Kade Challis - u1046309
Weston Prestwich - u0827235
Justin Newkirk - u1053607
November 27, 2020
Linear Algebra - Final Project

Markov Chains

Markov Chains are defined as a system of states where each possesses probabilities of transitioning to another associated state. While the transitioning between these states is completely probabilistic, the probabilities between the various states remain constant. This description allows us to predict future events where each event in the system is dependent on the outcome of the previous event.

There are many fields that have systems that fit this description and where Markov Chains are utilized to model these connections and deduce valuable, future information from them. Many of these systems can be wildly, if not impossibly complicated to model physically, but Markov Chains offer a much simpler solution to many of these systems, making them an indispensable tool for many industries. Fields where Markov Chains are used extensively include genetics, traffic flows modeling, text prediction, networking, and in that same vein, Google's page ranking algorithm. This is by no means a comprehensive list, but it gives a strong indication on how important this modeling can be in everyday life.

Linear algebra can be used to mathematically describe Markov Chains and show how they evolve from one iteration to the next, along with the long term behavior of a given system. The terms we use when describing Markov Chains in the context of Linear Algebra include the *transition matrix*, the *initial state vector,* and the *steady state vector*.

The *transition matrix* is a matrix of the probabilities that describe the likelihood of jumping from one node in the system to another associated node. The columns represent each

node, and the rows similarly. The intersection of a given column and row describe the probability of jumping from the column node to the row node. These nodes could be different, or they could be the same node, implying that the node points at itself.

The *initial state vector* encapsulates the initial conditions for the system. This vector is an n node by 1 matrix where each row describes the initial condition for a given variable. This initial state vector is then multiplied to the transition matrix to give us the first *state vector value*, and using that, we can begin recursively multiplying the resulting *state vector value* to the *transition matrix.*

The *steady state vector* is the vector value that is converged upon after many iterations. This *steady state vector* gives us the long term probabilities of each event in relation to the other events.

Problem 1:

When given the transition matrix: $\mathbf{P} = \begin{vmatrix} 0.4 & 0.5 \\ 0.6 & 0.5 \end{vmatrix}$

and the initial state vector: $\begin{vmatrix} 1 \\ 0 \end{vmatrix}$

We are able to determine the value of the state vector by multiplying the transition matrix **P** and the previous state vector value. For the initial state vector value, we simply multiply the initial state vector by the transition matrix. Below are the first ten state vectors and their corresponding product results:

State Vector Number	State Vector Value
1	0.4 0.6
2	0.46

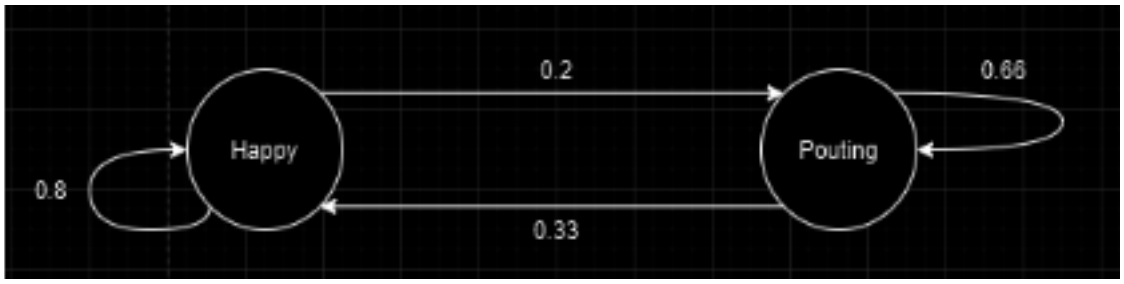
	0.54
3	0.454 0.546
4	0.4546 0.5454
5	0.45454 0.54546
6	0.454546 0.545454
7	0.4545454 0.5454546
8	0.45454546 0.54545454
9	0.454545454 0.545454546
10	0.4545454546 0.5454545454

From this table, we can see that as we continue to multiply the transition matrix by the previous state vector, we approach the approximate values 0.455 and 0.545 respectively. This gives us our steady state matrix for this particular problem.

Steady state matrix: $\begin{vmatrix} 0.455 \\ 0.545 \end{vmatrix}$

Problem 2:

In this problem, we have a description of a puppy, Gauss, where he is either happy or pouting on any given day along with probabilities on how he will be feeling on a future day based on how he's feeling on that day. The image below describes the relationship between the happy and pouting days and their corresponding probabilities:



From this diagram, we are able to determine our transition matrix.

