Dijkstra’s Algorithm and Applications

Parker Cuddeback, Caleb Way, Ethan Baldwin

Why we chose Dijkstra’s Algorithm

-We saw lots of potential in coding Dijkstra’s to create different visuals and real-life applications, given Parker and Caleb’s coding backgrounds.

- Balancing usefulness and exploring different applications we can create with the algorithm.

-Even before narrowing down our topic, we knew that with the right implementation, the results of our project could prove useful to any individual.

Our Initial Thoughts on Implementation

-At first, we pondered on studying something like different snow plow routes in St. Lawrence County, and seeing if we could optimise the routes(or possibly see if the routes match Dijkstra’s).

-After determining that this would be likely unreachable given the amount of time allotted for the project, we turned our eyes toward something more local, and we decided on working with the map of the St. Lawrence University campus.

-Refining our choice, we chose to optimize routes between different buildings on campus using strictly the campus roads and walkways, and create a sort of interactive element to our project.

Dijkstra’s Algorithm

1. Put a label of 0 on vertex u and shade vertex ‘u’.
2. Find the next closest vertex to vertex ‘u’ by finding the smallest sum of a label on any shaded vertex and a weight on an edge connecting a shaded vertex to an unshaded vertex.
3. Highlight the edge and shade the vertex found in step 2, and label the current vertex with this smallest sum.
4. If all vertices are shaded, stop.

Small Example

Following the previously outlined algorithm, here are the results for some shortest paths:

A-M:

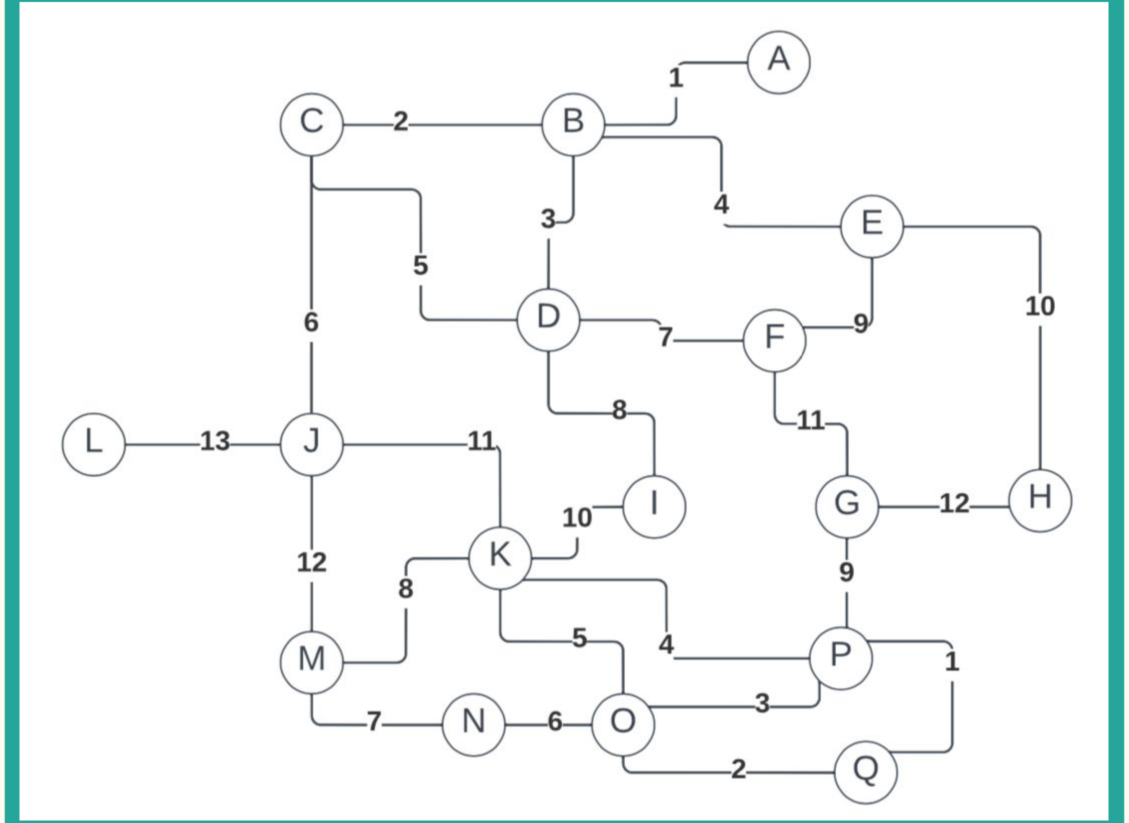
A, B, C, J, M (len=21)

L-H:

L, J, C, B, E, H (len=35)

O-C:

O, K, J, C (len=22)



General Implementation

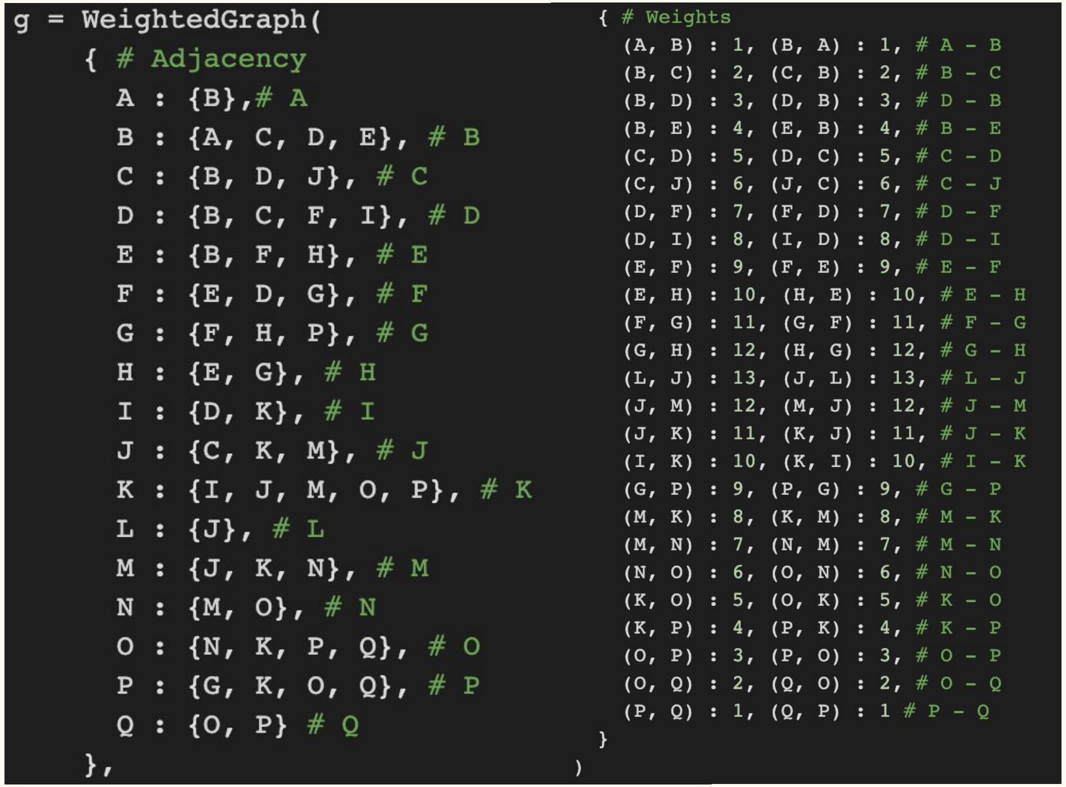
```python
# Defining the structure of a Weighted Graph in Python
class WeightedGraph(object):
    def __init__(self, adjacent, weight):
      self.adjacent = adjacent
      self.weight = weight
```

```python
# This function returns a best
# path to a vertex in a graph.
# The path starts at root of a path tree
# (represented by a parent map).
# If there is no path from this root to this vertex,
# the function returns None.
  def best_path(vertex, parent):
    if vertex not in parent:
      return None
    path = [vertex]
    while parent[vertex] is not None:
      vertex = parent[vertex]
      path.append(vertex)
    path.reverse()
    return path
```

```python
# Dijkstra's algorithm for undirected, weighted and connected graphs
# This function returns a parent map for a path tree from a vertex in a graph.
  def dijkstra_tree(vertex, graph):
    parent = {vertex: None} # Parent map constructed through for loop
    cost = {vertex: 0} # Weights
    new = [(0,vertex)] #
    old = set() # Set of seen vertices
    while len(new) > 0:
      (cv,v) = heappop(new)
      if v not in old: # ignore stale heap entries
        old.add(v)
        for a in graph.adjacent[v]: # For every adjacent vertex
          ca = cv + graph.weight[(v,a)] # Add the current weight and the weight of new path
          if a not in cost or ca < cost[a]:
            # If we havent seen this vertex or the cost is less than current cost:
            parent[a] = v # Redefine current
            cost[a] = ca # Redefine current path
            heappush(new, (ca,a)) # Push to stack
    return parent
```

