

**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**

05190000043, Ahmet Hakan Demirel

İçindekiler

[GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS 2](#_Toc62986201)

[1. AVL Tree Insertions (drawing), Heap Node Insertion and Removing 2](#_Toc62986202)

[2. B-Tree Insertion Method [Alternatives: AVL-Tree Insertion, Red-Black Trees, Huffman Encoding Tree] 4](#_Toc62986203)

[2.1 Kalemle Yazılan Algoritma Görüntüsü 4](#_Toc62986204)

[2.2 Yöntemin (ekleme işleminin) anlatımı / açıklaması / adımları 4](#_Toc62986205)

[2.3 Kaynak Kod 4](#_Toc62986206)

[2.4 (İşletim + Test) Ekran Görüntüsü 6](#_Toc62986207)

[3. Huffman Encoding Tree Generation 6](#_Toc62986208)

[4. Graph Algorithms 7](#_Toc62986209)

[4.1 Dijkstra’s Shortest Path [Source code + Screenshot for test] 7](#_Toc62986210)

[4.2 Prim’s MST [Source code + Screenshot for test] 8](#_Toc62986211)

[4.3 BFT (Breadth-First Traverse) or DFT (Depth-First Traverse) [Source code + Screenshot for test] 9](#_Toc62986212)

[4.4 Big-O Table (Time Complexities) 10](#_Toc62986213)

[5. Graph Drawing and Finding Shortest Path in Python 10](#_Toc62986214)

[5.1 Graph Drawing 10](#_Toc62986215)

[5.2 Finding Shortest Path 11](#_Toc62986216)

[5.3 Vertex Removing and repeating previous steps (Drawing and Finding SP) 12](#_Toc62986217)

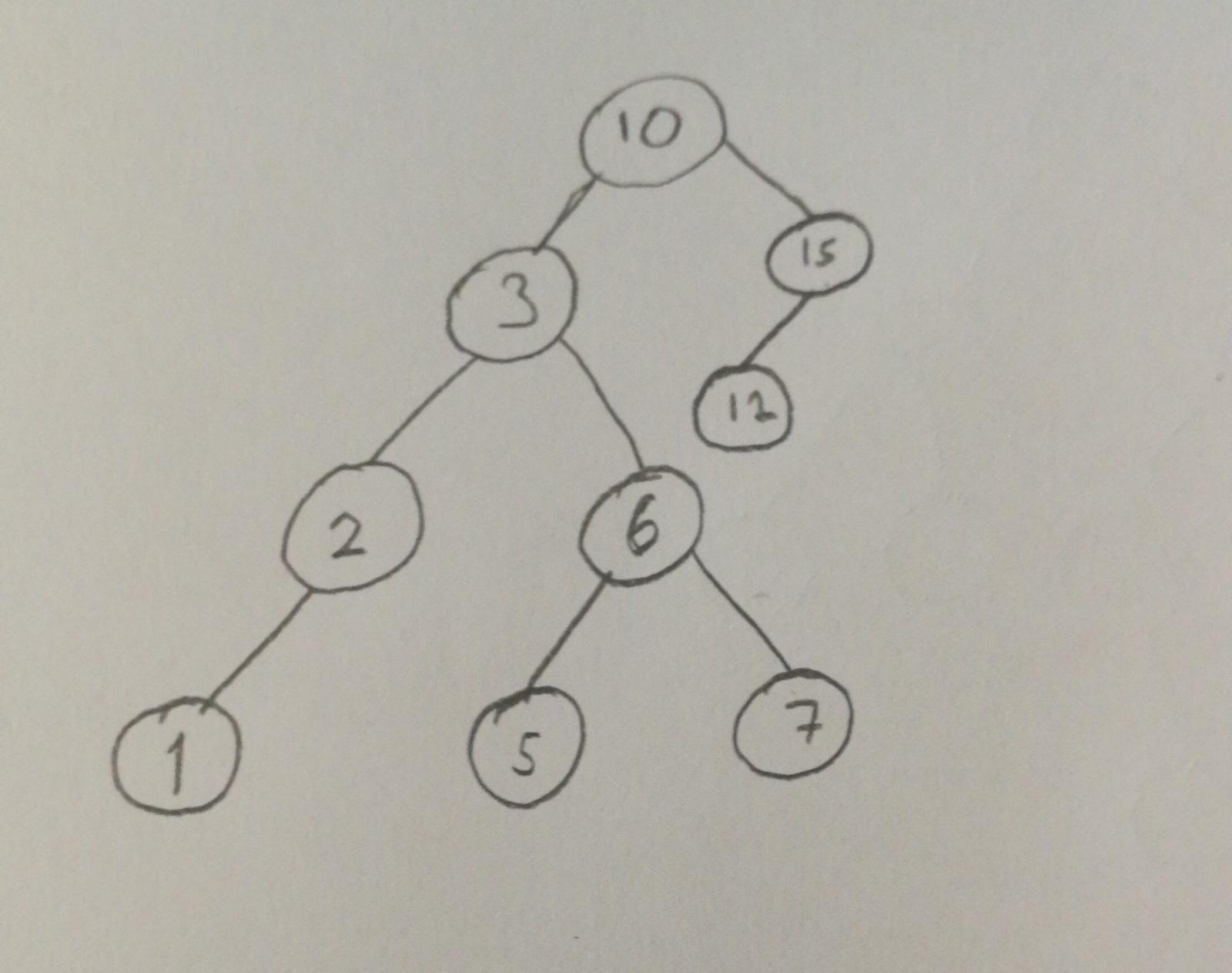
[6. Comparison of MST Algorithms 12](#_Toc62986218)