Transition matrix: $\begin{vmatrix} 0.8 & 0.2 \\ 0.33 & 0.66 \end{vmatrix}$

Initial state vector: $\begin{vmatrix} 1 \\ 0 \end{vmatrix}$

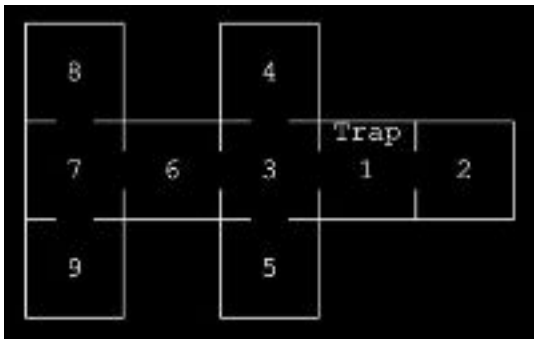
Here, the steady state matrix below through the same methods as the first problem. We calculated as many state vectors as needed for the numbers within them to converge to the numbers below.

Steady state matrix: $\begin{vmatrix} 0.625 \\ 0.375 \end{vmatrix}$

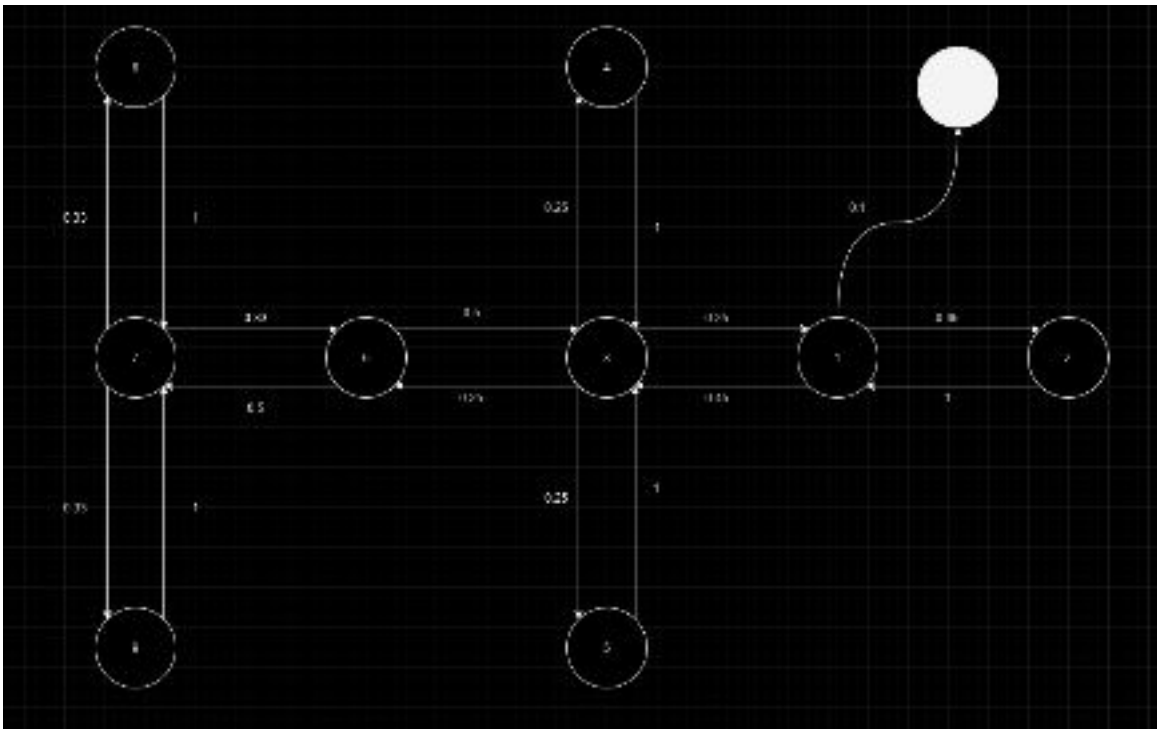
From our steady state matrix, we can determine that Gauss's odds of being happy on a given day are roughly 62.5%.

