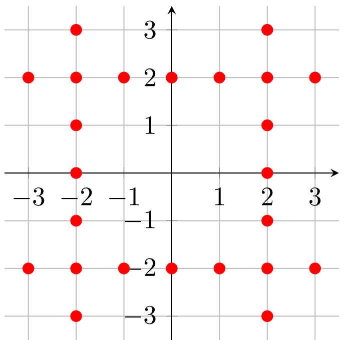
Part 1.

Final Answer: 4.5 seconds

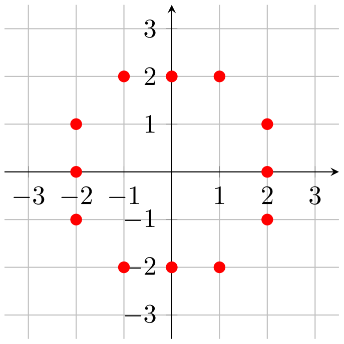
The solutions to parts 1 and 3 are based on the techniques found in [1].

For simplicity, let us work on the integer grid representing decimters in real life (so that the ant moves one space on the integer grid every second).

Let the red dots represent food:



Note that there are only 12 points which could be the first piece of food that the ant reaches; for example, the ant could not reach the point $(2, 2)$ without first passing through another point containing food. Since we only care about the first piece of food that the ant reaches, the problem remains identical if we remove all other pieces of food:



Now, let $t_{x,y}$ denote the expected time for the ant to reach a piece of food starting from the point (x, y) (where (x, y) is inside or on the boundary of food).

Note that we have 12 'trivial' boundary conditions – if we start on a piece of food, then the expected time to reach a piece of food is 0. Hence

$$t_{-1,-2} = t_{0,-2} = t_{1,-2} = t_{-2,-1} = t_{2,-1} = t_{-2,0} = t_{2,0} = t_{-2,1} = t_{2,1} = t_{-1,2} = t_{0,2} = t_{1,2} = 0.$$

Now let us consider the 9 non-trivial starting positions, namely where $-1 \le x, y \le 1$. Since the ant has not yet reached food, it must take one step in any direction, then 'start again' from the position it lands on. This gives us the following recurrence relation:

$$t_{x,y} = 1 + \frac{1}{4}\left(t_{x-1,y} + t_{x+1,y} + t_{x,y-1} + t_{x,y+1}\right)$$

which we can rewrite as

$$\frac{1}{4}t_{x,y-1} + \frac{1}{4}t_{x-1,y} - t_{x,y} + \frac{1}{4}t_{x+1,y} + \frac{1}{4}t_{x,y+1} = -1.$$

For example, considering the starting position $(1, 1)$ gives us the relation

$$\frac{1}{4}t_{1,0} + \frac{1}{4}t_{0,1} - t_{1,1} + \frac{1}{4}t_{2,1} + \frac{1}{4}t_{1,2} = -1.$$

We can write these 9 relations together with the 12 aforementioned boundary conditions as a system of linear equations $At = b$ with 21 variables, which fully encapsulates the problem:

$$\begin{bmatrix}
1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
p&0&0&p&-1&p&0&0&0&p&0&0&0&0&0&0&0&0&0&0&0\\
0&p&0&0&p&-1&p&0&0&0&p&0&0&0&0&0&0&0&0&0&0\\
0&0&p&0&0&p&-1&p&0&0&0&p&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&p&0&0&0&p&-1&p&0&0&0&p&0&0&0&0&0&0\\
0&0&0&0&0&p&0&0&0&p&-1&p&0&0&0&p&0&0&0&0&0\\
0&0&0&0&0&0&p&0&0&0&p&-1&p&0&0&0&p&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&p&0&0&0&p&-1&p&0&0&p&0&0\\
0&0&0&0&0&0&0&0&0&0&p&0&0&0&p&-1&p&0&0&p&0\\
0&0&0&0&0&0&0&0&0&0&0&p&0&0&0&p&-1&p&0&0&p\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}
\begin{bmatrix}
t_{-1,-2}\\ t_{0,-2}\\ t_{1,-2}\\ t_{-2,-1}\\ t_{-1,-1}\\ t_{0,-1}\\ t_{1,-1}\\ t_{2,-1}\\ t_{-2,0}\\ t_{-1,0}\\ t_{0,0}\\ t_{1,0}\\ t_{2,0}\\ t_{-2,1}\\ t_{-1,1}\\ t_{0,1}\\ t_{1,1}\\ t_{2,1}\\ t_{-1,2}\\ t_{0,2}\\ t_{1,2}
\end{bmatrix}
=
\begin{bmatrix}
0\\0\\0\\0\\-1\\-1\\-1\\0\\0\\-1\\-1\\-1\\0\\0\\-1\\-1\\-1\\0\\0\\0\\0
\end{bmatrix}$$

where $p = \frac{1}{4}$.

