Group

- With whom do you plan to work? Groups must be 2 – 3 members (you must speak with me if you would like to work with a different size group).
  - Cooper, Aayush, Mike, Hayden

Topic

- Which discrete topic(s) will you work with? / What real world topic(s) are you interested in working with?
  - Sets / subsets - Food classification in a grocery store
  - Graph theory - GPS navigation system
- What makes your real-world topic interesting/important to understand?
  - Sets and subsets are seen all over in the real world so it can be important to understand. In the case of fruit classification, it relates to inventory management, customer experience, supply chain efficiency, quality control, and marketing and sales.
  - Understanding how graph theory is applied in GPS navigation systems is crucial as it enables efficient route planning, real-time traffic management, and spatial representation of road networks. This knowledge enhances user experience, informs transportation infrastructure planning, and sheds light on emerging technologies like autonomous vehicles and smart cities. Overall, it illustrates the practical significance of mathematical concepts in solving real-world problems and improving modern navigation systems.

Building Blocks

- How will you model your real-world phenomenon with discrete math?
  - GPS navigational systems apply graph theory extensively, using the quickest most suitable routes keeping time as a major factor in choosing that route
  - Food in the grocery store is a simple yet effective example of sets and subsets as it shows a effective way to distinguish and categorize items

- What makes your project topic creative?
  - Overall the algorithms of the GPS relating to graphs will make it look so creative.
  - The fruit in the store might make it feel like a list of stuff to be found in the store which might make it look cool.

- Your project must demonstrate a deep understanding of the discrete topic(s) you chose – how do you intend to do this? In other words, you cannot rely solely on definitions, you must dig a little deeper or look at something in a creative new way.
    - To deeply understand graph theory in GPS navigation, we will explore the efficiency of algorithms and examine real-world examples such as Dijkstra's algorithm for finding the best path through a graph as well as how the road and traffic light system can be represented as a graph.
    - To deeply understand sets and their classification in the real world, we will analyze how sets interact, research case studies on their applications like customer segmentation, compare different classification approaches, and explore their uses.
- You will ultimately give a presentation on your work, what ideas do you have for making that presentation interactive for the audience?
    - ❖ We will try to make interesting powerpoint with cool activities and funny examples relating to the topic.

Roadblocks

- What roadblocks do you anticipate as you complete your project?
    - Graph theory for the GPS Navigational system is a newer subject and may take more time to understand dedicated more time to understanding a concept than the real world phenomenon itself
    - Fruits in the grocery store can be a huge category, as well as all fruit are not all the same ripeness and flavor. Categorizing these details can take very very specific sets and subsets to utilize

