

**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**

05180000091, Kutay Avcı

05180000056, Emre Balkaya

İçindekiler

[GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS 2](#_Toc64320804)

[1. AVL Tree Insertions (drawing), Heap Node Insertion and Removing 2](#_Toc64320805)

[2. B-Tree Insertion Method [Alternatives: AVL-Tree Insertion, Red-Black Trees, Huffman Encoding Tree] 3](#_Toc64320806)

[2.1 Kalemle Yazılan Algoritma Görüntüsü 3](#_Toc64320807)

[2.2 Yöntemin (ekleme işleminin) anlatımı / açıklaması / adımları 5](#_Toc64320808)

[2.3 Kaynak Kod 6](#_Toc64320809)

[2.4 (İşletim + Test) Ekran Görüntüsü 11](#_Toc64320810)

[3. Huffman Encoding Tree Generation 12](#_Toc64320811)

[4. Graph Algorithms 13](#_Toc64320812)

[4.1 Dijkstra’s Shortest Path [Source code + Screenshot for test] 13](#_Toc64320813)

[4.2 Prim’s MST [Source code + Screenshot for test] 15](#_Toc64320814)

[4.3 BFT (Breadth-First Traverse) or DFT (Depth-First Traverse) [Source code + Screenshot for test] 18](#_Toc64320815)

[4.4 Big-O Table (Time Complexities) 20](#_Toc64320816)

[5. Graph Drawing and Finding Shortest Path in Python 20](#_Toc64320817)

[5.1 Graph Drawing 20](#_Toc64320818)

[5.2 Finding Shortest Path 20](#_Toc64320819)

[5.3 Vertex Removing and repeating previous steps (Drawing and Finding SP) 21](#_Toc64320820)

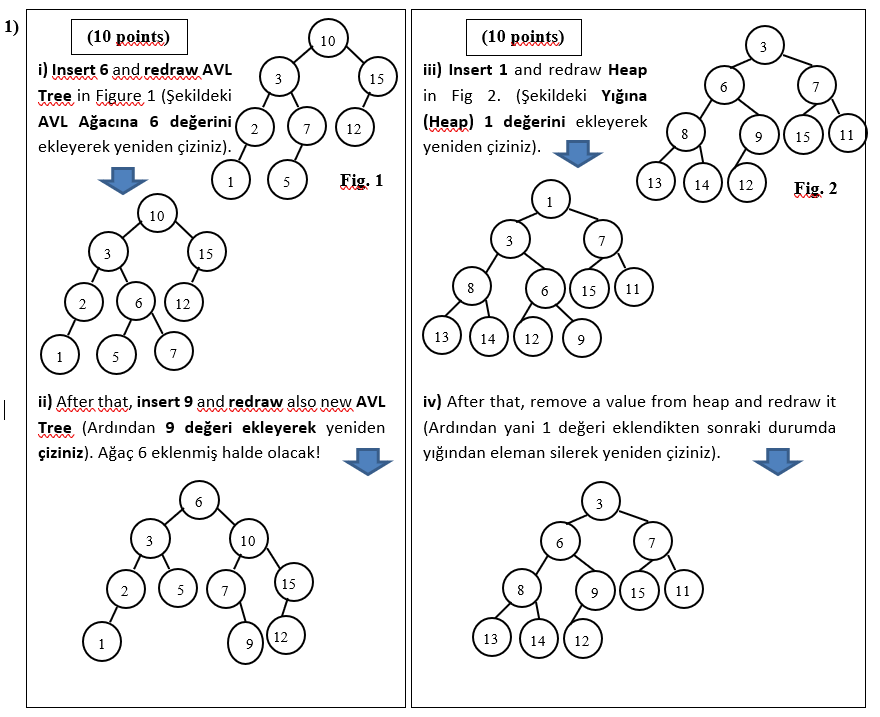
[6. Comparison of MST Algorithms 21](#_Toc64320821)

[7. Definitions 22](#_Toc64320822)

[SELF-ASSESSMENT TABLE 24](#_Toc64320823)

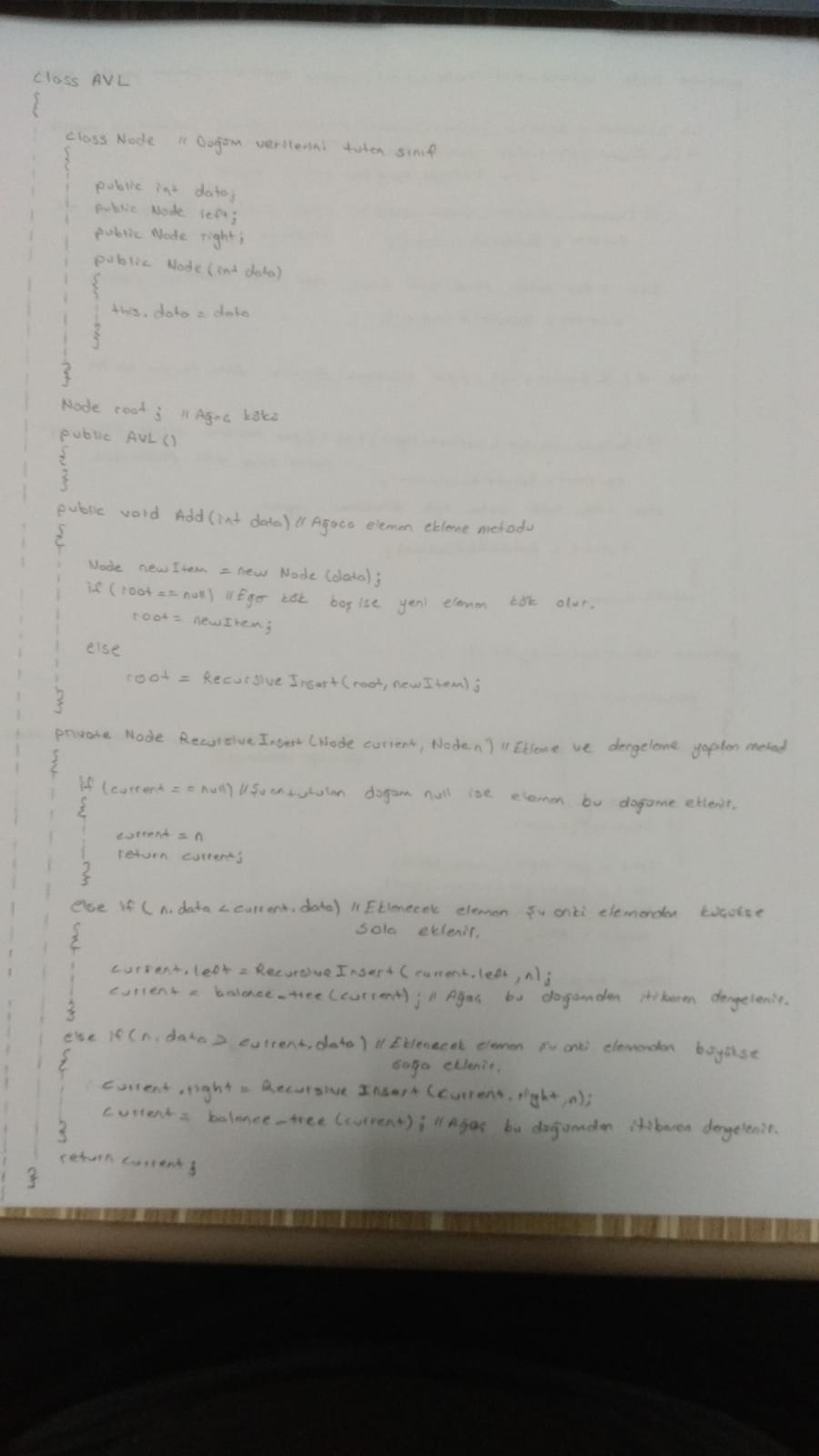
# GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS

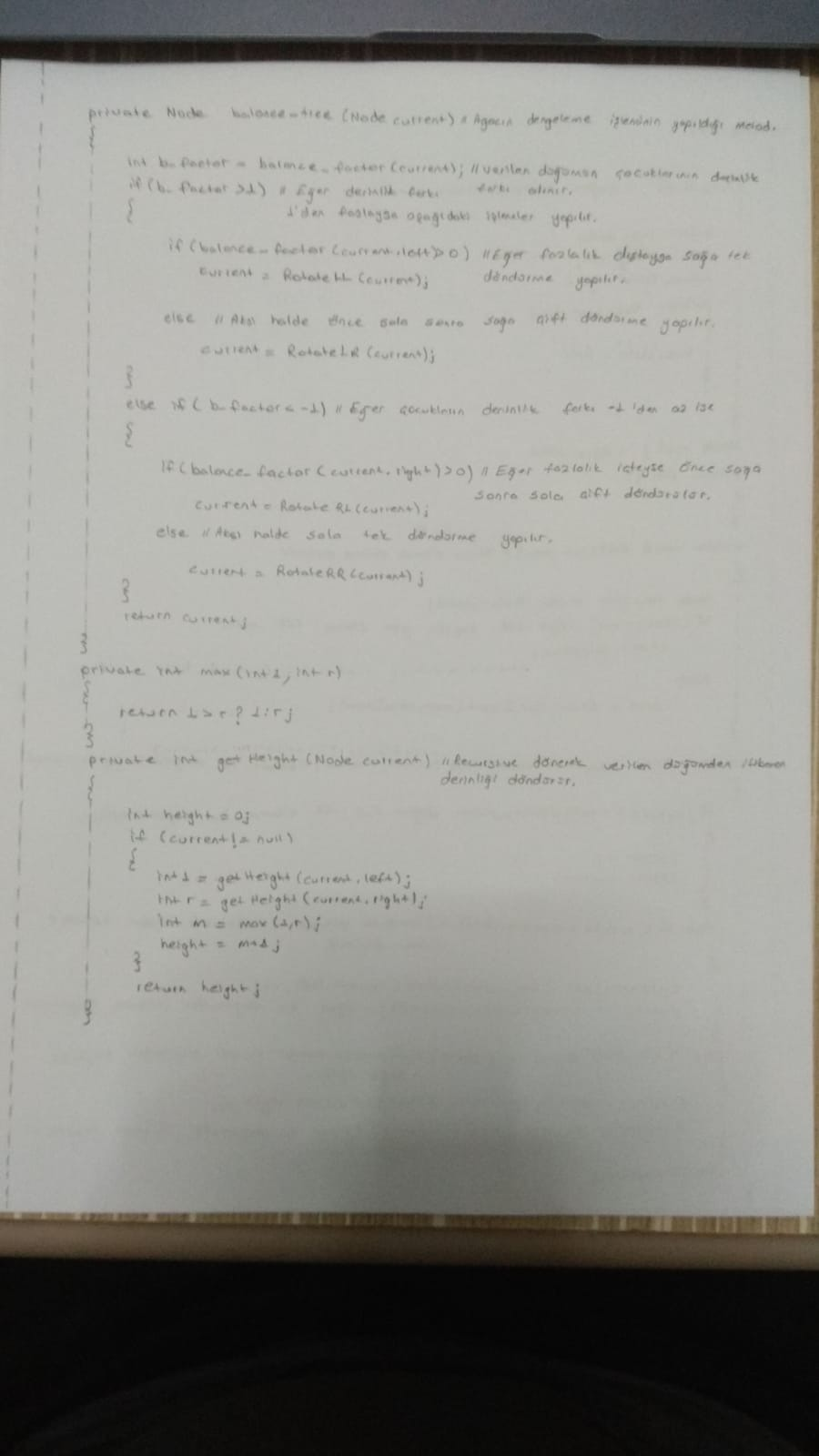
# 1. AVL Tree Insertions (drawing), Heap Node Insertion and Removing

