郭桐維教授

資料結構

作業四

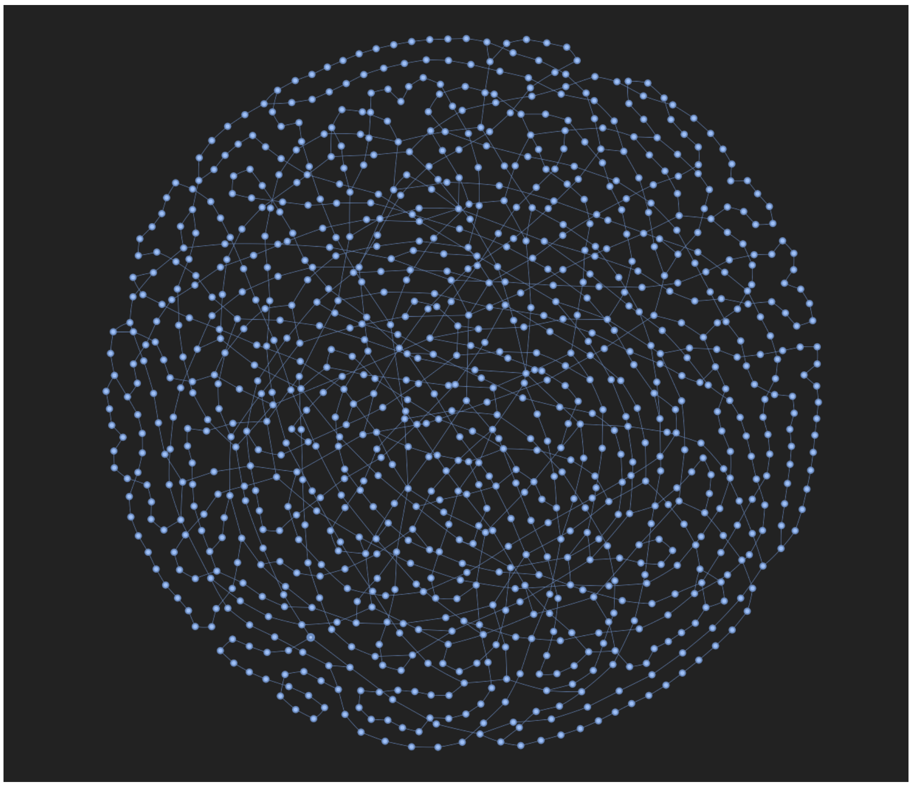
國貿二 郭許謙 110301034

一、實驗結果

實驗簡介

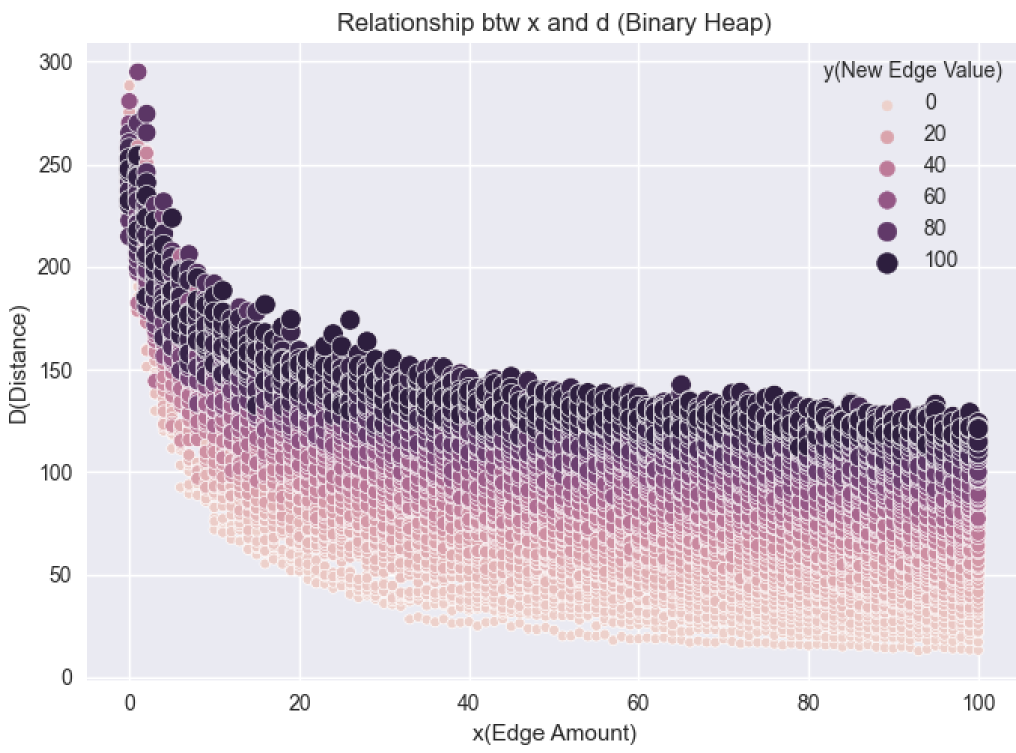
這次的實驗分別使用binary heap 和 Fibonacci heap 來實作 Dijkstra 演算法，並透過以下圖表來探討作業四的題目。

