EGE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

204 DATA STRUCTURES (3+1)

2020–2021 FALL SEMESTER

PROJECT-4 REPORT

(GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS)

DELIVERY DATE

16/02/2021

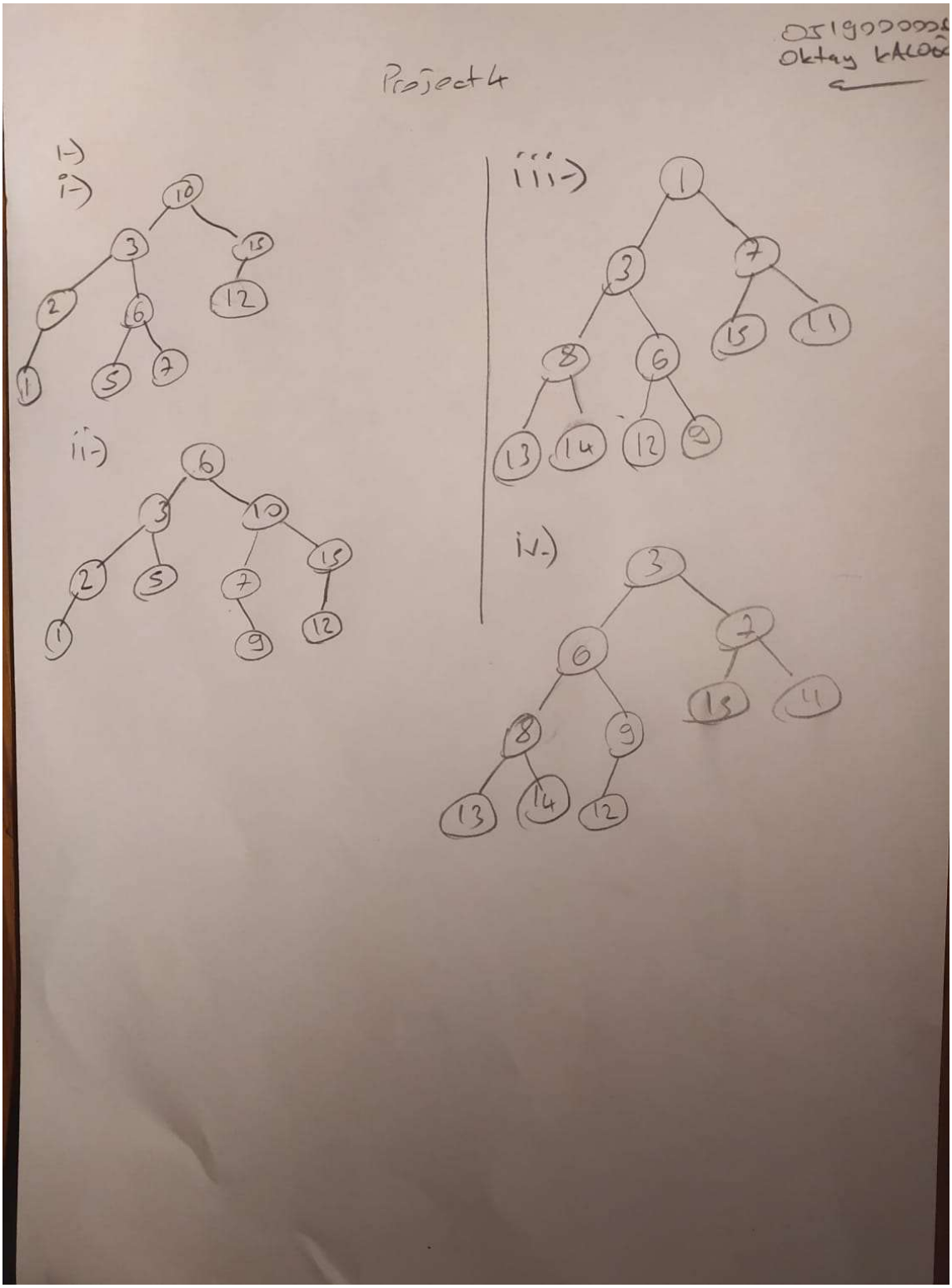
PREPARED BY

05190000061, Oktay Kaloğlu

İçindekiler

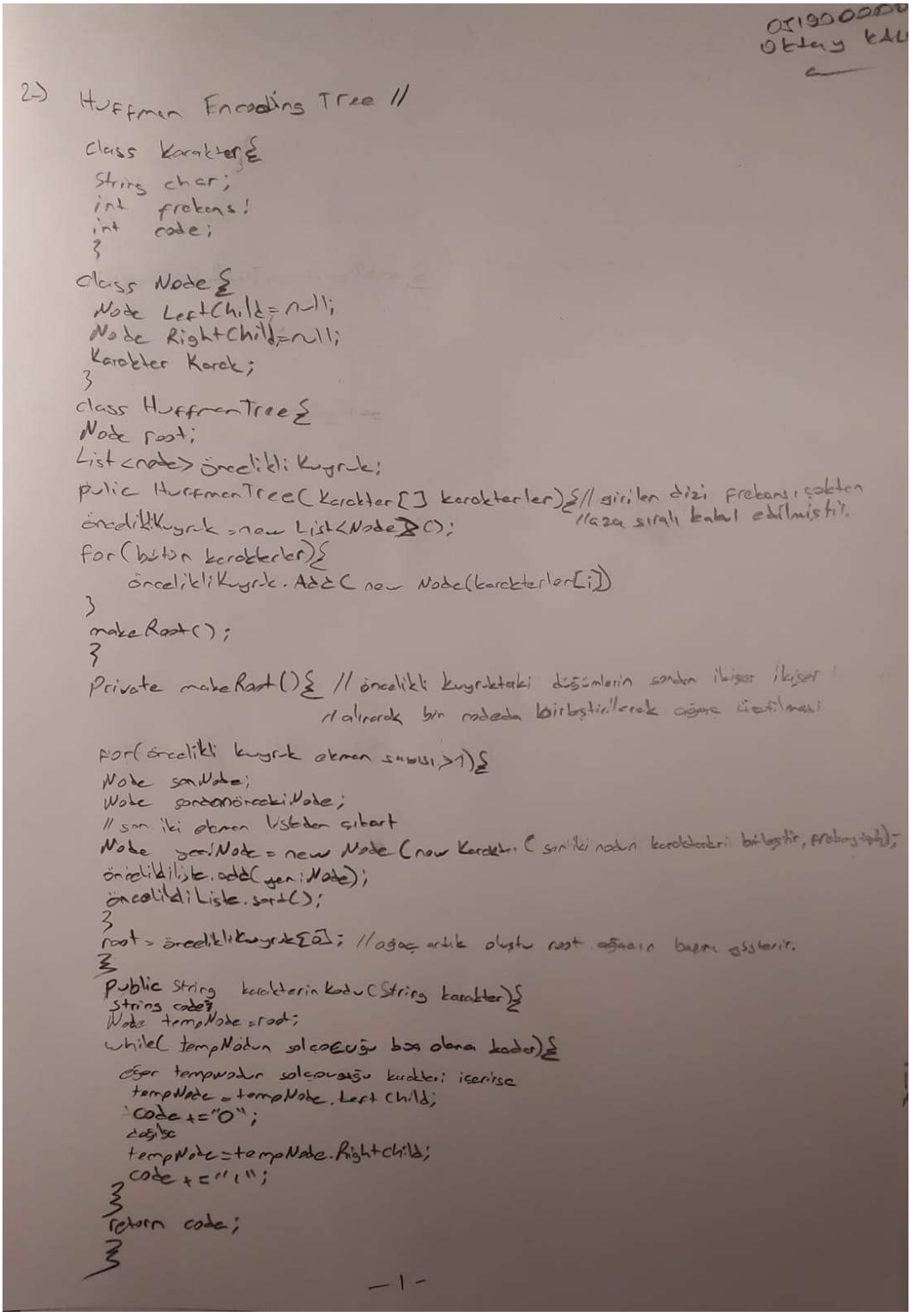
GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS