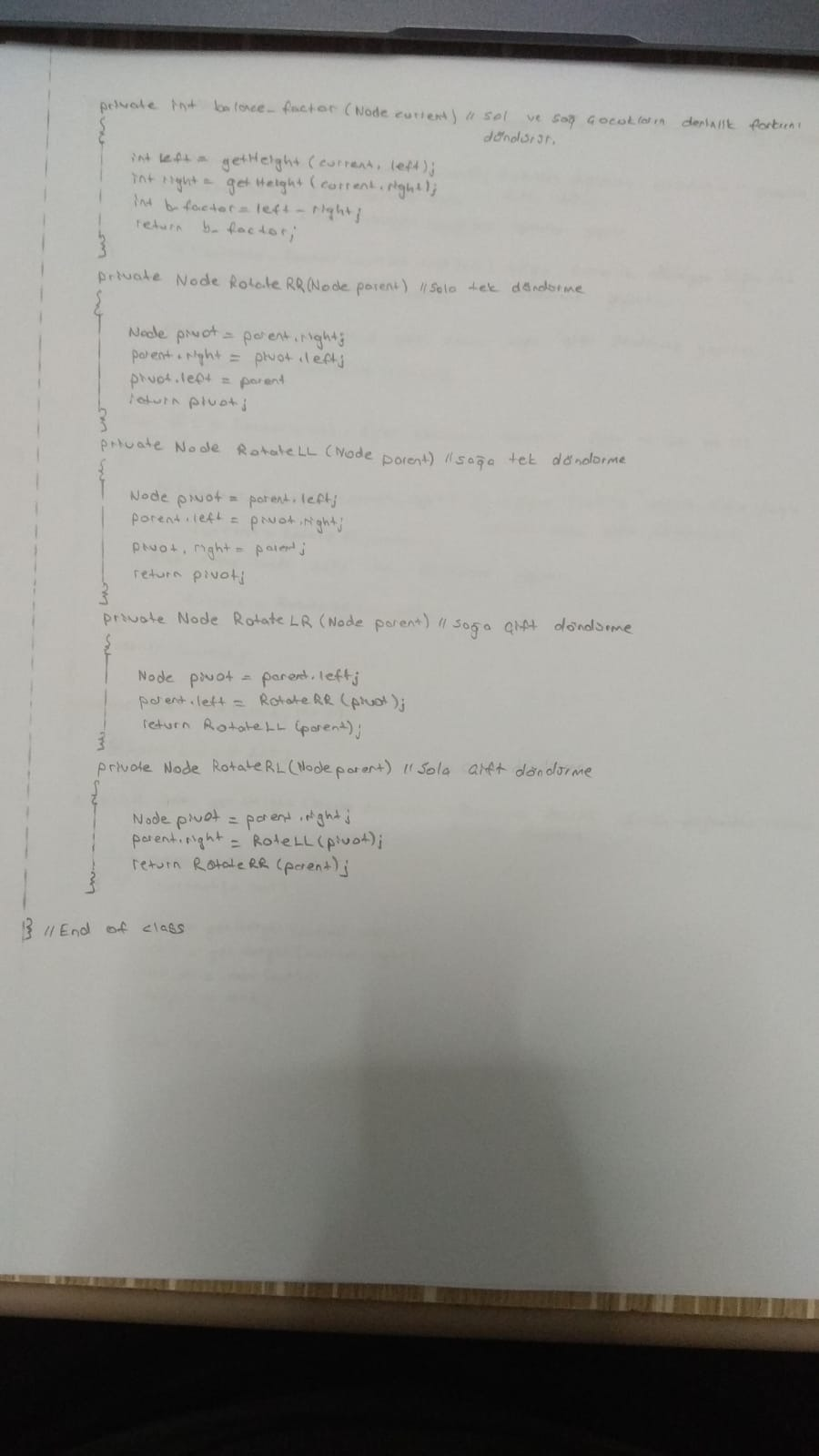


# 2. B-Tree Insertion Method [Alternatives: AVL-Tree Insertion, Red-Black Trees, Huffman Encoding Tree]

## 2.1 Kalemle Yazılan Algoritma Görüntüsü







## 2.2 Yöntemin (ekleme işleminin) anlatımı / açıklaması / adımları

AVL Tree’de ekleme işlemi yapıldıktan sonra çocuk düğümlerin derinlikleri farkı hesaplanır.Eğer sol çocuğun derinliği ve sağ çocuğun derinliği farkı 1’den büyükse ve fark çevre düğümlerde oluşuyorsa ağaca sağa doğru tek döndürme yapılır. Eğer fark iç kısımda oluşuyorsa çift döndürme işlemi yapılır.Sol çocuğun derinliği ve sağ çocuğun derinliği farkı -1’den küçükse ve fark çevre düğümlerde oluşuyorsa ağaca sola doğru tek döndürme yapılır.Eğer fark iç kısımda ise çift döndürme yapılır ve ağaç dengelenir.

## 2.3 Kaynak Kod

class AVL

{

class Node // Düğüm verilerini tutan sınıf

{

public int data;

public Node left;

public Node right;

public Node(int data)

{

this.data = data;

}

}

Node root; //Ağaç kökü

public AVL()

{

}

public void Add(int data) //Ağaca eleman ekleme metodu

{

Node newItem = new Node(data);

if (root == null) //Eğer kök boş ise yeni eleman kök olur

{

root = newItem;

}

else

{

root = RecursiveInsert(root, newItem);

}

}

private Node RecursiveInsert(Node current, Node n) //Ekleme ve dengeleme işleminin gerçekleştiği metot

{

if (current == null) //şuan tutulan düğüm null ise eklenecek eleman şuan tutulan düğüme atanır

{

current = n;

return current;

}

else if (n.data < current.data) //eklenecek eleman şuanki tutulan elemandan küçük ise sola eklenir

{

current.left = RecursiveInsert(current.left, n);

current = balance\_tree(current);//ağaç şuanki tutulan düğümden itibaren dengelenir

}

else if (n.data > current.data) //eklenecek eleman şuanki tutulan elemandan küçük ise sola eklenir

{

current.right = RecursiveInsert(current.right, n); //recursive olarak işlem devam eder

current = balance\_tree(current);//ağaç şuanki tutulan düğümden itibaren dengelenir

}

return current;

}

private Node balance\_tree(Node current)//Ağacı dengeleme işlemi asıl olarak burda gerçekleşir

{

int b\_factor = balance\_factor(current); // verilen düğümün sol ve sağ çocuklarının derinlik farkı alınır

if (b\_factor > 1)//eğer sol ve sağ çocuklarının derinlik farkı 1'den fazlaysa aşağıdaki işlemler yapılır

{

if (balance\_factor(current.left) > 0)//eğer fazlalık çevrede ise sağa tek döndürme yapılır.

