advent_of_code_day 17

January 14, 2022

1 Advent of Code

```
[ ]: # set up the environment
import numpy as np
```

1.1 Day 17: Trick Shot

You finally decode the Elves' message. HI, the message says. You continue searching for the sleigh keys.

Ahead of you is what appears to be a large ocean trench. Could the keys have fallen into it? You'd better send a probe to investigate.

The probe launcher on your submarine can fire the probe with any integer velocity in the x (forward) and y (upward, or downward if negative) directions. For example, an initial x,y velocity like 0,10 would fire the probe straight up, while an initial velocity like 10,-1 would fire the probe forward at a slight downward angle.

The probe's x,y position starts at 0,0. Then, it will follow some trajectory by moving in **steps**. On each step, these changes occur in the following order:

- The probe's x position increases by its x velocity.
- The probe's y position increases by its y velocity.
- Due to drag, the probe's x velocity changes by 1 toward the value 0; that is, it decreases by 1 if it is greater than 0, increases by 1 if it is less than 0, or does not change if it is already 0.
- Due to gravity, the probe's y velocity decreases by 1.

For the probe to successfully make it into the trench, the probe must be on some trajectory that causes it to be within a **target area** after any step. The submarine computer has already calculated this target area (your puzzle input). For example:

```
target area: x=20..30, y=-10..-5
```

This target area means that you need to find initial x,y velocity values such that after any step, the probe's x position is at least 20 and at most 30, and the probe's y position is at least -10 and at most -5.

Given this target area, one initial velocity that causes the probe to be within the target area after any step is 7,2:

```
.............#....#............
.......#..............#........
```

```
...............................
S........................#.....
...............................
...............................
...........................#...
...............................
....................TTTTTTTTTTT
....................TTTTTTTTTTT
....................TTTTTTTT#TT
....................TTTTTTTTTTT
....................TTTTTTTTTTT
....................TTTTTTTTTTT
```

In this diagram, S is the probe's initial position, 0,0. The x coordinate increases to the right, and the y coordinate increases upward. In the bottom right, positions that are within the target area are shown as T. After each step (until the target area is reached), the position of the probe is marked with #. (The bottom-right # is both a position the probe reaches and a position in the target area.)

Another initial velocity that causes the probe to be within the target area after any step is 6,3:

```
...............#..#............
...........#........#..........
...............................
......#..............#.........
...............................
...............................
S....................#.........
...............................
...............................
...............................
.....................#.........
....................TTTTTTTTTTT
....................TTTTTTTTTTT
....................TTTTTTTTTTT
....................TTTTTTTTTTT
....................T#TTTTTTTTT
....................TTTTTTTTTTT
```

Another one is 9,0:

```
S........#.....................
.................#.............
...............................
........................#......
...............................
....................TTTTTTTTTTT
....................TTTTTTTTTT#
....................TTTTTTTTTTT
....................TTTTTTTTTTT
```

```
....................TTTTTTTTTTT
....................TTTTTTTTTTT
```

One initial velocity that **doesn't** cause the probe to be within the target area after any step is 17,-4:

```
S..............................................................
...............................................................
...............................................................
...............................................................
.................#.............................................
....................TTTTTTTTTTT................................
....................TTTTTTTTTTT................................
....................TTTTTTTTTTT................................
....................TTTTTTTTTTT................................
....................TTTTTTTTTTT..#.............................
....................TTTTTTTTTTT................................
...............................................................
...............................................................
...............................................................
...............................................................
................................................#..............
...............................................................
...............................................................
...............................................................
...............................................................
...............................................................
...............................................................
..............................................................#
```

The probe appears to pass through the target area, but is never within it after any step. Instead, it continues down and to the right - only the first few steps are shown.

If you're going to fire a highly scientific probe out of a super cool probe launcher, you might as well do it with **style**. How high can you make the probe go while still reaching the target area?

In the above example, using an initial velocity of 6,9 is the best you can do, causing the probe to reach a maximum y position of 45. (Any higher initial y velocity causes the probe to overshoot the target area entirely.)

Find the initial velocity that causes the probe to reach the highest y position and still eventually be within the target area after any step. **What is the highest y position it reaches on this trajectory?**

[ ]:
```python
import re

with open('data/trick_shot.dat') as file:
    problem_target = file.readline().strip()

test_target = "target area: x=20..30, y=-10..-5"
```

```
def get_range_limits(range_str: str) -> tuple:
    x_range = re.search(r"x=(-?\d+)..(-?\d+)", range_str)
    x_min = int(x_range.group(1))
    x_max = int(x_range.group(2))
    y_range = re.search(r"y=(-?\d+)..(-?\d+)", range_str)
    y_min = int(y_range.group(1))
    y_max = int(y_range.group(2))

    return x_min, x_max, y_min, y_max
```

1.1.1 Answer part 1

As the projectile's velocity loss ascending is the same as it's gain descending it must return to the zero point with the same velocity it left with. The next increase should then take it to the max extent of the target. Given this the initial velocity must be: $v_{initial} = target_{min} - 1$

[ ]:
```
def step(x_pos, y_pos, x_vel, y_vel) -> tuple:
    x_pos += x_vel
    y_pos += y_vel
    if x_vel != 0:
        x_vel += -1 if x_vel > 0 else 1
    y_vel -= 1

    return x_pos, y_pos, x_vel, y_vel


def min_x_vel(x_min, x_max) -> tuple:
    final_vel = 0
    distance = 0
    steps = 0
    while not x_min <= distance <= x_max:
        final_vel += 1
        distance += final_vel
        steps += 1

    return final_vel, distance, steps

target = problem_target

x_min, x_max, y_min, y_max = get_range_limits(target)

x_initial_vel, x_distance, x_steps = min_x_vel(x_min, x_max)
print(f'{target}')
print(f'minimum x initial velocity: {x_initial_vel}, x distance: {x_distance},␣
↪steps: {x_steps}')
```

```
max_y_vel = (
    abs(y_min) - 1
)  # the max velocity is attained after completing the last increase.
height = 0
while max_y_vel != 0:
    height += max_y_vel
    max_y_vel -= 1

print(f"max height attained is {height}, with an initial velocity of␣
 ↪{abs(y_min) -1}")
```

```
target area: x=150..171, y=-129..-70
minimum x initial velocity: 17, x distance: 153, steps: 17
max height attained is 8256, with an initial velocity of 128
```

1.2 Part Two

Maybe a fancy trick shot isn't the best idea; after all, you only have one probe, so you had better not miss.