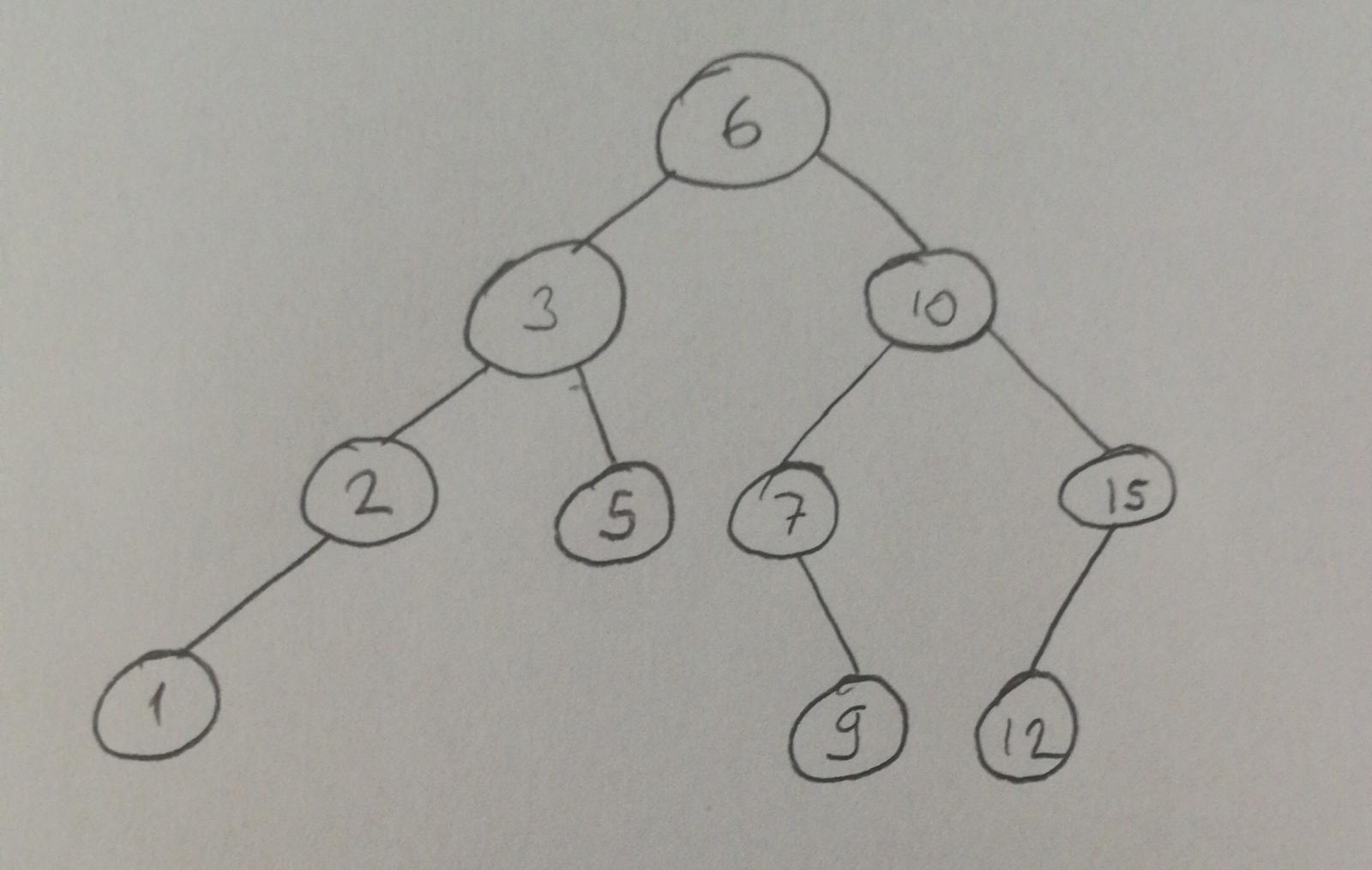
[7. Definitions 13](#_Toc62986219)