{

current = RotateLL(current);

}

else // aksi halde önce sola sonra sağa çift döndürme yapılır.

{

current = RotateLR(current);

}

}

else if (b\_factor < -1)//eğer sol ve sağ çocuklarının derinlik farkı -1'den az ise aşağıdaki işlemler yapılır

{

if (balance\_factor(current.right) > 0)//eğer fazlalık içerde ise önce sağa sonra sola çift döndürme yapılır.

{

current = RotateRL(current);

}

else // aksi halde sola tek döndürme yapılır.

{

current = RotateRR(current);

}

}

return current;

}

public void DisplayTree() //Ağacı bastırma işlemi

{

if (root == null)

{

Console.WriteLine("Tree is empty");

return;

}

InOrderDisplayTree(root);// InOrder olarak ağacı bastırır

Console.WriteLine();

}

private void InOrderDisplayTree(Node current)

{

if (current != null)

{

InOrderDisplayTree(current.left);

Console.Write("({0}) ", current.data);

InOrderDisplayTree(current.right);

}

}

private int max(int l, int r)

{

return l > r ? l : r;

}

private int getHeight(Node current) // recursive dönerek verilen düğümden itibaren derinliği döndürür

{

int height = 0;

if (current != null)

{

int l = getHeight(current.left);

int r = getHeight(current.right);

int m = max(l, r);

height = m + 1;

}

return height;

}

private int balance\_factor(Node current) //sol ve sağ çocuk derinlik farkını döndürür

{

int l = getHeight(current.left);

int r = getHeight(current.right);

int b\_factor = l - r;

return b\_factor;

}

private Node RotateRR(Node parent) //Sola tek döndürme

{

Node pivot = parent.right;

parent.right = pivot.left;

pivot.left = parent;

return pivot;

}

private Node RotateLL(Node parent)//sağa tek döndürme

{

Node pivot = parent.left;

parent.left = pivot.right;

pivot.right = parent;

return pivot;

}

private Node RotateLR(Node parent)//sağa çift döndürme

{

Node pivot = parent.left;

parent.left = RotateRR(pivot);

return RotateLL(parent);

}

private Node RotateRL(Node parent)//sola çift döndürme

{

Node pivot = parent.right;

parent.right = RotateLL(pivot);

return RotateRR(parent);

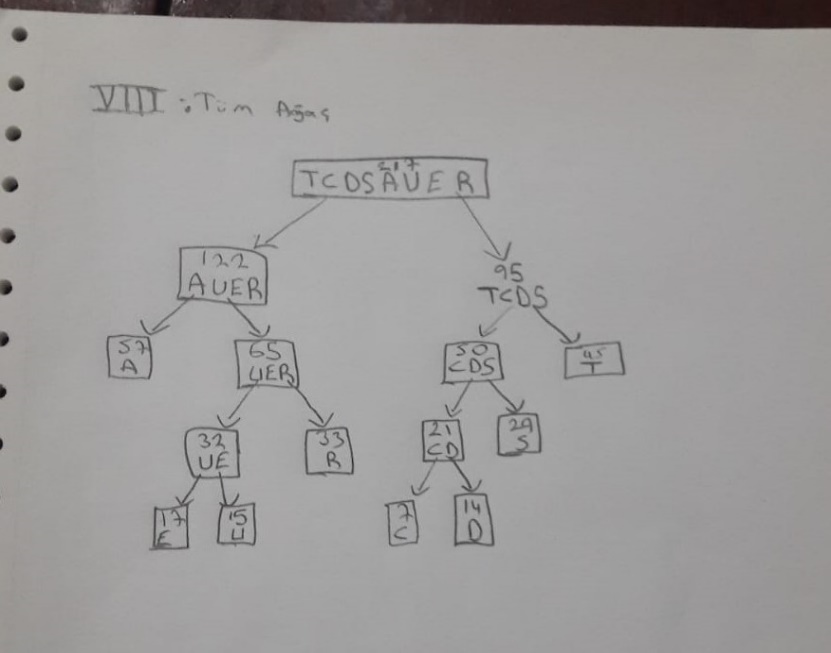
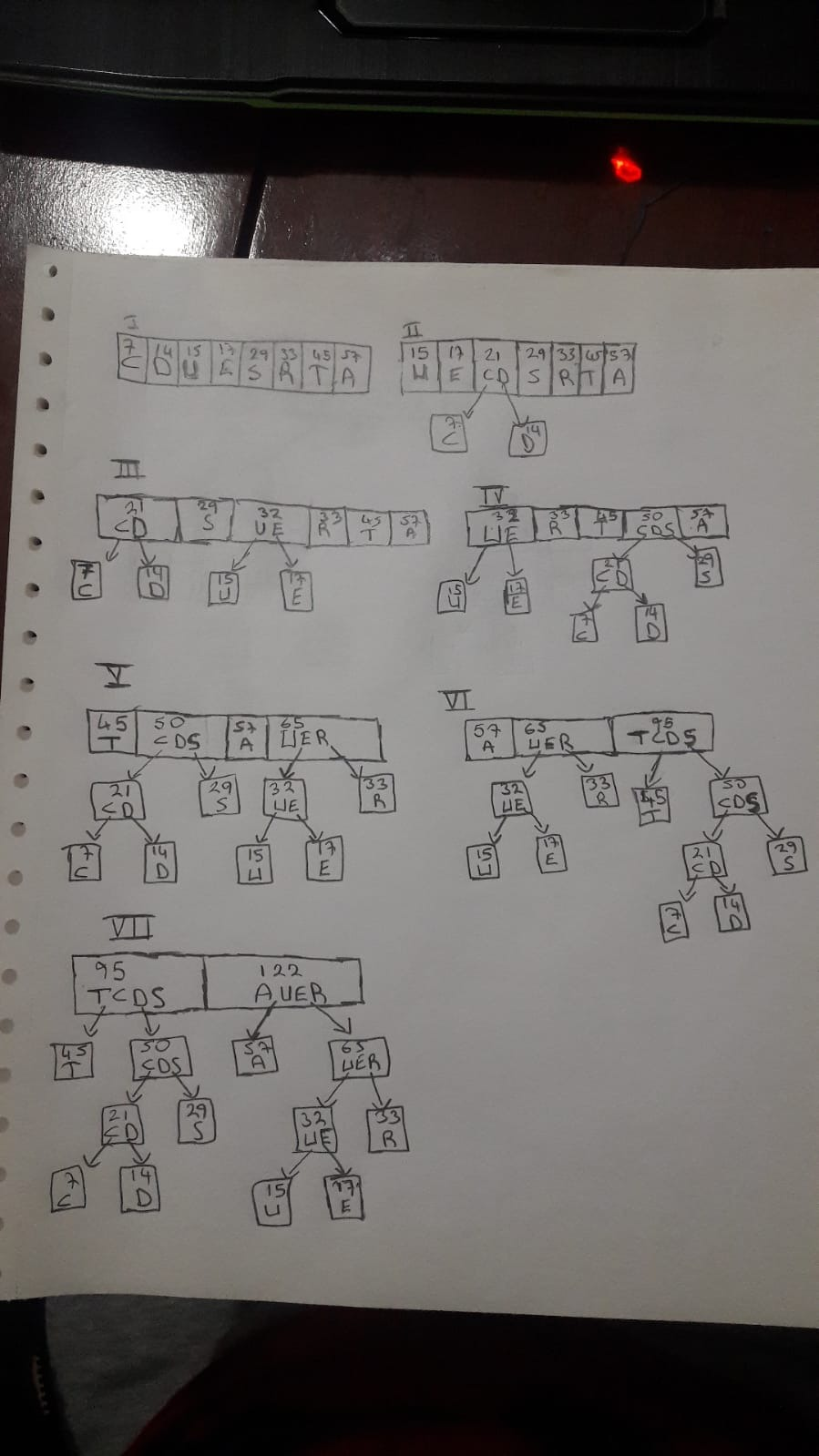
}

