CSE143 Section #19 Problems

Two-dimensional arrays are covered in chapter 7. The first index represents the row and the second index represents the column, as in data[3][5] for the value in row 3 and column 5. In a rectangular array, the number of columns is the same for each row. It is typically constructed as follows (constructing an array of four rows and six columns):

```
int[][] data = new int[4][6];
```

In a jagged array, the number of columns varies across rows. You construct one by first constructing the array of rows and then each individual row, as in:

```
int[][] data = new int[3][];
data[0] = new int[2];
data[1] = new int[3];
data[2] = new int[5];
```

This constructs an array of three rows where row 0 has 2 columns, row 1 has 3 columns, and row 2 has 5 columns.

For all problems involving a two-dimensional array, the contents should be indicated using the Arrays.deepToString format of nested bracketed lists. For example, given the following array:

```
int[][] data = {{8, 12, 14}, {7, 19, 4}, {8, 3, 42}};
```

Its contents should be displayed as follows:

```
[[8, 12, 14], [7, 19, 4], [8, 3, 42]]
```

1. Consider the following method:

```
public static int[][] mystery1(int n) {
    int[][] result = new int[n][2 * n - 1];
    for (int i = 0; i < result.length; i++) {
        for (int j = 0; j < result[i].length; j++) {
            result[i][j] = i + j;
        }
    }
    return result;
}
```

For each call below, indicate the contents of the two-dimensional array that is returned.

Method Call	Value Returned
mystery1(1)	_____
mystery1(2)	_____
mystery1(3)	_____
mystery1(4)	_____
	_____

2. Consider the following method:

```
public List<Integer> mystery2(int[][] data) {
    List<Integer> result = new LinkedList<Integer>();
    for (int i = 0; i < data.length; i++) {
        int sum = 0;
        for (int j = 0; j < data[i].length; j++) {
            sum = sum + j * data[i][j];
        }
        result.add(sum);
    }
    return result;
}
```

In the left-hand column below are specific two-dimensional arrays. Indicate in the right-hand column what values would be stored in the list returned by method mystery2 if the array in the left-hand column is passed as a parameter to mystery2.

Two-Dimensional Array	Contents of List Returned
[[1, 2, 3], [4, 5, 6]]	_____
[[3, 4], [1, 2, 3], [], [5, 6]]	_____
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]	_____

List<E> Methods (10.1)

add(value)	appends value at end of list
add(index, value)	inserts given value at given index, shifting subsequent values right
clear()	removes all elements of the list
indexOf(value)	returns first index where given value is found in list (-1 if not found)
get(index)	returns the value at given index
remove(index)	removes/returns value at given index, shifting subsequent values left
set(index, value)	replaces value at given index with given value
size()	returns the number of elements in list
addAll(list)	adds all elements from the given collection to the end of the list
contains(value)	returns true if the given value is found somewhere in this list
remove(value)	finds and removes the given value from this list
removeAll(list)	removes any elements found in the given collection from this list
iterator()	returns an object used to examine the contents of the list

Iterator<E> Methods (11.1)

hasNext()	returns true if there are more elements to be read from collection
next()	reads and returns the next element from the collection
remove()	removes the last element returned by next from the collection

Set<E> Methods (11.2)

add(value)	adds the given value to the set
contains(value)	returns true if the given value is found in the set
remove(value)	removes the given value from the set
clear()	removes all elements of the set
size()	returns the number of elements in the set
isEmpty()	returns true if the set's size is 0
addAll(collection)	adds all elements from the given collection to the set
containsAll(collection)	returns true if set contains every element from given collection
removeAll(collection)	removes any elements found in the given collection from this set
retainAll(collection)	removes any elements not found in the given collection from this set
iterator()	returns an object used to examine contents of the set

3. Write a method called acronymFor that takes a list of strings as a parameter and that returns the corresponding acronym. You form an acronym by combining the capitalized first letter of a series of words. For example, the list [laughing, out, loud] produces the acronym "LOL". The list [Computer, Science and, Engineering] produces the acronym "CSE". You may assume that all of the strings are nonempty. Your method is not allowed to change the list passed to it as a parameter. If passed an empty list, your method should return the empty string. You may construct iterators and strings, but you are not allowed to construct other structured objects.