1. AVL Tree Insertions (drawing), Heap Node Insertion and Removing

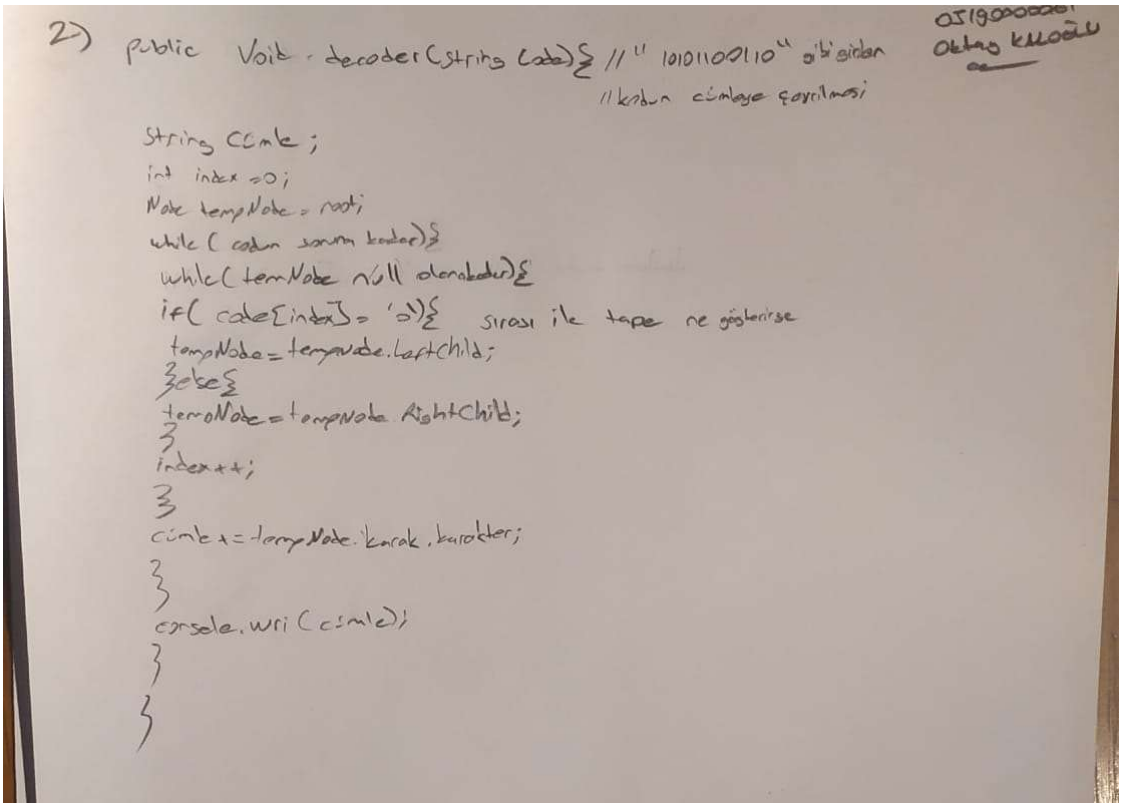


2. B-Tree Insertion Method [Alternatives: AVL-Tree Insertion, Red-Black Trees, Huffman Encoding Tree]

2.1 Kalemle Yazılan Algoritma Görüntüsü



```
2-) Huffman Encoding Tree //

class karakter{
 String char;
 int frekans;
 int code;
}
class Node{
 Node LeftChild=null;
 Node RightChild=null;
 Karakter Karak;
}
class HuffmanTree{
Node root;
List<node> öncelikli kuyruk;
pulic HuffmanTree(karakter[] karakterler){// girilen dizi frekansı çoktan
öncelikliKuyruk =new List<Node>();          //aza sıralı kabul edilmişti.
for(bütün karakterler){
    öncelikliKuyruk.Add( new Node(karakterler[i])
}
makeRoot();
}
Private makeRoot(){ // öncelikli kuyruktaki düğümlerin sondan ikişer ikişer
                    //alınarak bir noktada birleştirilerek ağaç üretilmesi

for(öncelikli kuyruk elemen sayısı>1){
Node sonNode;
Node sondanönceki Node;
// son iki elemanı listeden çıkart
Node yeni Node = new Node(new Karakter( sonlıki nodun karakterleri birleştir, frekans toplı);
öncelikliliste.add(yeni Node);
öncelikliListe.sort();
}
root = öncelikliKuyruk[0]; //ağaç artık oluştu root ağacın başını gösterir.
}
Public String karakterin kodu(String karakter){
 String code;
 Node tempNode=root;
 while( tempNodun sol çocuğu boş olana kadar){
  eğer tempNodun solçocuğu karakter içerirse
  tempNode = tempNode.Left Child;
  code+="0";
  değilse
  tempNode=tempNode.RightChild;
  code+="1";
 }
 return code;
}
```



2.2 Yöntemin (ekleme işleminin) anlatımı / açıklaması / adımları

//yazım kötü olduğu için kalemle yasmak istemedim.

//girilen karakter dizisi çoktan aza frekansda dizili kabul edilmiştir.

Girilen kelime nesnelerini içeren diziden frekanslara göre bir ağaç yapılarak her harfin kodlanamsıdır.

Giriş verisi bir metin de olabilir ona göre her karakterin frekansı bulunmalıdır. Bu frekanslara göre üretilen karakter nesneleri frekanslarına göre azdan çoğa doğru sıralandıktan sonra nesne içerisinde makeRoot ile öncelikli sırada yalnızda bir eleman kalana kadar sondaki iki düğümümn alınıp karakterlerinin birleştirilip, frekansları toplanarak yeni bir düğüm oluşturulur ve bu düğümün sağ socuğu listenin sondan alınan elemanı, diğeri ise sol çocuktur. Üretilen düğüm tekrardan sıralı şekilde listeye eklenir.Bu şekilde öncelikli kuyrukta 1 node kalana kadar tekrar edilir.