Constructing The Small Example in Python

1. WeightedGraph contains two components, adjacency and weight
2. Adjacency - dictionary, creates edges between the nodes of the graph
3. Weight - dictionary, creates weights associated with the edges in both the forward and backward direction



```
g = WeightedGraph(
    { # Adjacency
      A : {B},# A
      B : {A, C, D, E}, # B
      C : {B, D, J}, # C
      D : {B, C, F, I}, # D
      E : {B, F, H}, # E
      F : {E, D, G}, # F
      G : {F, H, P}, # G
      H : {E, G}, # H
      I : {D, K}, # I
      J : {C, K, M}, # J
      K : {I, J, M, O, P}, # K
      L : {J}, # L
      M : {J, K, N}, # M
      N : {M, O}, # N
      O : {N, K, P, Q}, # O
      P : {G, K, O, Q}, # P
      Q : {O, P} # Q
    },
```

```
    { # Weights
      (A, B) : 1, (B, A) : 1, # A - B
      (B, C) : 2, (C, B) : 2, # B - C
      (B, D) : 3, (D, B) : 3, # D - B
      (B, E) : 4, (E, B) : 4, # B - E
      (C, D) : 5, (D, C) : 5, # C - D
      (C, J) : 6, (J, C) : 6, # C - J
      (D, F) : 7, (F, D) : 7, # D - F
      (D, I) : 8, (I, D) : 8, # D - I
      (E, F) : 9, (F, E) : 9, # E - F
      (E, H) : 10, (H, E) : 10, # E - H
      (F, G) : 11, (G, F) : 11, # F - G
      (G, H) : 12, (H, G) : 12, # G - H
      (L, J) : 13, (J, L) : 13, # L - J
      (J, M) : 12, (M, J) : 12, # J - M
      (J, K) : 11, (K, J) : 11, # J - K
      (I, K) : 10, (K, I) : 10, # I - K
      (G, P) : 9, (P, G) : 9, # G - P
      (M, K) : 8, (K, M) : 8, # M - K
      (M, N) : 7, (N, M) : 7, # M - N
      (N, O) : 6, (O, N) : 6, # N - O
      (K, O) : 5, (O, K) : 5, # K - O
      (K, P) : 4, (P, K) : 4, # K - P
      (O, P) : 3, (P, O) : 3, # O - P
      (O, Q) : 2, (Q, O) : 2, # O - Q
      (P, Q) : 1, (Q, P) : 1 # P - Q
    }
)
```

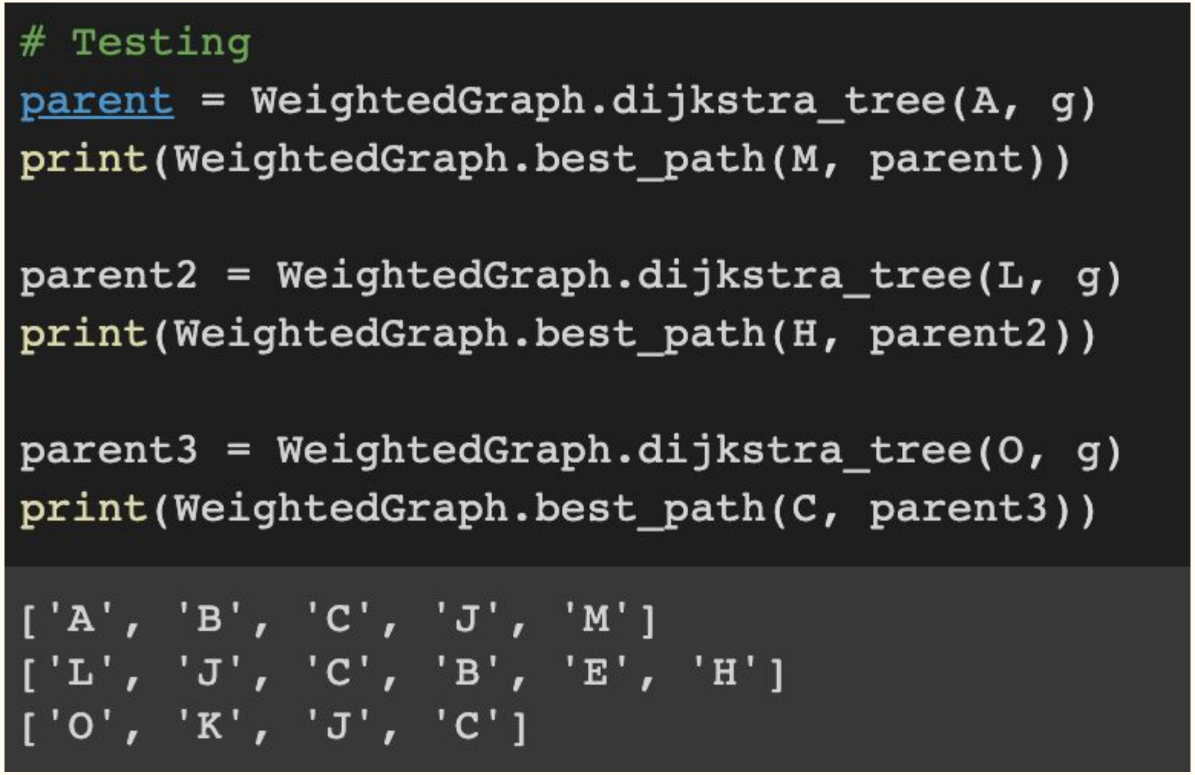
General Implementation Results - Small Example

Although this lightweight implementation does not currently have the capability to display total weight of the resulting shortest path, we can see the implementation selected the same paths we chose in doing the example by hand.