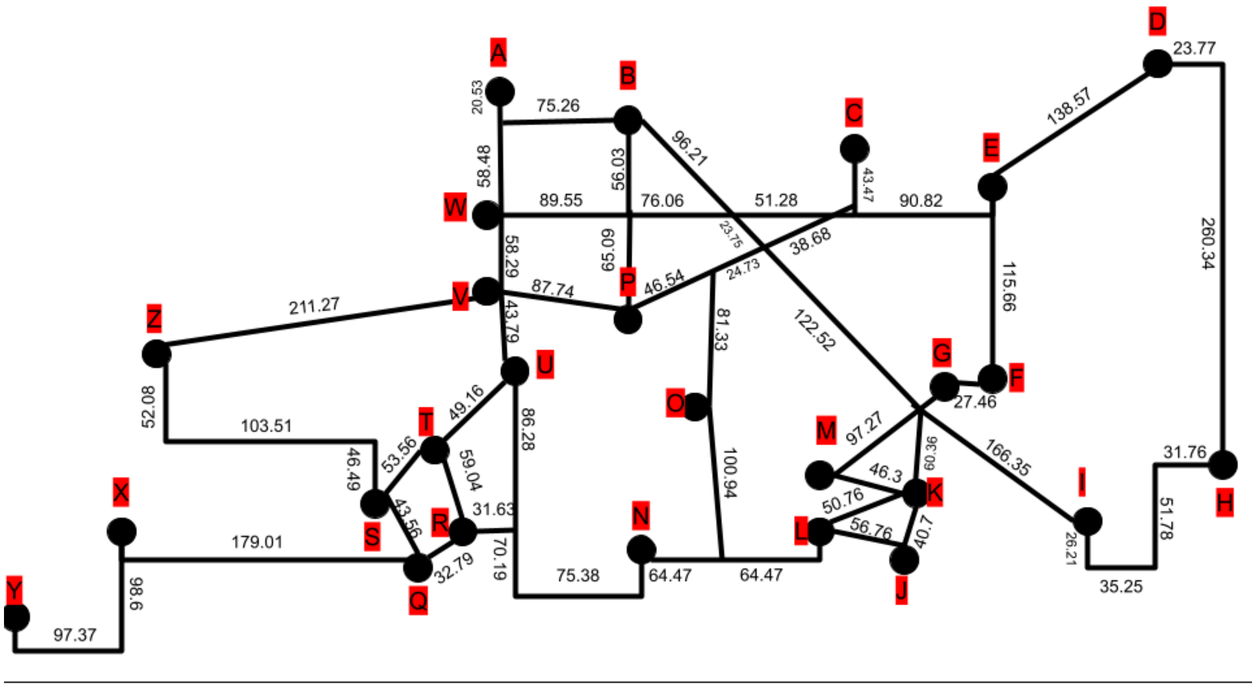
- Detail your plan for dealing with these roadblocks. How will you overcome them, or if you cannot, how will you modify your project?
    - Graph Theory for the GPS Navigation System would take extra days to understand and apply which is why we would allocate most of our beginning time to understanding graph theory much more precisely
    - Fruits in the grocery store can contain many sets and subsets to which all individuals could belong which is why it must be vital that we keep the categories to a minimum that accurately depict the fruit.
- Dijkstra Algorithm: https://youtu.be/XB4MIexjvY0?si=PPDc9aUqe-L1XCT7
    - Optimizing with traffic routes, road closures, and traffic conditions
    - Creating a code of a mini map that allows the user to travel from any two of there chosen points even if a traffic condition is tested

