DKC^3 2019 – Computing Problems

Important Reminders:

- Time remaining will be announced at 30 min., 5 min. and 1 min. left in the session.
- Assume leading zeros are used in all decimal notations. (ex: 0.167)
- Example input and output files are available in the **C:\DKC3** folder for each problem. These may be used to test your programs.
- Each program must read from its respective input file in the **C:\DKC3** folder and generate an output file in that same folder. The names of these files must match what is specified for each problem in this document. These output files will be retrieved electronically and scored by the judges at the end of the session.
- Do **NOT** store any of your code in the **C:\DKC3** folder. Anything stored there will be overwritten.
- At the end of the session, judges will overwrite the example input files with the official input files for each problem. Each file will contain 10 test cases. One member of each team will be asked by a competition organizer to run all completed programs. Programs may be recompiled at this point prior to being run, but no changes can be made to source code.
- Each program must complete execution within **two** minutes.
- After running your programs, be sure to close out of all IDEs.

DKC^3 2019 – Computing Problems

1. BOGGLE (75 points)

Boggle is a word game played with 16 lettered dice organized into a 4x4 grid. The grid is shaken to randomize the letters and then players have three minutes to write down as many words as they can that are formed from a sequence of adjacent (horizontally, vertically or diagonally) letters in the grid. The words can start anywhere on the grid and no die may be used twice. Players are then scored by the number and length of the words they were able to find (for the purposes of this problem we will ignore the rule that states words found by multiple players are not counted). The scoring is as follows:

- Words shorter than 3 letters do not earn points.
- 3-letter and 4-letter words are worth 1 point.
- 5-letter words are worth 2 points.
- 6-letter words are worth 3 points.
- 7-letter words are worth 5 points.
- 8-letter words and longer are worth 11 points.

There exists a die face for every letter A-Z except for Q - Instead, there exists a Qu die face since Q rarely appears without U in English words. When 'Qu' appears in words it counts as 2 letters despite coming from only one die – for example, the word QUICK would score 2 points as a 5-letter word despite being formed from a sequence of only 4 dice. In this problem you will be given a boggle grid and a list of words, and you must determine the list's score. Not every word in the list will be valid (people make mistakes and use the same letter twice, connect letters that are not adjacent and so on). Invalid words will score zero points and valid words will be scored based on their length as shown above. In real boggle the words must also be legitimate words in the English dictionary, but you will be able to ignore this requirement.

Input/Output information on the next page.

Input
The input will consist of ten test cases. Each test case will start by specifying the boggle grid for that test case (four lines of four die face letters, each separated by a tab). The fifth line of the test case will contain a list of up to 100 words, each separated by a space. Input words will contain no spaces or hyphens, but capitalization may vary. There will be a blank line between each test case.

Output
For each test case you must output a single integer – the total score of the list of words provided.

Input File: C:\DKC3\BoggleIn.txt
Output File: C:\DKC3\BoggleOut.txt

Examples:

Input:

```
N	I	E	R
M	N	C	R
F	A	I	S
T	E	B	O
mace tea mint cribs faces famine

Qu	Y	S	M
O	I	Z	I
O	N	I	R
D	V	A	E
sin quiz rare are rain rainy
```

Output:

```
7
4
```

2. Digi-Bowl (75 points)

In honor of the 20th anniversary of DKC3, the DKC3 committee has come up with a new variation of bowling, called Digi-Bowl. Your task will be to write a program that reads in lines of alpha-numeric characters that represents scores for each roll in a game of Digi-Bowl.

A single game of Digi-Bowl consists of ten frames. The object in each frame is to roll a ball at 20 bowling pins arranged in a triangle and knock down as many pins as possible. A maximum of 3 rolls is allowed per frame.

-- On the **first** roll:

- If a digi-bowler knocks down all 20 pins on the **first** roll the frame is scored as a strike. A **second** or **third** roll for that frame is **NOT** given.
- If a digi-bowler doesn't knock down all the pins, the number of pins knocked down is scored for the **first** roll of the frame.