}//TEST KODLARI KAYNAK KOD DOSYASINDA MAIN METODU İÇİNDE MEVCUTTUR.

## 2.4 (İşletim + Test) Ekran Görüntüsü



# 3. Huffman Encoding Tree Generation



# 4. Graph Algorithms

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

class Dijkstra

{

static int V;

int minDistance(int[] dist, bool[] marked)// Verilen uzaklık ve işaret dizilerine göre en kısa uzaklığı bulur.

{

int min = int.MaxValue, min\_index = -1; //mininum değeri sonsuz olarak belirterek daha sonrası için kolaylık sağlanır.

for (int v = 0; v < V; v++)

if (marked[v] == false && dist[v] <= min)//Eğer işaretlenmemişse ve minimum değerden küçükse index ve min değerleri bu değerler olarak atanır.

{

min = dist[v];

min\_index = v;

}

return min\_index;//en küçük değerin indexini döndürür

}

void printSolution(int[] dist)// Uzaklık dizisini bastırır

{

Console.Write("Vertex Distance "

+ "from Source\n");

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

Console.Write(i + " \t\t " + dist[i] + "\n");

}

public void dijkstra(int[,] graph, int src) // En kısa yol algoritmasının uygulanma metodu

{

V = graph.GetLength(0);

int[] dist = new int[V]; // Verilen kaynaktan diğerlerine en kısa uzaklıkları tutan uzaklık dizisi

bool[] marked = new bool[V]; //En kısa yolu bulunan düğümlerin işaretlendiği dizi

for (int i = 0; i < V; i++) // Daha sonra karşılaştırmada kolaylık olması için uzaklığın sonsuz işaretlerin ise false olarak atanması yapılmıştır.

{

dist[i] = int.MaxValue;

marked[i] = false;

}

dist[src] = 0; //Kendine olan uzaklığı 0 olarak verilir.

for (int count = 0; count < V - 1; count++) //Bütün düğümlere olan en kısa uzaklıkları bulmak için döngü

{

int u = minDistance(dist, marked);// minimum uzaklığı bulan method.İlk döngüde u her zaman kaynağa eşittir.

marked[u] = true; //en kısa uzaklığı bulunan düğüm işaretlenir.

for (int v = 0; v < V; v++)

if (!marked[v] && graph[u, v] != 0 &&

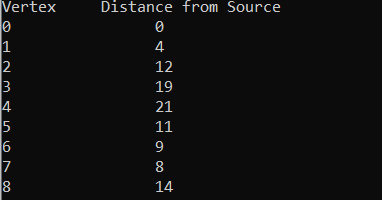
dist[u] != int.MaxValue && dist[u] + graph[u, v] < dist[v]) //Uzaklık dizisi gerekli kontrollerden sonra güncellenir

dist[v] = dist[u] + graph[u, v];

}

printSolution(dist);//Uzaklık dizisinin en son hali bastırılır.

}

} //TEST KODLARI KAYNAK KOD DOSYASI MAIN METODU İÇERİSİNDEDİR

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

class Prim\_s\_MST

{

static int V;

static int minKey(int[] key, bool[] marked) // minumum key değerinin indexini döndüren metot.

{

int min = int.MaxValue, min\_index = -1;

for (int v = 0; v < V; v++)

if (marked[v] == false && key[v] < min)//min değeri ile karşılaştırarak en küçük değer bulunur.

{

min = key[v];

min\_index = v;

}

return min\_index;

}

static void printMST(int[] parent, int[,] graph) //En küçük kapsayan ağacı bastıran metot.

{

Console.WriteLine("Edge \tWeight");

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

Console.WriteLine(parent[i] + " - " + i + "\t" + graph[i, parent[i]]);

}

public static void primMST(int[,] graph) //En küçük kapsayan ağacı oluşturan metot.