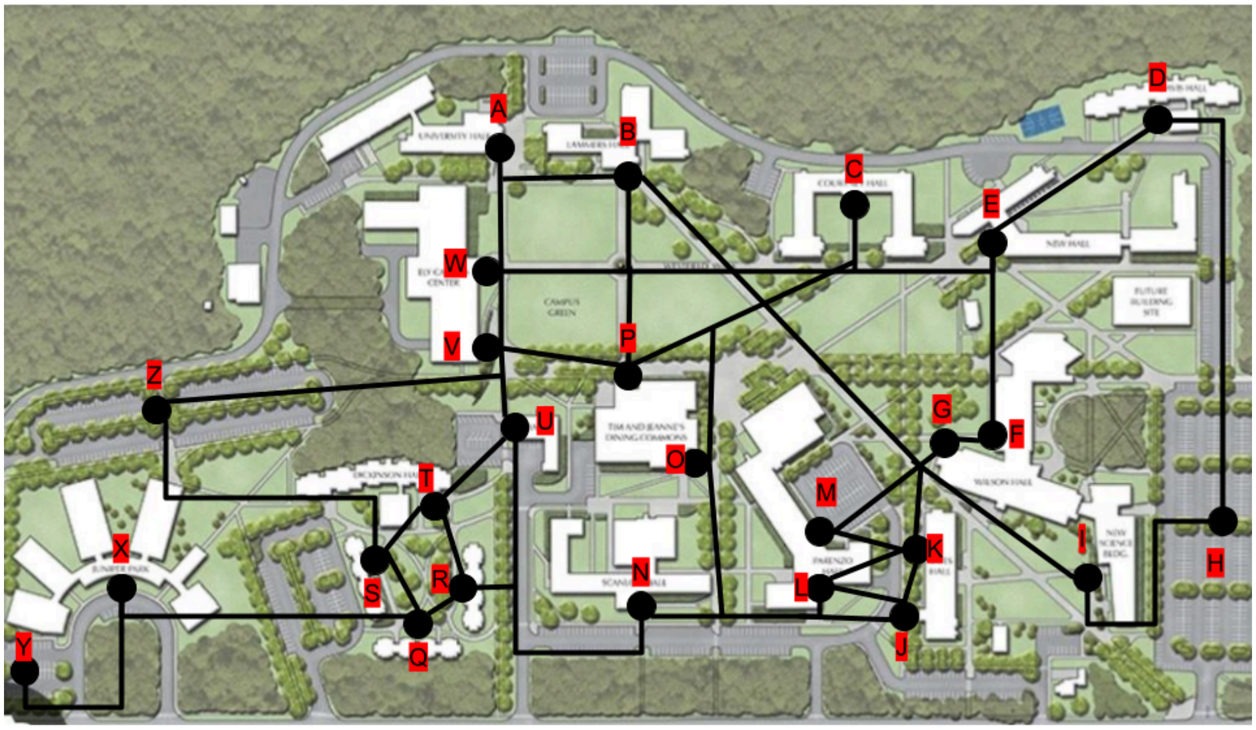
- Take into account that navigation algorithms often optimize their results based on time, i.e. fastest route, instead of based on distance, i.e. shortest route. So the navigation algorithms would have to take into account the different allowed speed limits on the various parts of the road network, and also handle the time-varying dynamic factor of traffic congestion, rush hours, accidents, etc. that occasionally cause traffic to move in speeds less than the allowed speed limit.
    - Because of this, a naive Dijkstra implementation very often is not enough, and the algorithm needs to be much more sophisticated (but still basically based on Dijkstra at the core).
- Model campus as a graph with buildings being nodes and routes between them edges
    - Explain different algorithms for shortest and fastest paths between destinations
    - Djikstras: shortest
    - Djikstra modified: fastest
        - Assign edge weights for different routes

Tasks:

- Graph overview
- How graph theory applies in different aspect of real world
- How graph theory more directly applies to navigation
- How westfield can be modeled as a graph (nodes, edges, and weights)
- Algorithm and explanation to determine shortest path around campus (code)
- Interactive: have people choose and guess different start/end locations

---


D
23.77
A
B
20.53
75.26
C
96.21
138.57
E
58.48
56.03
43.47
W
89.55
76.06
51.28
90.82
260.34
23.75
38.68
58.29
65.09
24.73
115.66
V
P
46.54
87.74
Z
211.27
43.79
81.33
122.52
U
G
F
27.46
52.08
49.16
86.28
O
103.51
T
97.27
M
96.09
166.35
I
H
46.49
53.56
59.04
46.3
31.76
X
K
50.76
S
43.56
R
31.63
N
L
56.76
40.7
179.01
51.78
26.21
Q
32.79
70.19
100.94
Y
98.6
75.38
64.47
64.47
J
35.25
97.37


A
B
C
D
E
W
V
P
Z
U
G
F
O
T
M
K
I
H
X
S
R
N
L
J
Y
Q
UNIVERSITY HALL
LAMMERS HALL
COURTNEY HALL
NEW HALL
CAMPUS GREEN
FUTURE BUILDING SITE
TIM AND JEANNE'S DINING COMMONS
DICKINSON HALL
WILSON HALL
NEW SCIENCE BLDG.
JUNIPER PARK
PARENZO HALL

https://docs.google.com/spreadsheets/d/1LzyrXOlbNcCNtpG_1A1QwvgBIF6cIoYLDVeoDw7-Tmk/edit?usp=sharing

This one allows edits^

POWERPOINT LINK: Discrete Final

CODE: https://github.com/cboard9/Westfield-Shortest-Path.git

Video for code https://youtu.be/SZXXnB7vSm4?si=aQURv32bzjeh8zoz
Explanation: https://www.youtube.com/watch?v=pVfj6mxhdMw
Implementation (pseudo): https://www.youtube.com/watch?v=pSqmAO-m7Lk