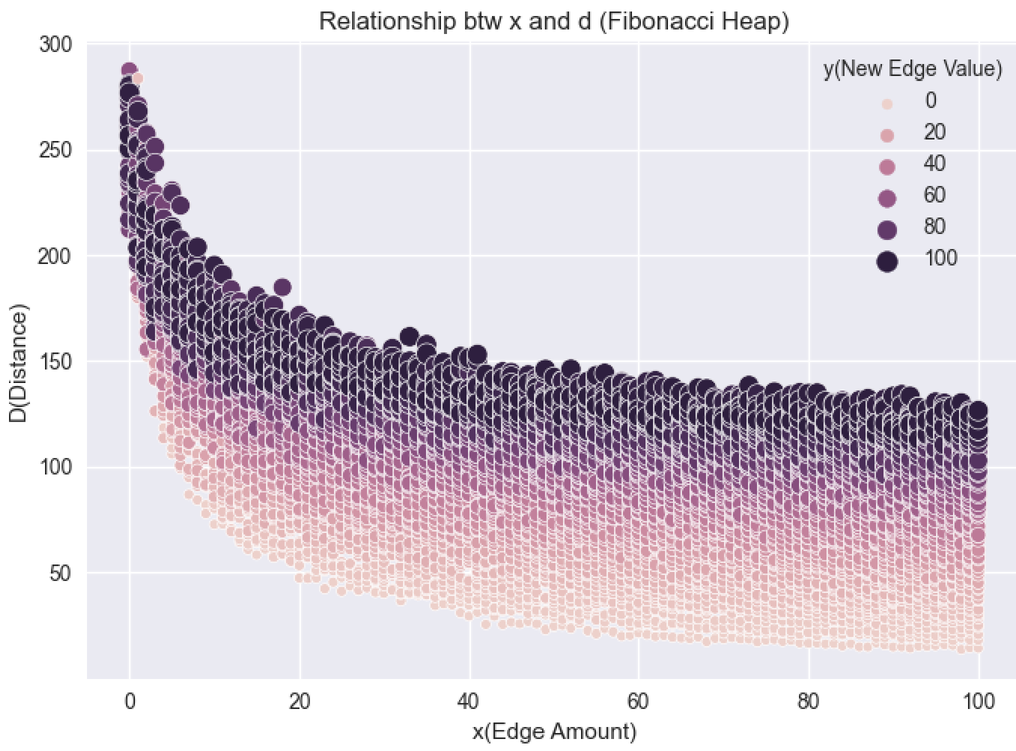
Graph Visualization with x = 100



The Relationship Between x and d

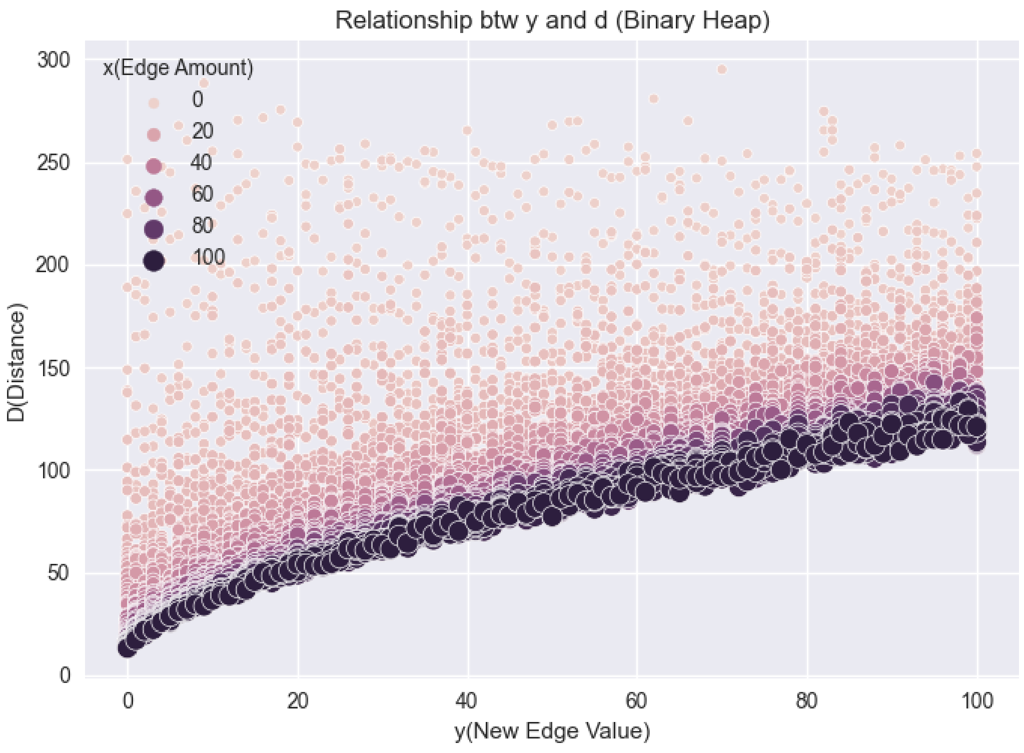
我們可以從上述兩個圖發現，基本上隨著edge　的數量增加，平均最小路徑會隨之減少，並且我們可以發現隨著新增edge 的value 增加，D 的下降趨勢會逐漸趨緩。

