

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**204 DATA STRUCTURES (3+1)**

**2021–2022 FALL SEMESTER**

**PROJECT-4 REPORT**

**(GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS)**

**DELIVERY DATE**

22/01/2022

**PREPARED BY**

05200000742 , Mehmet ÖKTEN

05180000063 , Burak KIZILAY

İçindekiler

[1.a AVL ağacına istenilen değerlerin eklenerek oluşan yeni AVL ağaçlarının çizimleri 3](#_Toc93185210)

[1.b Heapa istenilen değerin eklendikten ve silme işlemi yapıldıktan sonra heapin çizimi 3](#_Toc93185211)

[2.a B-Tree ekleme method (ya da AVL-Tree ekleme method/Red-Black tree/Huffman encoding tree) kodu 3](#_Toc93185212)

[2.b B-Tree ekleme kodunun adım adım açıklanması 3](#_Toc93185213)

[3.a Dijkstra kodu ve yapılan testler 3](#_Toc93185214)

[3.b Prim MST kodu ve yapılan testler 3](#_Toc93185215)

[3.c BFT ya da DFT kodu ve yapılan testler 3](#_Toc93185216)

[3.d Verilen Big-O tablosunun doldurulmuş hali 3](#_Toc93185217)

[4.a Verilen çizgenin python ortamında grafiksel olarak oluşturan kod 3](#_Toc93185218)

[4.b Verilen çizgede en kısa yol bulma sonuçları 3](#_Toc93185219)

[5.a. Prim ve Kruskal algoritması karşılaştırması 3](#_Toc93185220)

[5.b Verilen çizgede en kısa yol bulma sonuçları 3](#_Toc93185221)

[Özdeğerlendirme Tablosu 4](#_Toc93185222)

GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS

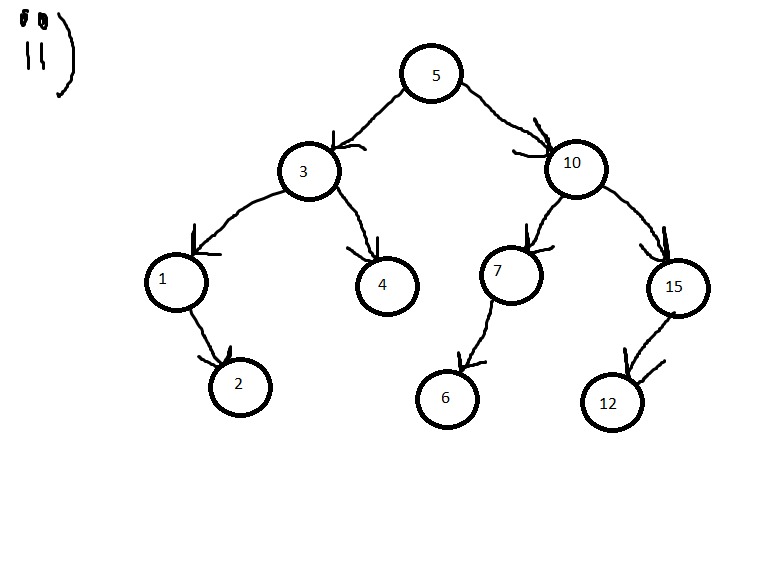
## 1.a AVL ağacına istenilen değerlerin eklenerek oluşan yeni AVL ağaçlarının çizimleri

FIGURE 1

( İ )

## 

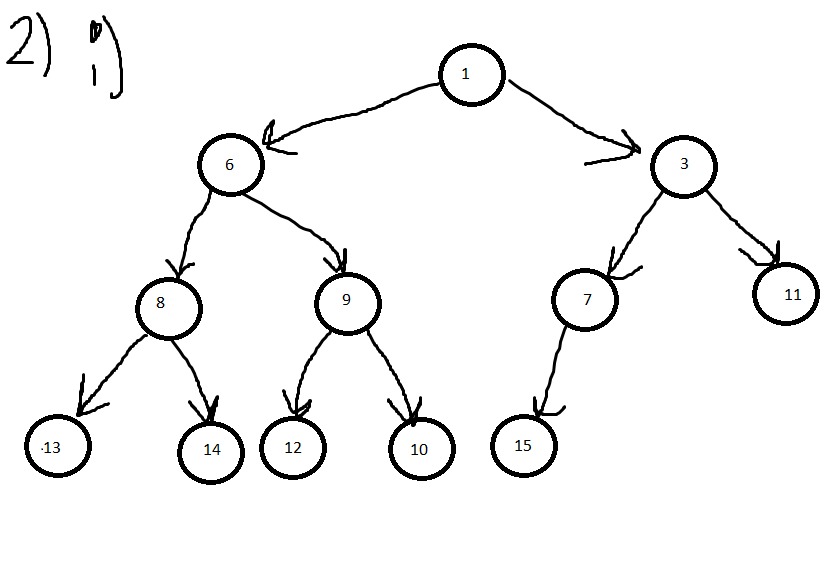
( İİ )



