Problem Statement

Suppose you are standing at the highest point, called M_0, of a mountainous landscape. As you look around, you wonder how many different points you could walk to starting from your lofty position without ever going uphill. The set of these points is called peak P_0, and the entire landscape can be divided into peaks according to a similar definition. For $i > 0$, let M_i be the highest point of the landscape not contained in peaks P_0 through P_{i-1}, and let peak P_i be the set of points to which there is a path from M_i that never goes uphill (but may remain level) and never touches points already contained in P_0 through P_{i-1}. The number of peaks in the landscape is the smallest value of n for which all points of the landscape are contained in P_0 through P_{n-1}.

You have a topographical map of a rectangular landscape, and you are interested in the area of its peaks. Write a class TopographicalImage with a method calcPeakAreas that takes a String[] **topoData** containing the height of the landscape at each x and y position and returns a int[] with the areas of each peak. The ASCII value of character x of element y of **topoData** is the height of the landscape at point (x,y). You can walk from a point to each of its vertical, horizontal, and diagonal neighbors. The return value should have a number of elements equal to the number of peaks in the landscape, and element i should be the number of points in P_i. If there is a tie between multiple points for maximum height when choosing M_i, choose the point with the smallest y-coordinate. If there is still a tie between points with the same y-coordinate, choose the point with the smallest x-coordinate.

Definition

Class:	TopographicalImage
Method:	calcPeakAreas
Parameters:	String[]
Returns:	int[]
Method signature:	int[] calcPeakAreas(String[] topoData)

(be sure your method is public)

Notes

- Point M_i is always contained in peak P_i, so the area of a peak is always at least 1.

Constraints

- **topoData** will contain between 1 and 50 elements, inclusive.
- Each element of **topoData** will contain between 1 and 50 characters, inclusive.
- Each element of **topoData** will contain the same number of characters.
- Each element of **topoData** will contain only characters with ASCII value between 33 and 126, inclusive.

Examples

0)

```
{
"...........",
"....i..i....",
"....i..i....",
".o..i..i..o.",
".o........o.",
"..oooooooo..",
"............"
}
```

```
Returns: { 78,  3,  3 }
```

1)

```
{
"............",
"....i..i....",
"....i..i....",
".S..i..i..Y.",
".M........E.",
"..ILEYSMIL..",
"............"
}
```

```
Returns: { 69,  3,  2,  5,  3,  1,  1 }
```

2)

```
{
"zzzzzzzzzzzzz",
"z...........z",
"z...c.b.c...z",
"z....bab.b..z",
"z...c.b.c...z",
"z...........z",
"zzzzzzzzzzzzz"
}
```

```
Returns: { 81,  6,  2,  1,  1 }
```