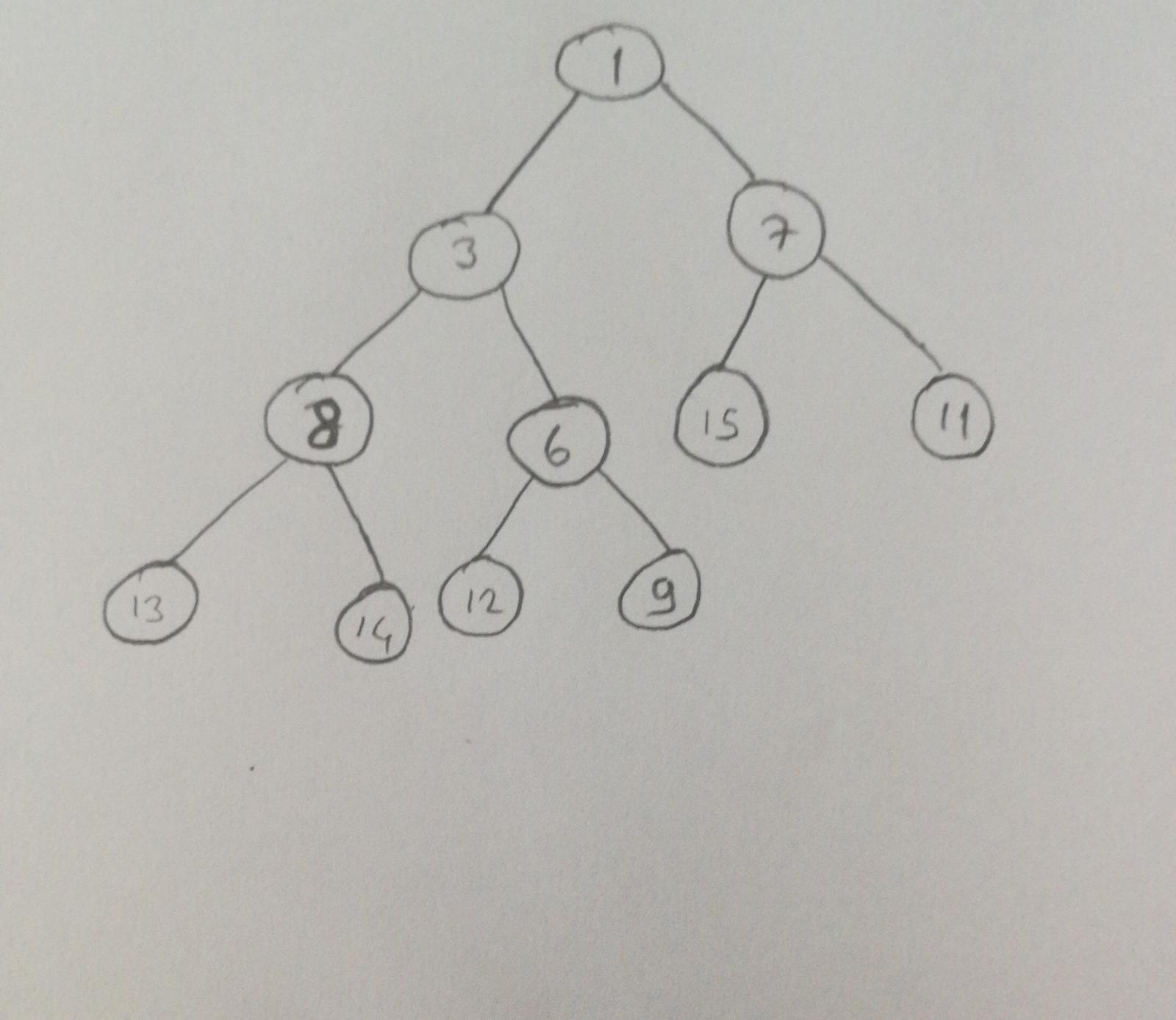
[SELF-ASSESSMENT TABLE 13](#_Toc62986220)

