The following tasks are inspired by tasks from https://adventofcode.com/.

1 Find Start-Of-Packet Marker (3 points)

Your task is to find the start-of-packet marker in a data stream. In the protocol being used, the start of a packet is indicated by a sequence of four characters that are all different.

Your code needs to identify the first position in the input string where the four characters were all different. Specifically, it needs to report the number of characters from the beginning of the string to the end of the first such four-character marker.

For example, suppose you receive the following string:

```
mjqjpqmgbljsphdztnvjfqwrcgsmlb
```

After the first three characters (mjq) have been received, there haven't been enough characters received yet to find the marker. The first time a marker could occur is after the fourth character is received, making the most recent four characters mjqj. Because j is repeated, this isn't a marker.

The first time a marker appears is after the seventh character arrives. Once it does, the last four characters received are jpqm, which are all different. In this case, your code should report the value 7, because the first start-of-packet marker is complete after 7 characters have been processed.

Here are a few more examples:

- b**vwbj**plbgvbhsrlpgdmjqwftvncz: first marker after character 5
- np**pdvj**thqldpwncqszvftbrmjlhg: first marker after character 6
- nznrnf**rfnt**jfmvfwmzdfjlvtqnbhcprsg: first marker after character 10
- zcfzfwz**zqfr**ljwzlrfnpqdbhtmscgvjw: first marker after character 11

How many characters need to be processed before the first start-of-packet marker is detected? Use this input: https://www.dfki.de/~steffen/advanced-java/marker.txt
Write a unit test to demonstrate your code.

Hint: You could use a class that implements the `java.util.Queue` interface, e.g. `java.util.LinkedList`.

2 Playing Bingo (5 points)

Bingo is played on a set of boards each consisting of a 5x5 grid of numbers. Numbers are chosen at random, and the chosen number is marked on all boards on which it appears. (Numbers may not appear on all boards.) If all numbers in any row or any column of a board are marked, that board wins. Diagonals don't count.

Your input consists of a random order in which to draw numbers and a random set of boards. For example:

```
7,4,9,5,11,17,23,2,0,14,21,24,10,16,13,6,15,25,12,22,18,20,8,19,3,26,1

22 13 17 11  0
 8  2 23  4 24
21  9 14 16  7
 6 10  3 18  5
 1 12 20 15 19

 3 15  0  2 22
 9 18 13 17  5
19  8  7 25 23
20 11 10 24  4
14 21 16 12  6

14 21 17 24  4
10 16 15  9 19
18  8 23 26 20
22 11 13  6  5
 2  0 12  3  7
```

You can use this input when developing and testing your code. It is available here: https://www.dfki.de/~steffen/advanced-java/bingo-demo.txt.

After the first five numbers are drawn (7, 4, 9, 5, and 11), there are no winners, but the boards are marked as follows (shown here adjacent to each other to save space):

```
22 13 17 11  0          3 15  0  2 22         14 21 17 24  4
 8  2 23  4 24          9 18 13 17  5         10 16 15  9 19
21  9 14 16  7         19  8  7 25 23         18  8 23 26 20
 6 10  3 18  5         20 11 10 24  4         22 11 13  6  5
 1 12 20 15 19         14 21 16 12  6          2  0 12  3  7
```

After the next six numbers are drawn (17, 23, 2, 0, 14, and 21), there are still no winners:

```
22 13 17 11  0          3 15  0  2 22         14 21 17 24  4
 8  2 23  4 24          9 18 13 17  5         10 16 15  9 19
21  9 14 16  7         19  8  7 25 23         18  8 23 26 20
 6 10  3 18  5         20 11 10 24  4         22 11 13  6  5
 1 12 20 15 19         14 21 16 12  6          2  0 12  3  7
```

Finally, 24 is drawn:

```
22 13 17 11  0         3 15  0  2 22         14 21 17 24  4
 8  2 23  4 24         9 18 13 17  5         10 16 15  9 19
21  9 14 16  7        19  8  7 25 23         18  8 23 26 20
 6 10  3 18  5        20 11 10 24  4         22 11 13  6  5
 1 12 20 15 19        14 21 16 12  6          2  0 12  3  7
```

At this point, the third board wins because it has at least one complete row or column of marked numbers (in this case, the entire top row is marked: `14 21 17 24 4`).