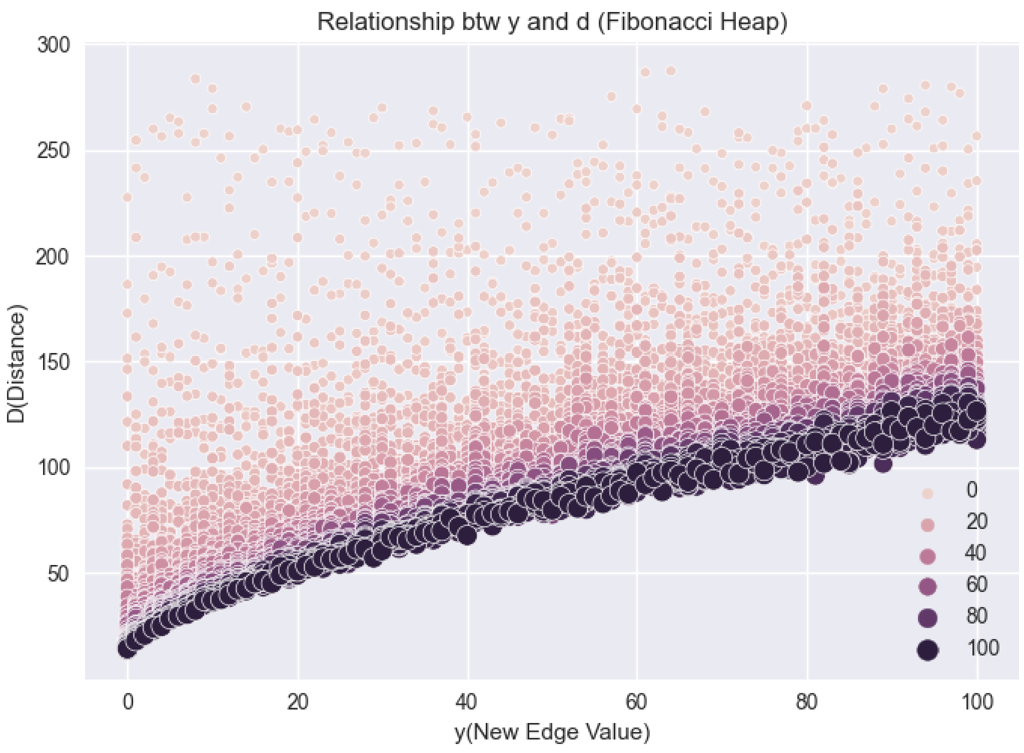

Relationship btw x and d (Binary Heap)
D(Distance)
x(Edge Amount)
y(New Edge Value)
0
20
40
60
80
100
0
50
100
150
200
250
300
0
20
40
60
80
100


Relationship btw x and d (Fibonacci Heap)
D(Distance)
x(Edge Amount)
y(New Edge Value)
0
20
40
60
80
100
50
100
150
200
250
300
0
20
40
60
80
100