4. Write a method called switchPairs that switches the order of values in a List of Strings in a pairwise fashion. Your method should switch the order of the first two values, then switch the order of the next two, switch the order of the next two, and so on. For example, if a list stores:

   [four, score, and, seven, years, ago]

   your method should switch the first pair (four, score), the second pair (and, seven) and the third pair (years, ago), to yield this list:

   [score, four, seven, and, ago, years]

   If there are an odd number of values in the list, the final element is not moved. For example, if the original list had been:

   [to, be, or, not, to, be, hamlet]

   It would again switch pairs of values, but the final value (hamlet) would not be moved, yielding this list:

   [be, to, not, or, be, to, hamlet]

5. Write a method stutter that doubles the size of the list by replacing every integer in the list with two of that integer. For example, if the list stores the following sequence of integers when the method is called:

   [1, 8, 19, 4, 17]

   It should store the following sequence of integers after stutter is called:

   [1, 1, 8, 8, 19, 19, 4, 4, 17, 17]

6. Write a method called reverse3 that takes a List of integer values as a parameter and that reverses each successive sequence of three values in the list. For example, suppose that a variable called list stores the following

sequence of values:

```
[3, 8, 19, 42, 7, 26, 19, -8, 193, 204, 6, -4]
```

and we make the following call:

```
reverse3(list);
```

Afterwards the list should store the following sequence of values:

```
[19, 8, 3, 26, 7, 42, 193, -8, 19, -4, 6, 204]
```

The first sequence of three values (3, 8, 19) has been reversed to be (19, 8, 3). The second sequence of three values (42, 7, 26) has been reversed to be (26, 7, 42). And so on. If the list has extra values that are not part of a sequence of three, those values are unchanged. For example, if the list had instead stored:

```
[3, 8, 19, 42, 7, 26, 19, -8, 193, 204, 6, -4, 99, 2]
```

The result would have been:

```
[19, 8, 3, 26, 7, 42, 193, -8, 19, -4, 6, 204, 99, 2]
```

Notice that the values (99, 2) are unchanged in position because they were not part of a sequence of three values.

7. Write a method hasOdd that takes a set of integers as a parameter and that returns true if the set contains at least one odd integer, false otherwise.

8. Write a method removeEvens that takes a set of integers as a parameter and that removes the even values from the set, returning those values as a new set. The new set should be ordered in increasing numerical order. For example, if a set s1 contains these values:

```
[0, 17, 16, 7, 10, 12, 13, 14]
```

and we make the following call:

```
Set<Integer> s2 = removeEvens(s1);
```

Then after the call s1 and s2 would contain the following values:

```
s1: [17, 7, 13]
s2: [0, 10, 12, 14, 16]
```

9. Write a method containsAll that takes two sets of integers as parameters and that returns true if the first set contains all of the values of the second set and that returns false otherwise. For example, if the two sets are:

```
s1: [17, 16, 7, 10, 12, 13, 14]
s2: [7, 12, 13]
```

then the call containsAll(s1, s2) would return true while the call containsAll(s2, s1) would return false. You are implementing a two-argument alternative to the standard Set method called containsAll, so you are not allowed to call that method to solve this problem. You are also not allowed to construct any structured objects to solve the problem (no set, list, stack, queue, string, etc). Your method should not change either set passed as a parameter.

10. Write a method called equals that takes two sets of integers as parameters and that returns true if the sets are equal. Two sets are considered equal if they store the same values. For example, given sets:

```
s1: [5, 3, 1, 0]
s2: [0, 1, 5, 3]
s3: [1, 0, 5, 3, 4]
```

The call equals(s1, s2) would return true while the calls equals(s1, s3) and equals(s2, s3) would return false. As in the examples above, you can not assume that the set values are ordered.