Ağaç oluşturulduktan sonra bir metinin kodlanması : girilen metin karakter karakter ağaçta aranır eğer sol çocuğu içerirse 0, değilse 1 eklenerek karakter aranır. Karakter bulunduğunda ki node dödürülecek stringe eklenir.

Verilen şifrenin çözülmesi : girilen şifrenin herbir rakamı sırası ile kullanılır. Yalnızca bir tane karaktere düşene kadar 0 için son çocuk, 1için sağ çocuk kullanılarak şifre çözülür. Çıkan karakter döndürülecek metine eklenir.

2.3 Kaynak Kod

```
class Karakter
    {
```

```
        public int frekans = 0;
        public String karakter;
        public string code;
        public Karakter(String car)
        {
            this.frekans = 1;
            this.karakter = car.ToString();
        }

        public Karakter(String car, int tekrar)
        {
            this.frekans = tekrar;
            this.karakter = car.ToString();
        }

        public string ToString()
        {
            return karakter + "     " + frekans + "     " + code;
        }


    }
    class HuffmanEncodingTree
    {
        private Node root;
        private List<Node> öncelikKuyruğu;
        private Karakter[] Karakterler;

        internal class Node
        {
            internal Karakter karak;

            internal Node leftChild;
            internal Node rightChild;

            public Node(Karakter kar)
            {
                this.karak = kar;
                this.leftChild = null;
                this.rightChild = null;


            }
        }
        public HuffmanEncodingTree(Karakter[] karakterler)//çok tekrardan az tekrara doğru sıralı karakterler neslelerini içeren dizi
girişi yapıldığı kabul edilmiştir.
        {
            Karakterler = karakterler;
            öncelikKuyruğu = new List<Node>();
            for (int i = 0; i < karakterler.Length; i++)
            {
                Node node = new Node(karakterler[i]);
                öncelikKuyruğu.Add(node);
            }
            makeRoot();

        }

        private void makeRoot()
        {
            int nodeSayısı = öncelikKuyruğu.Count;
            for (int i = nodeSayısı - 1; i > 0; i--)//öncelik kuğruğunda 1 tane item kalana kadar tekrarlanmalı. kkalan item ise bizim
kökümüz olmuştur.
            {
                Node son = öncelikKuyruğu[i];
                Node sondanÖnceki = öncelikKuyruğu[i - 1];

                öncelikKuyruğu.RemoveAt(i);//son iki itemın listeden çıkartılması
                öncelikKuyruğu.RemoveAt(i - 1);


                Karakter geriEklenecek = new Karakter(sondanÖnceki.karak.karakter + son.karak.karakter, son.karak.frekans +
sondanÖnceki.karak.frekans);
                Node geriEkelenecekNode = new Node(geriEklenecek);
                geriEkelenecekNode.rightChild = son;
                geriEkelenecekNode.leftChild = sondanÖnceki;
                bool durum = true;
                for (int y = 0; y < öncelikKuyruğu.Count; y++)
                {
                    if (öncelikKuyruğu[y].karak.frekans < geriEkelenecekNode.karak.frekans)
                    {
                        öncelikKuyruğu.Insert(y, geriEkelenecekNode);
                        durum = false;
                        break;
                    }
                }
                if (durum)
                {
                    öncelikKuyruğu.Add(geriEkelenecekNode);
                }

            }
            root = öncelikKuyruğu[0];
        }
        private string findCharCode(string karakter)
        {//verilen harfin ağaçtan faydalanarak kodunun üretilmesi
            Node tempNode = root;
            string code = "";
            while (tempNode != null && tempNode.leftChild != null)//node boş ise tek karaktere kadar inilmiştir.
```

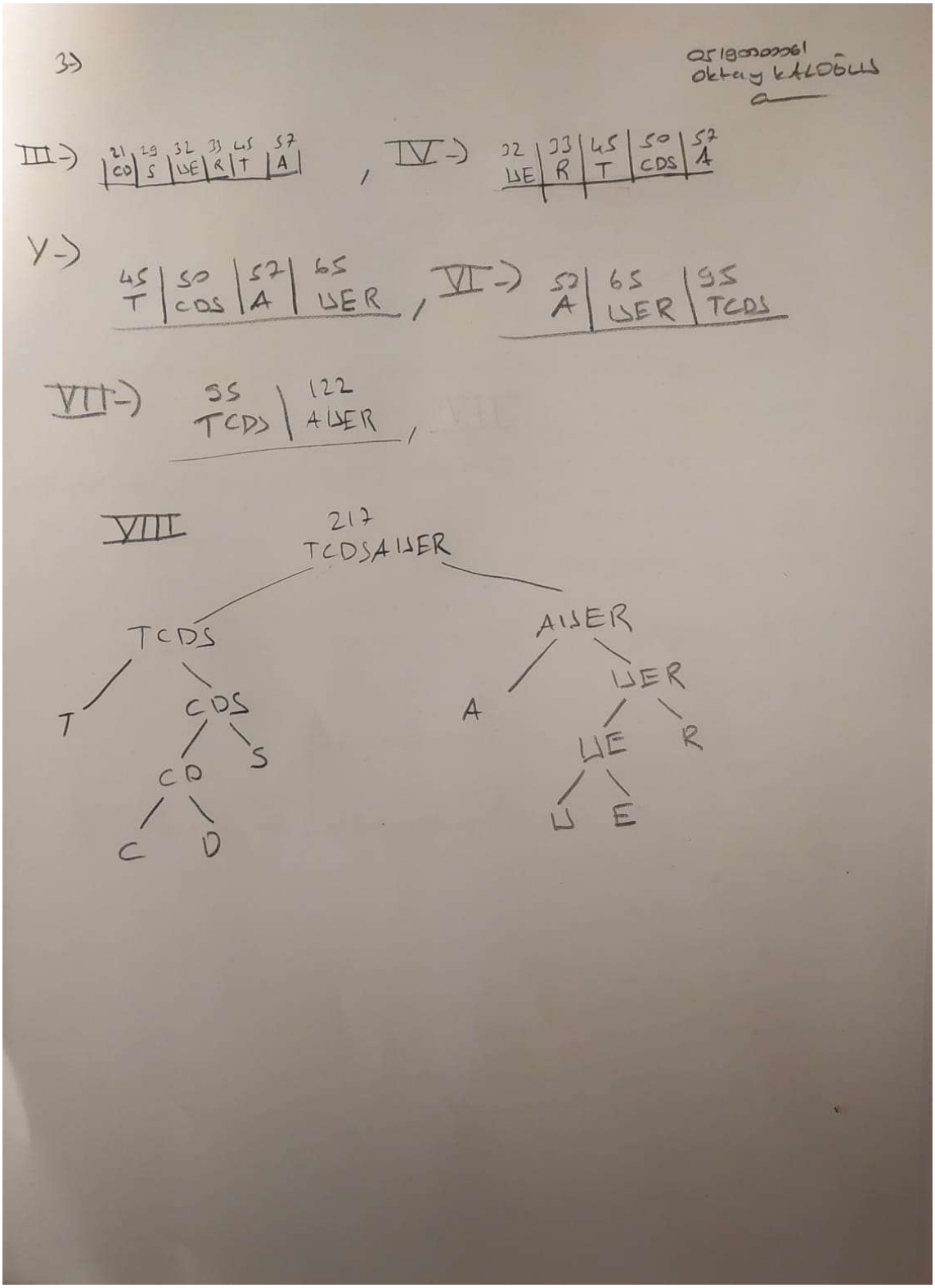
```
            {
                if (tempNode.leftChild.karak.karakter.Contains(karakter))
                {
                    tempNode = tempNode.leftChild;
                    code += "0";
                }
                else
                {
                    tempNode = tempNode.rightChild;
                    code += "1";
                }
            }
            return code;


        }
        public void harflerinCodeları()
        {
            Console.WriteLine("harf" + " " + "frekans" + " " + "code");
            for (int i = 0; i < Karakterler.Length; i++)
            {
                Karakterler[i].code = findCharCode(Karakterler[i].karakter);
                Console.WriteLine(Karakterler[i].ToString()); ;
            }
        }
        public string coder(string cümle)//cümle girilecek "asd ss ad". "010101001111" gibi string döndürücek
        {
            string code = "";


            for (int i = 0; i < cümle.Length; i++)
            {
                code += findCharCode(cümle.Substring(i, 1));

            }
            return code;
        }
        public string decoder(string code)//code girilecek "001101010" gibi
        {
            String cümle = "";
            int temp = 0;//her zaman başlanacak yeri gösterir
            Node tempNode = root;
            while (temp < code.Length)
            {
                while (tempNode.leftChild != null)//
                {
                    if (code[temp] != '0')
                    { //soldaki node değilse
                        temp++;
                        tempNode = tempNode.rightChild;
                    }
                    else//0 gelmiştir demmeki  sol çocuk aranılan harfi içerir.
                    {
                        temp++;
                        tempNode = tempNode.leftChild;

                    }
                }
                cümle += tempNode.karak.karakter;
                tempNode = root;//kodun devamı için tempnodun en başı göstermesi gerekir.

            }
            return cümle;


        }

    }
```