The Relationship Between y and d

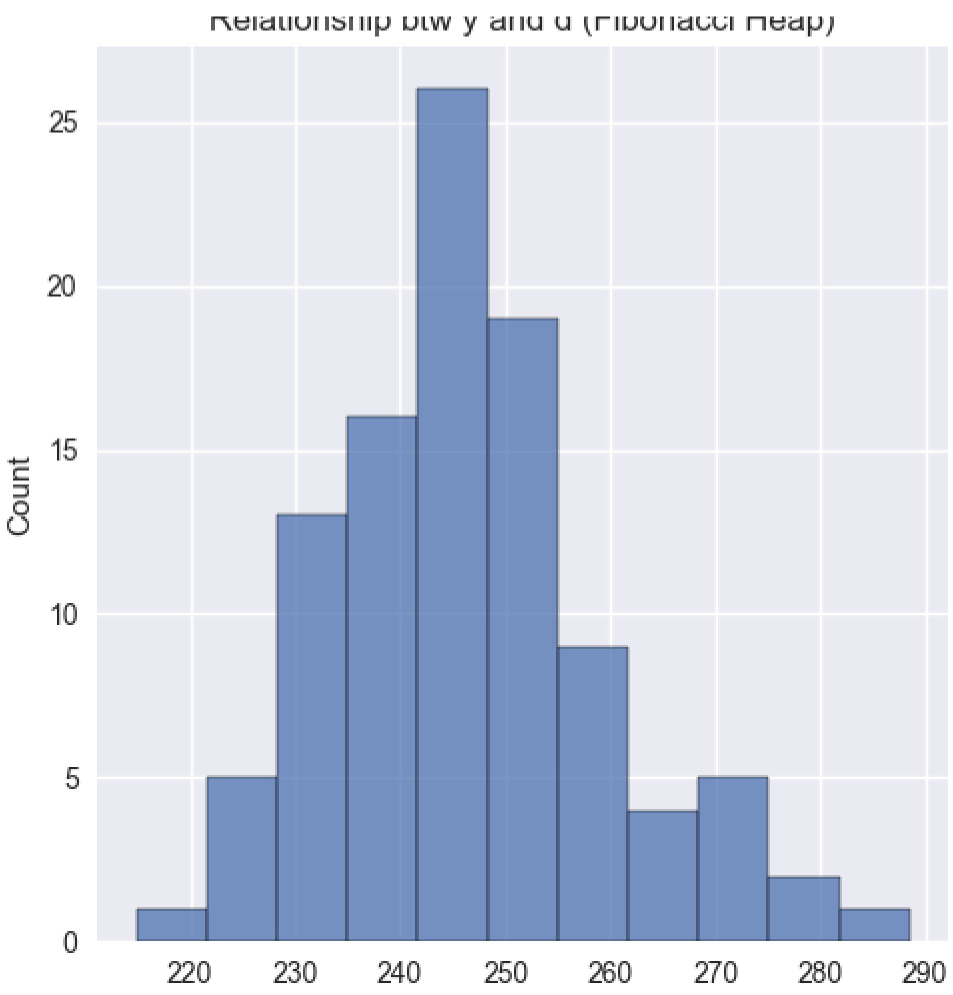
隨著y 增加 d也會增加，不過我們可以從下圖發現當 X很小的時候，y增加對d不會有太大的改變。