You are implementing a two-argument alternative to the standard Set method called equals, so you are not allowed to call that method or the containsAll method to solve this problem. You may construct iterator objects, but you are also not allowed to construct any structured objects to solve the problem (no set, list, stack, queue, string, etc). Your method should not change either of the sets passed as parameters.

11. Write a method called acronyms that takes a set of word lists as a parameter and that returns a map whose keys are acronyms and whose values are the word lists that produce that acronym. Acronyms are formed from each list as described in problem 1. Recall that the list [laughing, out, loud] produces the acronym "LOL". The list [League, of, Legends] also produces the acronym "LOL". Suppose that a variable called lists stores this set of word lists:

```
[[attention, deficit], [Star, Trek, Next, Generation],
 [laughing, out, loud], [International, Business, Machines],
 [League, of, Legends], [anno, domini], [art, director],
 [Computer, Science and, Engineering]]
```

Each element of this set is a list of values of type String. You may assume that each list is nonempty and that each string in a list is nonempty. Your method should construct a map whose keys are acronyms and whose values are sets of the word lists that produce that acronym. For example, the call acronyms(lists) should produce the following map:

```
{AD=[[attention, deficit], [anno, domini], [art, director]],
 CSE=[[Computer, Science and, Engineering]],
 IBM=[[International, Business, Machines]],
 LOL=[[laughing, out, loud], [League, of, Legends]],
 STNG=[[Star, Trek, Next, Generation]]}
```

Notice that there are 5 unique acronyms produced by the 8 lists in the set. Each acronym maps to a set of the word lists for that acronym. Your method should not make copies of the word lists; the sets it constructs should store references to those lists. As in the example above, the keys of the map that you construct should be in sorted order. You may assume that a working version of acronymFor as described in problem 4 is available for you to use no matter what you wrote for problem 4. Your method is not allowed to change either the set passed as a parameter or the lists within the set.

12. Write a method called deepCopy that takes as a parameter a map whose keys are strings and whose values are lists of integers and that creates and returns a new map that is a copy of the map parameter. For example, given a variable called map that stores the following information:

```
{"cse143"=[42, 17, 42, 42], "goodbye"=[3, 10, -5],
 "hello"=[16, 8, 0, 0, 106]}
```

the call deepCopy(map) should return a new map whose structure and content are identical to map. Any later modifications to map or the lists in map following this call should not be reflected in the copy. The map you construct should store keys in alphabetical order. Your method should not modify the contents of the map passed as a parameter. In constructing collection objects, you are required to use the 0-argument constructors.

13. Write a method called extractEqual that takes a set of Point objects and that returns a new set that contains all of the Point objects where the x and y values are equal to each other. For example, if a set called points contains the following values:

```
[[x=42,y=3], [x=4,y=2], [x=18,y=1], [x=7,y=8], [x=-2,y=-2], [x=3,y=3],
 [x=7,y=7], [x=0,y=82], [x=14,y=14], [x=3,y=13], [x=-3,y=4], [x=1,y=3]]
```

then the call extractEqual(points) should return the following set:

```
[[x=-2,y=-2], [x=3,y=3], [x=7,y=7], [x=14,y=14]]
```

The original set should be unchanged and you should not construct any new Point objects in solving this problem. As a result, both sets will end up referring to the Point objects in which the x and y coordinates are equal. Your method is expected to have reasonable efficiency in that it shouldn't lead to more set operations than it needs to.

Solution to CSE143 Section #19 Problems

1.

Method Call	Value Returned
mystery1(1)	[[0]]
mystery1(2)	[[0, 1, 2], [1, 2, 3]]
mystery1(3)	[[0, 1, 2, 3, 4], [1, 2, 3, 4, 5], [2, 3, 4, 5, 6]]
mystery1(4)	[[0, 1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6, 7], [2, 3, 4, 5, 6, 7, 8], [3, 4, 5, 6, 7, 8, 9]]

2.