-- On the **second** roll:

- If a digi-bowler knocks down all the remaining pins, the **second** roll of the frame is scored as a spare. A **third** roll is **NOT** given.
- If a digi-bowler doesn't knock down all the remaining pins the number of pins knocked down is scored for the **second** roll of the frame. At this point, if the digi-bowler has knocked down at least 10 pins in this particular frame, a **third** roll is given.

-- On the **third** roll:

- If the digi-bowler knocks down all the remaining pins, the **third** roll is scored as a spare.
- If a digi-bowler doesn't knock down all the pins, the number of pins knocked down is scored for the **third** roll of the frame.

-- If a digi-bowler scores a strike in the 10th frame the digi-bowler is allowed **2** more rolls.
-- If a digi-bowler scores a spare in the 10th frame the digi-bowler is allowed **1** more roll.

The score for a Digi-Bowl game consists of the sum of the scores for each frame. The score for each frame is the total number of pins knocked down in the frame plus bonuses for strikes and spares.

-- If a digi-bowler scores a strike in a frame the score for that frame is 20 plus the sum of the next 2 rolls.
-- If a digi-bowler scores a spare in a frame the score for that frame is 20 plus the score of the next roll.

The maximum possible score in a game of Digi-Bowl (strikes in all 10 frames plus 2 extra strikes for the 10th frame strike) is 600.

Input/Output information on the next page.

Input
The input will consist of ten test cases, each on its own line. Each test case will contain the scores for a single game, with the scores for each roll of the ball separated by a single space. The score of a single roll will be represented by the following – either a number indicating the number of pins knocked down (0-19), a '/' for a spare, or a 'X' for a strike.

Output
The program should output the total game score for each game in the input file. The game scores should be printed on a separate line in the order corresponding to the order of the games in the input.

Input File: C:\DKC3\BowlingIn.txt
Output File: C:\DKC3\BowlingOut.txt

Examples:

Input:

```
8 5 3 14 / 10 5 / 6 2 18 / X 9 7 2 0 7 12 4 / 9 6 / 12
19 / X X X 15 2 2 16 / X 18 / 19 0 / X 15 2
```

Output:

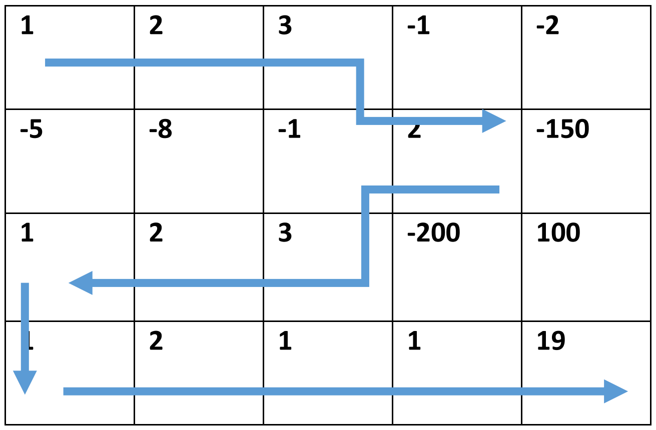
```
242
407
```

DKC^3 2019 – Computing Problems

3. High Score! (75 points)

The latest edition of the game Dungeon Diver just hit the local arcade, and you are after the high score. Dungeon Diver is a deceptively simple game. You begin on the top floor of a dungeon and can only go left, right, or down. Each square you move into increases or decreases your total score as you fight monsters and claim treasures. After you have visited a square, it no longer changes your score as either the monster is defeated, or the treasure is claimed.

A level of Dungeon Diver can be represented as a grid, with the player deciding which room on the first floor they want to start in. An example of a perfect game of Dungeon Diver is below



1	2	3	-1	-2
-5	-8	-1	2	-150
1	2	3	-200	100
1	2	1	1	19

Score: 37

Your task is to create a program that will find the highest possible score for a game of Dungeon Diver.

Input/Output information on the next page.