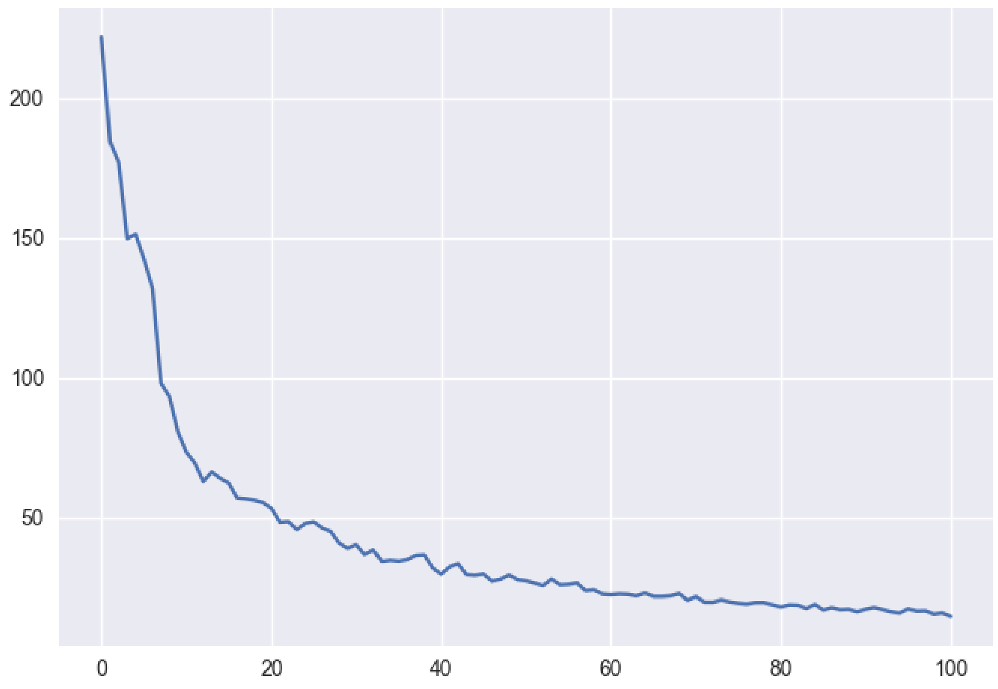




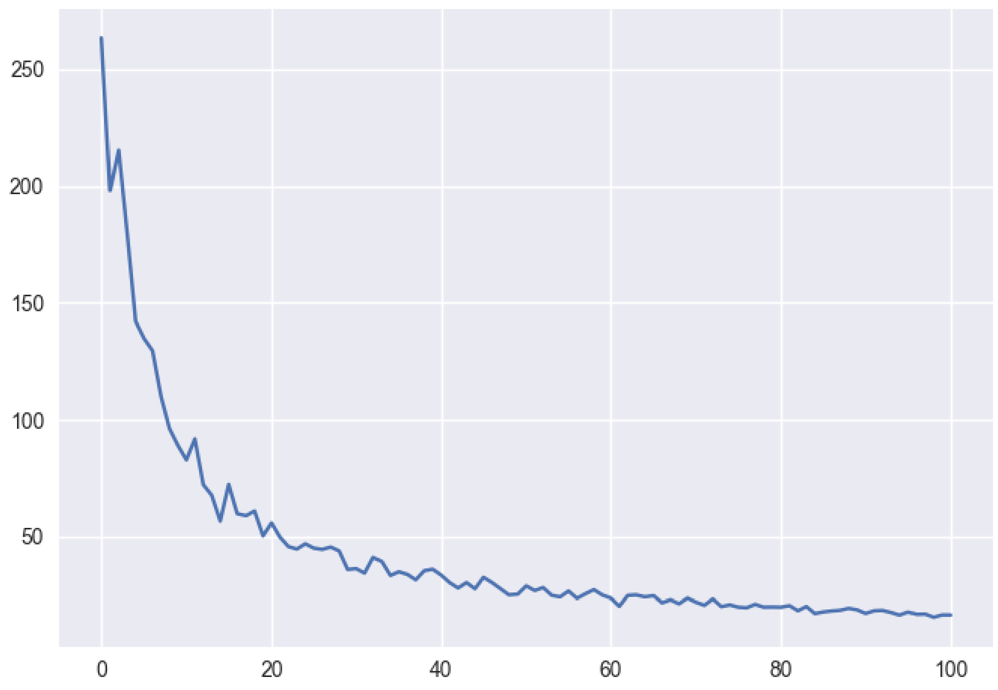
The reason why I choose 100 as z

我選擇設100當z 的原因有二，第一是因為就抽樣來說基本上要30以上才會有效力，而另外一個原因是考量到隨機新增edge會使d 不一定會隨者x 變大而變小，而為了能更精確的呈現我選擇100。