{

V = graph.GetLength(0);

int[] parent = new int[V]; //Oluşturulan MST'yi depolayan dizi

int[] key = new int[V]; // En küçük ağırlıkları tutan dizi

bool[] marked = new bool[V];//İşaret dizisi.

for (int i = 0; i < V; i++)//Karşılaştırma için minimum değer sonsuz olarak ve işaret false olarak atanır.

{

key[i] = int.MaxValue;

marked[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++)

{

int u = minKey(key, marked); //MST'de bulunmayan en küçük ağırlık değerini tutar.

marked[u] = true;//Bu düğümü işaretler

for (int v = 0; v < V; v++)

if (graph[u, v] != 0 && marked[v] == false

&& graph[u, v] < key[v])//Gerekli karşılaştırmaları yaparak MST'yi ve ağırlıkları tutan diziyi günceller

{

parent[v] = u;

key[v] = graph[u, v];

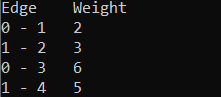
}

}

printMST(parent, graph);//Oluşturulan en küçük kapasayan ağacı bastırır.

}

}//TEST KODLARI KAYNAK KOD DOSYASI MAIN METODU İÇERİSİNDEDİR



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

class DFT

{

private int V; // Düğüm sayısı

private List<int>[] adj; //Komşuluk listesini tutan dizi

public DFT(int v)//Constructor Metot

{

V = v;

adj = new List<int>[v];

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

adj[i] = new List<int>();

}

public void AddEdge(int v, int w)//Graph'a kenar ekleme metodu

{

adj[v].Add(w);

}

void DFSUtil(int v, bool[] visited)//DFS fonskiyonun gerçekleştirimi

{

visited[v] = true;//Mevcut düğüm işaretlenir ve bastırılır.

Console.Write(v + " ");

List<int> vList = adj[v]; // Düğümün komşularını bir liste olarak tutar.

foreach (var n in vList)

{

if (!visited[n]) //İşaretlenmiş bir düğüm bulunana dek recursive olarak işlem devam eder.

DFSUtil(n, visited);

}

}

public void DFS(int v)//Depth First Search(Traversal) fonskiyonu

{

bool[] visited = new bool[V]; //Ziyaret edilen düğümleri işaretlemek için dizi

DFSUtil(v, visited);//Gerçekleştirim metodu.

}

}//TEST KODLARI KAYNAK KOD DOSYASI MAIN METODU İÇERİSİNDEDİR

## 4.4 Big-O Table (Time Complexities)

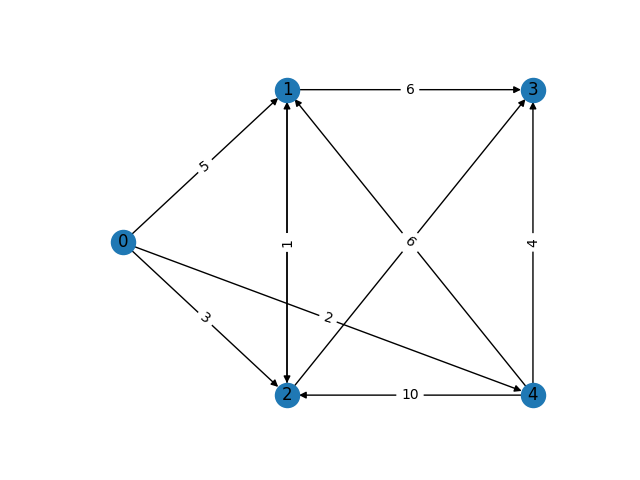
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Dijkstra’s SP** | Prim’s MST | DFT | **Heap Insertion** |
| **Big-O** (Zaman Karmaşıklığı  Big-O Notasyonuna Göre) | O(V+E) | O(V^2) | O(V + E) | O(logN) |

# 5. Graph Drawing and Finding Shortest Path in Python

## 5.1 Graph Drawing

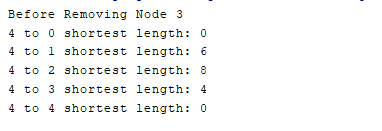
**import** networkx **as** nx  
**import** matplotlib.pyplot **as** plt  
graph = nx.DiGraph()  
nodes = [0,1,2,3,4]  
graph.add\_nodes\_from(nodes)  
weigted\_edges = [(0,1,5),(0,4,2),(0,2,3),(1,3,6),(2,1,1),(2,3,2),(4,1,6),(4,3,4),(4,2,10),(1,2,2)]  
graph.add\_weighted\_edges\_from(weigted\_edges)  
  
arc\_weight=nx.get\_edge\_attributes(graph,**'weight'**)  
graph.nodes[0][**'pos'**] = (0,0)  
graph.nodes[1][**'pos'**] = (2,2)  
graph.nodes[2][**'pos'**] = (2,-2)  
graph.nodes[3][**'pos'**] = (5,2)  
graph.nodes[4][**'pos'**] = (5,-2)  
node\_pos=nx.get\_node\_attributes(graph,**'pos'**)  
pos=nx.spring\_layout(graph)  
nx.draw\_networkx\_edge\_labels(graph, node\_pos, edge\_labels=arc\_weight)  
nx.draw(graph, node\_pos, node\_size=300, with\_labels=**True**)

plt.show()