Input
There will be ten test cases, each consisting of multiple lines. The first line of each test case will be ***n*** and ***m*** where ***n*** is the number of lines, and ***m*** is the number of values in each of the following lines. Each value represents the point value of a grid you visit. The last line (***n***+1) will be a '*'. Each test case will be separated by a blank line.

Output
For each test case, print out the highest possible score one can achieve.

Input File:	C:\DKC3\HighScoreIn.txt
Output File:	C:\DKC3\HighScoreOut.txt

Examples:

Input:

```
4 5
1 2 3 -1 -2
-5 -8 -1 2 -150
1 2 3 -200 100
1 2 1 1 19
*


2 2
-87 4
4 7
*
```

Output:

```
37
15
```

4. Mars MORS (75 points)

In previous years your school's teams have worked to help NASA with problems regarding the rovers they have on Mars. This year your school will have another chance to help NASA. NASA has put a permanent satellite in orbit around Mars. It's called Mors (Mars ORbiting Satellite), and it has made an exciting discovery. With a vast array of recording equipment, sensors, sonars and radio equipment, it has discovered a series of what appear to be roads buried under several feet of Martian soil. Many of the roads connect to each other in "clumps". While there are many of these clumps scattered around the planet, most of them do not connect to each other. Could these clumps have been cities at one point in time?

The roads follow a grid design and have intersections every kilometer. Some intersections act only as a terminator of the road, some intersections only connect two roads and some connect three or four roads. NASA is looking for a correct count of the number of clumps, or cities as they are calling them now.

Your job, should you choose to accept it, is to read in a number of road segments and calculate how many separate clumps, or cities, there were on the surface at some point thousands of years ago.

Input/Output information on the next page.

Input
Each test case will be a list of the pairs of coordinates that represent road sections on a grid. The middle of the grid will be (0,0) and will run in all directions. The range for X and Y will be -10,671 through 10671 inclusive. There will be 10 test cases and each test case will be separated by a blank line. No test case will have more than 5,000 pairs of coordinates.

Output
The number of individual clumps/cities that are not connected to any other clump/city.

Input File: C:\DKC3\MarsIn.txt
Output File: C:\DKC3\MarsOut.txt

Examples:

Input:

```
(0,1) (1,1)
(1,0) (1,1)
(1,1) (1,2)
(1,1) (2,1)
(5,0) (5,1)
(5,1) (5,2)
(4,1) (5,1)
(5,1) (6,1)

(2,2) (3,2)          /* picture representation included for this test case, except for the 9000 segment */
(3,2) (3,3)
(3,3) (4,3)
(4,3) (4,4)
(4,4) (3,4)
(4,4) (5,4)
(5,4) (5,5)
(5,5) (6,5)
(6,5) (6,6)
(6,6) (6,7)
(6,7) (6,8)
(6,8) (6,9)
(6,9) (6,10)
(6,5) (6,4)
(6,4) (7,4)
(7,4) (8,4)
(8,4) (9,4)
(9,4) (9,3)
(9,3) (8,3)
(8,3) (8,4)
```

```
(6,5) (7,5)
(7,5) (7,4)
(7,5) (8,5)
(8,5) (9,5)
(8,5) (8,6)
(9,5) (9,6)
(9,6) (8,6)
(2,-2) (3,-2)
(3,-2) (3,-3)
(3,-3) (4,-3)
(4,-3) (4,-4)
(4,-4) (3,-4)
(4,-4) (5,-4)
(5,-4) (5,-5)
(5,-5) (6,-5)
(6,-5) (6,-6)
(6,-6) (6,-7)
(6,-7) (6,-8)
(6,-8) (6,-9)
(6,-9) (6,-10)
(6,-5) (6,-4)
(6,-4) (7,-4)
(7,-4) (8,-4)
(8,-4) (9,-4)
(9,-4) (9,-3)
(9,-3) (8,-3)
(8,-3) (8,-4)
(6,-5) (7,-5)
(7,-5) (7,-4)
(7,-5) (8,-5)
(8,-5) (9,-5)
(8,-5) (8,-6)
(9,-5) (9,-6)
(9,-6) (8,-6)
(9000,9000) (9001,9000)
```

Output:

```
2
3
```