Two-Dimensional Array	Contents of List Returned
[[1, 2, 3], [4, 5, 6]]	[8, 17]
[[3, 4], [1, 2, 3], [], [5, 6]]	[4, 8, 0, 6]
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]	[8, 17, 26]

3. One possible solution appears below.

```
public String acronymFor(List<String> words) {
    String acronym = "";
    for (String next : words) {
        acronym += next.charAt(0);
    }
    return acronym.toUpperCase();
}
```

4. Two possible solutions appear below.

```
public void switchPairs(List<String> list) {
    for (int i = 0; i < list.size() - 1; i += 2) {
        String first = list.remove(i);
        list.add(i + 1, first);
    }
}

public void switchPairs(List<String> list) {
    int i = 0;
    while (i < list.size() - 1) {
        String first = list.get(i);
        list.set(i, list.get(i + 1));
        list.set(i + 1, first);
        i += 2;
    }
}
```

5. One possible solution appears below.

```
public void stutter(List<Integer> list) {
    for (int i = 0; i < list.size(); i += 2) {
        list.add(i, list.get(i));
    }
}
```

6. Two possible solutions appear below.

```
public void reverse3(List<Integer> list) {
    for (int i = 0; i < list.size() - 2; i += 3) {
        int n1 = list.get(i);
        int n3 = list.get(i + 2);
        list.set(i, n3);
        list.set(i + 2, n1);
    }
}

public void reverse3(List<Integer> list) {
    for (int i = 0; i < list.size() - 2; i += 3) {
        list.add(i, list.remove(i + 2));
        list.add(i + 2, list.remove(i + 1));
```

```
        }
    }
```

7. One possible solution appears below.

```
    public boolean hasOdd(Set<Integer> set) {
        for (int value : set) {
            if (value % 2 != 0) {
                return true;
            }
        }
        return false;
    }
```

8. One possible solution appears below.

```
    public Set<Integer> removeEvens(Set<Integer> s) {
        Set<Integer> result = new TreeSet<Integer>();
        Iterator<Integer> i = s.iterator();
        while (i.hasNext()) {
            int n = i.next();
            if (n % 2 == 0) {
                result.add(n);
                i.remove();
            }
        }
        return result;
    }
```

9. One possible solution appears below.

```
    public boolean containsAll(Set<Integer> s1, Set<Integer> s2) {
        Iterator<Integer> i = s2.iterator();
        while (i.hasNext()) {
            if (!s1.contains(i.next())) {
                return false;
            }
        }
      return true;
    }
```

10. One possible solution appears below.

```
    public boolean equals(Set<Integer> s1, Set<Integer> s2) {
        if (s1.size() != s2.size()) {
            return false;
        }
        for (int n : s1) {
            if (!s2.contains(n)) {
                return false;
            }
        }
        return true;
    }
```

11. One possible solution appears below.

```
    public Map<String, Set<List<String>>> acronyms(
            Set<List<String>> lists) {
        Map<String, Set<List<String>>> result =
            new TreeMap<String, Set<List<String>>>();
        for (List<String> words : lists) {
            String acronym = acronymFor(words);
            if (!result.containsKey(acronym)) {
                result.put(acronym, new HashSet<List<String>>());
            }
            result.get(acronym).add(words);
        }
        return result;
    }
```

12. One possible solution appears below.

```
public Map<String, List<Integer>> deepCopy(
        Map<String, List<Integer>> map) {
    Map<String, List<Integer>> copy =
        new TreeMap<String, List<Integer>>();
    for (String key : map.keySet()) {
        List<Integer> newList = new ArrayList<Integer>();
        for (int n : map.get(key)) {
            newList.add(s);
        }
        copy.put(key, newList);
    }
    return copy;
}
```