Program: Allows users to pick any two nodes from a graph as automatically finds the shortest path between points
If there is time:

- We could add a variable that closes off one of the paths to prove that it will still find the shortest and isn't a fixed value

In Meters
A: University Hall
B: Lammers Hall
C: Courtney Hall
D: Davis Hall
E: New Hall / Marketplace
F: Owl Cafe
G: WIlson Academic Hall
H: Commuter Lot 1
I: New Science Academic Hall
J: Bates Hall Entrance 1
K: Bates Hall Entrance 2
L: Parenzo Hall Entrance 1
M: Parenzo Hall Entrance 2
N: Scanlon Hall
O: Bistro Opening
P: Dining Commons
Q: Apartments 2
R: Apartments 1
S: Apartments 3
T: Dickinson Hall
U: Maintenance Building
V: Dunkin Donuts
W: Ely Hall
X: Dower Center
Y: South Lot Entrance

Z: Commuter Lot 2

- The two graphs above is westfield state campus and the outline I made of it, but after I did this I realized it's not the campus that actually came out as there are some parts of it that are misinformed such as the sidewalk to goes from the globe to Ely center, on the actual campus that isn't there
- In the shape of the campus which is just the black lines and dots it is measured in meters, I used google globe in order to get better measurements
- I labeled all the dots with letters that later correspond with which building is which dot so that it helps you orient the shape better
- In the google sheets link I found every direct line from every letter to letter so that we could create a 2d array with that information
- I also showed that there's a lot of combos for what the user could enter there's 351 so i tried to be as precise as I could
- In the code video it shows at the beginning the developer copying and printing some pre written array which is what is written below, he only has 6 letters but the westfield state is 26 letters so it is much bigger
- In order to read it it works like this {A, B, C, D}
    - The first organized information would be {AA, AB, AC, AD} and AA would be 0 since that's the distance from oneself so it would be {O, x, x, x} x representing some sort of distance
    - The next organized pair would be {BA, BB, BC, BD} which BA would be the same as AB and BB would be 0 so like this {AB, 0, x, x} x again representing some sort on length
- In between the 2 vertical lines is what should be able to just be copy and paste into the code unless i made a mistake, If you want to look over it the slide is above and I didnt use any other information on it
- I also worked on a simplified version of the map shape, it still looks complicated but does represent the more active nodes
- -MIKE

A {0, 95.79, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 79.01, 0, 0, 0}

B {95.79, 0, 190.96, 0, 238.31, 0, 242.48, 0, 408.65, 0, 302.84, 0, 339.75, 0, 226.02, 121.12, 0, 0, 0, 0, 0, 0, 145.58, 0, 0, 0}

C {0, 190.92 , 0, 0, 134.29, 0, 202.67, 0, 369.02, 0, 263.03, 0, 299.94, 0, 188.21, 153.42, 0, 0, 0, 0, 0, 0, 260.36, 0, 0, 0}

D {0, 0, 0, 0, 138.57, 0, 0, 284.11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

E {0, 238.31, 134.29, 138.57, 0, 115.66, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

F {0, 0, 0, 0, 115.66, 0, 27.46, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,} G {0, 242.48, 202.67, 0, 0, 27.46, 0, 0, 166.35, 0, 60.36, 0, 97.27, 0, 228.58, 193.79, 0, 0, 0, 0, 0, 0, 311.88, 0, 0, 0}

H {0, 0, 0, 284.11, 0, 0, 0, 0, 145, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

I {0, 408.65, 369.02, 0, 0, 0, 166.35, 145, 0, 0, 226.71, 0, 263.62, 0, 394.93, 360.14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

J {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 40.7, 56.76, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }

K {0, 302.84, 263.03, 0, 0, 0, 60.36, 0, 226.71, 40.7, 0, 50.76, 46.3, 0, 288.94, 254.15, 0, 0, 0, 0, 0, 0, 372.74, 0, 0, 0}