3)

```
{"!"}
```

```
Returns: { 1 }
```

4)

```
{
"AAAAAAABBBBCCCDEFGHHIIJIIHGFEDDCCCBBBBBBBBBBAAAAAA",
"AAAAABBBBBCCDDEEFGHIJJJJIIHGFEDDCCCCCCCCCBBBBBAAAA",
"AAAABBBBCCCDDEEFGHIIJJJJJIHGFEDDDDDDDDDCCCCBBBBAAA",
"AAABBBBCCDDEEFFGHHIJJJJJJIHGFEEDDDDDEEDDDDCCBBBBAA",
"AABBBCCDDEEFFGGHHIIJJJJJIHHGFEEEEEEEEFFFEEDDCCBBBAA",
"BBBBCCDDEFFGHHHIIIIJJJIIIHGFFEEEEFFGGGGGFEEDCCBBBA",
"BBBCCDEEFGHIIIJJJJIIIIIHHGGFFEEFFGGHHHHHGGFEDCCBBB",
"BBCCDEEGHIJJKKKKJJJIIHHGGFFEEEEFGGHIIJJIIHGFEDCCBB",
"CCCDEEFHIJKLMMMLKKJIHHGGFFEEEEFFGHIJJKKJJIHGFEDCBB",
"CDDEEFHIJLMNNNNMLKJIHGFFEEEDEEFFGIJKKLLLKJIHFEDCCB",
"DDEFFGIJLMNOPPONMLJIHGFEEDDDDEFGHIJKLMMMLKJIGFEDCB",
"EEFFGHIKMNOQQQPONLKIHFEEDDDDDEFGHIKLMMNMMLKIHGEDCC",
"FFGGHIJLMOPQRRQPNMKIGFEDDCCDDEFGHIKLMNNNNMLJIGFEDC",
"GHHHIJKLNOQRRRQPOMKIGFEDDCCDDEFGHIKLMNNNNMLKIHFEDC",
"HIIIJJKLNOPQRRQPNLKIGFEDDCCDDEFGHJKLMNOONNMKJHGFDC",
"IJJJJJKLMOPQQQPONLJHGFEDDDDDEEFGIJKLMNOONNMLJIGFED",
"JJJJJKKLMNOOPPONMKJHGFEDDDDEEFGHIJKLMNNONNMLJIGFED",
"JKKJJJKKLMMNNNNMLJIHFFEEEEEFGGHIJKLMMNNNNMMKJIGFED",
"KKKJJJJJKKLLMLLKJIHGFFEEEFFGHIJKKLMMNNNNNMLKJHGFED",
"JJJJIIIIIJJJKKJJIIHGFFFFFGHIJKLMMNNNNNNMMLKJIHGEDC",
"JJJIIHHHHHHIIIIIHHGGGGGGHIJKLMNOOOOONNMMLKJIHGFEDC",
"IIIHHGGGGGGGGHHHGGGGGGHIIJLMNOPQQQQPONMLKJIHGFEDDC",
"HHHGGFFFFFFFFFGGGGGGHHIJKMNOQRSSSSRQPNMLKIHGFFEDCC",
"GGGFFEEEEEEEEEFFFGGGHIJKLMOPRSTUUUTSRPNMKJHGFFEDCCB",
"FFFEEEEDDDDEEEEFGGHIJKLNOQRTUVWWWVTRPNLJIHFEEDCCBB",
"EEEEDDDDDDDDEEEFGHIJKLNOQRTVWXYYXWUSPNLJHGFEDCCBBB",
"DDDDDDDDDDDEEEFFGHIKLNOQRTVWXYZYYWURPMKIGFEDCCBBBB",
"CDDDDDDEEEEEEFFGHIJKMOPRSUWXYZZZXWTROMJHGEDCCBBBBA",
"CCDDDEEEFFFFFGGHHJKLNOQRTVWXYZZYXVTQNLIGFEDCBBBAAA",
"CCDDEFFGGGGHHHHIIJKMNPQSTVWXYYYXVURPMKIGEDCBBBAAAA",
"CDDEFGGHIIIIIIIJJKLMOPQSTUVWWXWVUSQNLJHFECCBBBAAAA",
"CDEFGHIJKKKKKKKKKLMNOPQRSTUVVVUTSQOMJHGEDCBBBAAAAA",
"CDEGHIKLMMMMMMLLLMMNOPQRSSTTTTSRQOMKIGFDCCBBAAAAAA",
"DEFGIKLMNOOOONNMMMNNOPQQRRRRRRQPNMKIHFEDCBBBAAAAAA",
"DEGHJLMOPQQQPPOONNNOOPPPQQQPPONMLKIHFEDCBBBAAAAAAA",
"DEGIKMNPQRRRRQPOOOOOOOPPPOOONMLKJIHFEDCCBBAAAAAAAA",
"DFGIKMOQRSSSRRQPOOOOOOOOONMMLKJIHGFEDCCBBBAAAAAAAA",
"DFGIKMOQRSSSRRQPOOOOONNNMMLKJIIHGFEDCCBBBAAAAAAAAA",
"DEGIJLNPQRRRRQPOONNNNNMMLLKJIHGFEEDCCBBBAAAAAAAAAA",
"DEFHJKMOPQQQQPOONNMMMMLLKJIHGGFEDDCCBBBAAAAAAAAAAA",
"CDFGIJLMNOOOONNMMLLLLKKJIHGFEEDCCCBBBAAAAAAAAAAAAA",
"CDEFGIJKLMMMMMLLKKKKKJJIIHGFEDDCCBBBBAAAAAAAAAAAAA",
"CCDEFGHIJKKKKKJJJIIIIIHHGGFEDDCCBBBBAAAAAAAAAAAAAA",
"BCCDEFGHHIIIIIHHHHHHGGGFFEDDCCBBBAAAAAAAAAAAAAAAAA",
"BBCCDEEFFGGGGGGFFFFFFFFEEDDCCCBBBAAAAAAAAAAAAAAAAA",
"BBBCCDDEEEEEEEEEEEEEEEDDDCCBBBBAAAAAAAAAAAAAAAAAAA",
"ABBBCCCCDDDDDDDDDDDDDDDCCCCBBBBAAAAAAAAAAAAAAAAAAA",
"AABBBBBCCCCCCCCCCCCCCCCCCBBBBBAAAAAAAAAAAAAAAAAAAA",
"AAABBBBBBBBBBBBBBBBBBBBBBBBAAAAAAAAAAAAAAAAAAAAAAA",
"AAAAAABBBBBBBBBBBBBBBBBBBBAAAAAAAAAAAAAAAAAAAAAAAA"
}

Returns: { 1918,  65,  483,  5,  5,  24 }
```