Manim Tutorial

The most convenient way to create manim animations is to use the linux terminal. If you are using windows, you can use WSL as usual. The needed commands are given in the following slides.

Installing and using manim

```
sudo apt update && sudo apt upgrade -y

sudo apt install python3 python3-pip python3-venv -y

python3 -m venv manim-venv

source manim-venv/bin/activate

sudo apt install -y libcairo2-dev libpango1.0-dev ffmpeg pkg-config
python3-dev

pip install manim
```

Create animation using python default version

```python
from manim import *
class DijkstraDemo(Scene):
    def construct(self):
        # Define graph layout
        positions = {
            "A": LEFT * 4 + UP * 2,
            "B": LEFT * 1 + UP * 2,
            "C": RIGHT * 2 + UP * 2,
            "D": LEFT * 3 + DOWN * 1,
            "E": RIGHT * 0 + DOWN * 1,
            "F": RIGHT * 3 + DOWN * 2,
        }
        edges = [
            ("A", "B", 4),
            ("A", "D", 2),
            ("B", "C", 6),
            ("B", "E", 3),
            ("C", "F", 1),
            ("D", "E", 1),
            ("E", "C", 2),
            ("E", "F", 4),
        ]
        nodes = {name: Dot(positions[name], color=BLUE) for name in positions}
        labels = {name: Text(name).scale(0.5).next_to(nodes[name], UP) for name in positions}
        for name in nodes:
            self.add(nodes[name], labels[name])
        edge_lines = {}
        weight_labels = {}
        for u, v, w in edges:
            line = Line(positions[u], positions[v], color=GRAY)
            edge_lines[(u, v)] = edge_lines[(v, u)] = line
            weight = Text(str(w)).scale(0.4).move_to(line.get_center()).shift(0.2 * UP)
            weight_labels[(u, v)] = weight_labels[(v, u)] = weight
            self.add(line, weight)
        self.wait(1)
        # Simulate Dijkstra's Algorithm from node A
        visited = set()
        distance = {node: float('inf') for node in positions}
        distance["A"] = 0
        path_tree = {}
        steps = []
        while len(visited) < len(positions):
            current = min((node for node in distance if node not in visited), key=lambda x: distance[x])
            visited.add(current)
            nodes[current].set_color(GREEN)
            self.play(nodes[current].animate.set_color(GREEN), run_time=0.5)
            self.wait(0.2)
            for (u, v, w) in edges:
                if u == current and v not in visited:
                    if distance[current] + w < distance[v]:
                        distance[v] = distance[current] + w
                        path_tree[v] = u
                        steps.append(((u, v), distance[v]))
                if v == current and u not in visited:
                    if distance[current] + w < distance[u]:
                        distance[u] = distance[current] + w
                        path_tree[u] = v
                        steps.append(((v, u), distance[u]))
        # Highlight the shortest path tree
        for (u, v), d in steps:
            if path_tree.get(v) == u or path_tree.get(u) == v:
                self.play(edge_lines[(u, v)].animate.set_color(YELLOW), run_time=0.5)
                self.wait(0.2)
        self.wait(2)
```

Example code that uses manim to create an animation

```
from manim import *
class PrimAlgorithmLight(Scene):
    def construct(self):
        # Light theme setup
        self.camera.background_color = WHITE
        # Node positions
        positions = {
            "A": LEFT * 4 + UP * 2,
            "B": LEFT * 1 + UP * 2,
            "C": RIGHT * 2 + UP * 2,
            "D": LEFT * 3 + DOWN * 1,
            "E": RIGHT * 0 + DOWN * 1,
            "F": RIGHT * 3 + DOWN * 2,
        }
        edges = [
            ("A", "B", 4),
            ("A", "D", 2),
            ("B", "C", 6),
            ("B", "E", 3),
            ("C", "F", 1),
            ("D", "E", 1),
            ("E", "C", 2),
            ("E", "F", 4),
        ]
        # Draw nodes and labels
        nodes = {name: Dot(positions[name], color=BLACK) for name in positions}
        labels = {name: Text(name, color=BLACK).scale(0.5).next_to(nodes[name], UP) for name in 
positions}
        for name in nodes:
            self.add(nodes[name], labels[name])

        # Draw edges
        edge_lines = {}
        weight_labels = {}
        for u, v, w in edges:
            line = Line(positions[u], positions[v], color=GRAY)
            edge_lines[(u, v)] = edge_lines[(v, u)] = line
            weight = Text(str(w), color=BLACK).scale(0.4).move_to(line.get_center()).shift(0.2 * 
UP)
            weight_labels[(u, v)] = weight_labels[(v, u)] = weight
            self.add(line, weight)

        self.wait(1)

        # Prim's Algorithm Step-by-Step (visually)
        mst_edges = [("A", "D"), ("D", "E"), ("E", "C"), ("C", "F"), ("E", "B")]
        for u, v in mst_edges:
            line = edge_lines[(u, v)]
            self.play(line.animate.set_color(ORANGE), run_time=0.8)
            self.wait(0.2)

        # Final emphasis
        self.wait(2)
```

Running manim to generate the animation

```
source manim-venv/bin/activate
```

mp4:

```
manim -pql dij.py DijkstraDemo
```

gif:

```
manim -pql dij.py DijkstraDemo --format=gif -o dijkstra.gif
```