L {0, 0, 0, 0, 0, 0, 0, 0, 0, 56.76, 50.76, 0, 0, 128.94, 165.41, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

M { 0, 339.75, 299.94, 0, 0, 0, 97.72, 0, 263.62, 0, 46.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

N {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128.94, 0, 0, 165.41, 0, 0, 177.2, 0, 0, 231.85, 0, 0, 0, 0, 0}

O {0, 262.02, 188.21, 0, 0, 0, 228.58, 0, 394.93, 0, 288.94, 165.41, 0, 165.41, 0, 127.87, 0, 0, 0, 0, 0, 0, 282.51, 0, 0, 0}

P {0, 121.12, 153.42, 0, 0, 0, 193.79, 0, 360.14, 0, 254.15, 0, 0, 0, 127.87, 0, 0, 0, 0, 0, 0, 87.74, 0, 0, 0, 0}

Q {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 32.79, 43.56, 0, 0, 0, 0, 179.01, 0, 0, }

R {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 177.2, 0, 0, 32.79, 0, 0, 59.04, 117.91, 0, 0, 0, 0, 0}

S {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 43.56, 0, 0, 53.56, 0, 0, 0, 0, 0, 202.08}

T {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59.04, 53.56, 0, 49.16, 0, 0, 0, 0, 0}

U {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 231.85, 0, 0, 0, 117.91, 0, 49.16, 0, 43.79, 0, 0, 0, 0}

V {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 282.51, 87.74, 0, 0, 0, 0, 43.79, 0, 58.29, 0, 0, 211.27}

W {79.01, 145.58, 260.36, 0, 0, 0, 311.88, 0, 0, 0, 372.24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 58.29, 0, 0, 0, 0}

X { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 179.01, 0, 0, 0, 0, 0, 0, 0, 195.97, 0}

Y {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 195.97, 0, 0}

Z { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 202.08, 0, 0, 211.27, 0, 0, 0, 0, }

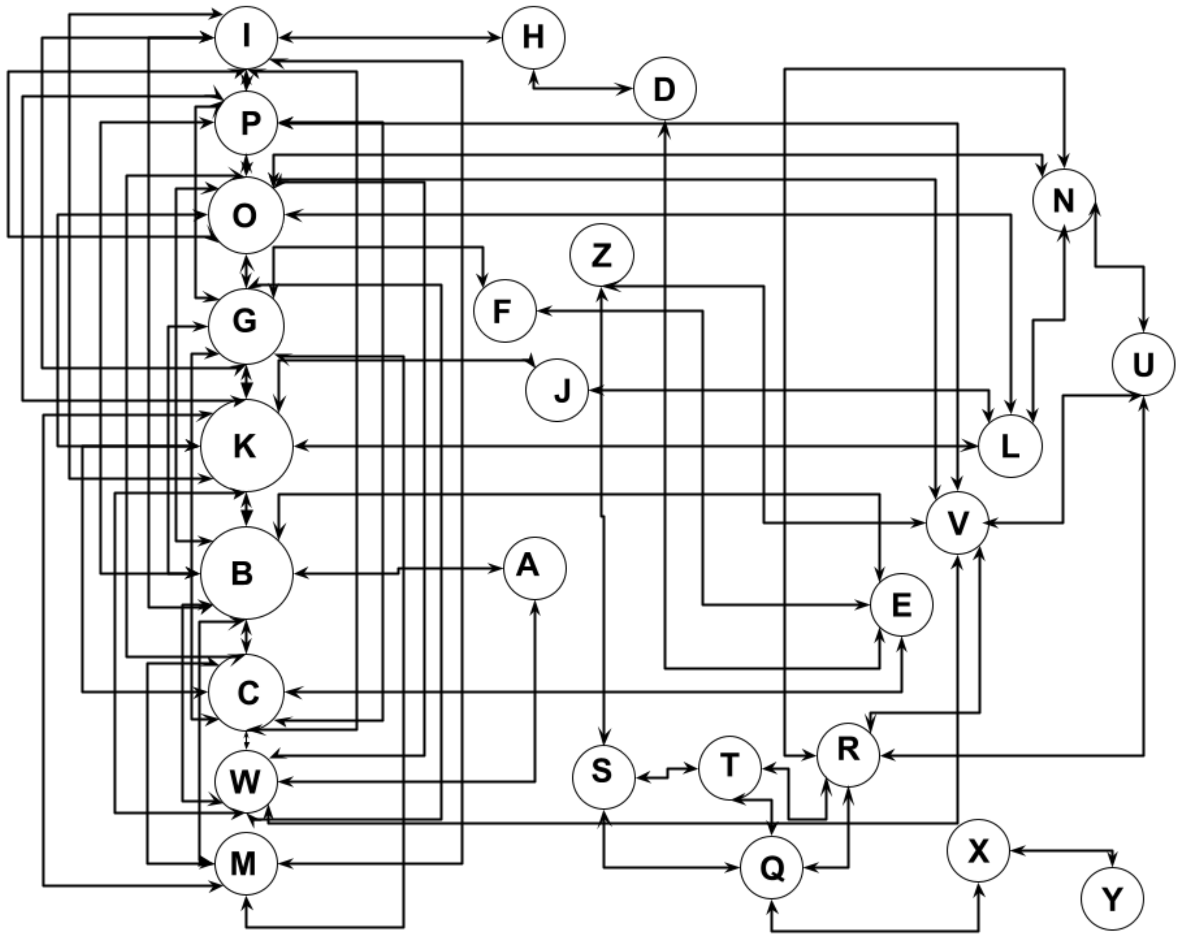
S array to Y

```
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 43.56, 0, 0, 53.56, 0, 0, 0,
0, 0, 202.08},
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 195.97,
0, 0},
```

