Past-paper walk-through

2D Array Traversals ▪ 4.12

eXercise

Questions in this set

- 2024 Q4
- 2022 Q4
- 2021 Q4
- 2019 Q4
- 2018 Q4
- 2017 Q4
- 2016 Q3
- 2015 Q1
- 2014 Q3

Reading the mark scheme

- **B1** a mark that stands on its own – the point is either made or not.
- **M1** a *method* mark: the right approach, even if the number is wrong.
- **A1** an *answer* mark, and it usually needs its M mark first.
- **C1** a working mark on the way to the answer.
- // separates answers the examiner accepts equally.
- **ecf** = error carried forward: a later part is marked on your own earlier answer.
- **owtte** = "or words to that effect" – the wording need not match.

Question 4

2024 ▪ Q4

This question involves a path through a two-dimensional (2D) array of integers, where the path is based on the values of elements in the array. When an element of the 2D array is accessed, the first index is used to specify the row and the second index is used to specify the column. The following `Location` class represents a row and column position in the 2D array.

```
k+kdpublic+w k+kdclass n+ncLocation
p
+w    k+kdprivate+w k+ktint+w ntheRowp;
+w    k+kdprivate+w k+ktint+w ntheColp;

+w    k+kdpublic+w n+nfLocationp(k+ktint+w nrp,+w k+ktint+w ncp)
+w    p
+w        ntheRow+w o=+w nrp;
+w        ntheCol+w o=+w ncp;
+w    p
+w    k+kdpublic+w k+ktint+w n+nfgetRowp(p)
+w    p+w  kreturn+w ntheRowp;+w  p

+w    k+kdpublic+w k+ktint+w n+nfgetColp(p)
+w    p+w  kreturn+w ntheColp;+w  p
```

Question 4

The following GridPath class contains the 2D array and methods to use to determine a path through the array. You will write two methods of the GridPath class.

```
k+kdpublic+w k+kdclass n+ncGridPath
p
+w    c+cm/** Initialized in the constructor with distinct values that never change */
+w    k+kdprivate+w k+ktinto[o]o[o]+w ngridp;

+w    c+cm/**
c+cm     * Returns the Location representing a neighbor of the grid element at row and col,
c+cm     * as described in part (a)
c+cm     * Preconditions: row is a valid row index and col is a valid column index in grid.
c+cm     *      row and col do not specify the element in the last row and last column of grid.
c+cm     */
+w    k+kdpublic+w nLocation+w n+nfgetNextLocp(k+ktint+w nrowp,+w k+ktint+w ncolp)
+w    p+w  c+cm/* to be implemented in part (a) */+w  p
+w    c+cm/**
c+cm     * Computes and returns the sum of all values on a path through grid, as described in
c+cm     * part (b)
c+cm     * Preconditions: row is a valid row index and col is a valid column index in grid.
c+cm     *      row and col do not specify the element in the last row and last column of grid.
```

Question 4

(a) Write the `getNextLoc` method, which returns a `Location` object that represents the smaller of two neighbors of the `grid` element at `row` and `col`, according to the following rules.

- The two neighbors that are considered as the element below the given element and the element to the right of the given element, if they exist.
- If both neighbors exist, the `Location` of the neighbor with the smaller value is returned. Two neighbors will always have different values.
- If only one neighbor exists, the `Location` of the existing neighbor is returned.

For example, assume that `grid` contains the following values.

	0	**1**	**2**	**3**	**4**
0	12	3	4	13	5
1	11	21	2	14	16
2	7	8	9	15	0
3	10	17	20	19	1
4	18	22	30	25	6

Question 4

The following table shows some sample calls to `getNextLoc`.

Method Call	Explanation
`getNextLoc(0, 0)`	Returns the neighbor to the right (the `Location` representing the element at row 0 and column 1), since $3 < 11$.
`getNextLoc(1, 3)`	Returns the neighbor below (the `Location` representing the element at row 2 and column 3), since $15 < 16$.
`getNextLoc(2, 4)`	Returns the neighbor below (the `Location` representing the element at row 3 and column 4), since the given element has no neighbor to the right.
`getNextLoc(4, 3)`	Returns the neighbor to the right (the `Location` representing the element at row 4 and column 4), since the given element has no neighbor below.