All that remains is to compute $t = A^{-1}b$. Performing this computation yields

$$t = \begin{bmatrix}
t_{-1,-2}\\ t_{0,-2}\\ t_{1,-2}\\ t_{-2,-1}\\ t_{-1,-1}\\ t_{0,-1}\\ t_{1,-1}\\ t_{2,-1}\\ t_{-2,0}\\ t_{-1,0}\\ t_{0,0}\\ t_{1,0}\\ t_{2,0}\\ t_{-2,1}\\ t_{-1,1}\\ t_{0,1}\\ t_{1,1}\\ t_{2,1}\\ t_{-1,2}\\ t_{0,2}\\ t_{1,2}
\end{bmatrix}
=
\begin{bmatrix}
0\\0\\0\\0\\2.75\\3.5\\2.75\\0\\0\\3.5\\4.5\\3.5\\0\\0\\2.75\\3.5\\2.75\\0\\0\\0\\0
\end{bmatrix}.$$

In particular, the value we are interested in is $t_{0,0} = 4.5$.

Part 2.

Final Answer: ∞

Let us use the movement of the ant to move a marker on the integer number line. For each position of the ant (x, y), define the position of the marker to be $x + y$. Then the ant reaching the line $x + y = 1$ on the plane (where the food is located) corresponds to the marker reaching the point $+1$ on the integer line.

Now, note that whenever the ant moves up or to the right, the marker moves one place to the right; and whenever the ant moves down or to the left, the marker moves one place to the left. Hence, at each step, the marker moves one step either left or right with equal probability. Thus we can rephrase the question as follows:

Consider a marker on the line of integers starting at 0. *Each second, the marker moves either one step to the left or one step to the right with equal probability. What is the average amount of time needed for the marker to reach the point* $+1$*?*

We conjecture that the average number of steps is infinite; to make the solution rigorous, let us instead assume that the marker moves right with probability $p \in (\frac{1}{2}, 1)$ and left with probability $1 - p$. Let t_x denote the expected number of steps required to reach the point $+1$ starting at $x \leq 1$.

By a similar argument to Part 1, t_x satisfies the recurrence relation

$$t_x = 1 + pt_{x+1} + (p-1)t_{x-1}$$

for all $x \leq 1$. In particular,

$$\begin{aligned} t_0 &= 1 + pt_1 + (p-1)t_{-1} \\ &= 1 + (p-1)t_{-1}. \end{aligned}$$

Now, if the marker starts at -1 instead of 0, it has to cover twice the distance to reach 1, and thus (by the Markov property of random walks) $t_{-1} = 2t_0$. Hence

$$t_0 = 1 + 2(p-1)t_0$$

which rearranges to

$$t_0 = \frac{1}{2p-1}.$$

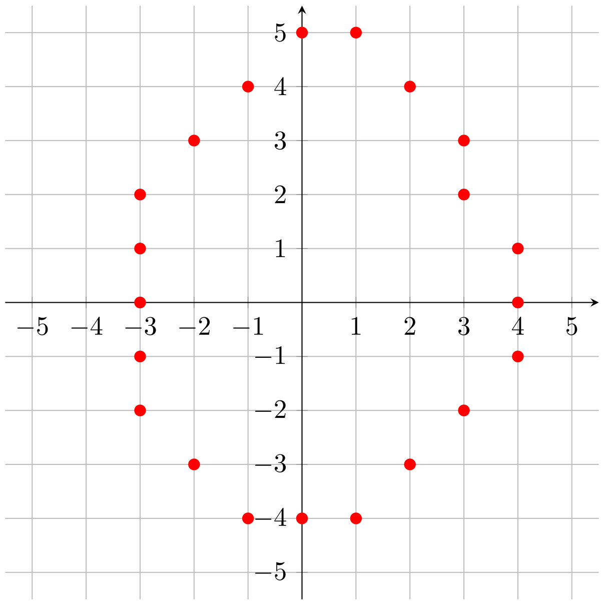
Finally,

$$\lim_{p \to \frac{1}{2}^+} \frac{1}{2p-1} = \infty.$$

Part 3.