A, B, C, J, M (len=21)

L, J, C, B, E, H (len=35)

O, K, J, C (len=22)

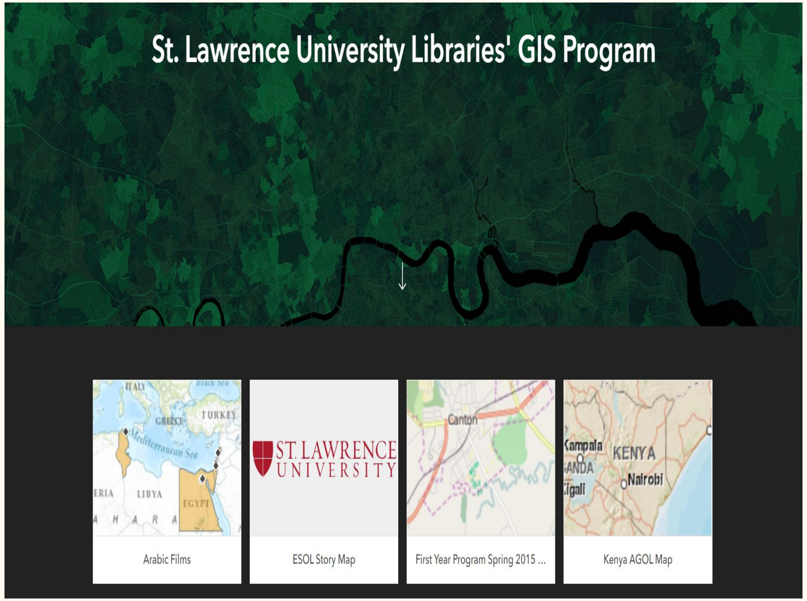


```
# Testing
parent = WeightedGraph.dijkstra_tree(A, g)
print(WeightedGraph.best_path(M, parent))

parent2 = WeightedGraph.dijkstra_tree(L, g)
print(WeightedGraph.best_path(H, parent2))

parent3 = WeightedGraph.dijkstra_tree(O, g)
print(WeightedGraph.best_path(C, parent3))
```

```
['A', 'B', 'C', 'J', 'M']
['L', 'J', 'C', 'B', 'E', 'H']
['O', 'K', 'J', 'C']
```

Using ArcGIS to map out the SLU Campus

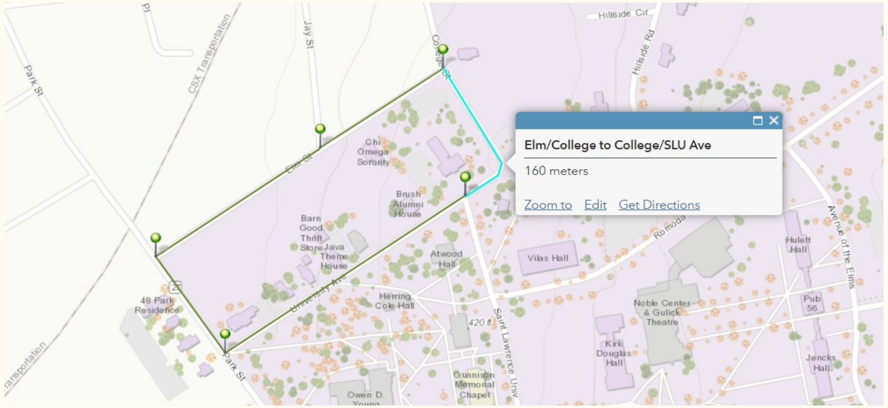
-Dr. Lock referred us to Carol Cady, who runs the SLU Geographic Information Services Program. We reached out to Carol, asking if she had any datasets that contained the road/walkway distances between each point on campus. We scheduled a meeting, and after giving us a crash course on using the program, we were given the option of using a premade campus map that already had all of the distances and nodes burned into the program, but we chose to map it out ourselves and have control over how we create our map.


St. Lawrence University Libraries' GIS Program
Arabic Films
ESOL Story Map
First Year Program Spring 2015 ...
Kenya AGOL Map

ArcGIS homepage

Laying out our Map in GIS

-Within ArcGIS, we are able to place pinpoints(vertices) on each intersection and building we would include in our map. After laying out our vertices, we were able to create edges between nodes, and use the measuring feature to map out our distances(weights) between vertices.

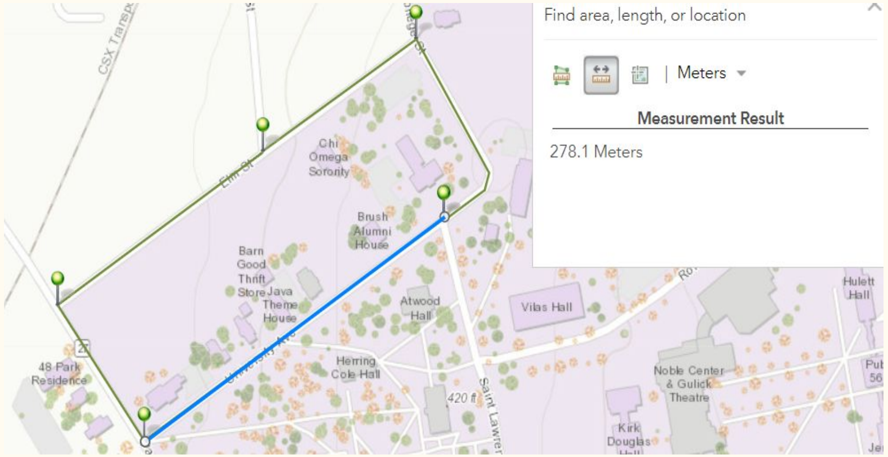


Elm/College to College/SLU Ave

160 meters

Zoom to Edit Get Directions

Mapping Ex.



Find area, length, or location

Meters

Measurement Result

278.1 Meters

Measuring Ex.