2.4 (İşletim + Test) Ekran Görüntüsü

```
char[] car = { ' ', 'a', 'b', 'c', 'd', 'e', 'f', 'g' };
int[] tekrar = { 55, 50, 45, 40, 35, 30, 25, 20 };
Console.WriteLine("abc ab abb : " + ağac.coder("abc ab abb".ToLower()));
Console.WriteLine("00000101111000001110000001001 çözülmüşü : " +
ağac.decoder("00000101111000001110000001001"));
```

```
harf frekans code
      55     11
a     50     000
b     45     001
c     40     011
d     35     100
e     30     101
f     25     0100
g     20     0101
abc ab abb : 00000101111000001110000001001
00000101111000001110000001001 çözülmüşü : abc ab abb
```

3. Huffman Encoding Tree Generation



4. Graph Algorithms

4.1 Dijkstra's Shortest Path [Source code + Screenshot for test]

```
class Dijkstra
    {
        // A utility function to find the
        // vertex with minimum distance
        // value, from the set of vertices
        // not yet included in shortest
        // path tree
        static int V = 8;
        int minDistance(int[] dist,
                        bool[] sptSet)
        {
            // Initialize min value
            int min = int.MaxValue, min_index = -1;

            for (int v = 0; v < V; v++)
                if (sptSet[v] == false && dist[v] <= min)
                {
                    min = dist[v];
                    min_index = v;
                }

            return min_index;
        }

        // A utility function to print
        // the constructed distance array
        void printSolution(int[] dist)
        {
            Console.Write("Vertex \t\t Distance "
                          + "from Source\n");
            for (int i = 0; i < V; i++)
                Console.Write(i + " \t\t " + dist[i] + "\n");
        }

        // Funtion that implements Dijkstra's
        // single source shortest path algorithm
        // for a graph represented using adjacency
        // matrix representation
        public void dijkstra(int[,] graph, int src)
        {
            V = graph.GetLength(0);
            int[] dist = new int[V]; // The output array. dist[i]
                                     // will hold the shortest
                                     // distance from src to i

            // sptSet[i] will true if vertex
            // i is included in shortest path
            // tree or shortest distance from
            // src to i is finalized
            bool[] sptSet = new bool[V];

            // Initialize all distances as
            // INFINITE and stpSet[] as false
            for (int i = 0; i < V; i++)
            {
                dist[i] = int.MaxValue;
                sptSet[i] = false;
            }

            // Distance of source vertex
            // from itself is always 0
            dist[src] = 0;

            // Find shortest path for all vertices
            for (int count = 0; count < V - 1; count++)
            {
                // Pick the minimum distance vertex
                // from the set of vertices not yet
                // processed. u is always equal to
                // src in first iteration.
                int u = minDistance(dist, sptSet);

                // Mark the picked vertex as processed
                sptSet[u] = true;

                // Update dist value of the adjacent
                // vertices of the picked vertex.
                for (int v = 0; v < V; v++)

                    // Update dist[v] only if is not in
                    // sptSet, there is an edge from u
                    // to v, and total weight of path
                    // from src to v through u is smaller
                    // than current value of dist[v]
                    if (!sptSet[v] && graph[u, v] != 0 && dist[u] != int.MaxValue && dist[u] + graph[u, v] < dist[v])
                        dist[v] = dist[u] + graph[u, v];
            }

            // print the constructed distance array
            printSolution(dist);
```

```
        }
    }
```

```
                                        //a,b,c,d,e,f,g,h,j,
    int[,] graph = new int[,] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },//a
                                { 4, 0, 8, 0, 0, 0, 0, 11, 0 },//b
                                { 0, 8, 0, 7, 0, 4, 0, 0, 2 },//c
                                { 0, 0, 7, 0, 9, 14, 0, 0, 0 },//d
                                { 0, 0, 0, 9, 0, 10, 0, 0, 0 },//e
                                { 0, 0, 4, 14, 10, 0, 2, 0, 0 },//f
                                { 0, 0, 0, 0, 0, 2, 0, 1, 6 },//g
                                { 8, 11, 0, 0, 0, 0, 1, 0, 7 },//h
                                { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };//j
```

```
Dijkstra's Shortest Path
Vertex              Distance from Source
0                   0
1                   4
2                   12
3                   19
4                   21
5                   11
6                   9
7                   8
8                   14
```