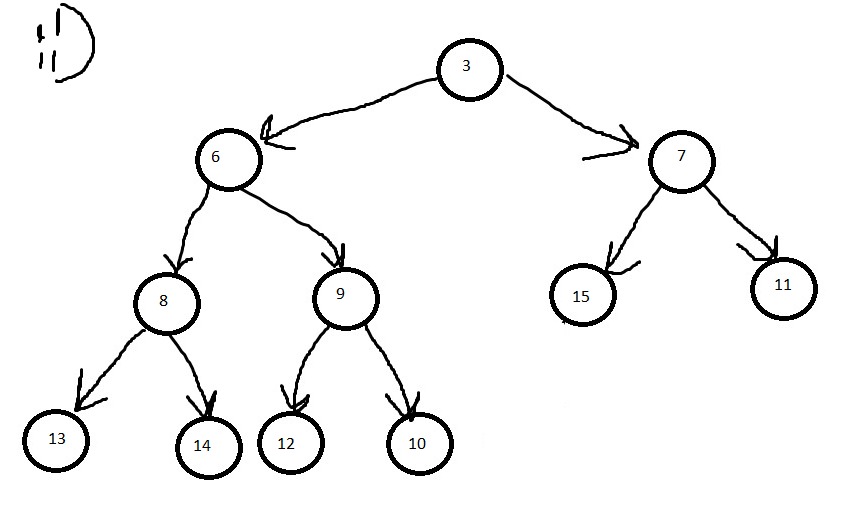
## 1.b Heapa istenilen değerin eklendikten ve silme işlemi yapıldıktan sonra heapin çizimi

FIGURE 2

( İİİ )



( İV )



## 2.a B-Tree ekleme method (ya da AVL-Tree ekleme method/Red-Black tree/Huffman encoding tree) kodu

## ----------------------------------------------------

## 2.b B-Tree ekleme kodunun adım adım açıklanması

## ----------------------------------------------------

## 3.a Dijkstra kodu ve yapılan testler

using System;

namespace Dijktras\_shortest\_path

{

class Program

{

static int inf = 1000;

static void Main(string[] args)

{

int[,] cost = new int[5, 5] {

{ 1000, 5, 3, 1000, 2},

{ 1000, 1000, 2, 6, 1000},

{ 1000, 1, 1000, 2, 1000},

{ 1000, 1000, 1000, 1000, 1000},

{ 1000, 6, 10, 4, 8} };

int[] minDistance = new int[5];

int sehirSayısı = 5;

MinDistance(sehirSayısı, cost, minDistance);

for (int i = 0; i < minDistance.Length; ++i)

{

Console.Write(minDistance[i]);

if (i != (minDistance.Length - 1))

{

Console.Write(", ");

}

}

}

public static void MinDistance(int N, int[,] cost, int[] D)

{

int w, v, min;

bool[] visited = new bool[N];

D[0] = 0;

for (v = 1; v < N; v++)

{

visited[v] = false;

D[v] = cost[0, v];

}

for (int i = 1; i < N; ++i)

{

min = inf;

for (w = 1; w < N; w++)

if (!visited[w])

if (D[w] < min)

{

v = w;

min = D[w];

}

visited[v] = true;

for (w = 1; w < N; w++)

if (!visited[w])

if (min + cost[v, w] < D[w])

D[w] = min + cost[v, w];

}

}

}

}



## 3.b Prim MST kodu ve yapılan testler

using System;

using System.Collections.Generic;

namespace minimum\_spanning\_tree

{

class Program

{

static void Main(string[] args)