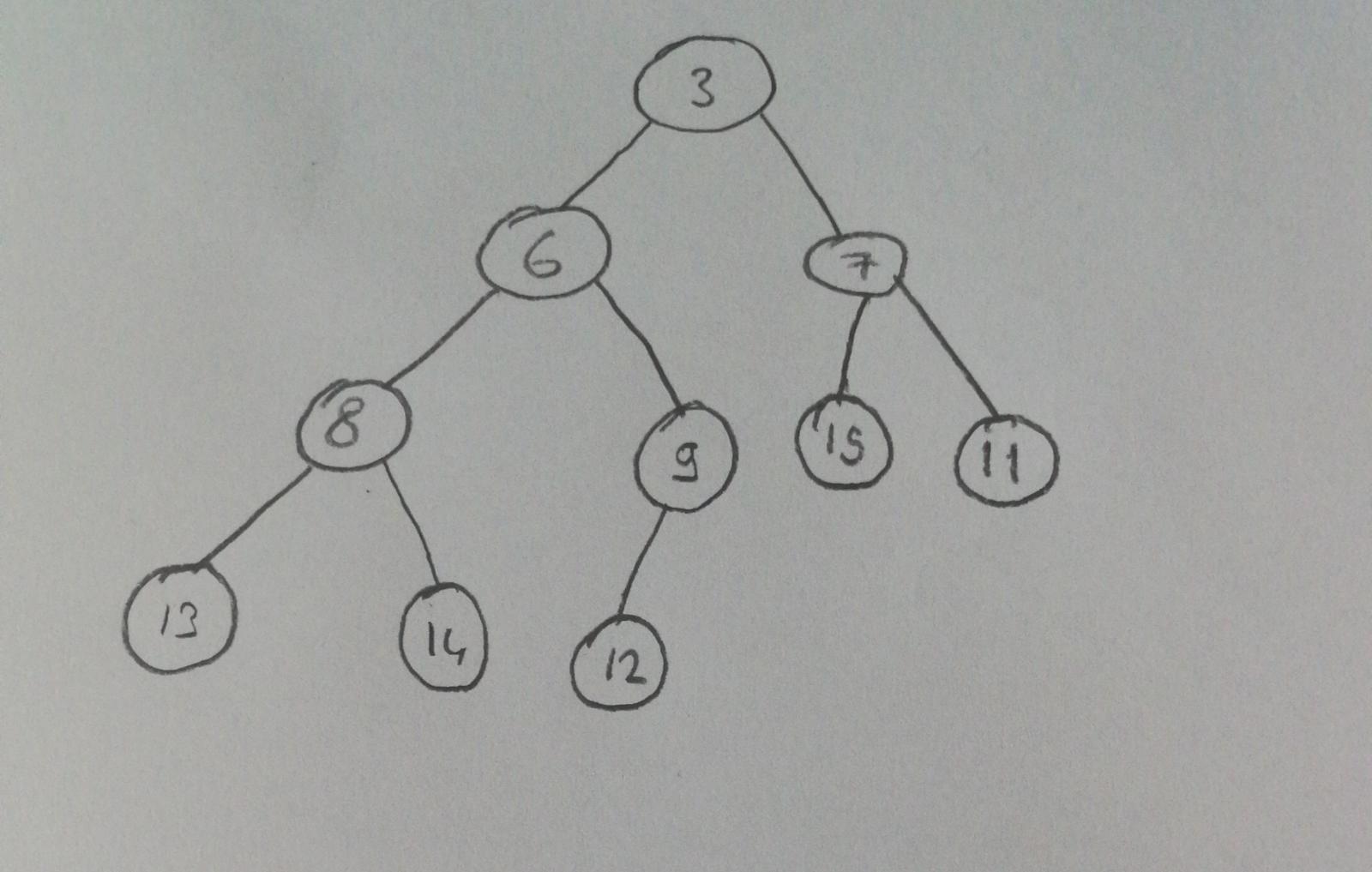
# GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS

# 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

## 2.3 Kaynak Kod

public class Node

{

public int key, height;

public Node left, right;

public Node(int d)

{

key = d;

height = 1;

}

}

public class AVLTree

{

public Node root;

// Ağacın yüksekliğini bulmak için kullandığımız fonksiyon

public int height(Node N)

{

if (N == null)

return 0;

return N.height;

}

// İki sayının maksimumunu bulma fonksiyonu

public int max(int a, int b)

{

return (a > b) ? a : b;

}

// Kökü y olan ağacı sağa döndürme fonksiyonu

public Node rightRotate(Node y)

{

Node x = y.left;

Node T2 = x.right;

x.right = y;

y.left = T2;

// Yükseklikleri güncelliyoruz

y.height = max(height(y.left),

height(y.right)) + 1;

x.height = max(height(x.left),

height(x.right)) + 1;

// Yeni rootu returnledik.

return x;

}

//Kökü x olan ağacı sola döndürme fonksiyonu

public Node leftRotate(Node x)

{

Node y = x.right;

Node T2 = y.left;

y.left = x;

x.right = T2;

x.height = max(height(x.left),

height(x.right)) + 1;

y.height = max(height(y.left),

height(y.right)) + 1;

return y;

}

// Balance faktörü alma fonksiyonu

public int getBalance(Node N)

{

if (N == null)

return 0;

return height(N.left) - height(N.right);

}

public Node insert(Node node, int key)

{

// BST'ye normal bir şekilde yerleştirme yapıyor.

if (node == null)

return (new Node(key));

if (key < node.key)

node.left = insert(node.left, key);

else if (key > node.key)

node.right = insert(node.right, key);

else // Yinelenen keylere izin vermiyor.

return node;

// Ata node'nin yüksekliği güncelleniyor.

node.height = 1 + max(height(node.left),

height(node.right));

int balance = getBalance(node);

if (balance > 1 && key < node.left.key)

return rightRotate(node);

if (balance < -1 && key > node.right.key)

return leftRotate(node);

if (balance > 1 && key > node.left.key)

{

node.left = leftRotate(node.left);

return rightRotate(node);

}

if (balance < -1 && key < node.right.key)

{

node.right = rightRotate(node.right);

return leftRotate(node);

}

return node;

}

public void preOrder(Node node)

{

if (node != null)

{

Console.Write(node.key + " ");

preOrder(node.left);

preOrder(node.right);