4.2 Prim's MST [Source code + Screenshot for test]

```
class MST//Prim's MST
    {

        // Number of vertices in the graph
        private int V = 9;

        // A utility function to find
        // the vertex with minimum key
        // value, from the set of vertices
        // not yet included in MST
        public MST(int[,] graph)
        {
            this.V = graph.GetLength(0);
            primMST(graph);
        }
        private int minKey(int[] key, bool[] mstSet)
        {

            // Initialize min value
            int min = int.MaxValue, min_index = -1;

            for (int v = 0; v < V; v++)
                if (mstSet[v] == false && key[v] < min)
                {
                    min = key[v];
                    min_index = v;
                }

            return min_index;
        }

        // A utility function to print
        // the constructed MST stored in
        // parent[]
        private void printMST(int[] parent, int[,] graph)
        {
            Console.WriteLine("Edge" + " \t\t " + "Weight");
            for (int i = 1; i < V; i++)
                Console.WriteLine(parent[i] + " - " + i +" \t\t "+ graph[i, parent[i]]);
        }

        // Function to construct and
        // print MST for a graph represented
        // using adjacency matrix representation
        public  void primMST(int[,] graph)
        {

            // Array to store constructed MST
            int[] parent = new int[V];
```

```
        // Key values used to pick
        // minimum weight edge in cut
        int[] key = new int[V];

        // To represent set of vertices
        // included in MST
        bool[] mstSet = new bool[V];

        // Initialize all keys
        // as INFINITE
        for (int i = 0; i < V; i++)
        {
            key[i] = int.MaxValue;
            mstSet[i] = false;
        }

        // Always include first 1st vertex in MST.
        // Make key 0 so that this vertex is
        // picked as first vertex
        // First node is always root of MST
        key[0] = 0;
        parent[0] = -1;

        // The MST will have V vertices
        for (int count = 0; count < V - 1; count++)
        {

            // Pick thd minimum key vertex
            // from the set of vertices
            // not yet included in MST
            int u = minKey(key, mstSet);

            // Add the picked vertex
            // to the MST Set
            mstSet[u] = true;

            // Update key value and parent
            // index of the adjacent vertices
            // of the picked vertex. Consider
            // only those vertices which are
            // not yet included in MST
            for (int v = 0; v < V; v++)

                // graph[u][v] is non zero only
                // for adjacent vertices of m
                // mstSet[v] is false for vertices
                // not yet included in MST Update
                // the key only if graph[u][v] is
                // smaller than key[v]
                if (graph[u, v] != 0 && mstSet[v] == false
                    && graph[u, v] < key[v])
                {
                    parent[v] = u;
                    key[v] = graph[u, v];
                }
        }

        // print the constructed MST
        printMST(parent, graph);
    }
}
```

```
                                    //a,b,c,d,e,f,g,h,j,
        int[,] graph = new int[,] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },//a
                                    { 4, 0, 8, 0, 0, 0, 0, 11, 0 },//b
                                    { 0, 8, 0, 7, 0, 4, 0, 0, 2 },//c
                                    { 0, 0, 7, 0, 9, 14, 0, 0, 0 },//d
                                    { 0, 0, 0, 9, 0, 10, 0, 0, 0 },//e
                                    { 0, 0, 4, 14, 10, 0, 2, 0, 0 },//f
                                    { 0, 0, 0, 0, 0, 2, 0, 1, 6 },//g
                                    { 8, 11, 0, 0, 0, 0, 1, 0, 7 },//h
                                    { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };//j
```

```
Prim's MST
Edge              Weight
0 - 1             4
1 - 2             8
2 - 3             7
3 - 4             9
2 - 5             4
5 - 6             2
6 - 7             1
2 - 8             2
```

4.3 BFT (Breadth-First Traverse) or DFT (Depth-First Traverse) [Source code + Screenshot for test]

4.4 Big-O Table (Time Complexities)

	Dijkstra's SP	Prim's MST	BFT	**Heap Insertion**
Big-O (Zaman Karmaşıklığı Big-O Notasyonuna Göre)	O(V^2), V=köşe sayısı	O(V^2)	O(V+E) E=kenar	O(LogN) N=node sayısı

// binary heap yardımıyla hem Dijkstra hem de prims in zaman karmaşıklığı O(ELogV) ye düşürülebilir.

