Minimum Spanning Trees Problem - Writeup

For this problem, I use **adjacency matrix** to represent the graph, because
1. The number of edge = (n, 2), which is the maximum possible amount, so the memory space O(n^2) won't be wasted.
2. We can take advantage of O(1) for looking up edges in adjacency matrix instead of O(n) for adjacency list.

I use **Prim's algorithm** to find the Minimun Spannig Tree.

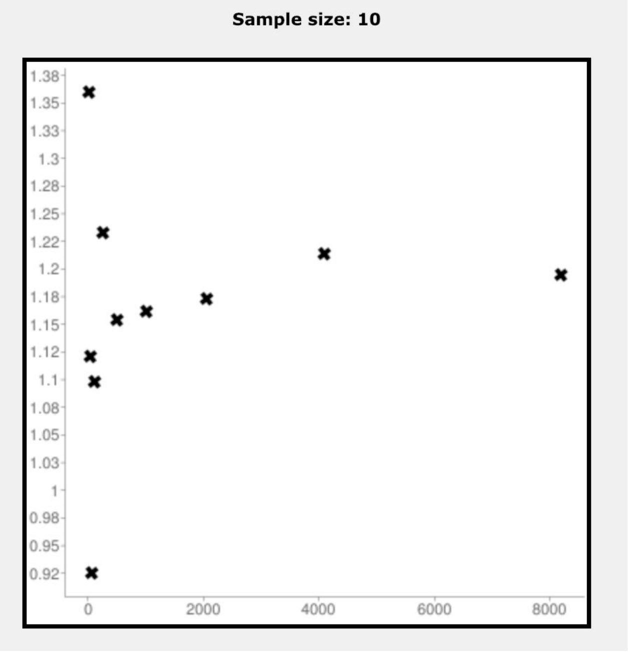
Here is the average weight results for Type 1 and Type 2

n	**Trials**	**Type 1**	**Type 2**
16	5	1.3597730275336273	2.5085712436649876
32	5	1.1212376000587858	3.372591797539171
64	5	0.9254493022136657	4.494382429976256
128	5	1.0982575870490734	6.845114514464262
256	5	1.2330741769387092	8.685414682025163
512	5	1.1535733874020297	12.980927639877535
1024	5	1.161729320662434	18.25250461819802
2048	5	1.1727194705397217	26.01205605537134
4096	5	1.213575229626299	35.903402738555265
8192	5	1.1947567104112742	51.803734145937185

Growth rate f(n) estimation:

- **Type1** :

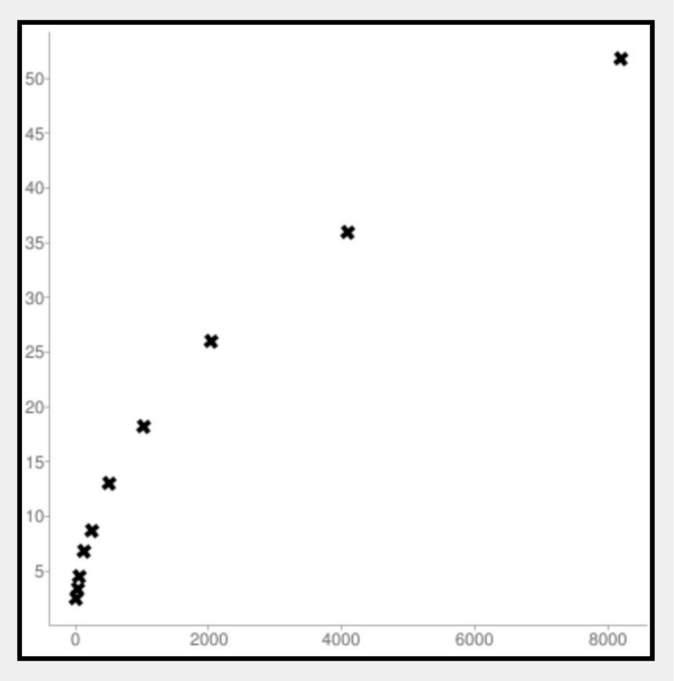
The average weight of MST fluctuate around and gradually approaches to 1.2, and there isn't any obvious relationship between number of nodes and weights. S.t. I estimate f(n) = 1.2



The growth rate seems reasonable as the egedes are randomly generated, and directly from [0,1]