To get the best idea of what your options are for launching the probe, you need to find **every initial velocity** that causes the probe to eventually be within the target area after any step.

In the above example, there are **112** different initial velocity values that meet these criteria:

```
23,-10  25,-9   27,-5   29,-6   22,-6   21,-7   9,0     27,-7   24,-5
25,-7   26,-6   25,-5   6,8     11,-2   20,-5   29,-10  6,3     28,-7
8,0     30,-6   29,-8   20,-10  6,7     6,4     6,1     14,-4   21,-6
26,-10  7,-1    7,7     8,-1    21,-9   6,2     20,-7   30,-10  14,-3
20,-8   13,-2   7,3     28,-8   29,-9   15,-3   22,-5   26,-8   25,-8
25,-6   15,-4   9,-2    15,-2   12,-2   28,-9   12,-3   24,-6   23,-7
25,-10  7,8     11,-3   26,-7   7,1     23,-9   6,0     22,-10  27,-6
8,1     22,-8   13,-4   7,6     28,-6   11,-4   12,-4   26,-9   7,4
24,-10  23,-8   30,-8   7,0     9,-1    10,-1   26,-5   22,-9   6,5
7,5     23,-6   28,-10  10,-2   11,-1   20,-9   14,-2   29,-7   13,-3
23,-5   24,-8   27,-9   30,-7   28,-5   21,-10  7,9     6,6     21,-5
27,-10  7,2     30,-9   21,-8   22,-7   24,-9   20,-6   6,9     29,-5
8,-2    27,-8   30,-5   24,-7
```

How many distinct initial velocity values cause the probe to be within the target area after any step?

1.2.1 Answer Part 2

We are now faced with a combinatorial problem. We can break it down into 2 parts: 1. All initial velocities in the y plane will eventually go past the target area if not stopped. Starting with the maximum positive velocity that will not overshoot the target area (1 less than the distance to the lower y bound) fire a test shot and see how man steps are within the target area then store the steps and the velocity. 2. Repeat the above decreasing the velocity by one each time until the velocity is equal to the lower bound of the target area. We should now have a list of all possible steps and the relevant velocity required to be in the target area on that step. 3. Get the maximum number of steps recorded above; any velocity in the x plane that last longer than this is not a solution so it sets an upper bound on the x steps. 4. Repeat the exercise in 1. & 2. for firings in the x plane. In this case the we know the minimum x velocity and the maximum x velocity is equal to the max x bound. - In this case determine if the x firing overshoots in the x plane. If not then it is stationary in the target area and all firings in the y plane that have at least this number of steps are valid.

We now have 2 lists for firings in the x and y direction. Combine the velocity elements from both lists where the step element matches. This will now, in all likelihood have duplicate elements so remove these.

This final list should now have all the possible distinct firing velocities.

[ ]:
```
target = problem_target

x_min, x_max, y_min, y_max = get_range_limits(target)

x_velocity = x_max  # hits far edge in 1!
y_velocity = abs(y_min) - 1  # hits lower edge in 1 after returning to zero␣
 ↪(see part 1)
x_min_velocity = min_x_vel(x_min, x_max)[0]
y_min_velocity = y_min
x_steps = []
y_steps = []


def test_x_velocity(v: int, min_x: int, max_x: int) -> tuple:
    """
    Determines the number of x steps that are in the target and if it␣
 ↪overshoots.
    """
    distance = 0
    step = 0
    steps = []
    overshot = True
    while distance <= max_x:
        if min_x <= distance <= max_x:
            steps.append(step)
        distance += v
        v -= 1
        step += 1
        if v == 0:
            overshot = False
            break

    return steps, overshot
```

```python
def test_y_velocity(v: int, min_y: int, max_y: int) -> list:
    """
    Determines the number of y steps that are in the target.
    """
    distance = 0
    step = 0
    steps = []
    while distance >= min_y:
        if min_y <= distance <= max_y:
            steps.append(step)
        distance += v
        v -= 1
        step += 1

    return steps

while y_velocity >= y_min_velocity:
    steps = test_y_velocity(y_velocity, y_min, y_max)
    for step in steps:
        y_steps.append([step, y_velocity])

    y_velocity -= 1

max_y_steps = max(y_steps)[0]

while x_velocity >= x_min_velocity:
    steps, overshot = test_x_velocity(x_velocity, x_min, x_max)
    if steps and not overshot:
        for x in range(max(steps), max_y_steps):
            steps.append(x+1)
    for step in steps:
        x_steps.append([step, x_velocity])
    x_velocity -= 1

possible_launches = []
for y in y_steps:
    for x in x_steps:
        if x[0] == y[0]:
            possible_launches.append([x[1], y[1]])

actual_launches = [list(t) for t in set(tuple(element) for element in␣
 ↪possible_launches)]

print(f'Number of unique possible launches is {len(actual_launches)}')
```

```
Number of unique possible launches is 2326
```