5. Graph Drawing and Finding Shortest Path in Python

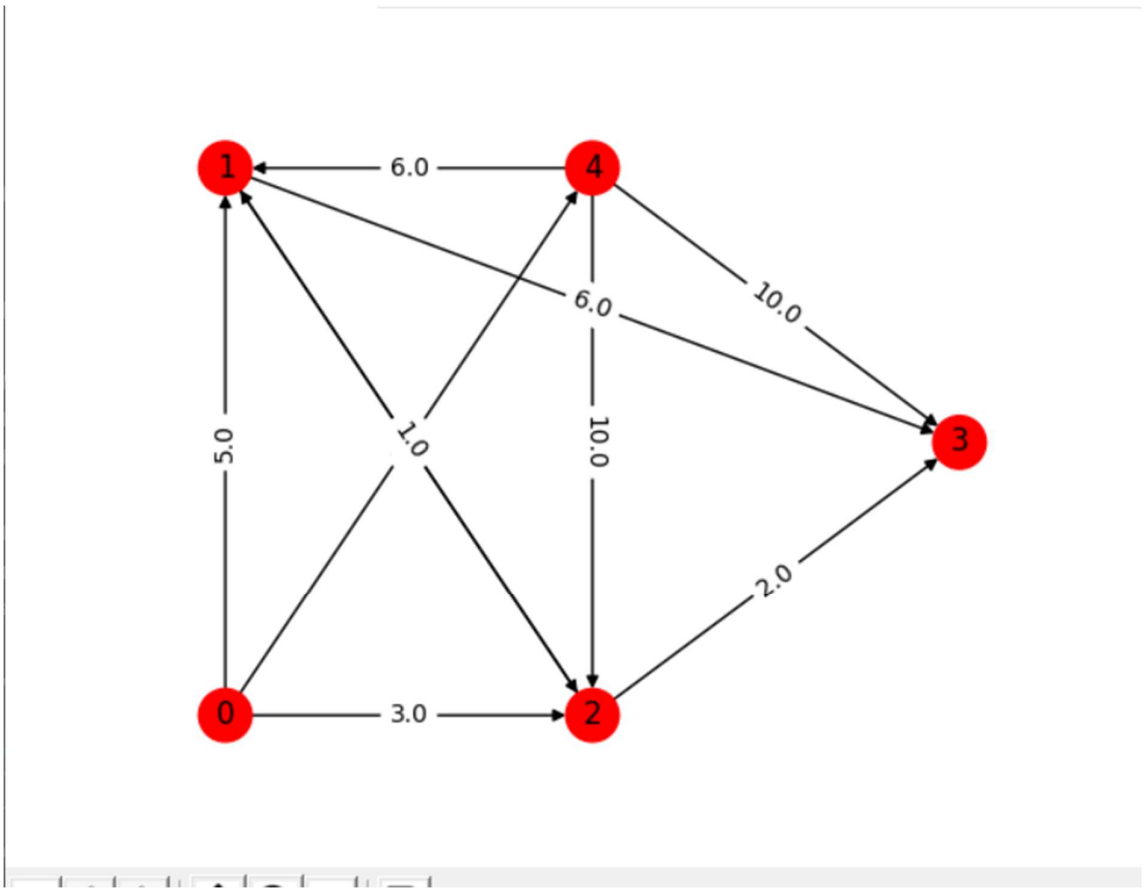
5.1 Graph Drawing

// Kaynak kod burada yer alacak

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.DiGraph()
list_nodes = [0, 1, 2, 3, 4]
G.add_nodes_from(list_nodes)
G.nodes()
list_arcs = [(0, 1, 5.0), (0, 4, 2.0), (0, 2, 3.0), (4, 1, 6.0), (4, 2, 10.0), (4, 3, 10.0), (2, 3, 2.0), (2, 1, 1.0),(1, 2, 2.0), (1, 3, 6.0)]
G.add_weighted_edges_from(list_arcs)
G.edges()
G.nodes[0]['pos'] = (0, 0)
G.nodes[1]['pos'] = (0, 2)
G.nodes[2]['pos'] = (2, 0)
G.nodes[3]['pos'] = (4, 1)
G.nodes[4]['pos'] = (2, 2)
node_pos = nx.get_node_attributes(G, 'pos')
arc_weight = nx.get_edge_attributes(G, 'weight')

node_col = ['white' if not node in G else 'red' for node in G.nodes()]
edge_col = ['black' if not edge in G else 'red' for edge in G.edges()]
nx.draw_networkx(G, node_pos, node_color=node_col, node_size=450)
```

```
nx.draw_networkx_edge_labels(G, node_pos, edge_labels=arc_weight)
plt.axis('off')
plt.show()
```

// Çizim burada yer alacak

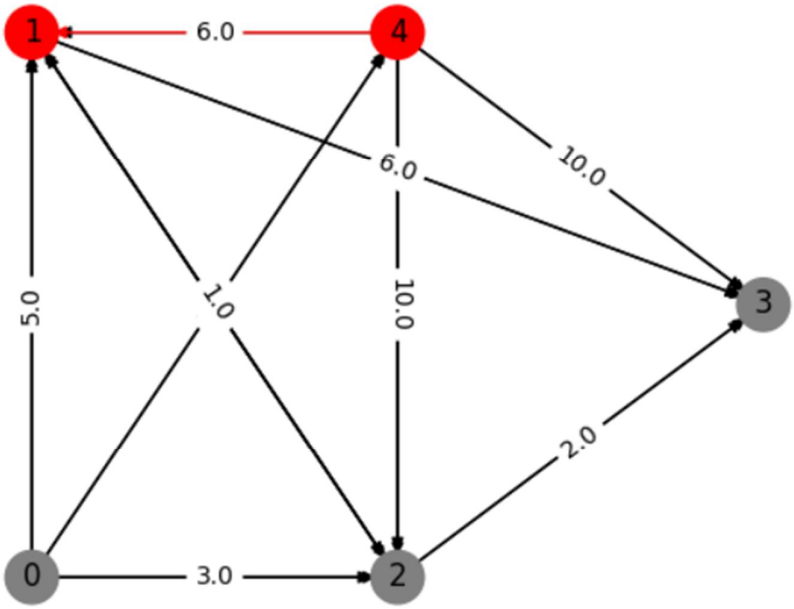


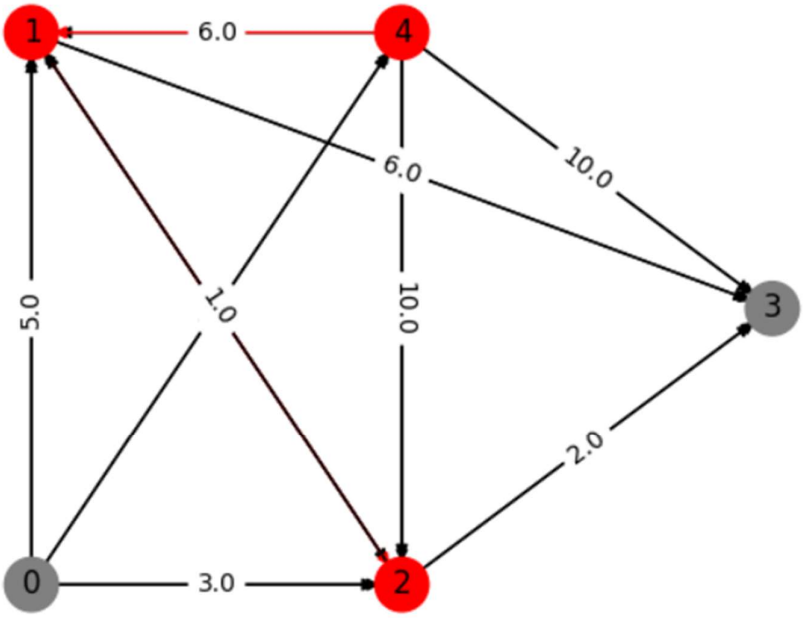
5.2 Finding Shortest Path

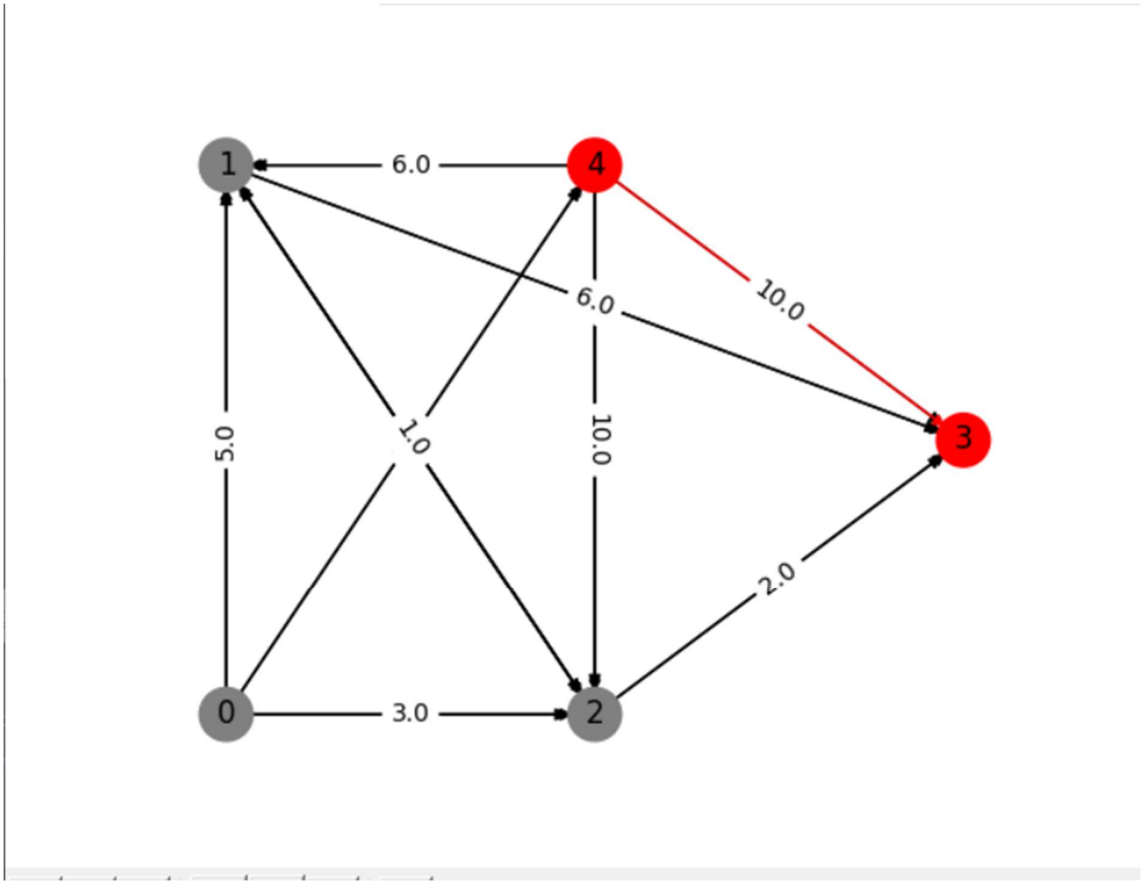
// Kaynak kod burada yer alacak