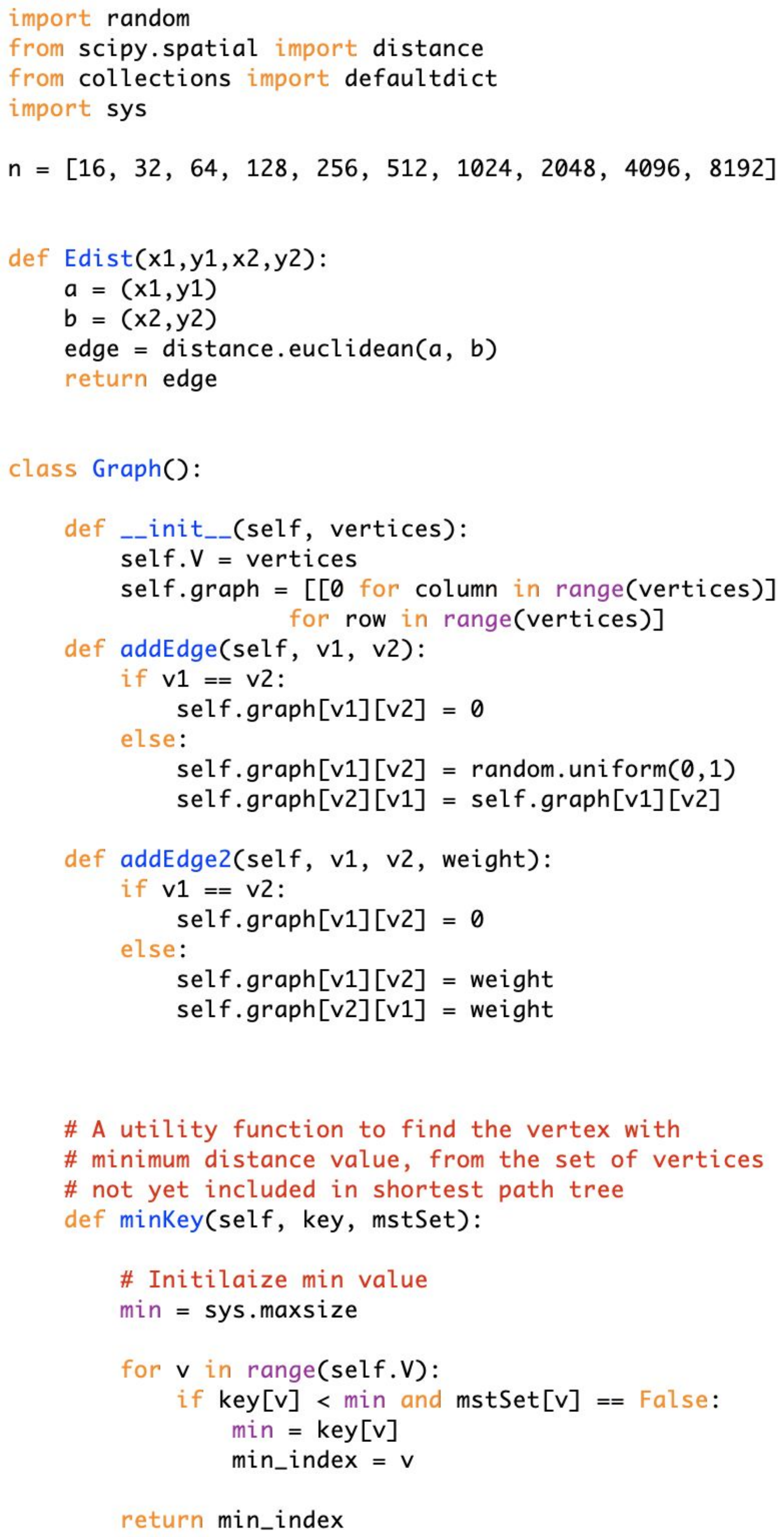
- **Type2:**

It could be seen clearly on the graph that as n increases, the average weight also increased exponentially. I estimate f(n) = 0.65n^0.5



The growth rate seems reasonable as its might related to how the edges are generated from x-axis and y-axis which create dimension of 2. Euclidean distance is computed by square rooting (x square + y square), and that might affect the result that average weight is growing exponentially.

Here is my code written in **Python** :



```python
import random
from scipy.spatial import distance
from collections import defaultdict
import sys

n = [16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192]


def Edist(x1,y1,x2,y2):
    a = (x1,y1)
    b = (x2,y2)
    edge = distance.euclidean(a, b)
    return edge


class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                    for row in range(vertices)]
    def addEdge(self, v1, v2):
        if v1 == v2:
            self.graph[v1][v2] = 0
        else:
            self.graph[v1][v2] = random.uniform(0,1)
            self.graph[v2][v1] = self.graph[v1][v2]

    def addEdge2(self, v1, v2, weight):
        if v1 == v2:
            self.graph[v1][v2] = 0
        else:
            self.graph[v1][v2] = weight
            self.graph[v2][v1] = weight



    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minKey(self, key, mstSet):

        # Initilaize min value
        min = sys.maxsize

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

        return min_index
    def primMST(self):
        key = [sys.maxsize] * self.V
        parent = [None] * self.V
        key[0] = 0
        mstSet = [False] * self.V

        parent[0] = -1

        for cout in range(self.V):
            u = self.minKey(key, mstSet)
            mstSet[u] = True
            for v in range(self.V):
                if self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.graph[u][v]:
                        key[v] = self.graph[u][v]
                        parent[v] = u
        totalweight = 0
        numnode = 0
        for i in range(1,self.V):
            totalweight +=self.graph[i][ parent[i] ]
        return totalweight

weightlist =[]
weightlist2 =[]
def main():
    print("Average weight list result for TYPE 1")
    for z in n:
        g = Graph(z)
        for i in range(z):
            for j in range(z):
                g.addEdge(i, j)
        weights = 0
        for i in range(5):
            weights += g.primMST()
        weightlist.append(weights / 5)
    print(weightlist)
    print("Average weight list result for TYPE 2")
    for z in n:
        g = Graph(z)
        for i in range(z):
            for j in range(z):
                x = [random.uniform(0,1),random.uniform(0,1)]
                y = [random.uniform(0,1),random.uniform(0,1)]
                weight = Edist(x[0],y[0],x[1],y[1])
                g.addEdge2(i,j,weight)
        weights = 0
        for i in range(5):
            weights += g.primMST()
        aveweight= weights / 5
        weightlist2.append(aveweight)

    print(weightlist2)

if __name__ == '__main__':
   main()
```