Einschränkungen

- Sie dürfen **keine** zusätzlichen eigenen Hilfsmethoden oder globalen Variablen verwenden.
- Die vorgegebenen Methodenköpfe dürfen **nicht** erweitert oder geändert werden.
- Für die Implementierung der rekursiven Methode dürfen **keine** Schleifen verwendet werden.
- Sie dürfen Strings **nicht** per Referenz vergleichen.
- Sie dürfen **nicht** die Methoden `clone` und `System.arraycopy` verwenden.
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `Arrays` verwenden: `deepToString`, `toString`
- Sie dürfen **nur** folgende Methode(n) aus der Klasse `String` verwenden: `charAt`, `equals`, `isEmpty`, `length`, `substring`

Aufgabenstellung

Deklarieren und initialisieren Sie in `main` die folgende(n) Variable(n):

```
int[][] data0 = {{3, 0}, {0, 1}, {2, 2}};
int[][] data1 = {{0, 1, 0, 0, 1, 0}, {}, {2, 2, 2, 2, 0, 1}};
int[] target1 = {0, 0, 0};
int[] target2 = {9, 9, 9, 9};
```

Implementieren Sie folgende Methoden:

- **`int[][] labelPath(int n, int[][] points)`** erstellt ein neues `n` Mal `n` Array und gibt dieses zurück. Jede Zeile in `points` beschreibt eine Stelle in dem neuen Array. Die Spalte 0 steht dabei immer für den Zeilenindex und die Spalte 1 für den Spaltenindex dieser Stelle. Das Rückgabearray enthält an jeder Stelle in `points` den Wert -1. An allen anderen Stellen enthält das Rückgabearray den Wert `n`.
  Vorbedingung(en): `n >= 1`, `points != null`, `0 <= points[i][j] < n` für alle gültigen Indizes `i,j`.
  Wird die Methode z.B. mit `n=4` und `points=data0` aufgerufen, entsteht folgendes Array:

4	-1	4	4
4	4	4	4
4	4	-1	4
-1	4	4	4

- **`void findMatches(int[][] data, int[] pattern, int[] target)`** bestimmt für jede Zeile `i` in `data`, wie oft die Folge der Werte in `pattern` vorkommt. Die jeweilige Anzahl wird in `target` am Index `i` abgelegt.
  Vorbedingung(en): `pattern.length > 0`, `target.length >= data.length`, `data != null`, `data[i] != null` für alle gültigen Indizes `i`.
  Wird die Methode mit `data=data1`, `pattern={0, 1}` und `target=target1` aufgerufen, entsteht folgendes Array:

2	0	1

- **`String insertMiddle(String input, String seps)`** fügt Zeichen aus `seps` in `input` ein. In der Mitte der Rückgabe befindet sich das erste Zeichen von `seps`. In der Mitte beider Hälften befindet sich dann das zweite Zeichen von `seps`, usw. Diese Vorgehensweise wird wiederholt, solange noch Zeichen zu vergeben sind und `input` in kleinere Teile geteilt werden kann. Ansonsten wird `input` zurückgegeben. Falls die Mitte nicht eindeutig bestimmt ist, wird so geteilt, dass die linke Hälfte die kürzere ist. Die Buchstaben aus `seps` werden bei der Bestimmung der Mitte nicht einbezogen.
  Diese Methode muss rekursiv implementiert werden.
  Vorbedingung(en): `input != null`, `seps != null`

Deklarieren Sie auch neue Arrays, die für die Tests benötigt werden.
Testen Sie alle Methoden und deren Seiteneffekte in `main` mit zumindest folgenden Aufrufen und weiteren Aufrufen (z.B. mit `deepToString`) für die Ausgaben.

Aufruf	**Ausgabe in main auf der Konsole**
`labelPath(3, new int[][] {})`	[[3, 3, 3], [3, 3, 3], [3, 3, 3]]
`labelPath(4, data0)`	[[4, -1, 4, 4], [4, 4, 4, 4], [4, 4, -1, 4], [-1, 4, 4, 4]]
`findMatches(data0, data0[1], target1)`	[0, 1, 0]
`findMatches(data1, data0[1], target1)`	[2, 0, 1]
`findMatches(data1, data0[2], target2)`	[0, 0, 3, 9]
`insertMiddle("XY", "abc")`	XaY
`insertMiddle("01234", "abc")`	0b1a2b3c4
`insertMiddle("01234567890123", "./-")`	0-12/34-56.7-89/01-23

Methode	**Bewertungsgrundlage**	**Punkt(e)**
`main`	Deklarationen	/ 2
	Testfälle korrekt implementiert	/ 4
`labelPath`	Korrekte Erstellung des Arrays (beide Dimensionen)	/ 2
	Einträge `n` korrekt	/ 4
	Einträge `-1` korrekt	/ 6
	Array enthält nur `-1` oder `n`	/ 2
	Korrekte Handhabung von `n=1`, Mehrfachvorkommen in `points`	/ 2
`findMatches`	Korrekte Schleifen	/ 6
	Einzelne Vergleiche mit `pattern` korrekt	/ 2
	Gefundene Werte korrekt in `target` eingetragen	/ 4
	Korrekte Reihenfolge in `target`	/ 2
	Nicht betroffene Indizes in `target` bleiben unverändert	/ 2
	Korrekte Handhabung von zu großem / nicht vorhandenem `pattern`	/ 2
`insertMiddle`	Korrekter Methodenansatz (Rückgabe vorhanden)	/ 2
	Basisfall vorhanden	/ 2
	Basisfall korrekt	/ 4
	Forschritt der Rekursion vorhanden	/ 2
	Forschritt der Rekursion korrekt	/ 4
	Korrekte Aufteilung des Strings	/ 2
	Korrekte Rückgabe	/ 4
Gesamt		/ 60