Final Answer: 14 seconds

The program essentially automates the same process used to solve Part 1. We just need all the points containing food which could viably be the first piece of food that the ant reaches. For the scenario where food is contained outside of the region $\left(\frac{x-2.5\text{cm}}{30\text{cm}}\right)^2 + \left(\frac{y-2.5\text{cm}}{40\text{cm}}\right)^2 < 1$, the relevant points are as follows:



The program then proceeds analogously to Part 1, assigning a variable to each of the boundary and interior points then encoding the problem into a system of linear equations.

```
import numpy as np
from shapely.geometry import Point, Polygon


# returns the extremal points of the boundary.
def get_boundary_properties(boundary_input):
    boundary_size = len(boundary_input)
    temp_vector = [boundary_input[i][0] for i in range(boundary_size)]
    x_min_return, x_max_return = min(temp_vector), max(temp_vector)
    temp_vector = [boundary_input[i][1] for i in range(boundary_size)]
    y_min_return, y_max_return = min(temp_vector), max(temp_vector)
    return x_min_return, x_max_return, y_min_return, y_max_return


boundary = [[0, -4], [1, -4], [2, -3], [3, -2], [4, -1], [4, 0], [4, 1], [3, 2], [3, 3], [2, 4
                                                ], [1, 5], [0, 5], [-1, 4], [-2, 3], [-3, 2], [
                                                -3, 1], [-3, 0], [-3, -1], [-3, -2], [-2, -3],
                                                [-1, -4]]

x_min, x_max, y_min, y_max = get_boundary_properties(boundary)

# finds the interior points.
interior = []
boundary_polygon = Polygon(boundary)
for y in range(y_min, y_max + 1):
    for x in range(x_min, x_max + 1):
        position = [x, y]
        point = Point(position)
        if boundary_polygon.contains(point):
            interior.append(position)

# creates a bijection between all the boundary/interior points and the integers 0, 1, ..., [
                                                total no. points] - 1.
# we need this to index the matrix elements later.
position_index = np.zeros((x_max - x_min + 1, y_max - y_min + 1)).astype(int)
index = 0
for y in range(y_min, y_max + 1):
    for x in range(x_min, x_max + 1):
        position = [x, y]
        if position in boundary or position in interior:
            position_index[position[0] - x_min][position[1] - y_min] = index
            index = index + 1
        else:
            position_index[position[0] - x_min][position[1] - y_min] = -1

# generates the system of linear equations which encodes the problem.
dimension = len(boundary) + len(interior)
A = np.zeros((dimension, dimension))
b = np.zeros(dimension)
for y in range(y_min, y_max + 1):
    for x in range(x_min, x_max + 1):
        position = [x, y]
        if position in boundary:
            A[position_index[x - x_min][y - y_min]][position_index[x - x_min][y - y_min]] = 1
        elif position in interior:
            A[position_index[x - x_min][y - y_min]][position_index[x - x_min][y - y_min]] = -1
            A[position_index[x - x_min][y - y_min]][position_index[x - x_min][y - 1 - y_min]]
                                                            = 0.25
            A[position_index[x - x_min][y - y_min]][position_index[x - 1 - x_min][y - y_min]]
                                                            = 0.25
            A[position_index[x - x_min][y - y_min]][position_index[x + 1 - x_min][y - y_min]]
                                                            = 0.25
            A[position_index[x - x_min][y - y_min]][position_index[x - x_min][y + 1 - y_min]]
                                                            = 0.25
            b[position_index[x - x_min][y - y_min]] = -1

# solves the system of linear equations.
t = np.linalg.solve(A, b)

print("Average time to reach boundary: {} seconds".format(t[position_index[-x_min][-y_min]]))
```

The above program returns an average time of 13.992053058411821 seconds.

References

[1] Miky Wright (2015), *Boundary Problems for One and Two Dimensional Random Walks*.