Problem 3:

In this problem, we are given a house with nine rooms and a mouse trap in room number one. There is a mouse that enjoys exploring the house, but when the mouse enters the room with the mouse trap, it has a probability **p** = 0.1 of getting caught in the trap. The floor plan for this house is seen here:



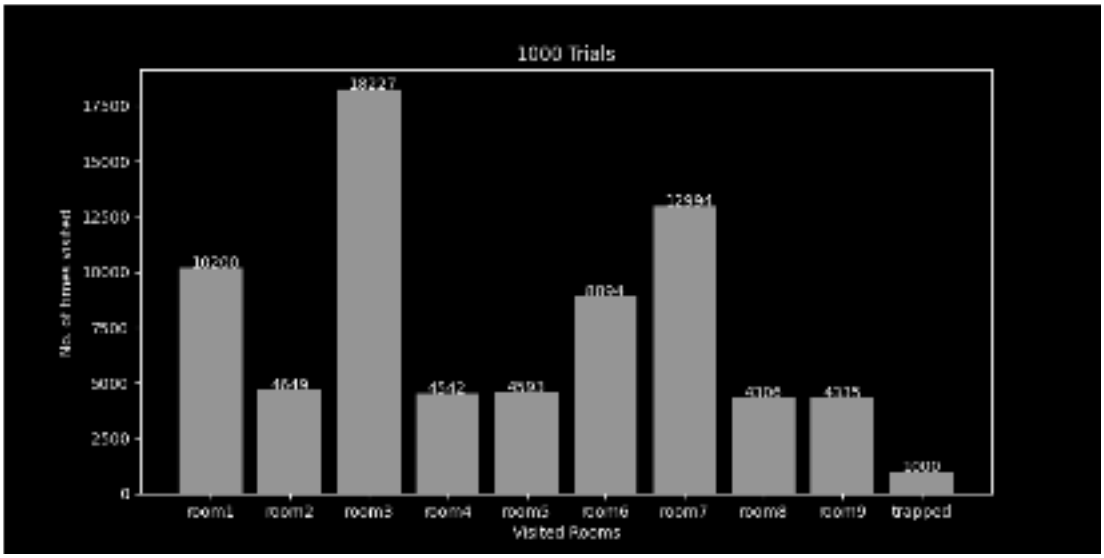
To model how the mouse will move through the house, we can use a Markov Chain containing the probabilities of the mouse entering a particular room given the room the mouse is in. The Markov Chain is modeled like so:



Using this Markov Chain model of the house, we can now find our *transition matrix* for this problem. The transition matrix for the floor plan is:

		From:									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Trap**
To:	**1**	0	**1**	**0.25**	0	0	0	0	0	0	0
	2	**0.45**	0	0	0	0	0	0	0	0	0
	3	**0.45**	0	0	**1**	**1**	**0.5**	0	0	0	0
	4	0	0	**0.25**	0	0	0	0	0	0	0
	5	0	0	**0.25**	0	0	0	0	0	0	0
	6	0	0	**0.25**	0	0	0	**0.33**	0	0	0
	7	0	0	0	0	0	**0.5**	0	**1**	**1**	0
	8	0	0	0	0	0	0	**0.33**	0	0	0
	9	0	0	0	0	0	0	**0.33**	0	0	0
	Trap	**0.1**	0	0	0	0	0	0	0	0	0

To understand the general behavior of this system over time, a simulation with this *transition matrix* was executed where the mouse was placed in a random room and would run around the house until it was caught in the trap. To get a decent distribution, this experiment was executed 1,000 times. Below is a graph of the rooms and how often they were visited:



As we can see from the graph, the mouse visited the rooms 3 and 7 with the greatest frequency. This makes sense as our Markov Chain model as those two nodes acting as intersections of 4 and 3 other nodes respectively. This means while the mouse is traversing the house, it will likely visit these rooms most often. The next two most visited rooms were 1 and 6. This also makes sense as both of these were also acting as an intersection of 2 other nodes, which also explains why their distribution is roughly equal. As expected, rooms 2, 4, 5, 8, and 9 are the least frequented rooms as those are rooms that can only be entered and then immediately exited and lead to no other nodes other than the single node that connects to it.

From this data we can see that intersections in Markov Chains between multiple nodes will typically be visited the most often and nodes that do not lead anywhere other than themselves will be visited less often.