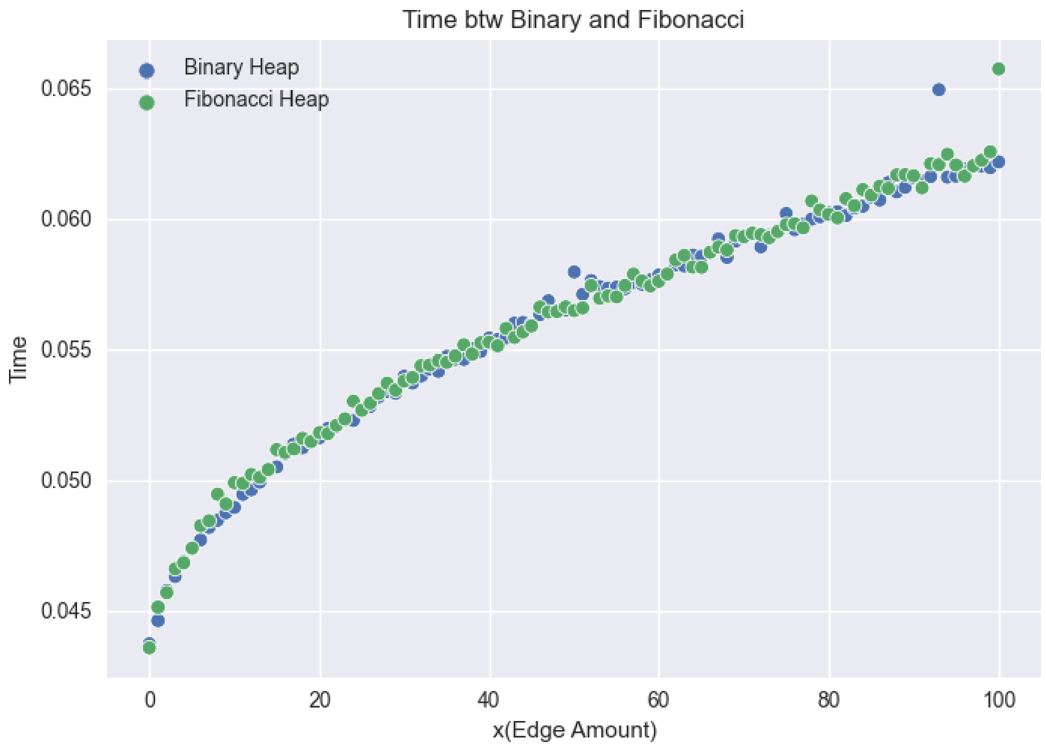
z = 100



z = 50

Implementation Time (Binary vs Fibonacci Heap)

