DSC 40B - Homework 05

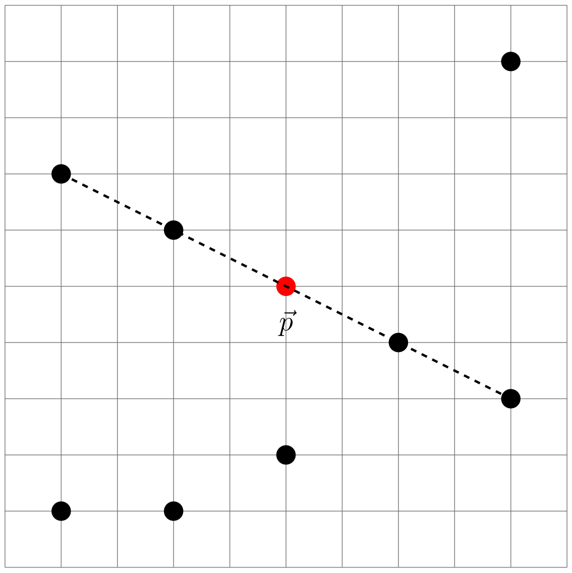
Due: Wednesday, May 8

Write your solutions to the following problems by either typing them up or handwriting them on another piece of paper. Unless otherwise noted by the problem's instructions, show your work or provide some justification for your answer. Homeworks are due via Gradescope at 11:59 p.m.

Programming Problem 1.

Consider the problem of finding collinear points. Suppose you are given a set X of n points in $\mathbb{R}^2$, along with a query point $\vec{p} \in \mathbb{R}^2$. Your goal is to calculate the maximum number of points in X that fall on the same straight line containing $\vec{p}$. That is, considering all of the (infinitely) many lines containing $\vec{p}$, find the one that contains the most points from X.

For example, in the picture below, the black points are those in X and the red point represents $\vec{p}$. The maximum number of points in X that falls on a line containing $\vec{p}$ is four (those on the dashed line).



In a file named `max_points_on_line.py`, write a function `max_points_on_line(X, p)` which accepts the following arguments:

- `X`: A list of points, each point being a 2-tuple of integers.
- `p`: A query point represented as a 2-tuple of integers.

Your function should return the maximum number of points in `X` that fall on the same straight line containing $\vec{p}$. Your answer should be an integer. If `X` is empty, return `0`. If a point $\vec{x}$ in `X` is such that $\vec{x} = \vec{p}$, you should consider to be on the same line as $\vec{p}$ (even if there are no other points).

Your algorithm should take expected time that is linear in the number of points in `X`.

Example (using the points in the picture above):

```
>>> X = [
... (1, 1),
... (3, 1),
... (5, 2),
... (3, 6),
... (1, 7),
... (7, 4),
... (9, 3),
... (9, 9)
... ]
>>> p = (5, 5)
>>> max_points_on_line(X, p)
4
```

Solution: This is an optimization problem that at first might look undoable in a finite amount of time: there are infinitely many lines that contain the point $\vec{p}$, and we don't have time to try all of them.

The key insight is that two points $\vec{x}_1$ and $\vec{x}_2$ will fall on the same line with $\vec{p}$ if and only if the slope between $\vec{x}_1$ and $\vec{p}$ is the same as the slope between $\vec{x}_2$ and $\vec{p}$. This is a consequence of the fact that any line containing $\vec{p}$ is uniquely determined by its slope.

Therefore, the solution proceeds by making a dictionary (hash map) whose keys are slopes and whose values are integers counting the number of points we've seen whose slope with $\vec{x}$ is equal to the key. We loop through all points in `X` and compute the slope between each point and $\vec{p}$. We index into the dictionary with this slope and increment the count stored. If this new value is the largest so far, we update our idea of the maximum.

Working Python code is below:

```python
def max_points_on_line(X, p):
    """Compute the maximum number of points in X that fall on any line that includes p.

    Parameters
    ----------
    X
        A list of 2-tuples containing integers.
    p
        A point as a 2-tuple of integers.

    Returns
    -------
    int

    Example
    -------

    >>> X = [
    ... (1, 1),
    ... (3, 1),
    ... (5, 2),
    ... (3, 6),
    ... (1, 7),
    ... (7, 4),
    ... (9, 3),
    ... (9, 9)
    ... ]
    >>> p = (5, 5)
    >>> max_points_on_line(X, p)
    4
    """
    # counts_by_slope is a dictionary whose keys are slopes and whose values are counts
    # of the times we see a point in X on the line with p with the given slope. By
    # convention, two distinct points on the same line will have a slope of infinity,
    # and two identical points will have a slope of None
    #
    # we'll initialize the dictionary with a key of None because we will be counting the
    # number of points in X that are identical to the query point p in order to handle
    # that corner case; see the comment above the return statement below for more info.
    counts_by_slope = {None: 0}

    maximum = 0

    for x in X:
        m = slope_between(x, p)

        # Count of points equivalent to p
        if m is None:
            counts_by_slope[None] += 1
            continue

        # Initialize the count for this slope if we haven't seen it before
        if m not in counts_by_slope:
            counts_by_slope[m] = 0

        # Update the count for this slope
        count = counts_by_slope[m] = counts_by_slope[m] + 1

        if count > maximum:
            maximum = count

    # if a point in X equals the query point p, it is automatically on any line
    # containing p, but it wasn't counted by the code above. Instead, it was counted in
    # counts_by_slope[None], thus we add it here
    return maximum + counts_by_slope[None]


def slope_between(p, q):
    """Computes the slope between 2 dimensional points, p and q.

    If p and q are on the same vertical line, we adopt the convention that the
    slope between them is infinity. However, if p and q are identical, we'll
    return None so that we can distinguish this corner case.

    """

    p_x, p_y = p
    q_x, q_y = q

    dx = q_x - p_x
    dy = q_y - p_y

    if dx == 0 and dy == 0:
```

```python
        return None

    if dx == 0:
        return float("inf")

    return dy / dx
```