

**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**

05200000099, Arda Bozkurt

05200000043, Emir Gültekin

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 5](#_Toc93185211)

[2.a B-Tree ekleme method (ya da AVL-Tree ekleme method/Red-Black tree/Huffman encoding tree) kodu 6](#_Toc93185212)

[2.b B-Tree ekleme kodunun adım adım açıklanması 11](#_Toc93185213)

[3.a Dijkstra kodu ve yapılan testler 11](#_Toc93185214)

[3.b Prim MST kodu ve yapılan testler 11](#_Toc93185215)

[3.c BFT ya da DFT kodu ve yapılan testler 13](#_Toc93185216)

[3.d Verilen Big-O tablosunun doldurulmuş hali 16](#_Toc93185217)

[4.a Verilen çizgenin python ortamında grafiksel olarak oluşturan kod 16](#_Toc93185218)

[4.b Verilen çizgede en kısa yol bulma sonuçları 16](#_Toc93185219)

[5.a. Prim ve Kruskal algoritması karşılaştırması 18](#_Toc93185220)

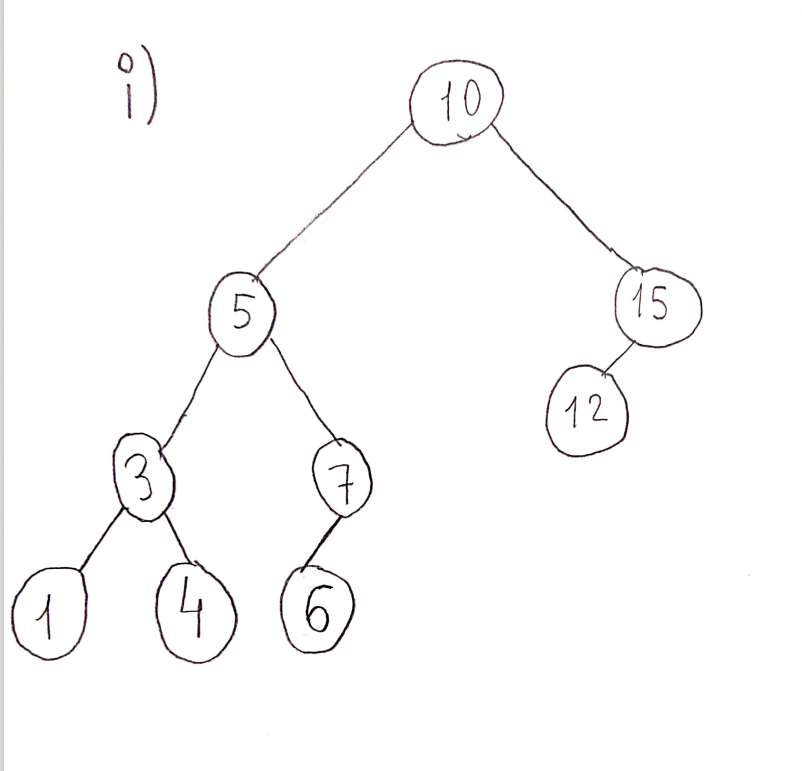
[5.b Verilen çizgede en kısa yol bulma sonuçları 18](#_Toc93185221)

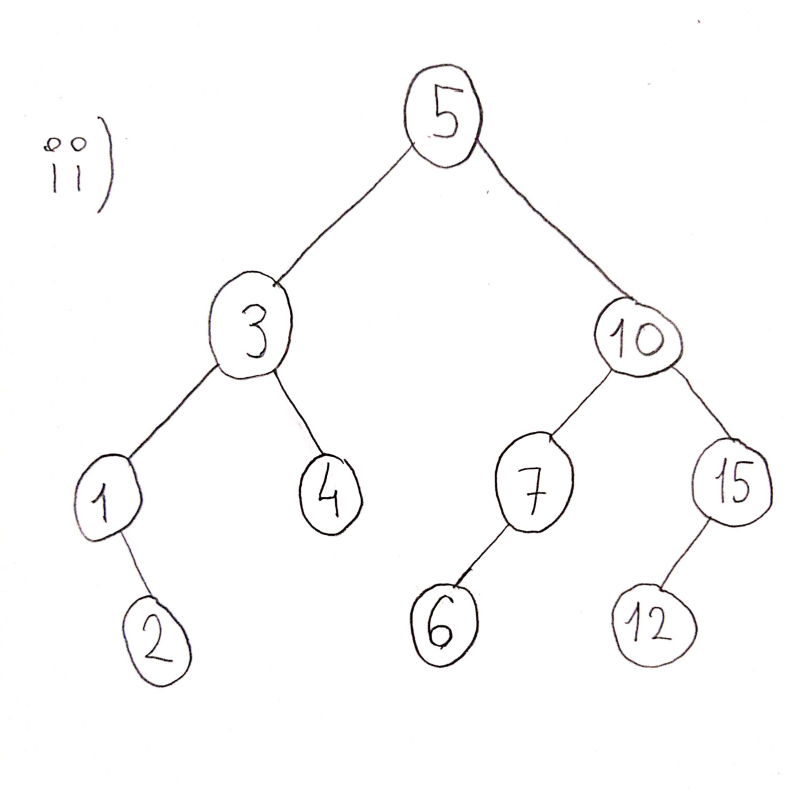
[Özdeğerlendirme Tablosu 21](#_Toc93185222)

GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS

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

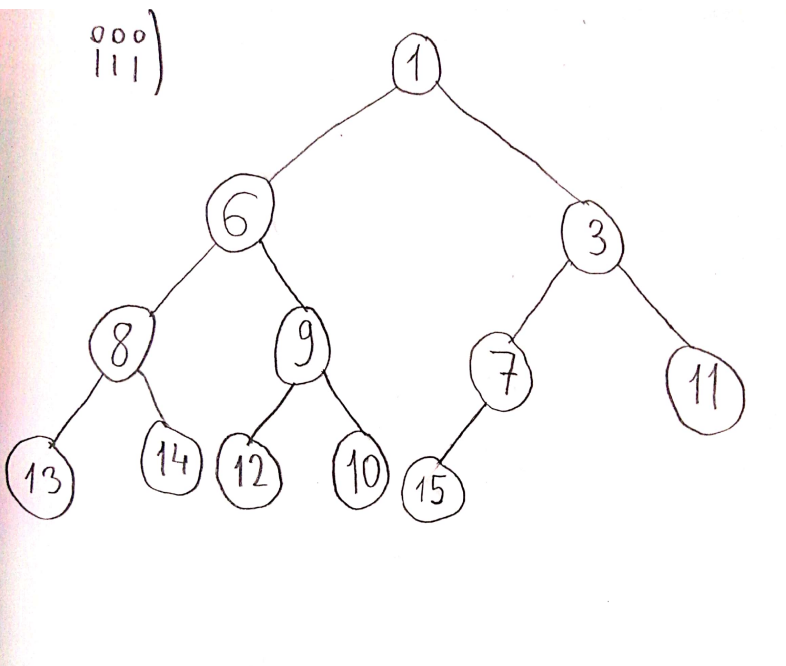
// i. ve ii. maddelerde istenilen ağaçların çizimini ekleyiniz

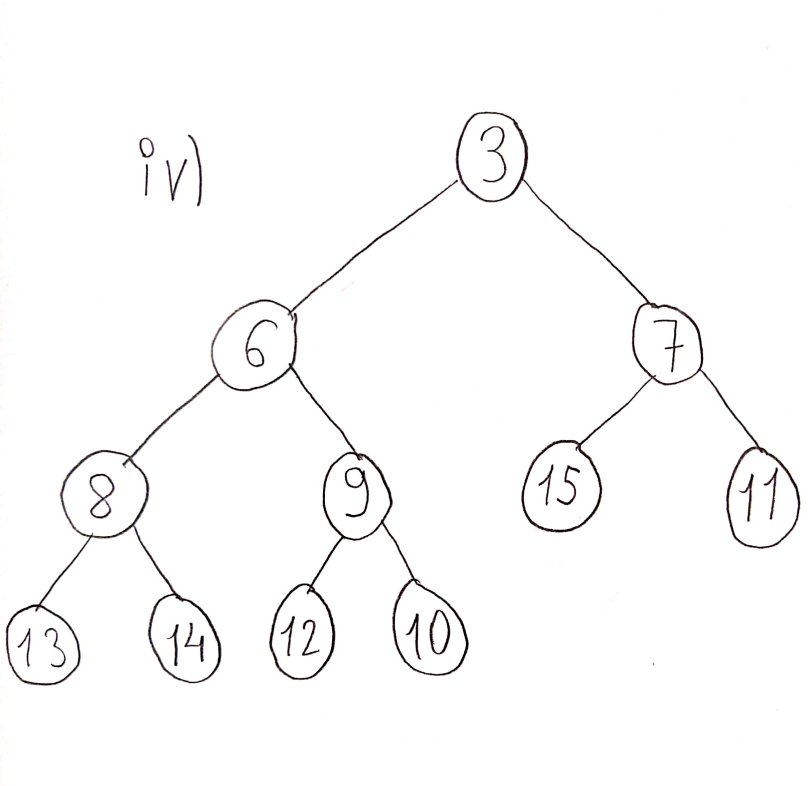




## 1.b Heapa istenilen değerin eklendikten ve silme işlemi yapıldıktan sonra heapin çizimi

// iii. ve iv. maddelerde istenilen işlemlerden sonra heapin çizimlerini ekleyiniz 1.b.2 Ekran görüntüleri





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

// öğrenip gerçekleştirdiğiniz method ekleyiniz

public class AVL {

Node root;

//

int height(Node N) {

if (N == null)

return 0;

return N.height;

}

// 2 integer arasındaki maximum değeri bulan fonksiyon

int max(int a, int b) {

return (a > b) ? a : b;

}

// y ile köklenmiş bir alt-ağacı sağa döndürme methodu.