A to M, G is used as an intersection point

```
A to Y out of bounds exception
B to Y out of bounds exception
C to Y out of bounds exception
D to Y out of bounds exception
E to Y out of bounds exception
F to Y out of bounds exception
```

```
{0, 95.79, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 79.01, 0, 0, 0}
{95.79, 0, 190.96, 0, 238.31, 0, 242.48, 0, 408.65, 0, 302.84, 0, 339.75, 0, 226.02, 121.12, 0, 0, 0, 0, 0, 0, 145.58, 0, 0, 0}
{0, 190.92 , 0, 0, 134.29, 0, 202.67, 0, 369.02, 0, 263.03, 0, 299.94, 0, 188.21, 153.42, 0, 0, 0, 0, 0, 0, 260.36, 0, 0, 0}
{0, 0, 0, 0, 138.57, 0, 0, 284.11, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
{0, 238.31, 134.29, 138.57, 0, 115.66, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
{0, 0, 0, 0, 115.66, 0, 27.46, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,}
{0, 242.48, 202.67, 0, 0, 27.46, 0, 0, 166.35, 0, 60.36, 0, 97.27, 0, 228.58, 193.79, 0, 0, 0, 0, 0, 0, 311.88, 0, 0, 0}
{0, 0, 0, 284.11, 0, 0, 0, 0, 145, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
{0, 408.65, 369.02, 0, 0, 0, 166.35, 145, 0, 0, 226.71, 0, 263.62, 0, 394.93, 360.14, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 40.7, 56.76, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
{0, 302.84, 263.03, 0, 0, 0, 60.36, 0, 226.71, 40.7, 0, 50.76, 46.3, 0, 288.94, 254.15, 0, 0, 0, 0, 0, 0, 372.74, 0, 0, 0}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 56.76, 50.76, 0, 0, 128.94, 165.41, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
{ 0, 339.75, 299.94, 0, 0, 0, 97.72, 0, 263.62, 0, 46.3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 128.94, 0, 0, 165.41, 0, 0, 177.2, 0, 0, 231.85, 0, 0, 0, 0, 0}
{0, 262.02, 188.21, 0, 0, 0, 228.58, 0, 394.93, 0, 288.94, 165.41, 0, 165.41, 0, 127.87, 0, 0, 0, 0, 0, 0, 282.51, 0, 0, 0}
{0, 121.12, 153.42, 0, 0, 0, 193.79, 0, 360.14, 0, 254.15, 0, 0, 0, 127.87, 0, 0, 0, 0, 0, 0, 87.74, 0, 0, 0, 0}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 32.79, 43.56, 0, 0, 0, 0, 179.01, 0, 0, }
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 177.2, 0, 0, 32.79, 0, 0, 59.04, 117.91, 0, 0, 0, 0, 0}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 43.56, 0, 0, 53.56, 0, 0, 0, 0, 0, 202.08}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59.04, 53.56, 0, 49.16, 0, 0, 0, 0, 0}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 231.85, 0, 0, 0, 117.91, 0, 49.16, 0, 43.79, 0, 0, 0, 0}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 231.85, 0, 0, 0, 117.91, 0, 49.16, 0, 43.79, 0, 0, 0, 0}
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 282.51, 87.74, 0, 0, 0, 0, 43.79, 0, 58.29, 0, 0, 211.27}
{79.01, 145.58, 260.36, 0, 0, 0, 311.88, 0, 0, 0, 372.24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 58.29, 0, 0, 0, 0}
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 179.01, 0, 0, 0, 0, 0, 0, 0, 0, 195.97, 0}
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 195.97, 0, 0}
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 202.08, 0, 0, 211.27, 0, 0, 0, 0, }
```