}

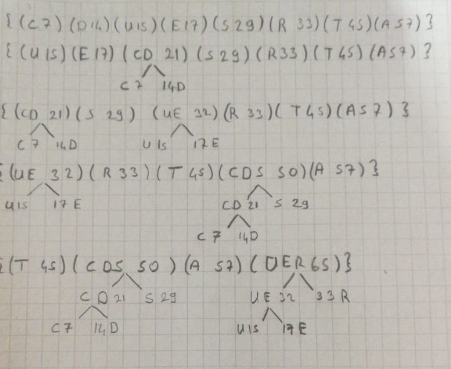
}

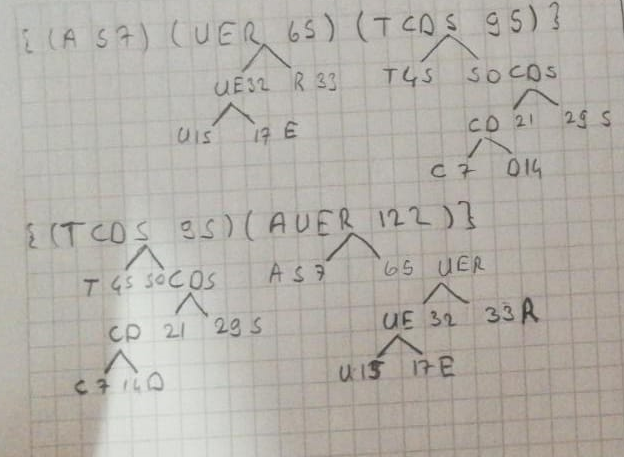
}

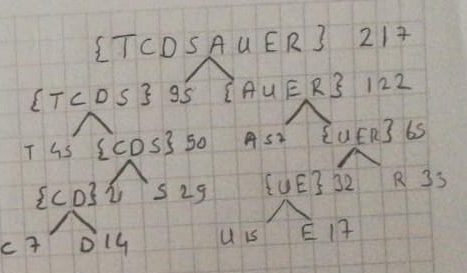
## 2.4 (İşletim + Test) Ekran Görüntüsü



# 3. Huffman Encoding Tree Generation







# 4. Graph Algorithms

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

class GFG

{

int V = 9;

int minDistance(int[] dist,

bool[] sptSet)

{

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;

}

void printSolution(int[] dist)

{

Console.Write("Vertex \t\t Kaynaktan "

+ "Uzaklık\n");

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

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

}

public void dijkstra(int[,] graph, int src)

{

int[] dist = new int[V];

bool[] sptSet = new bool[V];

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

{

dist[i] = int.MaxValue;

sptSet[i] = false;

}

dist[src] = 0;

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

{

int u = minDistance(dist, sptSet);

sptSet[u] = true;

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

if (!sptSet[v] && graph[u, v] != 0 && dist[u] != int.MaxValue && dist[u] + graph[u, v] < dist[v])

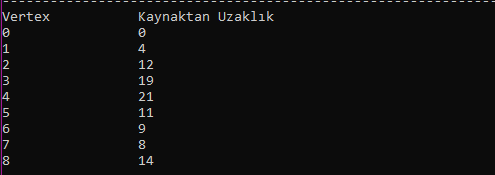
dist[v] = dist[u] + graph[u, v];

}

printSolution(dist);

}

}



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

class MST

{

public int V = 5;

public int minKey(int[] key, bool[] mstSet)

{

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;

}

public void printMST(int[] parent, int[,] graph)

{

Console.WriteLine("Kenar \tWeight ");

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

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

}

public void primMST(int[,] graph)

{

int[] parent = new int[V];

int[] key = new int[V];

bool[] mstSet = new bool[V];

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

{

key[i] = int.MaxValue;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

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

{

int u = minKey(key, mstSet);

mstSet[u] = true;

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

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

&& graph[u, v] < key[v])

{

parent[v] = u;

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

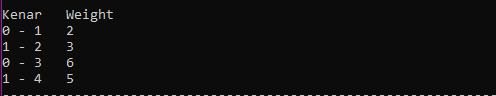
}

}

printMST(parent, graph);

}

}



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

class Graph

{

public int \_V;

public LinkedList<int>[] \_adj;

public Graph(int V)

{

\_adj = new LinkedList<int>[V];

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

{

\_adj[i] = new LinkedList<int>();

}

\_V = V;

}

public void AddEdge(int v, int w)

{

\_adj[v].AddLast(w);

}

public void BFS(int s)

{

bool[] visited = new bool[\_V];

for (int i = 0; i < \_V; i++)

visited[i] = false;

LinkedList<int> queue = new LinkedList<int>();

visited[s] = true;

queue.AddLast(s);

while (queue.Any())

{

s = queue.First();

Console.Write(s + " ");

queue.RemoveFirst();

LinkedList<int> list = \_adj[s];

foreach (var val in list)

{

if (!visited[val])

{

visited[val] = true;

queue.AddLast(val);