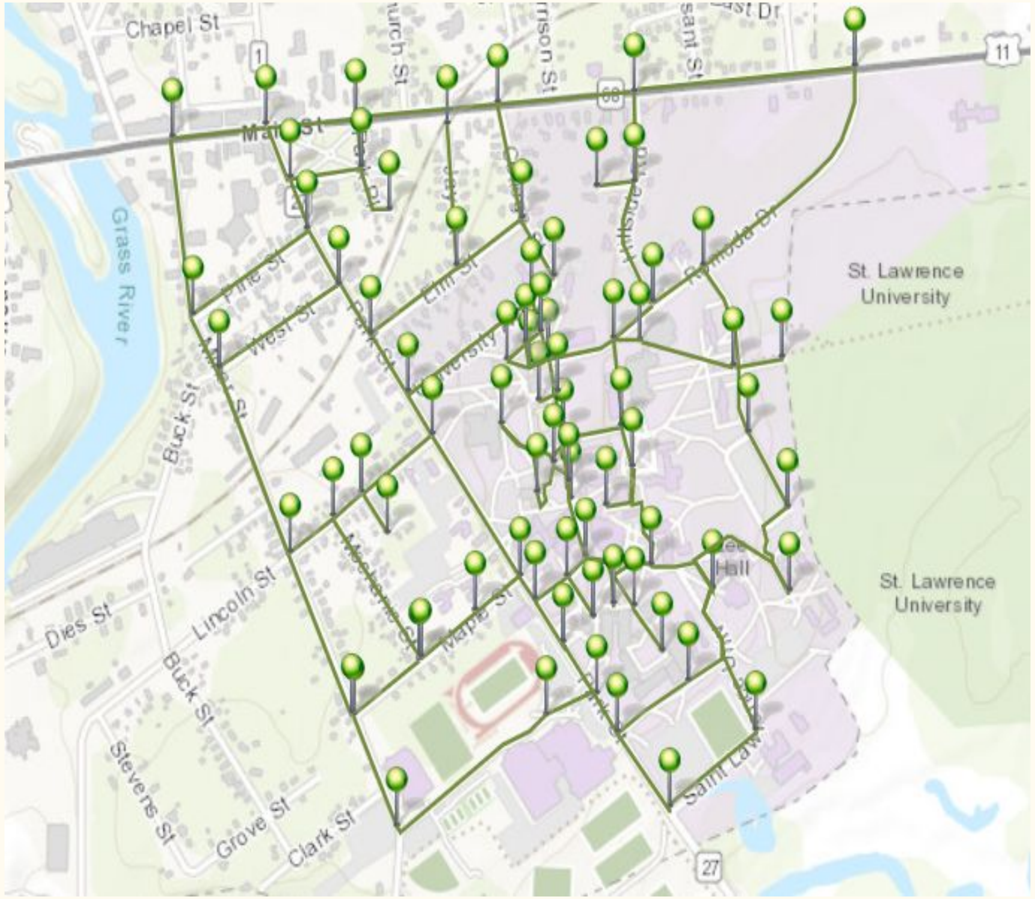
Final Map using ArcGIS

Our mapping includes all relevant walkways on campus, using Main Street and Miner Street as boundaries.

-In total, our final result included:

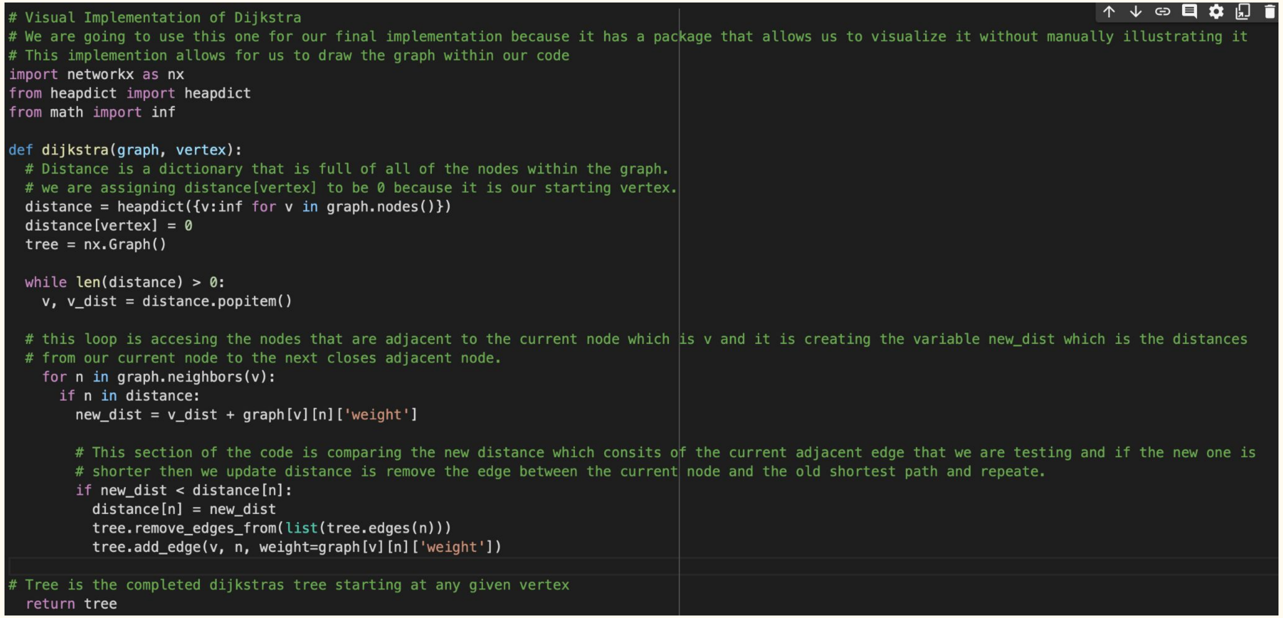
-73 nodes

-93 edges



Final Mapping of our Campus

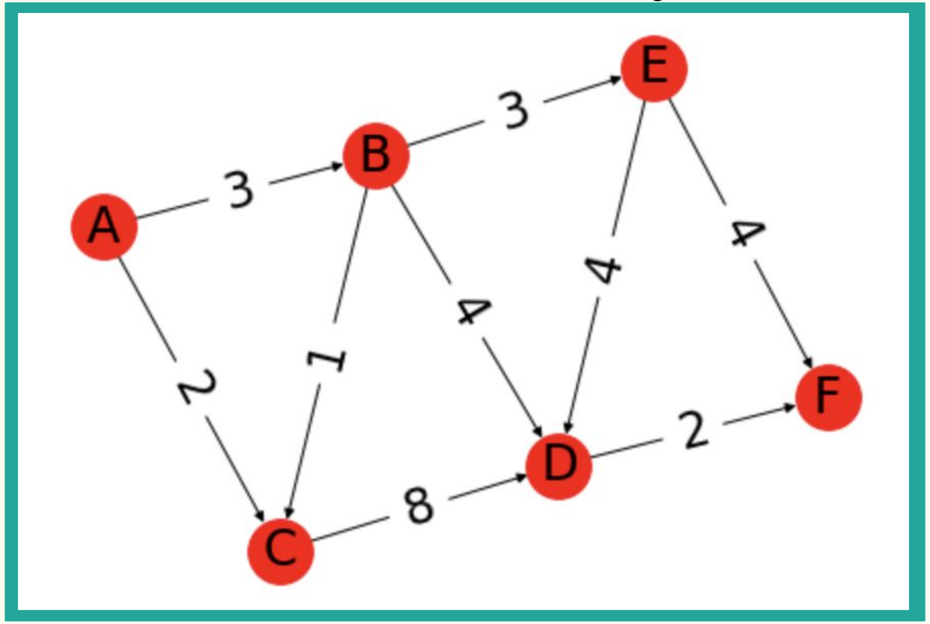
Visual Implementation

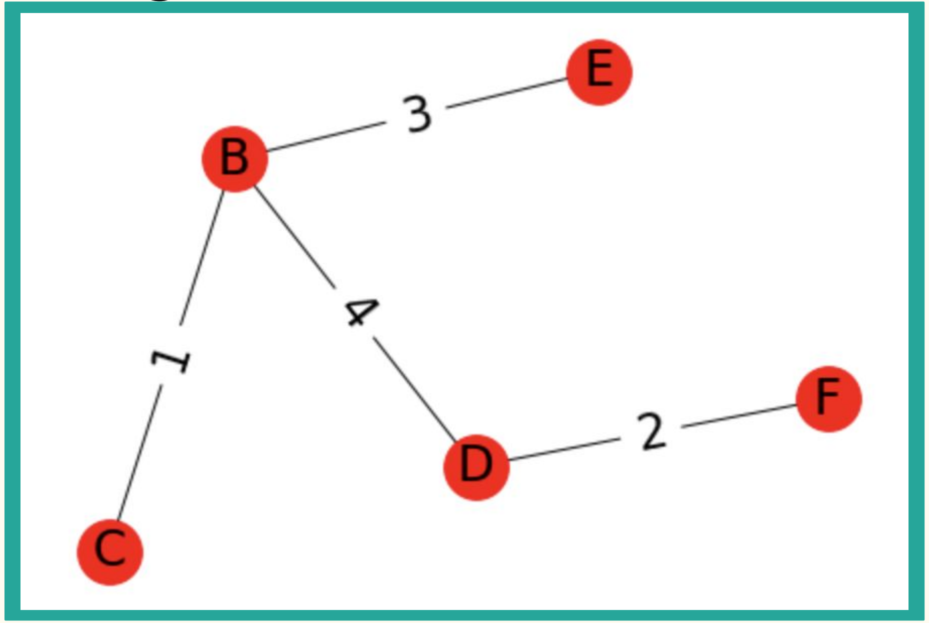


```python
# Visual Implementation of Dijkstra
# We are going to use this one for our final implementation because it has a package that allows us to visualize it without manually illustrating it
# This implemention allows for us to draw the graph within our code
import networkx as nx
from heapdict import heapdict
from math import inf

def dijkstra(graph, vertex):
  # Distance is a dictionary that is full of all of the nodes within the graph.
  # we are assigning distance[vertex] to be 0 because it is our starting vertex.
  distance = heapdict({v:inf for v in graph.nodes()})
  distance[vertex] = 0
  tree = nx.Graph()

  while len(distance) > 0:
    v, v_dist = distance.popitem()

  # this loop is accesing the nodes that are adjacent to the current node which is v and it is creating the variable new_dist which is the distances
  # from our current node to the next closes adjacent node.
    for n in graph.neighbors(v):
      if n in distance:
        new_dist = v_dist + graph[v][n]['weight']

        # This section of the code is comparing the new distance which consits of the current adjacent edge that we are testing and if the new one is
        # shorter then we update distance is remove the edge between the current node and the old shortest path and repeate.
        if new_dist < distance[n]:
          distance[n] = new_dist
          tree.remove_edges_from(list(tree.edges(n)))
          tree.add_edge(v, n, weight=graph[v][n]['weight'])

# Tree is the completed dijkstras tree starting at any given vertex
  return tree
```

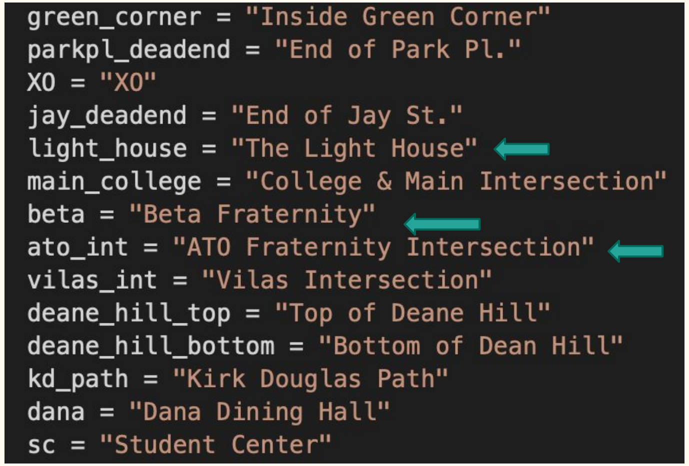
- This implementation performs Dijkstra’s algorithm while creating a dictionary, utilizing a package for the visual component and other built-in functions, such as best_path.

Visualization of Dijkstra Tree Algorithm

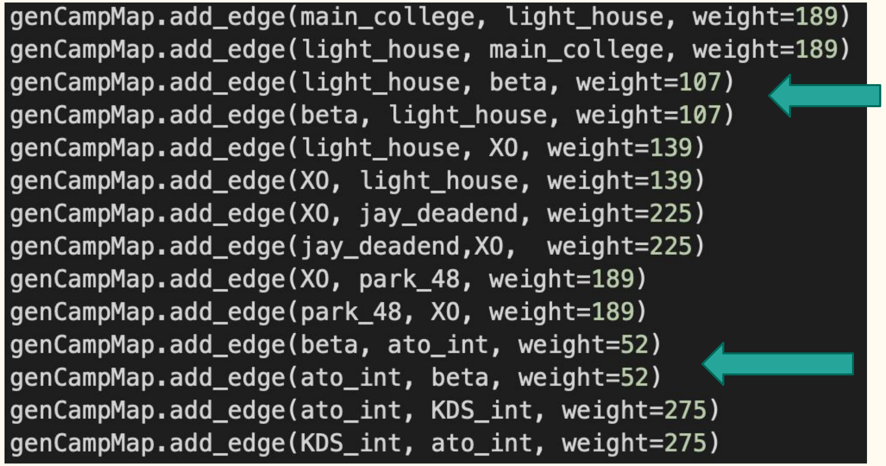




- As you can see, given a root vertex of point B, the result of the Dijkstra tree algorithm returns the shortest path from vertex B to any other vertex in the graph.

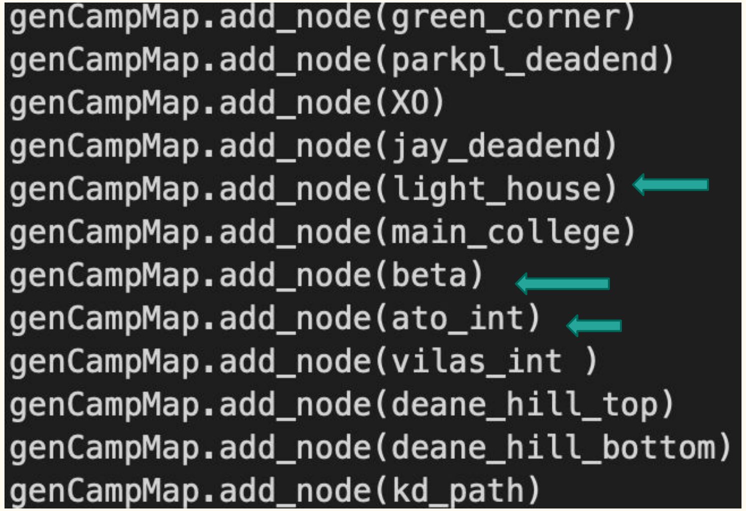


```
green_corner = "Inside Green Corner"
parkpl_deadend = "End of Park Pl."
XO = "XO"
jay_deadend = "End of Jay St."
light_house = "The Light House"
main_college = "College & Main Intersection"
beta = "Beta Fraternity"
ato_int = "ATO Fraternity Intersection"
vilas_int = "Vilas Intersection"
deane_hill_top = "Top of Deane Hill"
deane_hill_bottom = "Bottom of Dean Hill"
kd_path = "Kirk Douglas Path"
dana = "Dana Dining Hall"
sc = "Student Center"
```