Question 4

In the example, the `getNextLoc` method will never be called with row 4 and column 4, as those values would violate the precondition of the method.
Complete the `getNextLoc` method.

```
c+cm/**
c+cm * Returns the Location representing a neighbor of the grid element at row and col,
c+cm * as described in part (a)
c+cm * Preconditions: row is a valid row index and col is a valid column index in grid.
c+cm *      row and col do not specify the element in the last row and last column of grid
c+cm */
k+kdpublic+w nLocation+w n+nfgetNextLocp(k+ktint+w nrowp,+w k+ktint+w ncolp)
```

Question 4

Class information for this question

```
k+kdpublic+w k+kdclass n+ncLocation
k+kdprivate+w k+ktint+w ntheRow
k+kdprivate+w k+ktint+w ntheCol

k+kdpublic+w n+nfLocationp(k+ktint+w nrp,+w k+ktint+w ncp)
k+kdpublic+w k+ktint+w n+nfgetRowp(p)
k+kdpublic+w k+ktint+w n+nfgetColp(p)

k+kdpublic+w k+kdclass n+ncGridPath
k+kdprivate+w k+ktinto[o]o[o]+w ngrid

k+kdpublic+w nLocation+w n+nfgetNextLocp(k+ktint+w nrowp,+w k+ktint+w ncolp)
k+kdpublic+w k+ktint+w n+nfsumPathp(k+ktint+w nrowp,+w k+ktint+w ncolp)
```

Question 4

	0	1	2	3	4
0	12	3	4	13	5
1	11	21	2	14	16
2	7	8	9	15	0
3	10	17	20	19	1
4	18	22	30	25	6

Question 4

	0	1	2	3	4
0	12	30	40	25	5
1	11	3	22	15	43
2	7	2	9	4	0
3	8	33	18	6	1

Question 4

2024 ▪ Q4

This question involves a path through a two-dimensional (2D) array of integers, where the path is based on the values of elements in the array. When an element of the 2D array is accessed, the first index is used to specify the row and the second index is used to specify the column. The following `Location` class represents a row and column position in the 2D array.

```
k+kdpublic+w k+kdclass n+ncLocation
p
+w    k+kdprivate+w k+ktint+w ntheRowp;
+w    k+kdprivate+w k+ktint+w ntheColp;

+w    k+kdpublic+w n+nfLocationp(k+ktint+w nrp,+w k+ktint+w ncp)
+w    p
+w        ntheRow+w o=+w nrp;
+w        ntheCol+w o=+w ncp;
+w    p

+w    k+kdpublic+w k+ktint+w n+nfgetRowp(p)
+w    p+w  kreturn+w ntheRowp;+w  p
```

Question 4

The following GridPath class contains the 2D array and methods to use to determine a path through the array. You will write two methods of the GridPath class.

```
k+kdpublic+w k+kdclass n+ncGridPath
p
+w    c+cm/** Initialized in the constructor with distinct values that never change */
+w    k+kdprivate+w k+ktinto[o]o[o]+w ngridp;

+w    c+cm/**
c+cm     * Returns the Location representing a neighbor of the grid element at row and col,
c+cm     * as described in part (a)
c+cm     * Preconditions: row is a valid row index and col is a valid column index in grid.
c+cm     *      row and col do not specify the element in the last row and last column of grid.
c+cm     */
+w    k+kdpublic+w nLocation+w n+nfgetNextLocp(k+ktint+w nrowp,+w k+ktint+w ncolp)
+w    p+w  c+cm/* to be implemented in part (a) */+w  p
+w    c+cm/**
c+cm     * Computes and returns the sum of all values on a path through grid, as described in
c+cm     * part (b)
c+cm     * Preconditions: row is a valid row index and col is a valid column index in grid.
c+cm     *      row and col do not specify the element in the last row and last column of grid.
```

Question 4

(b) Write the `sumPath` method, which returns the sum of all values on a path in `grid`. The path begins with the element at `row` and `col` and is determined by successive calls to `getNextLoc`. The path ends when the element in the last row and last column of `grid` is reached.
For example, consider the following contents of `grid`. The shaded elements of `grid` represent the values on the path that results from the method call `sumPath(1, 1)`. The method call returns 19 because $3 + 2 + 9 + 4 + 0 + 1 = 19$.