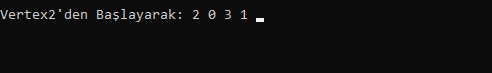
}

}

}

}

}



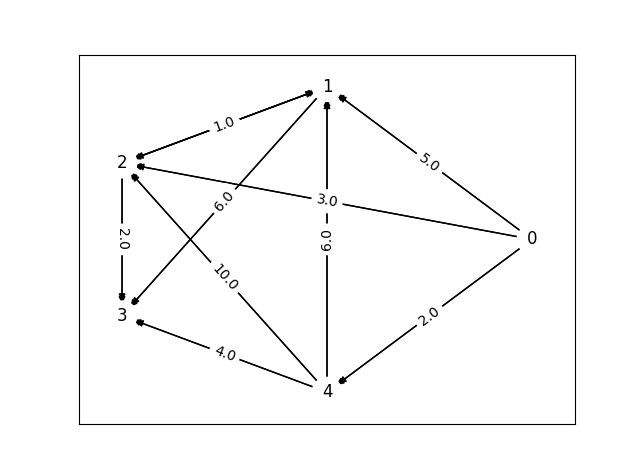
## 4.4 Big-O Table (Time Complexities)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Dijkstra’s SP** | Prim’s MST | BFT | **Heap Insertion** |
| **Big-O** (Zaman Karmaşıklığı  Big-O Notasyonuna Göre) | O(v^2) | O(v^2) | O(v^2) | O(1) |

# 5. Graph Drawing and Finding Shortest Path in Python

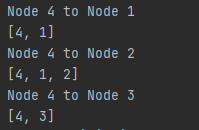
## 5.1 Graph Drawing

import networkx as nx  
import matplotlib.pyplot as plt  
  
graph\_1 = nx.DiGraph()  
  
graph\_1.add\_node(0)  
graph\_1.add\_node(1)  
graph\_1.add\_node(2)  
graph\_1.add\_node(3)  
graph\_1.add\_node(4)  
  
graph\_1.add\_edge(0, 1, weight=5.0)  
graph\_1.add\_edge(0, 2, weight=3.0)  
graph\_1.add\_edge(0, 4, weight=2.0)  
graph\_1.add\_edge(1, 2, weight=2.0)  
graph\_1.add\_edge(1, 3, weight=6.0)  
graph\_1.add\_edge(2, 1, weight=1.0)  
graph\_1.add\_edge(2, 3, weight=2.0)  
graph\_1.add\_edge(4, 1, weight=6.0)  
graph\_1.add\_edge(4, 2, weight=10.0)  
graph\_1.add\_edge(4, 3, weight=4.0)  
  
graph\_1.nodes[0]['pos'] = (0, 0)  
graph\_1.nodes[1]['pos'] = (-2, 2)  
graph\_1.nodes[2]['pos'] = (-4, 1)  
graph\_1.nodes[3]['pos'] = (-4, -1)  
graph\_1.nodes[4]['pos'] = (-2, -2)  
  
node\_pos = nx.get\_node\_attributes(graph\_1, 'pos')  
arc\_weight = nx.get\_edge\_attributes(graph\_1, 'weight')  
  
node\_color = ['white' for node in graph\_1.nodes()]  
edge\_color = ['black' for edge in graph\_1.edges()]  
  
nx.draw\_networkx(graph\_1, node\_pos, node\_color=node\_color, node\_size=450)  
nx.draw\_networkx\_edges(graph\_1, node\_pos , edge\_color= edge\_color)  
nx.draw\_networkx\_edge\_labels(graph\_1, node\_pos, edge\_labels=arc\_weight)  
plt.show()



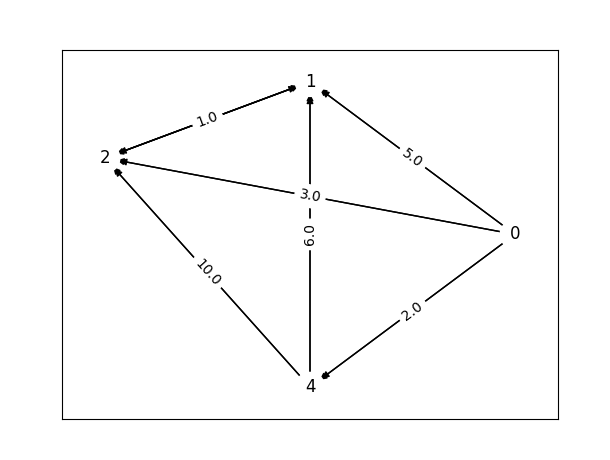
## 5.2 Finding Shortest Path

sp = nx.dijkstra\_path(graph\_1,source = 4, target = 1)  
print("Node 4 to Node 1")  
print(sp)  
sp = nx.dijkstra\_path(graph\_1,source = 4, target = 2)  
print("Node 4 to Node 2")  
print(sp)  
sp = nx.dijkstra\_path(graph\_1,source = 4, target = 3)  
print("Node 4 to Node 3")  
print(sp)