就實驗速度來看的話，從下方圖表可以看出他們並沒有顯著差異。



二、每個資料結構程式碼的來源 / 實作程式碼

```
import heapq
import pyvis
import random
import math
from matplotlib import pyplot as plt
from pyvis.network import Network
from IPython.core.display import display, HTML
import pandas as pd
import time
import fibheap
import seaborn as sns
```

```
"""*************************************************** Import DataSet
*************************************************************************************
*****"""


dis_data_binary = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 4_0117/hw4_data_distance_binary.csv")
dis_data_fib = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 4_0117/hw4_data_distance_fib.csv")
time_data_binary = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 4_0117/hw4_data_time_binary.csv")
time_data_fib = pd.read_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 4_0117/hw4_data_time_fib.csv")






"""*************************************************** Create Graph
(Default)****************************************************************************
**************"""

# Create graph
def make_default_graph():
   graph = dict()
   default_edge_value = 1

   # Create node

   for i in range(0,1000):
       graph[i] = dict()

   # Create edge

   for i in range(0,1000):
       if i == 0 :
           graph[i][i+1] = default_edge_value
           graph[i][999] = default_edge_value
       elif i == 999 :
           graph[i][0] = default_edge_value
           graph[i][i-1] = default_edge_value
       else:
           graph[i][i+1] = default_edge_value
           graph[i][i-1] = default_edge_value
   return graph




"""************************************************** Dijkstra Algorithm (Binary
Heap)*******************************************************************************
**********"""




def dijkstra(graph,node):
   distances={node:float('inf') for node in graph}
   distances[node]=0
   came_from={node:None for node in graph}
   queue=[(0,node)]

   while queue:
       current_distance,current_node=heapq.heappop(queue)
       # relaxation
       for next_node,weight in graph[current_node].items():
           distance_temp=current_distance+weight
           if distance_temp<distances[next_node]:
               distances[next_node]=distance_temp
               came_from[next_node]=current_node
               heapq.heappush(queue,(distance_temp,next_node))
   return distances,#came_from


# Compute average shortest path

def compute_shortest_path(your_graph, startpoint):
   shortest_path = 0
   dijk = dijkstra(your_graph,startpoint)
   for i in list(dijk[0]):
       shortest_path = shortest_path + dijk[0][i]
   return shortest_path

def compute_average_shortest_path(your_graph):
   total_shortest_path = list()
   for i in list(your_graph):
       total_shortest_path.append(compute_shortest_path(your_graph,i) )
```

```
    average = sum(total_shortest_path)/len(total_shortest_path)
    average = round(average, 3)
    return average



def compute_shortest_path_Zsamples(your_graph, z):
    average_shortest_path = list()
    for i in range(0,z):
        dijk = dijkstra(your_graph,random.choice(list(your_graph)))
        average_shortest_path.append(dijk[0][random.choice(list(dijk[0]))])
    average = sum(average_shortest_path)/len(average_shortest_path)
    average = round(average, 3)
    return average










"""************************************************** Dijkstra Algorithm (Fibonacci Heap)******************************************************************************"""




def fibque(fib):
    sorted_queue = list()
    for i in range(fib.num_nodes):
        sorted_queue.append(fibheap.fheappop(fib))
    for i in sorted_queue:
        fibheap.fheappush(fib, i)
    return sorted_queue


def dijkstra_fib(graph,node):
    distances={node:float('inf') for node in graph}
    distances[node]=0
    came_from={node:None for node in graph}
    queue = fibheap.makefheap()
```

```
   fibheap.fheappush(queue,(0,node))
   queue_list = fibque(queue)


   while queue_list:
       current_distance,current_node=fibheap.fheappop(queue)
       queue_list.remove((current_distance,current_node))
       # relaxation
       for next_node,weight in graph[current_node].items():
           distance_temp=current_distance+weight
           if distance_temp<distances[next_node]:
               distances[next_node]=distance_temp
               came_from[next_node]=current_node
               fibheap.fheappush(queue,(distance_temp,next_node))
               queue_list.append((distance_temp,next_node))


   return distances,#came_from


# Compute average shortest path

def compute_shortest_path_fib(your_graph, startpoint):
   shortest_path = 0
   dijk = dijkstra_fib(your_graph,startpoint)
   for i in list(dijk[0]):
       shortest_path = shortest_path + dijk[0][i]
   return shortest_path

def compute_average_shortest_path_fib(your_graph):
   total_shortest_path = list()
   for i in list(your_graph):
       total_shortest_path.append(compute_shortest_path_fib(your_graph,i) )
   average = sum(total_shortest_path)/len(total_shortest_path)
   average = round(average, 3)
   return average



def compute_shortest_path_Zsamples_fib(your_graph, z):
   average_shortest_path = list()
   for i in range(0,z):
       dijk = dijkstra_fib(your_graph,random.choice(list(your_graph)))
       average_shortest_path.append(dijk[0][random.choice(list(dijk[0]))])
   average = sum(average_shortest_path)/len(average_shortest_path)
   average = round(average, 3)
   return average




""""***************************************************** Graph Visualization (Default)
***********************************************************************************
*****"""
# Create network
'''

net = Network(bgcolor="#222222",
   font_color="white",
   height="750px",
   width="100%",
   select_menu=True,
   filter_menu=True)

# Create node

net.add_nodes(list(graph))

# Create edge

for i in list(graph):
   for j in list(graph[i]):
       net.add_edges([(i, j, graph[i][j])])


net.show_buttons(filter_=['physics'])
net.show('default.html')

'''
""""***************************************************** Experiment (Binary Heap)
***********************************************************************************
*****"""

'''

# Graph

graph_test_binary = make_default_graph()
```

```
## parameters

edge_amount_x = 100
add_edge_value_y = 100
sample_z = 100



## Add the random x edge

for j in range(0,add_edge_value_y+1):

   graph_test_binary = make_default_graph()

   for i in range(0,edge_amount_x+1):
       ### Selecting the edge
       if i != 0:
           edge_start = random.randint(0,999)
