1, 4, 1}};
B = new int[][]{{0, 1, 4}, {3, 1, 2}, {5, 0, 2}, {1, 4, 2}, {3, 0, 1}};
```

`multMatrices(A, B)` erzeugt →

```
36 21 30
22 18 23
27 21 21
```

- Implementieren Sie eine Methode `transposeMatrix`:

  `int[][] transposeMatrix(int[][] A)`

  Diese Methode nimmt ein ganzzahliges zweidimensionales Array `A` entgegen und baut es so um, sodass alle Einträge an der Hauptdiagonalen gespiegelt (transponiert)[4] werden.

  Zu überprüfende Bedingungen: `A != null`, `A` hat die Größe `m`×`n` und `m,n` $\in \{1, 2, ..., 30\}$

  Beispiele:

  ```
  int[][] A = new int[][]{{2, 3}, {4, 1}};
  ```

  `transposeMatrix(A)` erzeugt →

  ```
  2 4
  3 1
  ```

  ```
  int[][] A = new int[][]{{2, 1, 3, 4}, {1, 2, 1, 1}, {2, 1, 1, 1}};
  ```

  `transposeMatrix(A)` erzeugt →

  ```
  2 1 2
  1 2 1
  3 1 1
  4 1 1
  ```

  ```
  int[][] A = new int[][]{{2, 1, 3, 4}, {1, 2, 0, 1},
                          {2, 1, -6, 1}, {-3, 1, -6, 5}};
  ```

  `transposeMatrix(A)` erzeugt →

[4] https://de.wikipedia.org/wiki/Transponierte_Matrix

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

```
A = new int[][]{{2, 3, 1, 4, 6}, {2, 0, 3, 4, 1}, {0, 5, 1, 4, 1}};
```

`transposeMatrix(A)` erzeugt →

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