{

int[,] çizgeler = new int[7, 7] {

{ 1000, 5, 4, 1000, 23, 95,15},

{ 5 , 1000, 22, 6, 1000, 7,3},

{ 4 , 22, 1000, 2, 1000, 100,9},

{ 1000, 6, 2, 1000, 1000,1000,1000},

{ 23 , 1000, 1000, 1000, 1000,1000,1000},

{ 95 , 7, 100, 1000, 1000,1000,1 },

{ 15 , 3, 9, 1000, 1000,1,1000} };

bool[] gezildiMi = new bool[7];

for (int i = 1; i < 7; i++)

{

gezildiMi[i] = false;

}

gezildiMi[0] = true;

List<int> gezilenVertexler = new List<int>();

gezilenVertexler.Add(0);

for (int m = 0; m < 7; m++)

{

int enKısa = 1000;

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 7; j++)

{

if (gezildiMi[i] == true && gezildiMi[j] == false)

{

if (enKısa > çizgeler[i, j])

{

enKısa = çizgeler[i, j];

}

}

}

}

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 7; j++)

{

if (gezildiMi[i] == true && gezildiMi[j] == false)

{

if (enKısa == çizgeler[i, j])

{

gezilenVertexler.Add(j);

gezildiMi[j] = true;

}

}

}

}

}

for (int i = 0; i < 7; i++)

{

Console.Write(gezilenVertexler[i] + ", ");

}

}

}

## }

## 

## 3.c BFT ya da DFT kodu ve yapılan testler

using System;

using System.Collections.Generic;

namespace Breadth\_First\_Traverse

{

class Program

{

static void Main(string[] args)

{

int[,] çizgeler = new int[7, 7] {

{ 1000, 5, 4, 1000, 23, 95,15},

{ 5 , 1000, 22, 6, 1000, 7,3},

{ 4 , 22, 1000, 2, 1000, 100,9},

{ 1000, 6, 2, 1000, 1000,1000,1000},

{ 23 , 1000, 1000, 1000, 1000,1000,1000},

{ 95 , 7, 100, 1000, 1000,1000,1 },

{ 15 , 3, 9, 1000, 1000,1,1000} };

BFS(çizgeler, 0);

}

public static void BFS(int[,] uzaklıklarMatrisi, int başlancakNode)

{

List<int> stackim = new List<int>();

bool[] gezildiMi = new bool[7];

stackim.Add(başlancakNode);

gezildiMi[başlancakNode] = true;

for (int i = 0; i < 7; i++)

{

for (int j = 0; j < 7; j++)

{

if (uzaklıklarMatrisi[stackim[i], j] < 1000 && gezildiMi[j] == false)

{

stackim.Add(j);

gezildiMi[j] = true;

}

}

}

for (int i = 0; i < stackim.Count; i++)

{

Console.Write(stackim[i]);

if (i != stackim.Count - 1)

{

Console.Write(", ");

}

}

}

}

}

## 

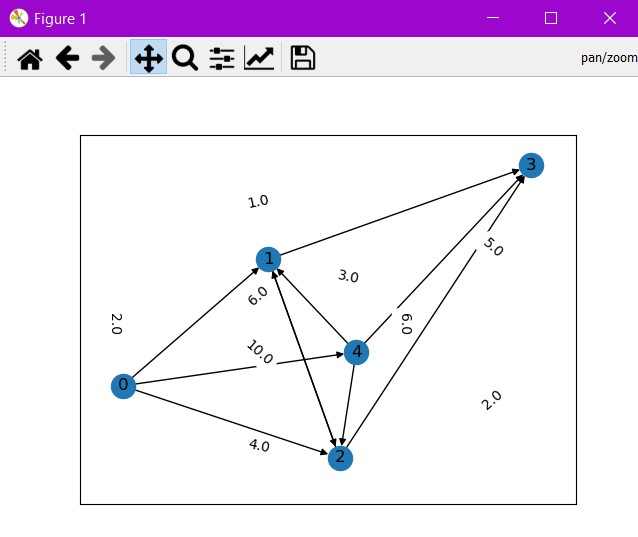
## 3.d Verilen Big-O tablosunun doldurulmuş hali

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Dijkstra’s SP** | Prim’s MST | BFT | **Heap Insertion** |
| **Big-O** (Zaman Karmaşıklığı  Big-O Notasyonuna Göre) | O(V2) or O(V + Elog(V)) | O(V2) or O((V+E)log(V)) | O(V+E) | O(log(n)) |