Revised Plans:

Initially, we planned to model the graph as nodes and edges, where each node represents a location on the campus map (such as buildings, parking lots, or landmarks), and each edge represents a pathway connecting two locations. This approach would involve defining classes for nodes and edges, with attributes such as location name, coordinates, and distance between nodes.

However, as we delved deeper into the project, we realized that representing the campus map with nodes and edges might introduce unnecessary complexity, especially when dealing with a large number of locations and pathways. Additionally, implementing algorithms like Dijkstra's algorithm using this approach would require managing connections between nodes and updating edge weights, which could become cumbersome.

To simplify the modeling process and enhance readability, we opted for an adjacency matrix representation instead. In this approach, we used a two-dimensional array to represent the graph, where each row and column correspond to a location on the campus map. The value stored in each cell of the array represents the distance (or weight) between two locations or 0 if there is none. This matrix-based representation is used in various other graph modeling and program related areas, while also offers a straightforward and intuitive way to visualize the campus map and its connections, making it easier to implement algorithms like Dijkstra's for finding shortest paths.

Additionally, our original plan involved going through the code line by line, explaining its functionality and how it contributes to the overall algorithm. However, as we considered the best approach for presenting the material, we realized that a more visual explanation might be more effective for understanding how Dijkstra's algorithm works.

Instead of dissecting the code directly, we decided to present the algorithm using pseudocode and visual aids on PowerPoint slides. Pseudocode provides a high-level description of the algorithm's steps without being tied to a specific programming language, making it accessible to a broader audience. We could then complement the pseudocode with diagrams or flowcharts illustrating each step of the algorithm, helping viewers grasp the concept visually.

By opting for this approach, we aimed to enhance comprehension and engagement among our audience, as they could follow along with the logical flow of the algorithm and visualize how it operates on a graph. This method allows for a more intuitive understanding of Dijkstra's algorithm, making it easier for viewers to grasp its mechanics and significance in the context of navigation on the campus map.

FINAL REPORT:

POWERPOINT LINK: Discrete Final

CODE: https://github.com/cboard9/Westfield-Shortest-Path.git

Introduction**:**
Our project delves into graph theory, specifically exploring Dijkstra's algorithm for finding the shortest path in a graph. We applied this algorithm to model our school campus, Westfield State, as a graph, enabling us to determine the shortest paths between various destinations on campus. Graph theory is fascinating as it provides a

mathematical framework for representing and analyzing networks, making it an essential tool in navigation systems and other real-world applications.