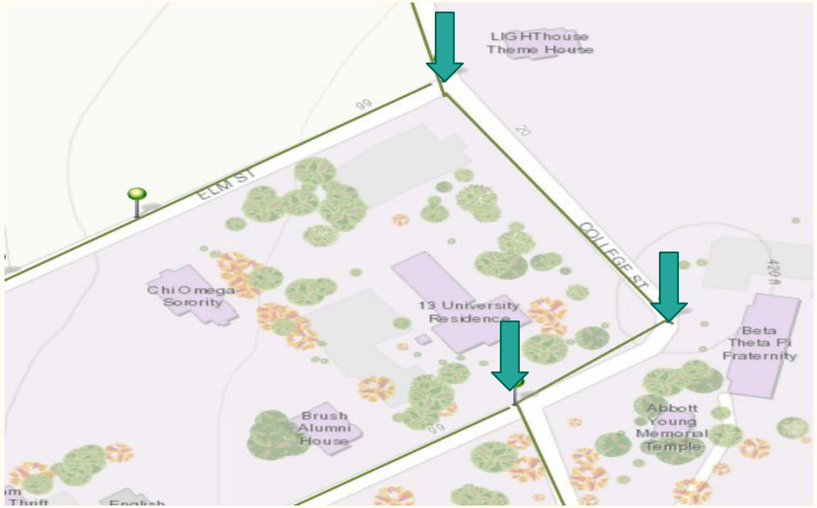


```
genCampMap.add_edge(main_college, light_house, weight=189)
genCampMap.add_edge(light_house, main_college, weight=189)
genCampMap.add_edge(light_house, beta, weight=107)
genCampMap.add_edge(beta, light_house, weight=107)
genCampMap.add_edge(light_house, XO, weight=139)
genCampMap.add_edge(XO, light_house, weight=139)
genCampMap.add_edge(XO, jay_deadend, weight=225)
genCampMap.add_edge(jay_deadend,XO,  weight=225)
genCampMap.add_edge(XO, park_48, weight=189)
genCampMap.add_edge(park_48, XO, weight=189)
genCampMap.add_edge(beta, ato_int, weight=52)
genCampMap.add_edge(ato_int, beta, weight=52)
genCampMap.add_edge(ato_int, KDS_int, weight=275)
genCampMap.add_edge(KDS_int, ato_int, weight=275)
```

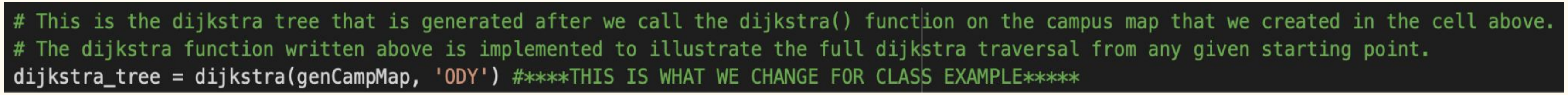
- 2 different Types of structures in the graph: nodes, edges



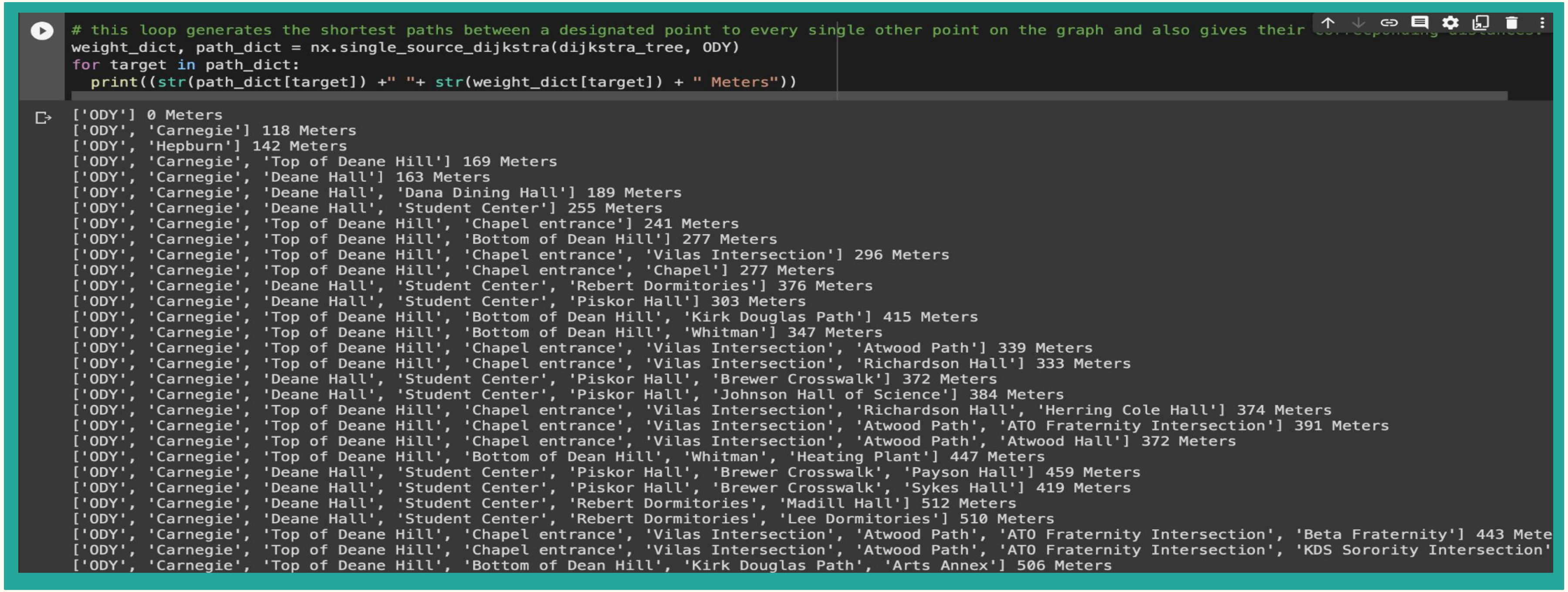
```
genCampMap.add_node(green_corner)
genCampMap.add_node(parkpl_deadend)
genCampMap.add_node(XO)
genCampMap.add_node(jay_deadend)
genCampMap.add_node(light_house)
genCampMap.add_node(main_college)
genCampMap.add_node(beta)
genCampMap.add_node(ato_int)
genCampMap.add_node(vilas_int )
genCampMap.add_node(deane_hill_top)
genCampMap.add_node(deane_hill_bottom)
genCampMap.add_node(kd_path)
```