## 4.a Verilen çizgenin python ortamında grafiksel olarak oluşturan kod

import networkx as nx  
import matplotlib.pyplot as plt

G1=nx.DiGraph()  
G1.add\_node(0,pos=(0,0))  
G1.add\_node(1,pos=(2,-1))  
G1.add\_node(2,pos=(1,-2))  
G1.add\_node(3,pos=(-1,-2))  
G1.add\_node(4,pos=(-2,-1))  
  
G1.add\_edge(0,4, weight = 2.0)  
G1.add\_edge(0,2, weight = 3.0)  
G1.add\_edge(0,1, weight = 5.0)  
G1.add\_edge(1,2, weight = 2.0)  
G1.add\_edge(1,3, weight = 6.0)  
G1.add\_edge(2,1, weight = 1.0)  
G1.add\_edge(2,3, weight = 2.0)  
G1.add\_edge(4,1, weight = 6.0)  
G1.add\_edge(4,2, weight = 10.0)  
G1.add\_edge(4,3, weight = 4.0)  
Pos = nx.circular\_layout(G1)  
nx.draw\_networkx(G1, with\_labels=True)  
labels = nx.get\_edge\_attributes(G1,'weight')  
nx.draw\_networkx\_edge\_labels(G1,Pos,edge\_labels=labels)  
plt.show()

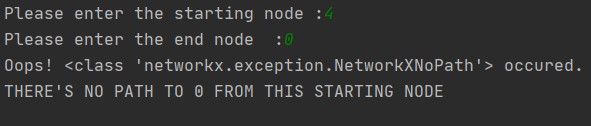
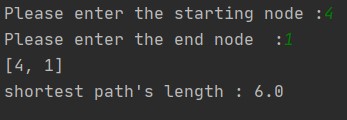


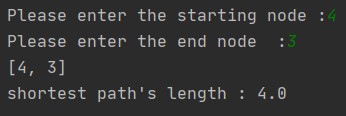
## 4.b Verilen çizgede en kısa yol bulma sonuçları

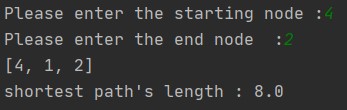
· İİ ŞIKKI

import networkx as nx  
import networkx.exception  
import matplotlib.pyplot as plt  
  
x = int(input("Please enter the starting node :"))  
y = int(input("Please enter the end node :"))  
  
G1=nx.DiGraph()  
G1.add\_node(0,pos=(0,0))  
G1.add\_node(1,pos=(2,-1))  
G1.add\_node(2,pos=(1,-2))  
G1.add\_node(3,pos=(-1,-2))  
G1.add\_node(4,pos=(-2,-1))  
  
G1.add\_edge(0,4, weight = 2.0)  
G1.add\_edge(0,2, weight = 3.0)  
G1.add\_edge(0,1, weight = 5.0)  
G1.add\_edge(1,2, weight = 2.0)  
G1.add\_edge(1,3, weight = 6.0)  
G1.add\_edge(2,1, weight = 1.0)  
G1.add\_edge(2,3, weight = 2.0)  
G1.add\_edge(4,1, weight = 6.0)  
G1.add\_edge(4,2, weight = 10.0)  
G1.add\_edge(4,3, weight = 4.0)  
Pos = nx.circular\_layout(G1)  
nx.draw\_networkx(G1, with\_labels=True)  
labels = nx.get\_edge\_attributes(G1,'weight')  
nx.draw\_networkx\_edge\_labels(G1,Pos,edge\_labels=labels)  
  
try:  
 f = (nx.dijkstra\_path(G1, x, y))  
 print(f)  
 if len(f) == 2:  
 path1 = labels[f[0], f[1]]  
 print("shortest path's length :", path1)  
 else:  
 path1 = labels[f[0], f[1]]  
 path2 = labels[f[1], f[2]]  
 print("shortest path's length :", path1 + path2)  
  
except:  
 print("Oops!", networkx.exception.NetworkXNoPath, "occured.")  
 print("THERE'S NO PATH TO" , y, "FROM THIS STARTING NODE")  
 print()  
  
plt.show()

**NOT : Try-Except kullanımımızın sebebi 0 node’unda hata almamak. Aynı zamanda kullanıcı girdili olarak yazdık ki kullanıcı sadece 4 node’unu değil diğer tüm nodeları da rahatlıkla elde edebilsin.**