Overview of Discrete Topics:

- *Graph Theory*: Graph theory is the study of graphs, which consist of vertices (nodes) connected by edges. It provides a mathematical framework for analyzing relationships and connections in networks. In our project, we utilized graph theory to model Westfield State campus as a graph, with buildings as nodes and pathways as edges.
- *Dijkstra's Algorithm:* Dijkstra's algorithm is a graph search algorithm used to find the shortest path between nodes in a weighted graph. It starts at a designated node and iteratively explores the neighboring nodes, updating the shortest distances as it progresses. Dijkstra's algorithm is widely used in navigation systems for finding optimal routes.

Overview of Real-World Topic:

Our real-world topic revolves around navigation systems, particularly GPS navigation. GPS navigation systems rely on graph theory and algorithms like Dijkstra's to provide users with optimal routes between locations. These systems are essential for modern transportation, enabling efficient route planning, real-time traffic updates, and navigation assistance.

Modeling Real-World Topic using Discrete:

To model our real-world topic using discrete mathematics, we represented Westfield State campus as a graph. Each building on campus was represented as a node, and pathways between buildings were represented as edges with weights corresponding to distances. By applying Dijkstra's algorithm to this graph, we were able to determine the shortest paths between any two locations on campus, simulating the functionality of a GPS navigation system.

Shortest Path Code Explanation:

1. *Initialization:*
    - The function takes the graph (**graph**), the number of nodes (**nodes**), the starting point (**start**), and the ending point (**end**) as parameters.
    - It initializes several data structures:
        - **visited**: An ArrayList to keep track of visited nodes.
        - **distance**: An ArrayList to store the distances from the starting node to each node.

        - **previous**: An ArrayList of ArrayLists of Nodes to store the previous nodes in the shortest path.
2. *Setting the Starting Point:*
    - It retrieves the index of the starting and ending nodes (**startIndex** and **endIndex**) from the **letterToIndex**HashMap.
    - Sets the distance of the starting node to 0 and adds it to the **previous** ArrayList.
3. *Min Heap Initialization:*
    - Initializes a min heap (**mh**) to keep track of nodes with the least distance.
    - Inserts the starting node into the min heap.
4. *Main Loop:*
    - The loop runs as long as the min heap is not empty.
    - In each iteration, it extracts the node with the shortest distance (**current**) from the min heap.
    - Marks the current node as visited.
    - Checks if the distance to the current node is already less than the calculated distance. If true, it skips further processing for this node.
    - Loops through the neighboring nodes of the current node:
        - If a node is already visited or there's no edge to it (distance is 0), it skips it.
        - Calculates the new distance to the neighboring node via the current node.
        - If the new distance is less than the current distance to the neighboring node, updates the distance and path accordingly.
        - Inserts the updated node into the min heap.
5. *Path Retrieval:*
    - After the main loop completes, the function returns the ArrayList of nodes representing the shortest path from the starting node to the ending node.

<u>Conclusion:</u>
Through our project, we gained a deeper understanding of graph theory and its practical applications in navigation systems. By modeling our campus as a graph and applying Dijkstra's algorithm, we demonstrated how mathematical concepts can be used to solve real-world problems effectively.

<u>Future Work:</u>
If we were to revisit this project, we would explore additional graph algorithms and optimization techniques to further enhance the functionality of our navigation system. Additionally, we could incorporate real-time data sources such as traffic conditions and

pedestrian traffic to provide more accurate route recommendations. Furthermore, expanding the scope of the project to include outdoor navigation and integration with mobile applications would be an exciting avenue for future development.

Sources:

https://youtu.be/SZXXnB7vSm4?si=aQURv32bzjeh8zoz
https://www.youtube.com/watch?v=pVfj6mxhdMw
https://www.youtube.com/watch?v=pSqmAO-m7Lk
https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/