DKC^3 2016 – Long Computing Problems

1. Heighway Dragon (75 points)

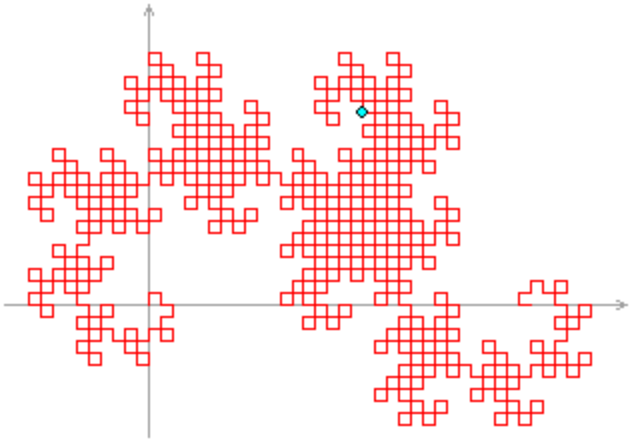
Let D_0 be the two-letter string "Fa". For n≥1, derive D_n from D_{n-1} by the string-rewriting rules:

"a" → "aRbFR"
"b" → "LFaLb"

Thus, D_0 = "Fa", D_1 = "FaRbFR", D_2 = "FaRbFRRLFaLbFR", and so on.

> These strings can be interpreted as instructions to a computer graphics program, with "F" meaning "draw forward one unit", "L" meaning "turn left 90 degrees", "R" meaning "turn right 90 degrees", and "a" and "b" **being** ignored. The initial position of the computer cursor is (0,0), pointing up towards (0,1).

Then D_n is an exotic drawing known as the Heighway Dragon of order n. For example, D_{10} is shown below; counting each "F" as one step, the highlighted spot at (18,16) is the position reached after 500 steps.



There will be 10 test cases, one line each. Each test case will consist of two integers separated by a space, the first being **n** and the second being a number of steps. Using this information you must find the coordinates after **n** steps as well as the final coordinates of $\mathbf{D_n}$ and how many steps to get there. The format should be as follows..

> D**{n}**: Position after **{step1}** steps is (**{x1}**,**{y1}**) and ends at (**{x2}**,**{y2}**) with **{step2}** steps.

1. **{n}** is number of iterations from the input
2. **{step1}** is the number of steps from the input
3. **{x1}** and **{y1}** are the coordinates after **{step1}** number of steps
4. **{x2}** and **{y2}** are the final coordinates after completing all steps in **{n}** iterations
5. **{step2}** is the final number of steps after completing all **{n}** iterations

DKC^3 2016 – Long Computing Problems

Input file: D:\DKC3\HeighwayDragonIn.txt
Output file: D:\DKC3\HeighwayDragonOut.txt

Examples:

Input:
0 1
10 892

Output:
D0: Position after 1 steps is (0,1) and ends at (0,1) with 1 steps.
D10: Position after 892 steps is (24,-10) and ends at (32,0) with 1024 steps.

DKC^3 2016 – Long Computing Problems

2. Can you crack the code? (75 points)

Write a program that reads in 2 integers (width and length of a codex) then reads in a list of characters, inserting them into the codex as they are read in. All characters are comma delimited. An example of this is 2,3,a,b,c,d,1,2 were read in, the codex would look like this…

a	b
c	d
1	2

But thanks to the paranoid fella who made this codex, a few more levels of complexity were added. Every letter should be replaced by its opposite letter (i.e. a is replaced with z, b is replaced with y, and so on) and then replaced again by its equivalent number (i.e. a = 0, b = 1, c = 2, and so on). The opposite process should happen for the numbers that were read in (0 = a, 1 = b, 2 = c, and so on) then they should be replaced by their opposite letter (as explained earlier).

Once you reach your first coordinate in the input line, which will be in the format where (0,0) = 'a' and (1,0) = 'b' in the graph above, all the proceeding inputs will be coordinates that need to be deciphered. Read in all of the coordinates and map them to the codex that you have created and the output should be a string of characters. There will be ten test cases, one test case per line.