ÇIKTI 1ÇIKTI 2

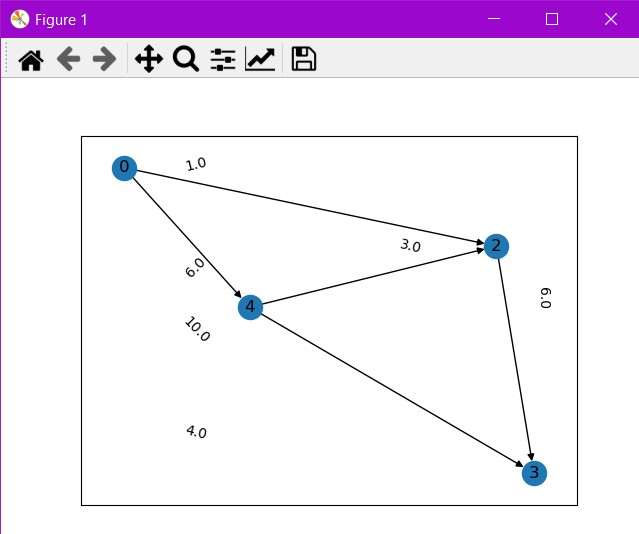
ÇIKTI 3

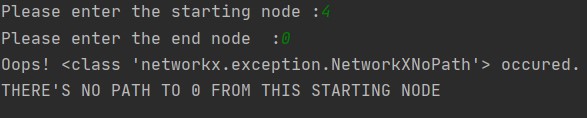
ÇIKTI 4

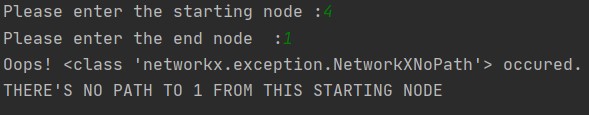
------------------------------------------------------------------------------

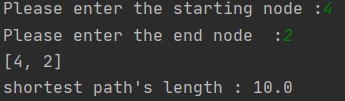
· İİİ ŞIKKI

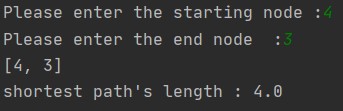
import networkx as nx  
import networkx.exception  
import matplotlib.pyplot as plt  
  
x = int(input("Please enter the starting node :"))  
y = int(input("Please enter the end node :"))  
  
G1=nx.DiGraph()  
G1.add\_node(0,pos=(0,0))  
G1.add\_node(1,pos=(2,-1))  
G1.add\_node(2,pos=(1,-2))  
G1.add\_node(3,pos=(-1,-2))  
G1.add\_node(4,pos=(-2,-1))  
  
G1.add\_edge(0,4, weight = 2.0)  
G1.add\_edge(0,2, weight = 3.0)  
G1.add\_edge(0,1, weight = 5.0)  
G1.add\_edge(1,2, weight = 2.0)  
G1.add\_edge(1,3, weight = 6.0)  
G1.add\_edge(2,1, weight = 1.0)  
G1.add\_edge(2,3, weight = 2.0)  
G1.add\_edge(4,1, weight = 6.0)  
G1.add\_edge(4,2, weight = 10.0)  
G1.add\_edge(4,3, weight = 4.0)  
Pos = nx.circular\_layout(G1)  
labels = nx.get\_edge\_attributes(G1,'weight')  
nx.draw\_networkx\_edge\_labels(G1,Pos,edge\_labels=labels)  
G1.remove\_node(1)  
nx.draw\_networkx(G1, with\_labels=True)  
  
try:  
 f = (nx.dijkstra\_path(G1, x, y))  
 print(f)  
 if len(f) == 2:  
 path1 = labels[f[0], f[1]]  
 print("shortest path's length :", path1)  
 else:  
 path1 = labels[f[0], f[1]]  
 path2 = labels[f[1], f[2]]  
 print("shortest path's length :", path1 + path2)  
  
except:  
 print("Oops!", networkx.exception.NetworkXNoPath, "occured.")  
 print("THERE'S NO PATH TO" , y, "FROM THIS STARTING NODE")  
 print()  
  
plt.show()

 NODE 1 SİLİNİNCE ÇİZGE

 ÇIKTI 1

ÇIKTI 2

ÇIKTI 3

ÇIKTI 4

## 5.a. Prim ve Kruskal algoritması karşılaştırması

## 

Prim algoritmasında yayılmacı bir aç gözlü tutum vardır. En başta rastgele bir düğüm seçilir. Ve sonrasında en kısa kenarı seçilip onun gittiği düğüm de alınır. Sonra iki düğüme bakılır… Bu şekilde bağlantılı düğümlerde çok iyi çalışır.