```
for i in (1,2,3):

  sp = nx.dijkstra_path(G, source=4, target=i)
  print(sp)
  G.nodes[0]['pos'] = (0, 0)
  G.nodes[1]['pos'] = (0, 2)
  G.nodes[2]['pos'] = (2, 0)
  G.nodes[3]['pos'] = (4, 1)
  G.nodes[4]['pos'] = (2, 2)
  node_pos = nx.get_node_attributes(G, 'pos')
  arc_weight = nx.get_edge_attributes(G, 'weight')
  red_edges = list(zip(sp, sp[1:]))


  node_col = ['grey' if not node in sp else 'red' for node in G.nodes()]
  edge_col = ['black' if not edge in red_edges else 'red' for edge in G.edges()]
  nx.draw_networkx(G, node_pos, node_color=node_col, node_size=450)
  nx.draw_networkx_edges(G, node_pos, edge_color=edge_col)
  nx.draw_networkx_edge_labels(G, node_pos, edge_labels=arc_weight)
  plt.axis('off')
  plt.show()
```

// Konsol / Ekran görüntüsü burada yer alacak


1
4
6.0
6.0
10.0
5.0
1.0
10.0
3
2.0
0
3.0
2


1
4
6.0
6.0
10.0
5.0
1.0
10.0
3
2.0
0
3.0
2



5.3 Vertex Removing and repeating previous steps (Drawing and Finding SP)

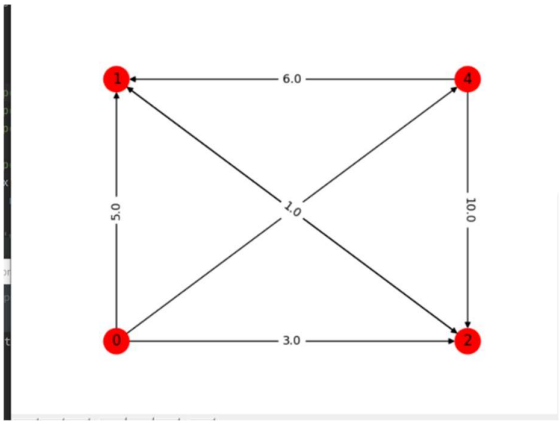
// Kaynak kodlar burada yer alacak

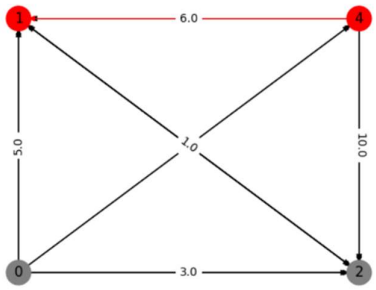
```
G.remove_node(3)


plt.show()

G.nodes[0]['pos'] = (0, 0)
G.nodes[1]['pos'] = (0, 2)
G.nodes[2]['pos'] = (2, 0)

G.nodes[4]['pos'] = (2, 2)
node_pos = nx.get_node_attributes(G, 'pos')
arc_weight = nx.get_edge_attributes(G, 'weight')

node_col = ['white' if not node in G else 'red' for node in G.nodes()]
edge_col = ['black' if not edge in G else 'red' for edge in G.edges()]
nx.draw_networkx(G, node_pos, node_color=node_col, node_size=450)
nx.draw_networkx_edge_labels(G, node_pos, edge_labels=arc_weight)
plt.axis('off')
plt.show()


for i in (1,2):

  sp = nx.dijkstra_path(G, source=4, target=i)
  print(sp)
  G.nodes[0]['pos'] = (0, 0)
  G.nodes[1]['pos'] = (0, 2)
  G.nodes[2]['pos'] = (2, 0)