	0	**1**	**2**	**3**	**4**
0	12	30	40	25	5
1	11	**3**	22	15	43
2	7	**2**	**9**	**4**	**0**
3	8	33	18	6	**1**

Question 4

[Grid diagram: 4x5 table of integers, rows 0-3 and columns 0-4 labelled. The shaded (bolded) path cells forming the path from sumPath(1,1) are: (1,1)=3, (2,1)=2, (2,2)=9, (2,3)=4, (2,4)=0, (3,4)=1, matching the sum $3 + 2 + 9 + 4 + 0 + 1 = 19$.]
Write the `sumPath` method. Assume `getNextLoc` works as intended, regardless of what you wrote in part (a). You must use `getNextLoc` appropriately in order to receive full credit.

```
c+cm/**
c+cm * Computes and returns the sum of all values on a path through grid, as described in
c+cm * part (b)
c+cm * Preconditions: row is a valid row index and col is a valid column index in grid.
c+cm *      row and col do not specify the element in the last row and last column of grid
c+cm */
k+kdpublic+w k+ktint+w n+nfsumPathp(k+ktint+w nrowp,+w k+ktint+w ncolp)
```

Question 4

Class information for this question

```
k+kdpublic+w k+kdclass n+ncLocation
k+kdprivate+w k+ktint+w ntheRow
k+kdprivate+w k+ktint+w ntheCol

k+kdpublic+w n+nfLocationp(k+ktint+w nrp,+w k+ktint+w ncp)
k+kdpublic+w k+ktint+w n+nfgetRowp(p)
k+kdpublic+w k+ktint+w n+nfgetColp(p)

k+kdpublic+w k+kdclass n+ncGridPath
k+kdprivate+w k+ktinto[o]o[o]+w ngrid

k+kdpublic+w nLocation+w n+nfgetNextLocp(k+ktint+w nrowp,+w k+ktint+w ncolp)
k+kdpublic+w k+ktint+w n+nfsumPathp(k+ktint+w nrowp,+w k+ktint+w ncolp)
```

Question 4

2022 ▪ Q4

This question involves a two-dimensional array of integers that represents a collection of randomly generated data. A partial declaration of the Data class is shown. You will write two methods of the Data class.

```
k+kdpublic+w k+kdclass n+ncData
p
+w    k+kdpublic+w k+kdstatic+w k+kdfinal+w k+ktint+w nMAX+w o=+w c+cm/* value not shown */p;
+w    k+kdprivate+w k+ktinto[o]o[o]+w ngridp;

+w    c+cm/** Fills all elements of grid with randomly generated values, as described in part (a)
c+cm     *  Precondition: grid is not null.
c+cm     *       grid has at least one element.
c+cm     */
+w    k+kdpublic+w k+ktvoid+w n+nfrepopulatep(p)
+w    p+w  c+cm/* to be implemented in part (a) */+w  p
+w    c+cm/** Returns the number of columns in grid that are in increasing order, as described
c+cm     *  in part (b)
c+cm     *  Precondition: grid is not null.
c+cm     *       grid has at least one element.
c+cm     */
+w    k+kdpublic+w k+ktint+w n+nfcountIncreasingColsp(p)
```

Question 4

Class information for this question:

```
k+kdpublic+w k+kdstatic+w k+kdfinal+w k+ktint+w nMAX+w o=+w c+cm/* value n
k+kdprivate+w k+ktinto[o]o[o]+w ngrid
k+kdpublic+w k+ktvoid+w n+nfrepopulatep(p)
k+kdpublic+w k+ktint+w n+nfcountIncreasingColsp(p)
```

Question 4

(a) Write the `repopulate` method, which assigns a newly generated random value to each element of `grid`. Each value is computed to meet all of the following criteria, and all valid values must have an equal chance of being generated.

- The value is between 1 and `MAX`, inclusive.
- The value is divisible by 10.
- The value is not divisible by 100.