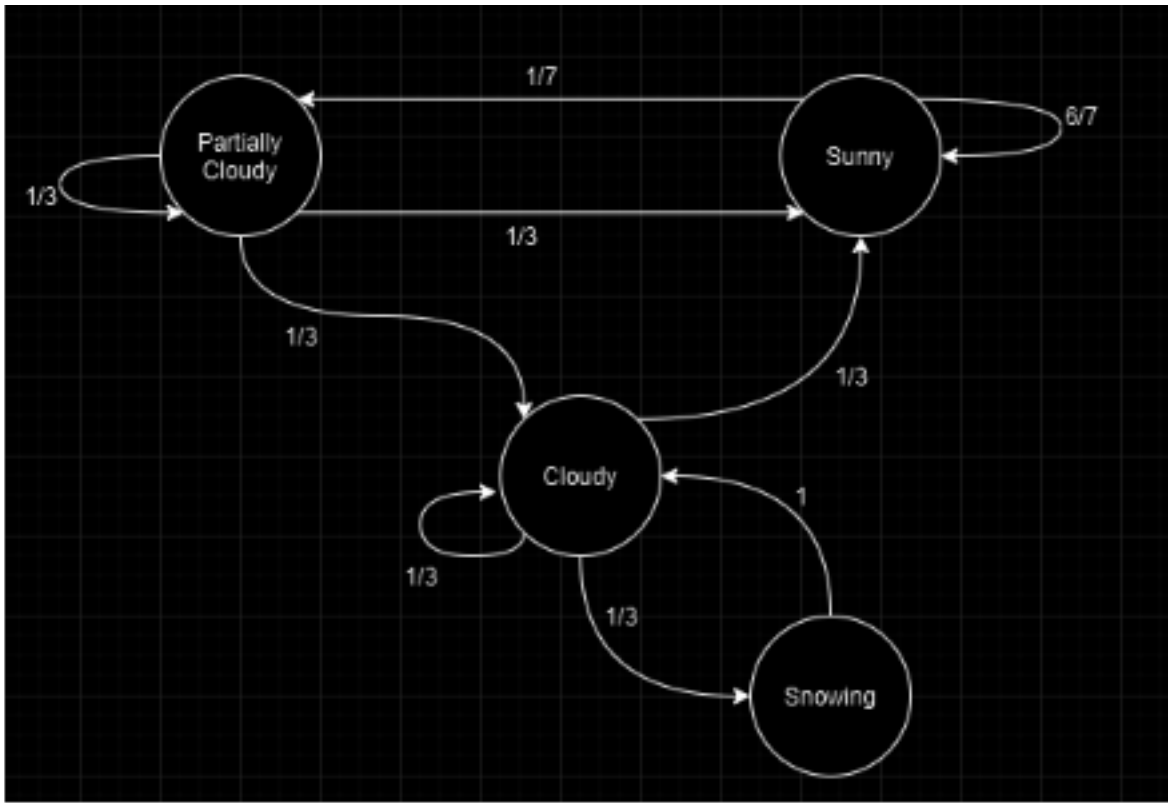
Problem 4:

For problem 4, we designed a Markov Chain that can predict the weather given a specific weather pattern that occurred. The patterns that were included included sunny, partially cloudy, cloudy, and snowing. For our initial data gathered, we used the predicted weather forecast for December 1 - 15, 2020 and got the following event order:

PC : Partially Cloudy, **C** : Cloudy, **Su** : Sunny, and **Sn** : Snowing

PC, PC, Su, Su, Su, Su, Su, Su, Su, PC, C, Sn, C, C, Su

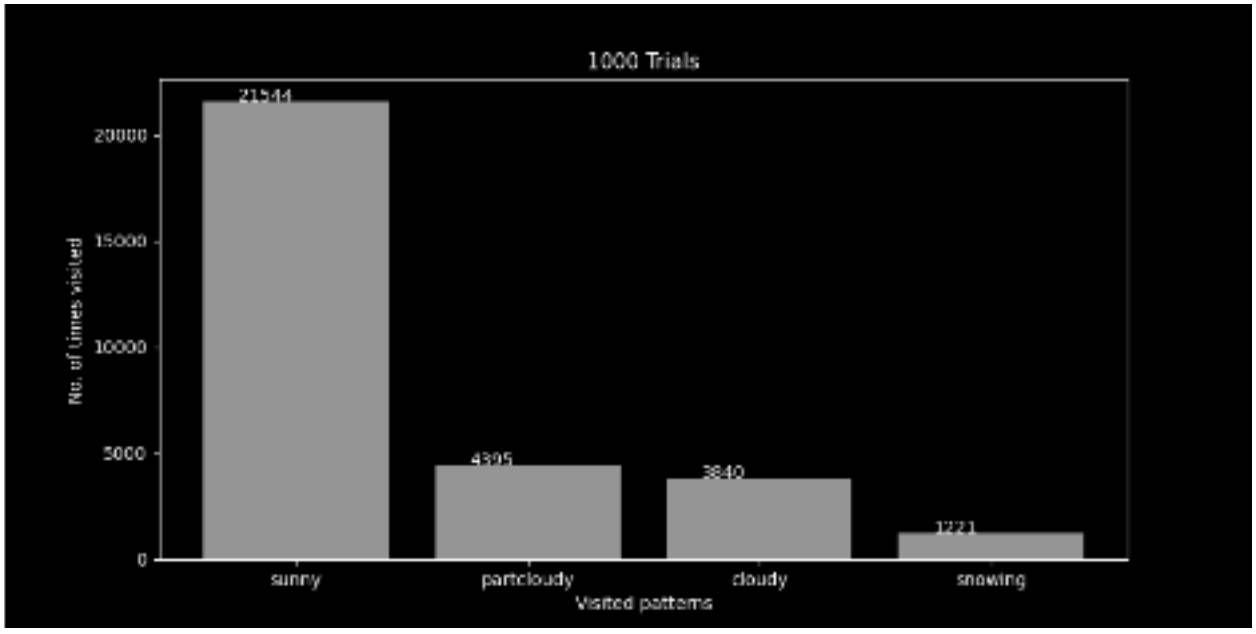
From this, we are able to build our Markov Chain model:



With this Markov Chain, we are now able to generate our *transition matrix*:

	PC	**C**	**Su**	**Sn**
PC	~ 0.33	0	~ 0.14	0
C	~ 0.33	~ 0.33	0	1
Su	~ 0.33	~ 0.33	~ 0.86	0
Sn	0	~ 0.33	0	0

Now that we have our *transition matrix*. From this, we are able to analyze the general behavior of this system. This is done by choosing a random weather node (essentially choosing a random *initial vector value*) and then traversing to 31 other nodes and counting how many times each node is visited. To get a decent distribution, this experiment was repeated 1,000 times. The resulting distribution is graphed as such:



Appendix A - Problem 3's Code:

```
from collections import Counter, defaultdict
import numpy as np
import matplotlib.pyplot as plt


MOUSE_PROB = {
    "room1": {
        "room1": 0,
        "room2": 0.45,
        "room3": 0.45,
        "room4": 0,
        "room5": 0,
        "room6": 0,
        "room7": 0,
        "room8": 0,
        "room9": 0,
        "trapped": 0.1,
    },
    "room2": {
        "room1": 1,
        "room2": 0,
        "room3": 0,
        "room4": 0,
        "room5": 0,
        "room6": 0,
        "room7": 0,
        "room8": 0,
        "room9": 0,
        "trapped": 0,
    },
    "room3": {
        "room1": 0.25,
        "room2": 0,
        "room3": 0,
        "room4": 0.25,
        "room5": 0.25,
        "room6": 0.25,
        "room7": 0,
        "room8": 0,
        "room9": 0,
        "trapped": 0,
    },
    "room4": {
        "room1": 0,
        "room2": 0,
        "room3": 1,
        "room4": 0,
        "room5": 0,
        "room6": 0,
        "room7": 0,
        "room8": 0,
        "room9": 0,
        "trapped": 0,
```

```
},
"room5": {
    "room1": 0,
    "room2": 0,
    "room3": 1,
    "room4": 0,
    "room5": 0,
    "room6": 0,
    "room7": 0,
    "room8": 0,
    "room9": 0,
    "trapped": 0,
},
"room6": {
    "room1": 0,
    "room2": 0,
    "room3": 0.5,
    "room4": 0,
    "room5": 0,
    "room6": 0,
    "room7": 0.5,
    "room8": 0,
    "room9": 0,
    "trapped": 0,
},
"room7": {
    "room1": 0,
    "room2": 0,
    "room3": 0,
    "room4": 0,
    "room5": 0,
    "room6": 0.3333,
    "room7": 0,
    "room8": 0.3333,
    "room9": 0.3334,
    "trapped": 0,
},
"room8": {
    "room1": 0,
    "room2": 0,
    "room3": 0,
    "room4": 0,
    "room5": 0,
    "room6": 0,
    "room7": 1,
    "room8": 0,
    "room9": 0,
    "trapped": 0,
},
"room9": {
    "room1": 0,
    "room2": 0,
    "room3": 0,
    "room4": 0,
    "room5": 0,
    "room6": 0,
    "room7": 1,
```

```
        "room8": 0,
        "room9": 0,
        "trapped": 0,
    },
    "trapped": {
        "room1": 0,
        "room2": 0,
        "room3": 0,
        "room4": 0,
        "room5": 0,
        "room6": 0,
        "room7": 0,
        "room8": 0,
        "room9": 0,
        "trapped": 1,
    },
}

MOUSE_ROOMS = list(MOUSE_PROB.keys())


def next_room(current):
    """
    Given the probabilities declared in MOUSE_PROB, determine which room the
mouse next visits.
    """

    return np.random.choice(MOUSE_ROOMS, p=list(MOUSE_PROB[current].values()))


def run_until_trapped(start):
    """
    Let the mouse run until he becomes trapped.

    Returns the sequence of rooms visited.
    """

    sequence = [start]

    current = start
    while current != "trapped":
        current = next_room(current)
        sequence.append(current)

    return sequence


def generate_plot(runs):
    """
    Generate a plot given some number of executions.
    """

    total = defaultdict(int)
    for _ in range(runs):
        counts = Counter(run_until_trapped("room1"))
        for room in MOUSE_ROOMS:
            total[room] += counts.get(room, 0)
```

```
    values = [total[x] for x in MOUSE_ROOMS]

    _ = plt.figure(figsize=(10, 5))
    plt.bar(MOUSE_ROOMS, values, label="Rooms")
    plt.title(f"{runs} Trials")
    plt.xlabel("Visited Rooms")
    plt.ylabel("No. of times visited")

    for i, val in enumerate(values):
        plt.text(i - 0.25, val, str(val))

    plt.savefig("genfig-3.png")


def output_single():
    """
    Produce the output for a single run through.
    """

    execution = run_until_trapped("room1")

    answer = f"""
    Problem 3:
        Rooms Before Trapped: {len(execution) - 1}
        Sequence Of Rooms: {execution}
    """

    print(answer)


output_single()
generate_plot(1000)
```