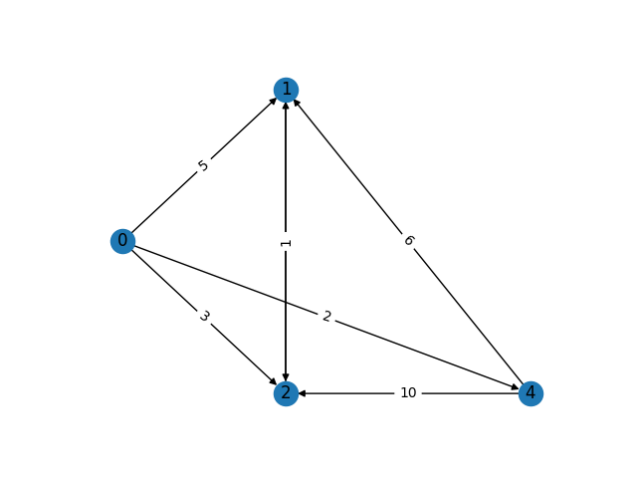
## 5.2 Finding Shortest Path

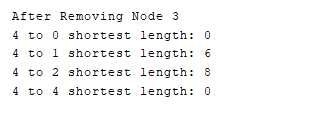
**for** x **in** range(0,5):  
 **try**:  
 sp = nx.dijkstra\_path\_length(graph,source = 4,target=x)  
 print( **"{} to {} shortest length: {}"**.format(4,x,sp))  
 **except** nx.exception.NetworkXNoPath:  
 print(**"{} to {} shortest length: 0"**.format(4,x))



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

graph.remove\_node(3)  
arc\_weight=nx.get\_edge\_attributes(graph,**'weight'**)  
pos=nx.spring\_layout(graph)  
nx.draw\_networkx\_edge\_labels(graph, node\_pos, edge\_labels=arc\_weight)  
nx.draw(graph, node\_pos, node\_size=300, with\_labels=**True**)  
print(**"After Removing Node 3"**)  
**for** x **in** range(0,5):  
 **try**:  
 **if**(x ==3):  
 **continue** sp = nx.dijkstra\_path\_length(graph,source = 4,target=x)  
 print( **"{} to {} shortest length: {}"**.format(4,x,sp))  
 **except** nx.exception.NetworkXNoPath:  
 print(**"{} to {} shortest length: 0"**.format(4,x))  
plt.show()

