Assignment objectives

Design and implement recursive algorithms in the pattern of "Decrease by 1", a subcategory of Decrease and Conquer algorithms.

Introduction

Middle of January is just about time for Santa's Elves to return from their post-Christmas vacations and get to work on preparing for next year's holiday season! First on the to-do list: deck the halls of Santa's Workshop with a dazzling display of Christmas lights.

Each (LED) light bulb can display three different colours (red, white, green), and the colours along a string are determined by a *pattern*. A pattern has one digit for each bulb, with red=0, white=1, green=2. For example, the following pattern for a string of 8 lights means that the 1st, 3rd, 5th, and 7th lights are red and the rest are green:

```
02020202
```

To animate the lights you can specify a sequence of these patterns. The light string will cycle through the patterns in the list. E.g. here is a sequence of three patterns that will make the colours look like they are marching along the string like weirdly-coloured ants.

```
01201201
12012012
20120120
```

Your assignment is to write a program that automatically generates *lists of light patterns*.

Class name

Please name your Java class "Lab3".

Part 1

Santa is a pattern-junkie and a bit of a completist, and he wants to see ALL the patterns. You need to write a function that will produce all the possible patterns for a given number (N) of lights. It does not matter what order the patterns appear on the list.

Use a recursive, decrease-and-conquer approach to generate the strings. Recall the common pattern of a "decrease by 1" decrease-and-conquer algorithm for a problem size N:

1. Recursively solve the problem for an input size of N-1.
2. Adjust, adapt, or extend the results for N-1 to obtain the results for input size N.

Your public function must meet the requirements below. It is OK to have private helper functions.

Name: generateAllPatterns
Input arguments: (int) length of one pattern; you may assume this is > 0
Return value: ArrayList of Strings – the list of all patterns of the given length

For example, if this function is called with an argument of 3, it would return the following list (27 patterns). *Please note that these are strings, not numbers*:

```
[000, 001, 002, 010, 011, 012, 020, 021, 022, 100, 101, 102, 110, 111,
     112, 120, 121, 122, 200, 201, 202, 210, 211, 212, 220, 221, 222]
```

Here is the list for N=4 (81 patterns):

```
[0000, 0001, 0002, 0010, 0011, 0012, 0020, 0021, 0022, 0100, 0101, 0102,
     0110, 0111, 0112, 0120, 0121, 0122, 0200, 0201, 0202, 0210, 0211,
     0212, 0220, 0221, 0222, 1000, 1001, 1002, 1010, 1011, 1012, 1020,
     1021, 1022, 1100, 1101, 1102, 1110, 1111, 1112, 1120, 1121, 1122,
     1200, 1201, 1202, 1210, 1211, 1212, 1220, 1221, 1222, 2000, 2001,
     2002, 2010, 2011, 2012, 2020, 2021, 2022, 2100, 2101, 2102, 2110,
     2111, 2112, 2120, 2121, 2122, 2200, 2201, 2202, 2210, 2211, 2212,
     2220, 2221, 2222]
```

Part 2

Your first program is a HUGE hit with the elves, but it reveals a hardware design flaw in the microprocessor that controls the light display: If the controller reads any pattern that contains two of the same digit in a row, it goes into an error condition that requires a “hard” reboot—i.e. somebody has to unplug the whole thing and plug it back in! This puts a serious dent in the elves’ productivity, so you decide to make an alternative list that contains *only the patterns that do not have any repeated, adjacent digits*. (Hopefully Santa won’t notice!)

Your public function must meet the requirements below. It is OK to have private helper functions.

Name: generatePatternsWithNoDoubleDigits
Input arguments: (int) length of one pattern, may assume > 0
Return value: ArrayList of Strings – the list of all patterns of the given length

There aren’t nearly as many of these patterns. Here is the list for N=4 (24 patterns):

```
[0101, 0102, 0120, 0121, 0201, 0202, 0210, 0212, 1010, 1012, 1020, 1021,
     1201, 1202, 1210, 1212, 2010, 2012, 2020, 2021, 2101, 2102, 2120,
     2121]
```

And for N=5 (48 patterns):

```
[01010, 01012, 01020, 01021, 01201, 01202, 01210, 01212, 02010, 02012,
     02020, 02021, 02101, 02102, 02120, 02121, 10101, 10102, 10120,
     10121, 10201, 10202, 10210, 10212, 12010, 12012, 12020, 12021,
     12101, 12102, 12120, 12121, 20101, 20102, 20120, 20121, 20201,
     20202, 20210, 20212, 21010, 21012, 21020, 21021, 21201, 21202,
     21210, 21212]
```

Important: You should NOT produce this list by generating all strings and then deleting items with repeated digits. This function should definitely NOT call generateAllPatterns. Write a standalone algorithm that follows the same Decrease and Conquer pattern (recursively solve for N-1, and then “extend” the list to obtain results for N).

Main program

You are not required to have a `main()` function in your submission. But of course you should thoroughly test your two Decrease and Conquer algorithms, and having a `main()` function in your code is an easy way to do that. The following information may be helpful:

```
Generating all patterns of a given length:
Length 1 produces 3 patterns.
Length 2 produces 9 patterns.
Length 3 produces 27 patterns.
Length 4 produces 81 patterns.
Length 5 produces 243 patterns.
Length 6 produces 729 patterns.
Length 7 produces 2187 patterns.
Length 8 produces 6561 patterns.
Length 9 produces 19683 patterns.
Length 10 produces 59049 patterns.

Generating patterns without double-digits:
Length 1 produces 3 patterns.
Length 2 produces 6 patterns.
Length 3 produces 12 patterns.
Length 4 produces 24 patterns.
Length 5 produces 48 patterns.
Length 6 produces 96 patterns.
Length 7 produces 192 patterns.
Length 8 produces 384 patterns.
Length 9 produces 768 patterns.
Length 10 produces 1536 patterns.
```

Submission information

Due date: As shown on Learning Hub.

Submit the following to the drop box on Learning Hub:

- Just your Java source code (*.java file).
- File name is not important.
- Please *do not zip* or otherwise archive your code. Plain Java files only.
- Please *do not zip* or include your entire project directory.

Marking information

This lab is worth 20 points.

5/20 of these points are reserved for compliance with the COMP 3760 Coding Requirements.

Virtual donut

This part is not required; it is just for fun.

Can you think of any other interesting light patterns that can be generated by using Decrease and Conquer algorithms of a type similar to the assignment?

Below are some ideas. **Caution**: I have not written algorithms for these, and I suspect at least one in particular actually is impossible to do *with the decrease-by-constant technique*.

- Generate all strings of length N that are palindromes. Hint: try to find a *decrease by 2* algorithm instead of decrease by 1.

- Suppose the light bulbs have *five* colours available, but you want to generate all the strings of length N that use *exactly three* colours.
- Generate all strings of length N that have exactly the same number of each colour bulb. Your function can assume that it will be given input N that is a multiple of 3.
- Invent your own!