Appendix B - Problem 4's Code:

```
from collections import Counter, defaultdict
import numpy as np
import matplotlib.pyplot as plt


WEATHER_PROB = {
    "sunny": {
        "sunny": 6 / 7,
        "partcloudy": 1 / 7,
        "cloudy": 0,
        "snowing": 0,
    },
    "partcloudy": {
        "sunny": 1 / 3,
        "partcloudy": 1 / 3,
        "cloudy": 1 / 3,
        "snowing": 0,
    },
    "cloudy": {
        "sunny": 1 / 3,
        "partcloudy": 0,
        "cloudy": 1 / 3,
        "snowing": 1 / 3,
    },
    "snowing": {
        "sunny": 0,
        "partcloudy": 0,
        "cloudy": 1,
        "snowing": 0,
    },
}

WEATHER_PAT = list(WEATHER_PROB.keys())


def next_pattern(current):
    """
    Given the probabilities declared in WEATHER_PAT, determine which weather
pattern occurs next.
    """

    return np.random.choice(WEATHER_PAT,
p=list(WEATHER_PROB[current].values()))


def run_from_start(start, transitions=30):
    """
    Let the weather shift for the number of transitions.

    Returns the sequence of weather patterns.
    """

    sequence = [start]
```

```
    current = start
    for _ in range(transitions):
        current = next_pattern(current)
        sequence.append(current)

    return sequence


def generate_plot(runs):
    """
    Generate a plot given some number of executions.
    """

    total = defaultdict(int)
    for _ in range(runs):
        counts = Counter(run_from_start("sunny"))
        for pattern in WEATHER_PAT:
            total[pattern] += counts.get(pattern, 0)

    values = [total[x] for x in WEATHER_PAT]

    _ = plt.figure(figsize=(10, 5))
    plt.bar(WEATHER_PAT, values, label="patterns")
    plt.title(f"{runs} Trials")
    plt.xlabel("Visited patterns")
    plt.ylabel("No. of times visited")

    for i, val in enumerate(values):
        plt.text(i - 0.25, val, str(val))

    plt.savefig("genfig-4.png")


def output_single():
    """
    Produce the output for a single run through.
    """

    execution = run_from_start("sunny")

    answer = f"""
    Problem 4:
        Sequence Of Weather Patterns: {execution}
    """

    print(answer)


output_single()
generate_plot(1000)
```