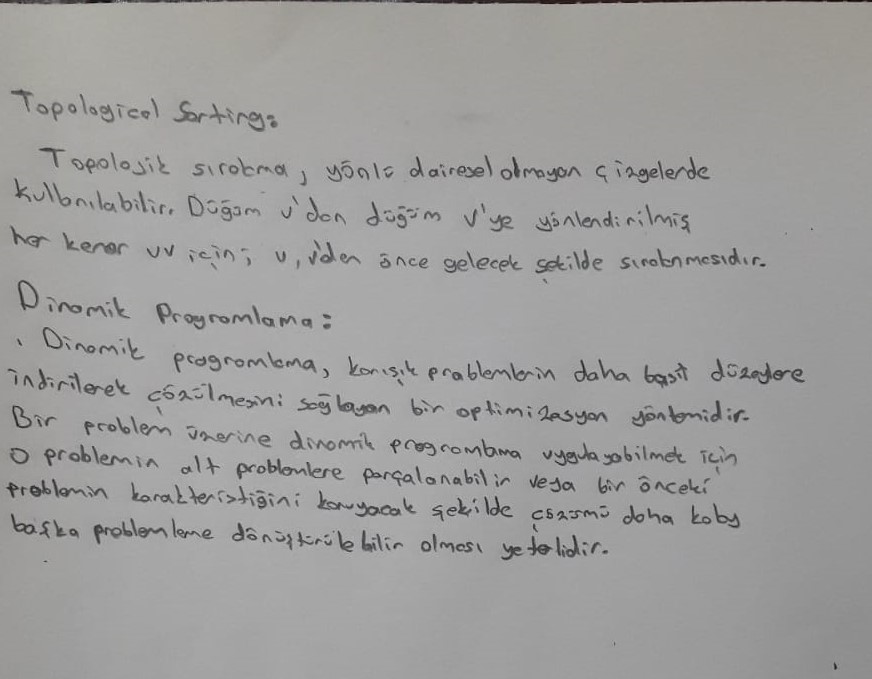


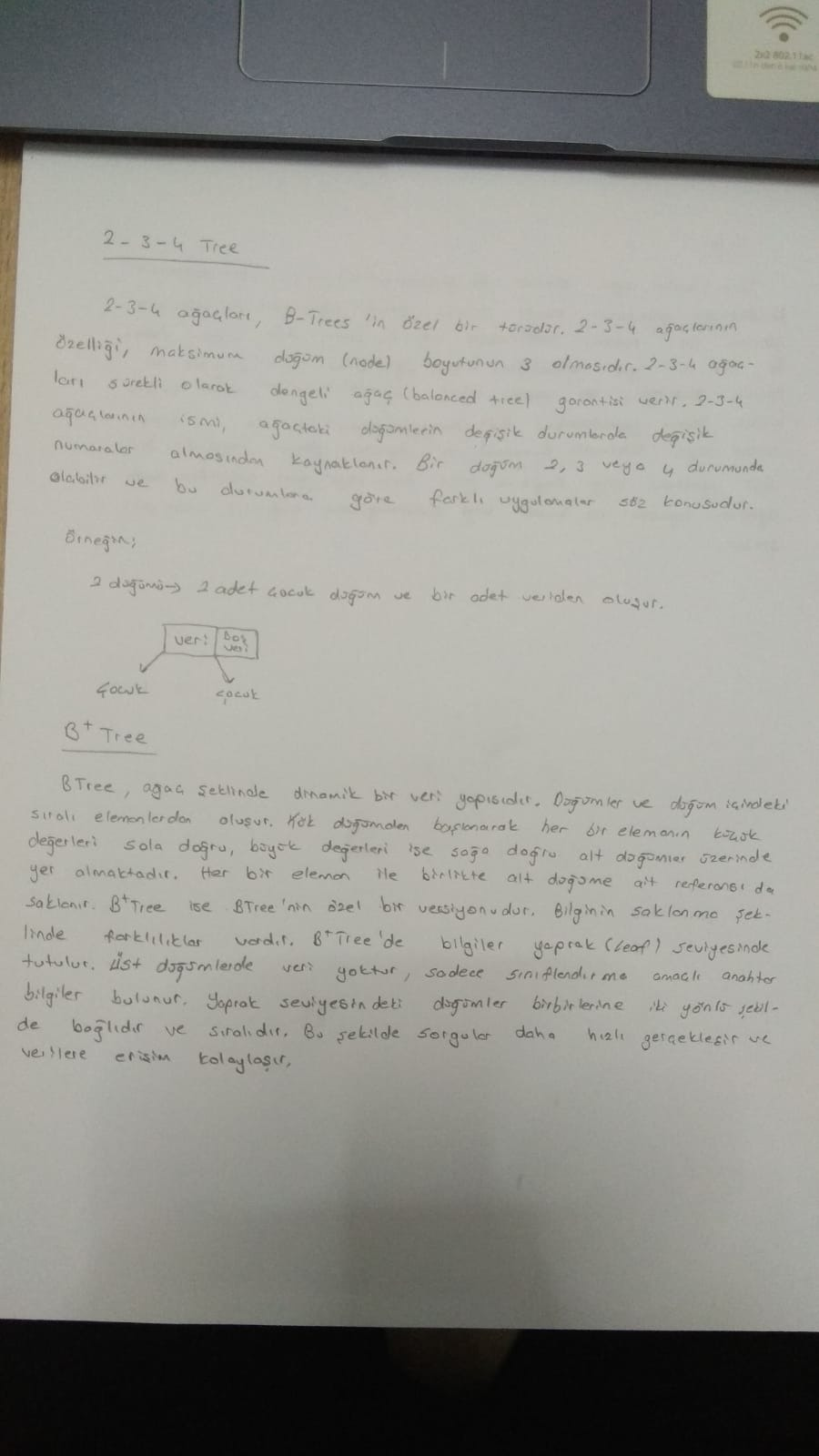


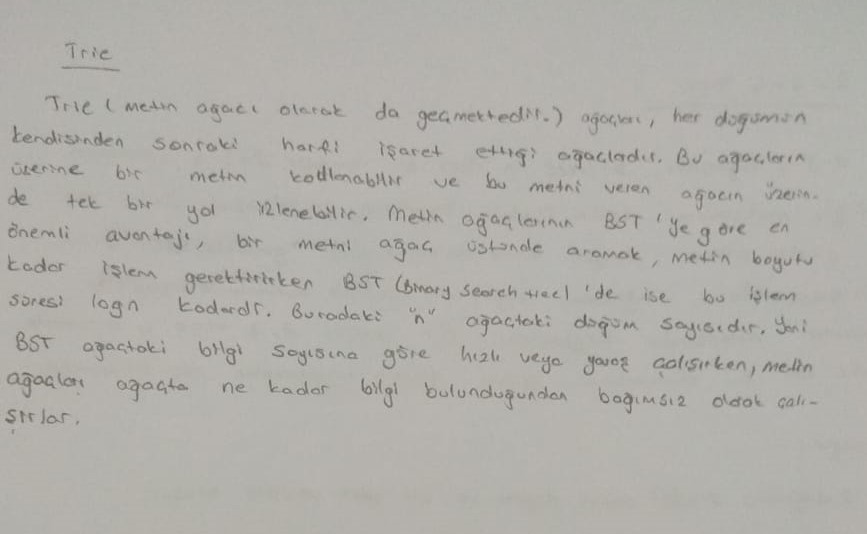
# 6. Comparison of MST Algorithms

Prim’in algoritması herhangi bir düğümden, Kruskal’ın algoritması ise ağırlığı en düşük olan düğümden başlar.Prim’in algoritmasında bir node birden fazla dolaşılabilir, Kruskal’da ise bir node sadece bir kez dolaşılabilir.Prim’in algoritması yoğun graphlarda daha hızlıyken Kruskal’ınki seyrek grafiklerde daha hızlıdır.Prim’in zaman karmaşıklığı O(V^2) iken Kruskal’ın zaman karmaşıklığı O(E logV)’dir.

# 7. Definitions







# SELF-ASSESSMENT TABLE

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Points** | **Estimated Grade** | **Explanation** |
| **1 a) AVL Tree** | **10** | **10** | **Ders videosu izlenerek ağaca ekleme işlemi yapıldı.** |
| **1 b) Heap** | **10** | **10** | **Bir önceki projeden bilgilerimizle ekleme ve çıkarma işlemi yapıldı.** |
| **2) B-Tree Insertion / AVL Tree Insertion / Red-Black Trees / Huffman Encoding Tree** | **10** | **10** | **Ders kitabı okunarak AVL ağaca ekleme metodu tasarlandı.** |
| 1. **Generating Huffman Encoding Tree** | **10** | **10** | **Ders videosu izlenerek adım adım ağaç oluşturuldu** |
| **4 a) Dijkstra’s shortest path code + test** | **5** | **5** | **Ders kitabı ve** |
| **4 b) Prim’s MST code + test** | **5** | **5** | **videosu izlenerek** |
| **4 c) BFT or DFT code + test** | **5** | **5** | **İstenen 3 yapı da tasarlandı.** |
| **4 d) Filling Big-O Table** | **5** | **5** | **Zaman karmaşıkları hesaplanarak tablo dolduruldu** |
| **5 i) Graph Drawing** | **5** | **5** | **Verilen web sitesi yardımı ile graph çizdirildi** |
| **5 ii) Finding Shortest Paths with Dijkstra’s** | **5** | **5** | **Verilen metod yardımıyla en kısa yol oluşturuldu.** |
| **5 iii) Node deletion and repeating i, ii.** | **5** | **5** | **Kütüphane silme metodu ile düğüm silindi ve işlemler tekrarlandı.** |
| **6) Comparison (Prim’s & Kruskal’s Algorithm)** | **5** | **5** | **Araştırılarak farkları öğrenilip karşılaştırıldı.** |
| **7) Explanations of 5 terms** | **10** | **10** | **Ders kitabı ve farklı kaynaklardan seçilen terimler açıklandı.** |
| **Demo Video for Source Codes and Tests of Q2, Q4 and Q5** | **5** | **5** | **Video çekildi.** |
| **Self-assessment Table** | **5** | **5** | **Tablo tamamlandı.** |
| **Total** | **100** | **100** |  |

**GÖREV DAĞILIMI:**

**Bütün sorular birlikte yapılmıştır.**

**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.**