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

graph\_1.remove\_node(3)  
graph\_1.remove\_nodes\_from("spam")  
  
print("3. NODE SİLİNDİKTEN SONRA")  
  
node\_pos = nx.get\_node\_attributes(graph\_1, 'pos')  
arc\_weight = nx.get\_edge\_attributes(graph\_1, 'weight')  
  
node\_color = ['white' for node in graph\_1.nodes()]  
edge\_color = ['black' for edge in graph\_1.edges()]  
  
nx.draw\_networkx(graph\_1, node\_pos, node\_color=node\_color, node\_size=450)  
nx.draw\_networkx\_edges(graph\_1, node\_pos , edge\_color= edge\_color)  
nx.draw\_networkx\_edge\_labels(graph\_1, node\_pos, edge\_labels=arc\_weight)  
plt.show()  
  
sp = nx.dijkstra\_path(graph\_1,source = 4, target = 1)  
print("Node 4 to Node 1")  
print(sp)  
sp = nx.dijkstra\_path(graph\_1,source = 4, target = 2)  
print("Node 4 to Node 2")  
print(sp)



# 6. Comparison of MST Algorithms

Prim’in algororitması Grafikteki herhangi bir tepe noktasından Minimum Genişleme Ağacı (Minimum Spanning Tree) oluşturmaya başlar. Kruskal’ın algoritması ise Grafikte minimum ağırlığı taşıyan tepe noktasından Minimum Genişleme Ağacı (Minimum Spanning Tree) oluşturmaya başlar. Ayrıca Prim’in algoritması minimum mesafeyi elde etmek için bir düğümü birden fazla kez geçerken Kruskal’ın algoritması bir düğümü yalnızca bir kez geçer. Prim'in algoritması, yalnızca bağlı grafikte çalıştığı gibi bağlantılı bileşeni de verir. Kruskal’ın algoritması her an bağlantısı kesilen bileşenler oluşturabilir ve bağlantısı kesilen bileşenler üzerinde çalışabilir.

# 7. Definitions

Dynamic Programming: Karmaşık bir problemi tekrarlanan alt problemlere bölerek, her bir alt problemi bir kere çözüp daha sonra bu çözümü kaydederek karmaşık problemin çözümünde kullanma yöntemidir.

Quadratic Probic: Özellikle özetleme fonksiyonlarının bilgileri sınıflandırması sırasında kullanılan formülün ikinci dereceden olması durumudur.Özetleme fonksiyonlarında, sık kullanılan linear probing yönteminin tersine, bir bilgiyi tasnif ederken, ardışık olarak veriler üzerinde hareket etmez, bunun yerine her defasında baktığı uzaklığı ikinci dereceden bir denklem ile arttırır.

2-3-4 Tree: 2-3-4 ağacı, b-trees’in özel bir halidir. bu ağacın özelliği, düğüm boyutunun 3 ile sınırlı olmasıdır. ağaç ayrıca sürekli olarak dengeli bir ağaç garantisi verir. 2-3-4 ağaçları, red-black trees’in eş şekillisi olarak da düşünülebilir.

Trie: Bir küme içinden belirli anahtarları bulmak için kullanılan bir ağaç veri yapısı olan bir arama ağacı türüdür.

B+ Tree: Düğüm başına değişken ancak genellikle çok sayıda çocuğu olan m-ary ağacıdır. Bir B + ağacı bir kök, iç düğümler ve yapraklardan oluşur. Kök, bir yaprak veya iki veya daha fazla çocuğu olan bir düğüm olabilir.

# 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** | **5** | **Yapıldı. Kağıda el ile yazılmadı.** |
| 1. **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** | **3** | **Video çekemedim.** |
| **Self-assessment Table** | **5** | **5** | **Yapıldı** |
| **Total** | **100** | **93** |  |

**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.**