Ama Kruskal algoritmasında en başta en düşük kenar seçilir. Ve bu şekilde döngü oluşturamayacak şekilde kalanlar arasından en düşük kenar şeklinde devam edeceği için bağlantısız şekillerde de çalışabilir.

## 5.b Verilen çizgede en kısa yol bulma sonuçları

// verilen terimlerden seçtiğiniz 5 tanesini burada açıklayınız.

## 2-3-4 Tree

2-3-4 Tree’ler, B Tree’lerin özel bir halidir. Yani 1 veri 2 çocuk, 2 veri 3 çocuk ya da 3 veri 4 çocuk olabilir. Red Black Tree’lerin eş şekillisi olarak da düşünülebilirler. Sürekli olarak dengeli bir ağaç yapısı sunarlar. Silme ve ekleme işlemleri o yüzden biraz zaman alır. Ama arama açısından hız kazandıran bir yapıdır.

## Quadratic Probing

Quadratic Probing, Hash table’larda verilen key’lere yapılan işlemler sonucu aynı register’a kayır edilemeyeceği için çıkmış bir yöntemdir. Bu yöntmede belli bir polinomik denklem vardır. Ve eğer gelinen register doluysa bu polinom sonucunun cevabı ardışık şekilde eklenerek başka register’a gidilir. Örneğin polinomumuz x2 ise ilk başta register doluysa 1 eklenir, sonra gittiğimiz de doluysa 4 eklenir, orası da doluysa 9 eklenir… şeklinde ilerleyerek çalışır.

## Warshall’s Algorithm

Warshall’s Algorithm, dinamik şekilde çalışan algoritmalara örnektir. Her seferinde uzaklıklar tablosu üzerinde küçük küçük düzeltmeler yaparak en sonunda tüm düğümler için tüm düğümlere en kısa yolları gözlerimizin önüne serer.

## Topological Sorting

Topological Sorting, sadece hangi düğümden başlarsanız başlayın tekrar aynı düğüme gelemediğiniz graphlar için kullanılan bir yöntemdir. Her düğüm ilk geçildiğinde ona bir görülme zamanı atanır. Bir de tüm alt düğümleri bitirdiğinde bir zaman atanır. İşte bu bitirme zamanı en yüksek olan topolojide en başta olan olacaktır Bu şekilde sıralanırlar.

## Dynamic Programming

Dynamic Programming yöntemi problemin tamamını çözmej için onu küçük küçük adımlara böler ve bu her küçük adımı sadece tek bir sefer yapar. En sonunda tüm küçük adımlar bittiğinde büyük problem çözülmüş olur. Warshalls’s Algorithm da bir Dynamic Programming örneğidir.

## Özdeğerlendirme Tablosu

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Points** | **Estimated Grade** | **Explanation** |
| **1 a) AVL Tree** | **10** |  |  |
| **1 b) Heap** | **10** |  |  |
| **2) B-Tree Insertion / AVL Tree Insertion / Red-Black Trees / Huffman Encoding Tree** | **10** |  |  |
| **3 a) Dijkstra’s shortest path code + test** | **5** |  |  |
| **3 b) Prim’s MST code + test** | **5** |  |  |
| **3 c) BFT or DFT code + test** | **5** |  |  |
| **3 d) Filling Big-O Table** | **5** |  |  |
| **4 i) Graph Drawing** | **5** |  |  |
| **4 ii) Finding Shortest Paths with Dijkstra’s** | **5** |  |  |
| **4 iii) Node deletion and repeating i, ii.** | **5** |  |  |
| **5 a) Comparison (Prim’s & Kruskal’s Algorithm)** | **5** |  |  |
| **5 b) Explanations of 4 terms** | **20** |  |  |
| **Demo Video for Source Codes and Tests of Q2 Q3a, Q3b and Q3c.** | **5** |  |  |
| **Self-assessment Table** | **5** |  |  |
| **Total** | **100** |  |  |
|  |  |  |  |