The score of the winning board can now be calculated. Start by finding the **sum of all unmarked numbers** on that board; in this case, the sum is 188. Then, add **the number that was just called** when the board won, 24, to get the final score, 188 + 24 = **212**.

Write code that plays bingo and reports the winning score using this input: https://www.dfki.de/~steffen/advanced-java/bingo.txt.
Write a unit test to demonstrate your code.

3 Free Disk Space (7 points)

You need to install an update on your computer. Unfortunately, you don't have enough disk space left. You browse around the file system to assess the situation and save the resulting terminal output (your input). For example:

```
$ cd /
$ ls
dir a
14848514 b.txt
8504156 c.dat
dir d
$ cd a
$ ls
dir e
29116 f
2557 g
62596 h.lst
$ cd e
$ ls
584 i
$ cd ..
$ cd ..
$ cd d
$ ls
4060174 j
8033020 d.log
5626152 d.ext
7214296 k
```

You can use this input when developing and testing your code. It is available here: https://www.dfki.de/~steffen/advanced-java/dir-finder-demo.txt.

The file system consists of a tree of files (plain data) and directories (which can contain other directories or files). The outermost directory is called /. You can navigate around the file system, moving into or out of directories and listing the contents of the directory you're currently in.

Within the terminal output, lines that begin with $ are **commands you executed**, very much like some modern computers:

- `cd` means **change directory**. This changes which directory is the current directory, but the specific result depends on the argument:
  - `cd x` moves **in** one level: it looks in the current directory for the directory named x and makes it the current directory.
  - `cd ..` moves **out** one level: it finds the directory that contains the current directory, then makes that directory the current directory.

  - `cd /` switches the current directory to the outermost directory, /.

- `ls` means **list**. It prints out all of the files and directories immediately contained by the current directory:
  - `123 abc` means that the current directory contains a file named `abc` with size `123` bytes
  - `dir xyz` means that the current directory contains a directory named `xyz`.

Given the commands and output in the example above, you can determine that the filesystem looks visually like this:

```
- / (dir)
  - a (dir)
    - e (dir)
      - i (file, size=584)
    - f (file, size=29116)
    - g (file, size=2557)
    - h.lst (file, size=62596)
  - b.txt (file, size=14848514)
  - c.dat (file, size=8504156)
  - d (dir)
    - j (file, size=4060174)
    - d.log (file, size=8033020)
    - d.ext (file, size=5626152)
    - k (file, size=7214296)
```

Here, there are four directories: / (the outermost directory), a and d (which are in /), and e (which is in a). These directories also contain files of various sizes.

The total size of a directory is the sum of the sizes of the files it contains, directly or indirectly. (Directories themselves do not count as having any intrinsic size.) The total sizes of the directories above can be found as follows:

- The total size of directory e is 584 because it contains a single file i of size 584 and no other directories.
- The directory a has total size 94.853 because it contains files f (size 29.116), g (size 2.557), and h.lst (size 62.596), plus file i indirectly (a contains e which contains i).
- Directory d has total size 24.933.642.
- As the outermost directory, / contains every file. Its total size is 48.381.165, the sum of the size of every file.

The total disk space available to your file system is 90.000.000 bytes. To run the update, you need unused space of at least 60.000.000 bytes. You need to find a directory you can delete that will free up enough space to run the update.

In the example above, the total size of the outermost directory (and thus the total amount of used space) is 48.381.165; this means that the size of the unused space must currently be 41.618.835, which isn't quite the 60.000.000 required by the update. Therefore, the update still requires a directory with total size of at least 18.381.165 to be deleted before it can run.

To achieve this, you have the following options:

- Delete directory e, which would increase unused space by 584.
- Delete directory a, which would increase unused space by 94.853.
- Delete directory d, which would increase unused space by 24.933.642.
- Delete directory /, which would increase unused space by 48.381.165.

Directories e and a are both too small; deleting them would not free up enough space. However, directories d and / are both big enough! Between these, choose the smallest: d, increasing unused space by 24.933.642.

Find the smallest directory and its size that, if deleted, would free up enough space on the file system to run the update. Use this input:
https://www.dfki.de/~steffen/advanced-java/dir-finder.txt.
Write a unit test to demonstrate your code.

Hints:

- You might want to use a `java.util.Stack` to keep track of the path to the current directory.
- Arbitrary instances of Java classes can be sorted when the class implements the `Comparable` interface.