All Possible Paths with O.D.Y as a Source



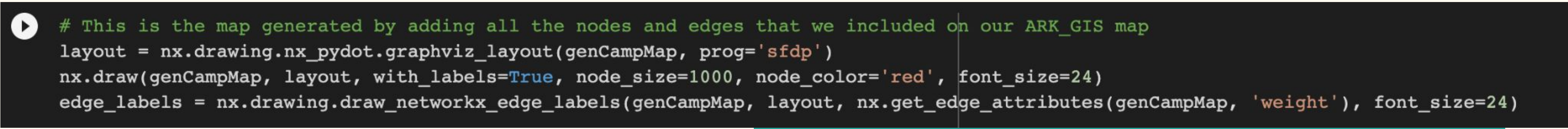
```
# This is the dijkstra tree that is generated after we call the dijkstra() function on the campus map that we created in the cell above.
# The dijkstra function written above is implemented to illustrate the full dijkstra traversal from any given starting point.
dijkstra_tree = dijkstra(genCampMap, 'ODY') #****THIS IS WHAT WE CHANGE FOR CLASS EXAMPLE*****
```



```
# this loop generates the shortest paths between a designated point to every single other point on the graph and also gives their corresponding distances.
weight_dict, path_dict = nx.single_source_dijkstra(dijkstra_tree, ODY)
for target in path_dict:
  print((str(path_dict[target]) +" "+ str(weight_dict[target]) + " Meters"))
```

```
['ODY'] 0 Meters
['ODY', 'Carnegie'] 118 Meters
['ODY', 'Hepburn'] 142 Meters
['ODY', 'Carnegie', 'Top of Deane Hill'] 169 Meters
['ODY', 'Carnegie', 'Deane Hall'] 163 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Dana Dining Hall'] 189 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center'] 255 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance'] 241 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Bottom of Dean Hill'] 277 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Vilas Intersection'] 296 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Chapel'] 277 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Rebert Dormitories'] 376 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Piskor Hall'] 303 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Bottom of Dean Hill', 'Kirk Douglas Path'] 415 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Bottom of Dean Hill', 'Whitman'] 347 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Vilas Intersection', 'Atwood Path'] 339 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Vilas Intersection', 'Richardson Hall'] 333 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Piskor Hall', 'Brewer Crosswalk'] 372 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Piskor Hall', 'Johnson Hall of Science'] 384 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Vilas Intersection', 'Richardson Hall', 'Herring Cole Hall'] 374 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Vilas Intersection', 'Atwood Path', 'ATO Fraternity Intersection'] 391 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Vilas Intersection', 'Atwood Path', 'Atwood Hall'] 372 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Bottom of Dean Hill', 'Whitman', 'Heating Plant'] 447 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Piskor Hall', 'Brewer Crosswalk', 'Payson Hall'] 459 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Piskor Hall', 'Brewer Crosswalk', 'Sykes Hall'] 419 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Rebert Dormitories', 'Madill Hall'] 512 Meters
['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Rebert Dormitories', 'Lee Dormitories'] 510 Meters
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Vilas Intersection', 'Atwood Path', 'ATO Fraternity Intersection', 'Beta Fraternity'] 443 Mete
['ODY', 'Carnegie', 'Top of Deane Hill', 'Chapel entrance', 'Vilas Intersection', 'Atwood Path', 'ATO Fraternity Intersection', 'KDS Sorority Intersection'
['ODY', 'Carnegie', 'Top of Deane Hill', 'Bottom of Dean Hill', 'Kirk Douglas Path', 'Arts Annex'] 506 Meters
```

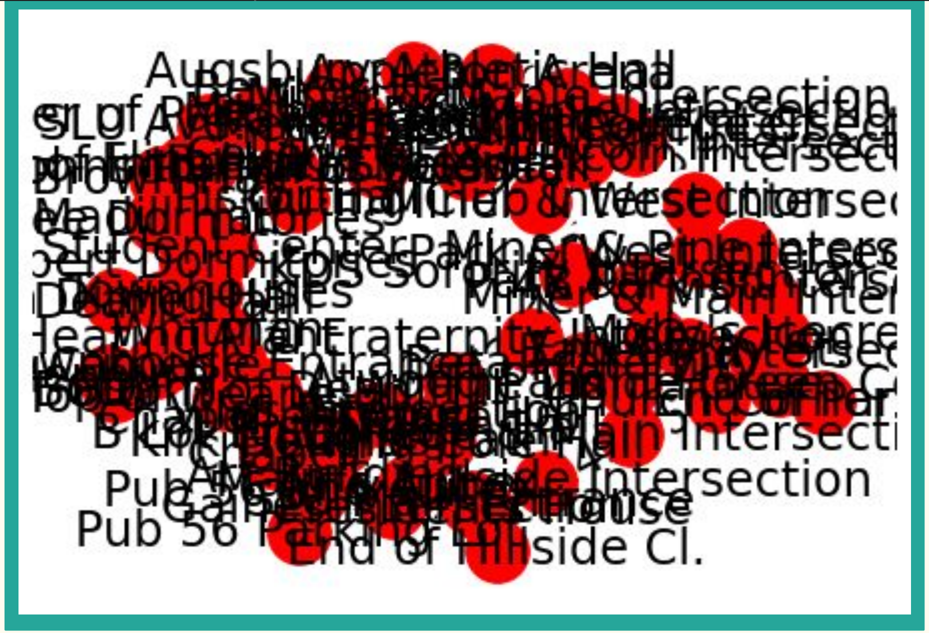
Our First Generated Result:



```
# This is the map generated by adding all the nodes and edges that we included on our ARK_GIS map
layout = nx.drawing.nx_pydot.graphviz_layout(genCampMap, prog='sfdp')
nx.draw(genCampMap, layout, with_labels=True, node_size=1000, node_color='red', font_size=24)
edge_labels = nx.drawing.draw_networkx_edge_labels(genCampMap, layout, nx.get_edge_attributes(genCampMap, 'weight'), font_size=24)
```

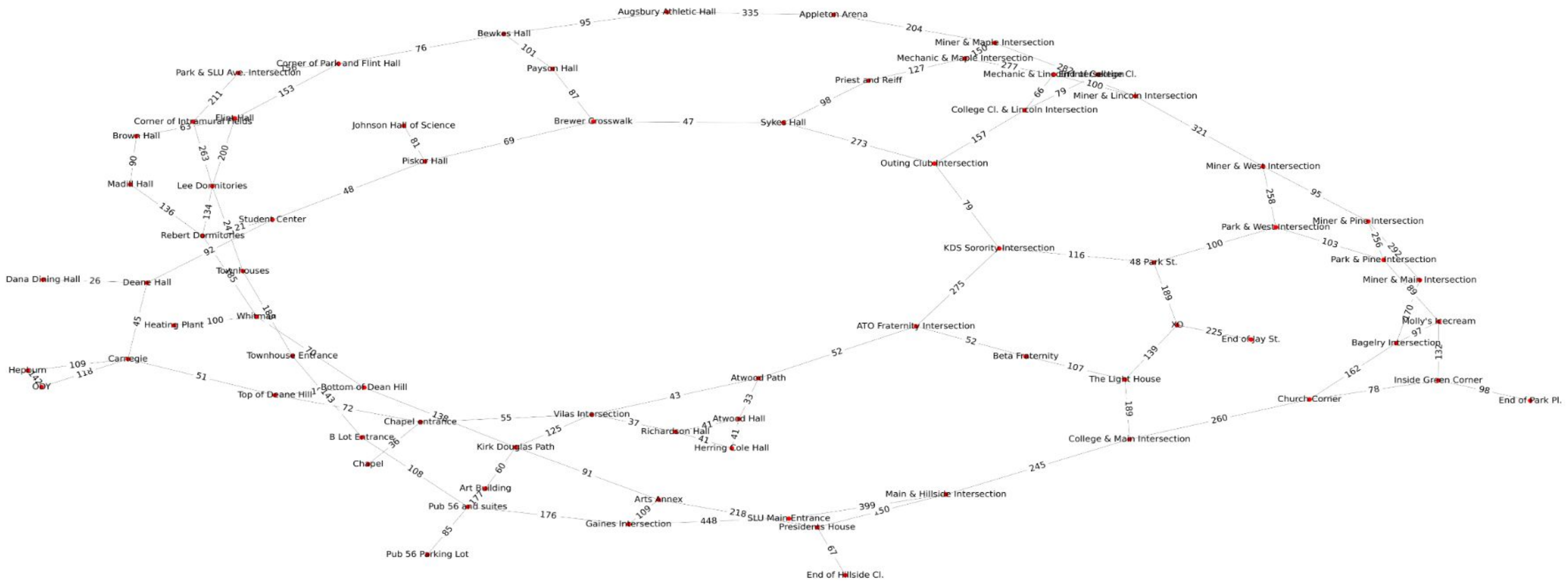
-As seen in our figure to the right, the output from this cell of code is very untidy and not legible as is.