Input file: D:\DKC3\CrackTheCodeIn.txt
Output file: D:\DKC3\CrackTheCodeOut.txt

Examples:

Input:
3,2,w,23,25,y,24,x,(2,0),(1,1),(1,0),(0,1),(2,1),(0,0)
3,4,6,8,25,18,v,21,x, ,17,12,7,3,(1,1),(0,2),(1,2),(2,2),(1,3),(1,2),(0,0),(0,1),(2,1),(1,2),(2,0),(0,3),(1,3),(2,3),(2,1),(1,0)

Output:
abc123
42 is the answer

DKC^3 2016 – Long Computing Problems

DKC^3 2016 – Long Computing Problems

3. Lack of Funding (75 points)

A number of cities across the U.S. are planning on building new roadway systems to replace their crumbling infrastructure, and make traffic more efficient within city limits. These cities originally thought they would be getting Federal Aid, however, Congress was unable to pass the bill that provides this funding. The city planners had originally designed the new roadway systems to have redundant routes and shortcuts. However, due to the lack of funds, they are being forced to reduce the cost of the road systems. Unfortunately, there isn't enough time to go and redesign the plans from scratch, so the only way to cut costs is to eliminate the redundant roads and be left with the bare minimum to get by. The only constraint they have is that it must be possible to get from any point in the city to any other point – it doesn't matter how long the travel time is, or how congested the roads are. You will be provided a "map" of the originally proposed roadways for each city and the cost of each individual road. It will be your job to provide the city planners with new maps that represent the roadway systems for each city that has the lowest possible cost and meets the constraint described above.

Input: Each test case will consist of a matrix that represents the roadways between each point in the city where there are between 2 and 10 points in each city. The values in the matrix represent the cost of the road that links the two points in millions of dollars. If there is no direct road between points or if it's the same point between row and column, the value will be zero. There will be ten test cases and each test case will be separated by an asterisk "*".

Output: The output should be a new matrix that represents the reduced roadway system, with the lowest possible cost. Each output matrix should be separated by an asterisk "*".

Input file: D:\DKC3\FundingIn.txt
Output file: D:\DKC3\FundingOut.txt

DKC^3 2016 – Long Computing Problems

Examples:

Input:

0 16 12 21 0 0 0

16 0 0 17 20 0 0

12 0 0 28 0 31 0

21 17 28 0 18 19 23

0 20 0 18 0 0 11

0 0 31 19 0 0 27

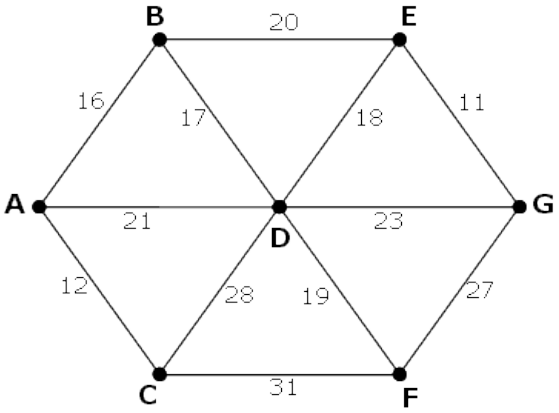
0 0 0 23 11 27 0

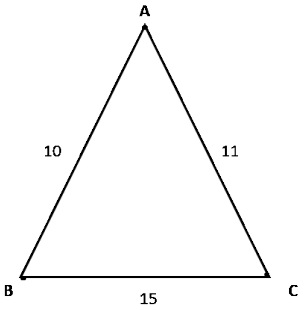
*

0 10 11

10 0 15

11 15 0





Output:

