A typical strategy in the game Starcraft is to mass up large amounts of low-tier units such as Zerglings, then throw them at your opponent and laugh maniacally as they overwhelm any opposition. However, when both players opt for the same strategy, the result can often become...quick, brutal and messy. Sadly, the game only allows for up to 400 Zerglings per player. In this problem, however, you will simulate games that could have more than 400 Zerglings.

The map on which Zerg rushes occur is represented by a $N \times N$ grid. Each Zergling occupies a grid square and no two Zerglings ever occupy the same grid square. Each Zergling has a certain number of hit points which starts off as 35. Its attack value is 5 plus the attack upgrade of the player that controls it. When one Zergling attacks another, the damage incurred equals to the attack value of the attacking Zergling minus the armour upgrade of the player that owns the Zergling being attacked. The number of hit points of the Zergling that is attacked is reduced by the amount of the damage.

Due to the inability of both players to manage their huge horde of Zerglings, the Zerglings make their decisions each turn using the following algorithm (Zerglings are not the brightest creatures, as you will see):

- If there is an opponent Zergling in one of the 8 horizontally, vertically, or diagonally adjacent grid squares, the Zergling will attack it. A Zergling attacks at most one opponent each turn; see below for the tie-breaking rules.
- Otherwise, if the other player has at least one Zergling remaining on the map, the Zergling will move to the horizontally, vertically, or diagonally adjacent square that is closest to the opponent's closest Zergling in terms of Manhattan distance. When more than one adjacent square is closest, the tie-breaking rules below are used. The Manhattan distance between two points is the sum of the differences in the x and y coordinates of the points.

When the above rules could cause the Zergling to attack in more than one direction, or to move in more than one direction, the following tie-breaking rule is used. The Zergling will prefer the first direction starting with north going clockwise. That is, the directions in order of preference are north, northeast, east, southeast, etc.

Once all Zerglings have made their decisions, all the attacks are conducted simultaneously and all the Zerglings with 0 or fewer hit points are marked as dead and removed from the map. Then all the movements of the Zerglings that didn't attack are conducted simultaneously. If the square to which a Zergling is moving is occupied by another Zergling that is not moving away in this turn, then the Zergling does not move. If two or more Zerglings try to move to the same grid square, then the Zergling in the northernmost row has the right of way and the other Zergling remains stationary. If there are multiple Zerglings in the northernmost row trying to move to the same grid square, then of these, the westernmost Zergling moves and the others remain stationary. Zerglings also have a remarkable regeneration rate. After each turn, all the Zerglings that are still alive and have less than 35 hitpoints will regain one hit point.

Input

The input file contains a number of test cases, ended by a case with $N = 0$. Each case begins with N between 2 and 150, followed by 2 pairs of 2 integers between 0 and 3, the attack and armour upgrades of the first player, followed by the attack and armour upgrades of the second player. This is followed by the initial game map, where '`.`' denotes an empty square, '`1`' a Zergling belonging to the first player and '`2`' a Zergling belonging to the second player. On the map, north is up (i.e. towards the first row) and west is left (i.e. towards the first column). Finally, the test case provides the number t of turns for which the Zerg rush is to be simulated, which is an integer between 0 and 400, inclusive.

Output

For each input case, output the map after t turns in the same format as above. Print one empty line between the output for consecutive test cases.

Sample Input

```
2
0 0
0 0
1.
..
0
2
0 0
0 0
1.
.2
100
0
```

Sample Output

```
1.
..

..
..
```