-We needed to take a few extra steps in order to tidy up our visual aid and make our graph legible and useful for our presentation.



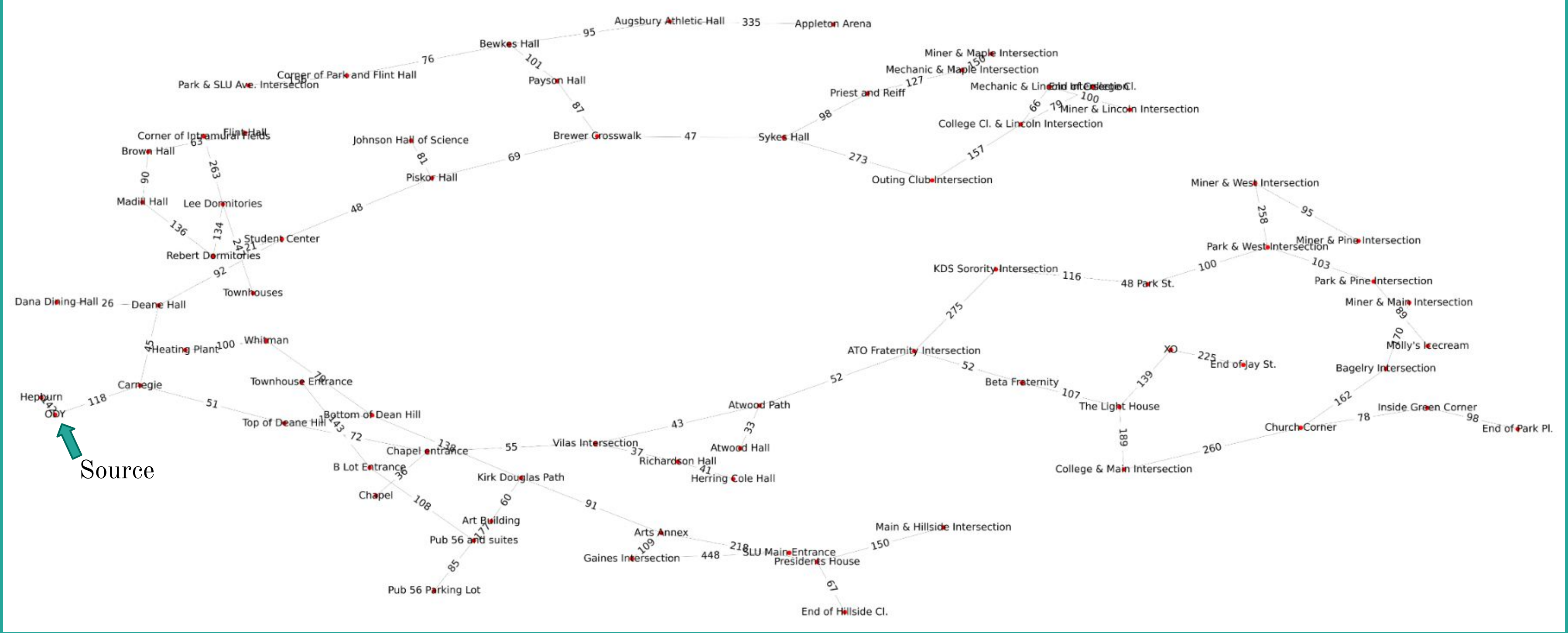
Initial visual aid

General Campus Map - Better Resolution



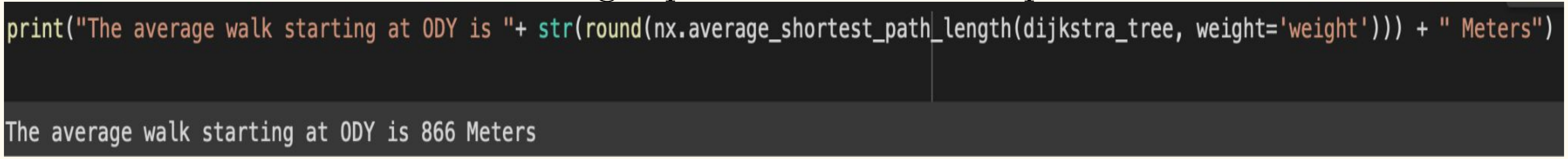
```
dijkstra_tree = dijkstra(genCampMap, 'ODY') #****THIS IS WHAT WE CHANGE FOR CLASS EXAMPLE*****
fig = plt.figure(1,figsize=(200,80),dpi=100)
nx.draw(dijkstra_tree, layout, with_labels=True, node_size=1200, node_color='red', font_size=80)
edge_labels = nx.drawing.draw_networkx_edge_labels(genCampMap, layout, nx.get_edge_attributes(dijkstra_tree, 'weight'), font_size=80)
```

Dijkstra Tree, with Owen D. Young Library as the Root



Our Applications

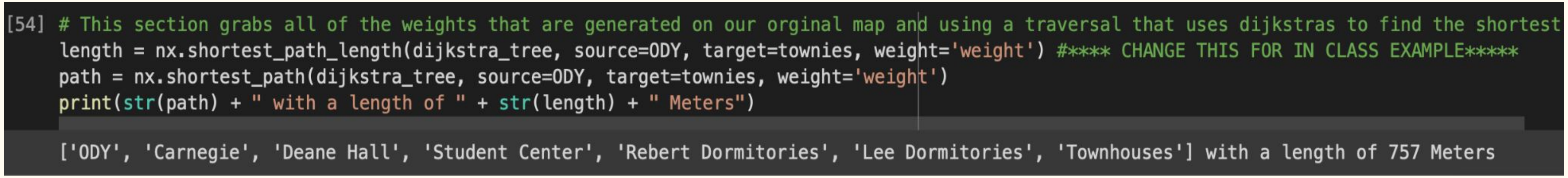
-Using our paths stemming from ODY for example, we can find the average distance of a walk from ODY, to some target point on SLU’s campus.



```
print("The average walk starting at ODY is "+ str(round(nx.average_shortest_path_length(dijkstra_tree, weight='weight'))) + " Meters")
```

```
The average walk starting at ODY is 866 Meters
```

-Using ODY as a starting point we are able to find the optimal path between ODY and any given destination with the total weight it takes to travel between the two points.



```
[54] # This section grabs all of the weights that are generated on our orginal map and using a traversal that uses dijkstras to find the shortest
     length = nx.shortest_path_length(dijkstra_tree, source=ODY, target=townies, weight='weight') #***** CHANGE THIS FOR IN CLASS EXAMPLE*****
     path = nx.shortest_path(dijkstra_tree, source=ODY, target=townies, weight='weight')
     print(str(path) + " with a length of " + str(length) + " Meters")
```

```
     ['ODY', 'Carnegie', 'Deane Hall', 'Student Center', 'Rebert Dormitories', 'Lee Dormitories', 'Townhouses'] with a length of 757 Meters
```

How to Further Develop our Project

-If we were able to keep developing this product, we would attempt to create a more user-friendly and developed interface to use in our applications.

-We would include a more developed map made within ArcGis that would encompass more secluded spots on and around campus.

- Possibly adapt our code to account for large open areas of grass as a possible shortcut, altering the length of our paths. (Think the KD Quad, Intramural Fields, etc)

-Create a more advanced visual that gives total distances from starting node at each edge

-Many more