0 16 12 0 0 0 0

16 0 0 17 0 0 0

12 0 0 0 0 0 0

0 17 0 0 18 19 0

0 0 0 18 0 0 11

0 0 0 19 0 0 0

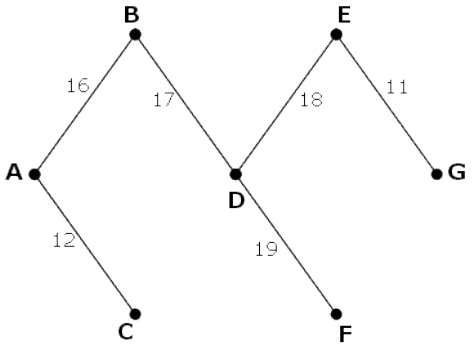
0 0 0 0 11 0 0

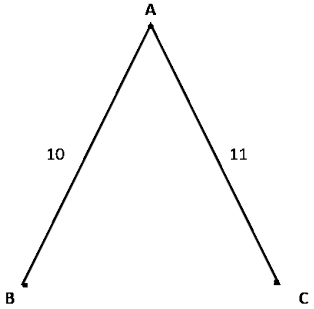
*

0 10 11

10 0 0

11 0 0



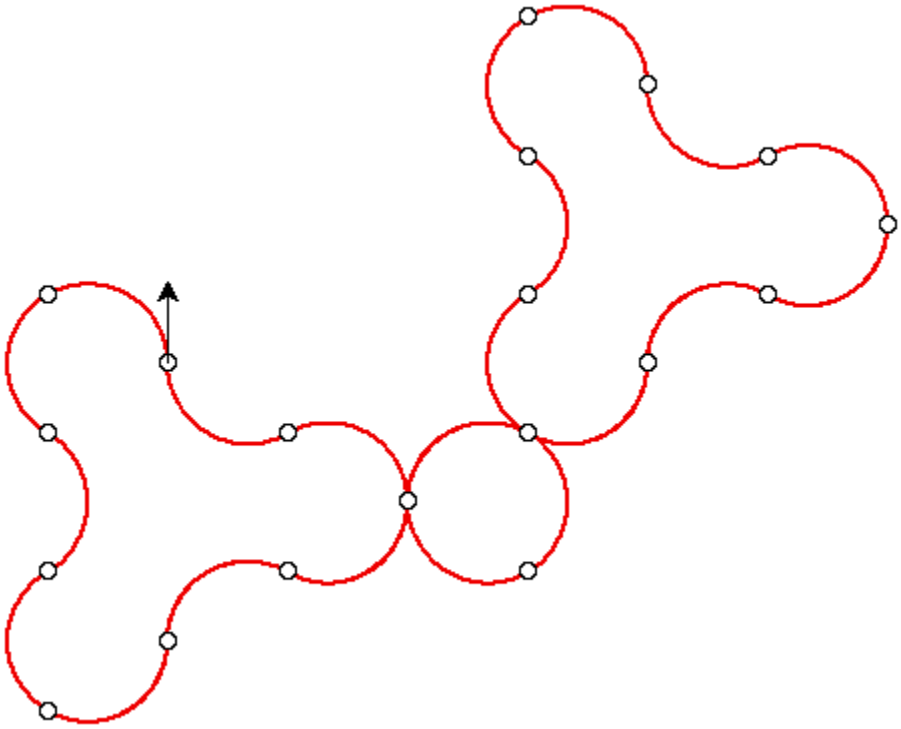


DKC^3 2016 – Long Computing Problems

4. Robot Walks (75 points)

A robot moves in a series of one-third circular arcs (120°), with a free choice of a clockwise or an anticlockwise arc for each step, but no turning on the spot.

One of 80096 possible closed paths of 21 arcs is



Given that the robot starts facing North, calculate the number of journeys the robot can take for a given number of arcs that return it, after the final arc, to its starting position. Any arc may be traversed multiple times during this journey.

Input: Each test case will be the number of arcs in the robot's journey. There will be ten test cases, one test case per line.

Output: Each test case's output will be a number between 1 and 1,000,000,000 on its own line, and the number will be the number of journeys the robot can with the given number of arcs.

Input file: D:\DKC3\RobotIn.txt
Output file: D:\DKC3\RobotOut.txt

DKC^3 2016 – Long Computing Problems

Examples:

Input:
21
9

Output:
80096
44