13. Two possible solutions appear below.

```
public Set<Point> extractEqual(Set<Point> data) {
    Set<Point> result = new HashSet<Point>();
    for (Point p : data) {
        if (p.getX() == p.getY()) {
            result.add(p);
        }
    }
    return result;
}

public Set<Point> extractEqual(Set<Point> data) {
    Set<Point> result = new HashSet<Point>();
    Iterator<Point> i = data.iterator();
    while (i.hasNext()) {
        Point p = i.next();
        if (p.getX() == p.getY()) {
            result.add(p);
        }
    }
    return result;
}
```

Optional - Sudoku Solver

The remaining problems involve writing code that involves two-dimensional arrays. The specific program we will consider is a Sudoku solver that uses recursive backtracking. As with the 8 queens problem, we will define a class that handles much of the low-level detail of the puzzle. In particular, we will have a class called Grid that keeps track of the current state of the 9x9 Sudoku grid. The basic class structure will be:

```
public class Grid {
    public static final int SIZE = 9;
    private int[][] grid;

    // methods
}
```

We are writing a fairly specific class with a size of 9, but it is best to use a named constant when possible to add to readability. The grid will store values between 1 and 9. We will assume that the value 0 means that the cell is empty (no value has been assigned to it).

Several of the methods involved use a cell number to refer to a particular location. The idea is that the cells are numbered starting at 0 in row/major order. In other words, we number all of the first row, then all of the second row, and so on. So the cell numbers are:

```
 0  1  2  3  4  5  6  7  8
 9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26
...
72 73 74 75 76 77 78 79 80
```

Because we are starting at 0, the highest cell number is 80.

3. Write a method called print that prints the grid contents to System.out with each row printed on a separate line and with a space after each grid value. Empty cells (cells that store a value of 0) should be printed as a dash. For example, if you have a Grid variable g and you make this call:

   ```
   g.print();
   ```

   you should get output that looks something like this:

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

   Instead of using the SIZE constant, write this method so that it would work no matter what the dimensions of the grid are and even if the rows had different numbers of elements in them.

4. Write a constructor for the Grid class that takes a Scanner as a parameter. The Scanner will contain values like those produced by print. In other words, cells that are occupied will be listed with a number between 1 and 9 and cells that are empty will be listed with some other token like a dash.

You may assume the Scanner is open and that it contains a legal sequence of tokens (81 of them).

5. Write a method called place that takes a cell number and a value n and that stores the value n in that cell. Your method will have to convert the cell number to a row and column in your grid.

6. Write a method called remove that takes a cell number and that removes the value in that cell (resetting it to 0). As with the place method, you will have to convert the cell number to a row and column in your grid.

7. Write a method called getUnassignedLocation. It should look through the grid in row/major order and return the cell number of the first unoccupied cell. If there are no unoccupied cells, it should return the value -1.

8. Write a method called noConflicts that takes a cell number and a value n and that returns true if it is possible to store n in that cell without generating any Sudoku conflicts. Remember that Sudoku does not allow repetition in any given row, column, or block.

9. Write the recursive backtracking code that will search all possible solutions to the Sudoku puzzle. You should write a method called explore that takes a Grid as a parameter and that attempts to fill the grid with a solution. It should return whether or not a solution is found. Below is a method that calls the explore method and reports the result:

```
public static void solve(Grid g) {
    if (!explore(g))
        System.out.println("No solution.");
    else {
        System.out.println("One solution is as follows:");
        g.print();
    }
}
```

The Grid object has the following public member functions available to you:

`getUnassignedLocation()`	returns the cell number of the first unoccupied cell (-1 if no such cell exists)
`noConflicts(cellNumber, n)`	returns whether it is legal to place the value n in the given cell without violating Sudoku constraints
`place(cellNumber, n)`	places the value n in the given cell
`remove(cellNumber)`	removes the value in the given cell, resetting it to be unoccupied

Solutions - Sudoku Solver

A complete solution to the Grid class and the Sudoku backtracking program can be found on the course website. It includes an extra feature not described in the problem statement of counting how many calls are made on the place method.

https://courses.cs.washington.edu/courses/cse143/22au/sections/Grid.java
https://courses.cs.washington.edu/courses/cse143/22au/sections/Sudoku.java