Complete the `repopulate` method.

```
c+cm/** Fills all elements of grid with randomly generated values, as described in part (a)
c+cm *  Precondition: grid is not null.
c+cm *      grid has at least one element.
c+cm */
k+kdpublic+w k+ktvoid+w n+nfrepopulatep(p)
```

Question 4

10	540	440	440
220	450	440	190

Question 4

10	50	40
20	40	20
30	50	30

Question 4

2022 ▪ Q4

This question involves a two-dimensional array of integers that represents a collection of randomly generated data. A partial declaration of the Data class is shown. You will write two methods of the Data class.

```
k+kdpublic+w k+kdclass n+ncData
p
+w    k+kdpublic+w k+kdstatic+w k+kdfinal+w k+ktint+w nMAX+w o=+w c+cm/* value not shown */p;
+w    k+kdprivate+w k+ktinto[o]o[o]+w ngridp;

+w    c+cm/** Fills all elements of grid with randomly generated values, as described in part (a)
c+cm     *  Precondition: grid is not null.
c+cm     *      grid has at least one element.
c+cm     */
+w    k+kdpublic+w k+ktvoid+w n+nfrepopulatep(p)
+w    p+w  c+cm/* to be implemented in part (a) */+w  p

+w    c+cm/** Returns the number of columns in grid that are in increasing order, as described
c+cm     *  in part (b)
c+cm     *  Precondition: grid is not null.
c+cm     *      grid has at least one element
```

Question 4

Class information for this question:

```
k+kdpublic+w k+kdstatic+w k+kdfinal+w k+ktint+w nMAX+w o=+w c+cm/* value n
k+kdprivate+w k+ktinto[o]o[o]+w ngrid
k+kdpublic+w k+ktvoid+w n+nfrepopulatep(p)
k+kdpublic+w k+ktint+w n+nfcountIncreasingColsp(p)
```

Question 4

(b) Write the `countIncreasingCols` method, which returns the number of columns in `grid` that are in increasing order. A column is considered to be in increasing order if the element in each row after the first row is greater than or equal to the element in the previous row. A column with only one row is considered to be increasing order.

The following examples show the `countIncreasingCols` return values for possible contents of `grid`.

The return value for the following contents of `grid` is 1, since the first column is in increasing order but the second and third columns are not.

10	**50**	**40**
20	40	20
30	50	30

Question 4

The return value for the following contents of `grid` is 2, since the first and third columns are in increasing order but the second and fourth columns are not.

10	**540**	**440**	**440**
220	450	440	190

Complete the `countIncreasingCols` method.

```
c+cm/** Returns the number of columns in grid that are in increasing order, as described
c+cm *  in part (b)
c+cm *  Precondition: grid is not null.
c+cm *       grid has at least one element.
c+cm */
k+kdpublic+w k+ktint+w n+nfcountIncreasingColsp(p)
```

Solution 4

(a) Mark scheme

- Model solution (repopulate, 4 pts):
- “‘java
- public void repopulate()
- {
- for (int row = 0; row < grid.length; row++)
- {
- for (int col = 0; col < grid[0].length; col++)
- {

Solution 4

- int rval = (int)(Math.random() * MAX) + 1;
- while (rval % 10 != 0 || rval % 100 == 0)
- {
- rval = (int)(Math.random() * MAX) + 1;
- }
- grid[row][col] = rval;
- }
- }
- }
- ‘‘‘

Solution 4

- (Alternate canonical: generate rval directly as a multiple of 10 via '((int)(Math.random() * (MAX / 10)) + 1) * 10' and reject only when rval % 100 == 0.) Points (1 each): (1) traverses grid with no bounds errors; (2) generates a random integer in a range based on MAX, cast to int; (3) ensures all produced values are divisible by 10 but not by 100 (e.g. via a rejection loop); (4) assigns valid, equally-distributed values in range to every element of grid (full algorithm).

Solution 4

(b) Mark scheme

- Model solution (countIncreasingCols, 5 pts):
- “‘java
- public int countIncreasingCols()
- {
- int count = 0;
- for (int col = 0; col < grid[0].length; col++)
- {
- boolean ordered = true;

Solution 4

- for (int row = 1; row < grid.length; row++)
- {
- if (grid[row][col] < grid[row-1][col])
- {
- ordered = false;
- }
- }
- if (ordered)
- {
- count++;

Solution 4

- }
- }
- return count;
- }
- ‘‘‘
- Points (1 each): (5) traverses grid in column-major order with no loop-header bounds errors; (6) compares two elements in the same column of grid (adjacent rows); (7) correctly determines whether a single column is in increasing order (each element after the first $>=$ the one above it), resetting per column; (8) counts all columns identified as increasing; (9) returns the calculated count.