  G.nodes[4]['pos'] = (2, 2)
  node_pos = nx.get_node_attributes(G, 'pos')
  arc_weight = nx.get_edge_attributes(G, 'weight')
  red_edges = list(zip(sp, sp[1:]))
  node_col = ['grey' if not node in sp else 'red' for node in G.nodes()]
  edge_col = ['black' if not edge in red_edges else 'red' for edge in G.edges()]
  nx.draw_networkx(G, node_pos, node_color=node_col, node_size=450)
```

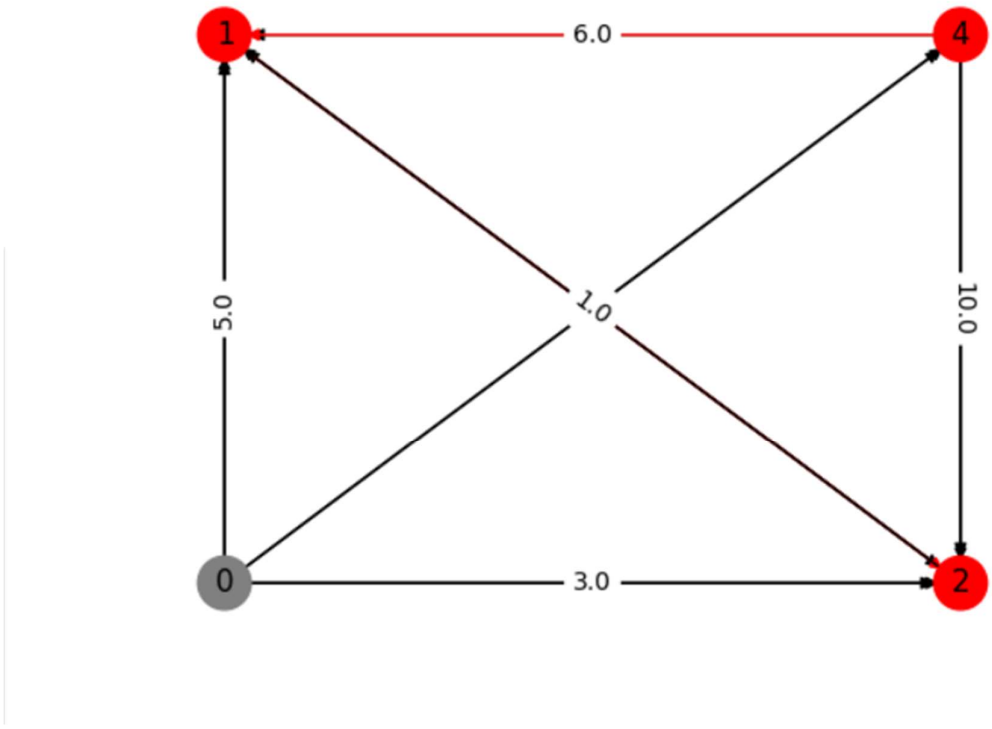
```
nx.draw_networkx_edges(G, node_pos, edge_color=edge_col)
nx.draw_networkx_edge_labels(G, node_pos, edge_labels=arc_weight)
plt.axis('off')
plt.show()
```

```
G.clear()
```

// Çizim ve konsol / ekran görüntüleri burada yer alacak







6. Comparison of MST Algorithms

//yazım kötü olduğu için kalemle yazmak istemedim.

Kruskal's Algorithm: MST yi en düşük fiyatlı olan köşeden başlayarak oluşturur. Herbir düğümden yalnızca bir kere geçer ve sıradaki kenar bulunulan köşede olamasa bile her zaman en ucuz olandır. Birbirinden kopuk elemanlarıda kullanır ve bir orman oluşturur. Daha ayrık graphlarda daha verimlidir.

Prim's Algorithm: MST yi istenilen köşeden başlayarak oluşturur. En ucuz yolu bulmak için aynı köşeyi birden fazla kez gezer. Yalnızca herbir köşe birbirine bağlı ise çalışabilir. Bir sonraki kenar o köşedeki en ucuz kenardır. Yoğun ve fazla kenarlı graphlarda daha hızlı çalışır.

7. Definitions

//yazım kötü olduğu için kalemle yazmak istemedim.

a-) B+Tree: ikili arama ağacının aksine ebebeyinler ikiden fazla çocuğa sahip olabilirler. Btree kendisini dengeleyen ve elemanlarını sıralı tutan bir ağaç yapısıdır bu nedenle arama, ekleme, çıkartma işlemleri O(LogN) süresiyle yapılabilir. Database gibi büyük verilerin depolama ve okunduğu sistemler için uygundur.

b-) 2-3-4 Tree: Kendisini dengeleyen ve ağacın her düğümün iki,üç veya dört çocuğu olan bir ağaçtır. bütün çocuklar aynı derinlik seviyesindedir. Arama, ekleme, çıkartma işlemleri O(LogN) süresiyle yapılabilir. Sözlüklerin verilerinin tutulması için kullanılabilir.

h-) Dynamic Programming: Problemin daha küçük alt paraçalara bölüp, bu küçük problemler için yapılan hesaplamaların bellekte tutulmasıdır(revursive göre bellekte fazladan yer tutar). Büyük problemin recursive çözümü için aynı değerler için birden fazla hesaplamalar yapılabilecektir, dynamic programming matematiksel bir yaklaşım ile recursive çözümü daha lineer hale getirir.

d. Quadratic Probing: hashing foksiyonları aynı index değerini birden fazla, farklı veri içinde üretebilir bu üretilen indexlere ekleme yapılamayacağı için bu veriler belirli bir düzende hashtable a eklenmelidir. Quadratic probing bu çakışma problemini aşmak için kullanılan bir yöntemdir. Üretilen indexte çakışma gerçekleşirse üretilen index e i^2 eklenerek boş bir yer aranır. (i =>1 , i tamsayıdır.)

g. Topological Sorting : yölü bir graphın her düğümünün, kendisinin gösterdiği her düğümden daha önce yer aldığı bir dizi haline getirilmesidir. Kullanım amacı genelde birbirini gerektiren uygulama adımlarının sıralanışının belirlenmesidir.s

SELF-ASSESSMENT TABLE

	Points	Estimated Grade	Explanation
1 a) AVL Tree	**10**	**10**	**Yapıldı**
1 b) Heap	**10**	**10**	**Yapıldı**
2) B-Tree Insertion / AVL Tree Insertion / Red-Black Trees / Huffman Encoding Tree	**10**	**10**	**Yapıldı**
2) Generating Huffman Encoding Tree	**10**	**10**	**Yapıldı**
4 a) Dijkstra's shortest path code + test	**5**	**5**	**Yapıldı**
4 b) Prim's MST code + test	**5**	**5**	**Yapıldı**
4 c) BFT or DFT code + test	**5**	**5**	**Yapıldı**
4 d) Filling Big-O Table	**5**	**5**	**Yapıldı**
5 i) Graph Drawing	**5**	**5**	**Yapıldı**
5 ii) Finding Shortest Paths with Dijkstra's	**5**	**5**	**Yapıldı**
5 iii) Node deletion and repeating i, ii.	**5**	**5**	**Yapıldı**
6) Comparison (Prim's & Kruskal's Algorithm)	**5**	**5**	**Yapıldı**
7) Explanations of 5 terms	**10**	**10**	**Yapıldı**
Demo Video for Source Codes and Tests of Q2, Q4 and Q5	**5**	**5**	**Yapıldı**
Self-assessment Table	**5**	**5**	**Yapıldı**
Total	**100**	**100**	

Açıklama kısmında yapıldı, yapılmadı bilgisi ve hangi maddelerin nasıl yapıldığı (ve nelerin yapılmadığı / yapılamadığı) yazılmalıdır. Tahmini not kısmına da ilgili maddeden kaç almayı beklediğinizi yazmalısınız.