           edge_end = random.randint(0,999)
           while edge_end in list(graph_test_binary[edge_start]):
               edge_end = random.randint(0,999)

           ### Building the edge

           graph_test_binary[edge_start][edge_end] = j
           graph_test_binary[edge_end][edge_start] = j

       ### Shortest Path
       start = time.time()
       shortest_path = compute_shortest_path_Zsamples(graph_test_binary,sample_z)
       end = time.time()

       ### Save the data
       dis_data_binary.iat[i,j+1] = shortest_path
       time_data_binary.iat[i,j+1] = end - start

       ## Count
       print(j,i)

print(dis_data_binary)
print(time_data_binary)
```

```
x = range(0,101)
y = shortest_path_list_x

plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.plot(x,y)
plt.show()
'''

"""*************************************************** Experiment (Fibonacci Heap)
*********************************************************************************
*****"""

'''

# Graph

graph_test_fib = make_default_graph()


## parameters

edge_amount_x = 100
add_edge_value_y = 100
sample_z = 100



## Add the random x edge

for j in range(0,add_edge_value_y+1):

   graph_test_fib = make_default_graph()

   for i in range(0,edge_amount_x+1):
       ### Selecting the edge
       if i != 0:
           edge_start = random.randint(0,999)
           edge_end = random.randint(0,999)
           while edge_end in list(graph_test_fib[edge_start]):
               edge_end = random.randint(0,999)

           ### Building the edge
```

```
            graph_test_fib[edge_start][edge_end] = j
            graph_test_fib[edge_end][edge_start] = j

        ### Shortest Path
        start = time.time()
        shortest_path = compute_shortest_path_Zsamples(graph_test_fib,sample_z)
        end = time.time()

        ### Save the data
        dis_data_fib.iat[i,j+1] = shortest_path
        time_data_fib.iat[i,j+1] = end - start

        ## Count
        print(j,i)

print(dis_data_fib)
print(time_data_fib)




x = range(0,101)
y = shortest_path_list_x

plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.plot(x,y)
plt.show()

'''




"""***************************************************** Save the Experiment data
***********************************************************************************
*****"""

'''

dis_data_binary.to_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 4_0117/hw4_data_distance_binary.csv",index=False)
```

```
dis_data_fib.to_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
4_0117/hw4_data_distance_fib.csv",index=False)
time_data_binary.to_csv("/Users/stevenkuo/College Course/大二/Data
Structure/Assignment 4_0117/hw4_data_time_binary.csv",index=False)
time_data_fib.to_csv("/Users/stevenkuo/College Course/大二/Data Structure/Assignment
4_0117/hw4_data_time_fib.csv",index=False)

'''

"""***************************************************** Experiment Visualization
(Data)
*********************************************************************************
*****"""

### Data (Binary & Fib)

datavis_new_edge_value = list()
datavis_edgeamount = list()
datavis_time = list()
datavis_distance = list()
datavis_heap = list()


for i in range(0,101):
   for j in range(1,102):
       datavis_distance.append(dis_data_binary.iat[i,j])
       datavis_time.append(time_data_binary.iat[i,j])
       datavis_edgeamount.append(i)
       datavis_new_edge_value.append(j-1)
       datavis_heap.append("Binary Heap")

for i in range(0,101):
   for j in range(1,102):
       datavis_distance.append(dis_data_fib.iat[i,j])
       datavis_time.append(time_data_fib.iat[i,j])
       datavis_edgeamount.append(i)
       datavis_new_edge_value.append(j-1)
       datavis_heap.append("Fibonacci Heap")


b_f_dict = {"x(Edge Amount)":datavis_edgeamount,"y(New Edge
Value)":datavis_new_edge_value,"Heap":datavis_heap,"Time":datavis_time,"D(Distance)
":datavis_distance}
b_f_df = pd.DataFrame(b_f_dict)
```

```
##Filter

filter1 = b_f_df["Heap"]== "Binary Heap"
filter2 = b_f_df["Heap"]== "Fibonacci Heap"

df = b_f_df[(filter1 & filter2)]




### Relationship btw x and d (Binary Heap)


plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")

sns.scatterplot(x="x(Edge Amount)", y="D(Distance)", hue= "y(New Edge Value)",
size="y(New Edge Value)", data=b_f_df[filter1])
plt.title("Relationship btw x and d (Binary Heap)")
plt.show()




### Relationship btw x and d (Fibonacci Heap)


plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")

sns.scatterplot(x="x(Edge Amount)", y="D(Distance)", hue= "y(New Edge Value)",
size="y(New Edge Value)", data=b_f_df[filter2])
plt.title("Relationship btw x and d (Fibonacci Heap)")
plt.show()



### Relationship btw y and d (Binary Heap)


plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")

sns.scatterplot(x="y(New Edge Value)", y="D(Distance)", hue= "x(Edge Amount)",
size="x(Edge Amount)", data=b_f_df[filter1])
plt.title("Relationship btw y and d (Binary Heap)")
plt.show()




### Relationship btw y and d (Fibonacci Heap)




sns.scatterplot(x="y(New Edge Value)", y="D(Distance)", hue= "x(Edge Amount)",
size="x(Edge Amount)", data=b_f_df[filter2])
plt.title("Relationship btw y and d (Fibonacci Heap)")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()


### how to choose z

filter3 = b_f_df["x(Edge Amount)"]== 0

df_z = b_f_df[(filter1 & filter3)]
z_list = list()

for i in range(0,101):
   z_list.append(df_z["D(Distance)"][i])




sns.displot(z_list)
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.title("Relationship btw y and d (Fibonacci Heap)")
plt.show()
```

```
## time between binary and fib

filter5 = b_f_df["y(New Edge Value)"] == 50

fast_df = b_f_df[(filter5)]


sns.scatterplot(x="x(Edge Amount)", y="Time", hue="Heap" , data=fast_df)
plt.title("Time btw Binary and Fibonacci")
plt.style.use("seaborn")
plt.legend()
plt.ticklabel_format(style="plain")
plt.show()





"""************************************************** Experiment Visualization
(Graph)
***********************************************************************************
*****"""

# Visualization

'''

# Create network

net_test_1 = Network(bgcolor="#222222",
   font_color="white",
   height="750px",
   width="100%",
   select_menu=True,
   filter_menu=True)

# Create node

net_test_1.add_nodes(list(graph_test_1))

# Create edge

for i in list(graph_test_1):
   for j in list(graph_test_1[i]):
        net_test_1.add_edges([(i, j, graph_test_1[i][j])])



net_test_1.show_buttons(filter_=['physics'])
net_test_1.show('demo.html')


'''
```

三、心得、疑問、遇到的困難

　　這學期的資料結構課程感覺一下子就過了，很謝謝老師的細心指導，我也從作業中學到很多，從獨立解決問題，到我在作業四的時候慢慢可以很自然地用OOP的概念coding 作為一個非本科系的學生，我真的覺得這門是我這學期收穫最多的一堂課。