Node rightRotate(Node y) {

Node x = y.left; // Tanımlar.

Node T2 = x.right;

// Döndürme işlemi

x.right = y;

y.left = T2;

// Yüksekliklerini güncelleme işlemi.

y.height = max(height(y.left), height(y.right)) + 1;

x.height = max(height(x.left), height(x.right)) + 1;

// Alt ağacın yeni kökünü geri döndürüyor.

return x;

}

// x ile köklenmiş bir alt-ağacı sola döndürme methodu.

Node leftRotate(Node x) {

Node y = x.right;

Node T2 = y.left;

// Döndürme işlemi

y.left = x;

x.right = T2;

// Yüksekliklerini güncelleme işlemi.

x.height = max(height(x.left), height(x.right)) + 1;

y.height = max(height(y.left), height(y.right)) + 1;

// Alt ağacın yeni kökünü geri döndürüyor.

return y;

}

// Verilen düğümün denge faktörünü bulma işlemi.

int getBalance(Node N) {

if (N == null)

return 0;

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

}

Node insert(Node node, int key) {

/\* 1. Binary search tree'ye göre yerleştirme işlemi. \*/

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

return node;

/\* 2. Ata düğümün yüksekliğini güncelliyoruz. \*/

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

height(node.right));

/\* 3. Ata düğümün denge faktörünü ölçüyoruz eğer dengesiz ise ona göre AVL Tree'ye uyduracağız. \*/

int balance = getBalance(node);

// Eğer düğüm dengesiz ise döndürme işlemlerinden birini yapıp dengeli yapacağız.

//Left Left problemi

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

return rightRotate(node);

// Right Right problemi

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

return leftRotate(node);

// Left Right problemi

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

node.left = leftRotate(node.left);

return rightRotate(node);

}

// Right Left problemi

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

node.right = rightRotate(node.right);

return leftRotate(node);

}

return node;

}

<https://www.geeksforgeeks.org/avl-tree-set-1-insertion/> //KAYNAK//

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

// yazdığınız kodu adım adım açıklayınız

İlk önce Binary search Tree gibi düşünülüp uygun yer bulunup insert edilir.

2.adım olarak ata düğümünün yüksekliği güncellenir.

Oluşan yeni durumda denge faktörü bulunur.

--Eğer denge faktörü 1 den büyükse solda düğüm birikmesi olmuştur.

Ve eğer Düğümün de solundaysa bu left left durumudur sağa döndürme ile düzeltilir.

--Eğer denge faktörü 1 den küçükse sağda düğüm birikmesi olmuştur.

Ve eğer Düğümün de sağındaysa bu right right problemidir sola döndürme ile düzeltilir.

--Eğer denge faktörü 1 den büyükse ve key düğümün sağındaysa bu left right problemidir.

Bu solda birikme anlamına gelir fakat düğümün sağında olduğu için ilk önce sola döndürülür sonra sağa döndürülür.

--Eğer denge faktörü 1 den küçükse ve key düğümün solundaysa

Bu sağda birikme anlamına gelir fakat düğümün solunda olduğu için önce sağa döndürülür sonra sola döndürülür.

Dengesizliklerin hepsini bu sırada kontrol edip uygun işlemler yapılır.

## 3.a Dijkstra kodu ve yapılan testler

int[,] cost = {

{ INFINITY, 5, 3, INFINITY, 2},

{ INFINITY, INFINITY, 2, 6, INFINITY},

{ INFINITY, 1, INFINITY, 2, INFINITY},

{ INFINITY, INFINITY, INFINITY, INFINITY, INFINITY},

{ INFINITY, 6, 10, 4, INFINITY} };

int[] distances = new int[N];

int[] previous = Distance(N, cost, distances, SRC);

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

if (distances[i] != INFINITY)

Console.WriteLine(distances[i]);

else Console.WriteLine("INFINITY");

public static int[] Distance(int N, int[,] cost, int[] D, int src)

{

int w, v, min;

bool[] visited = new bool[N];

int[] previous = new int[N]; //for tracking shortest paths (güzergah)

//initialization of D[], visited[] and previous[] arrays according to src node

for (v = 0; v < N; v++)

{

if (v != src)

{

visited[v] = false;

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

if (D[v] != INFINITY) //there is a connection between src and v

{

previous[v] = src;

}

else //no path from source

{

previous[v] = -1;

}

}

else

{

visited[v] = true;

D[v] = 0;

previous[v] = -1;

}

}

// Searching for shortest paths

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

{

min = INFINITY;

for (w = 0; w < N; w++)

if (!visited[w])

if (D[w] < min)

{

v = w;

min = D[w];

}

visited[v] = true;

for (w = 0; w < N; w++)

if (!visited[w])

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

{

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

previous[w] = v; //update the path info

}

}

return previous;

}

## }

## 3.b Prim MST kodu ve yapılan testler

int[,] graph = new int[,] { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

## { 0, 5, 7, 9, 0 } };

// Number of vertices in the graph

static int V = 5;

// A utility function to find

// the vertex with minimum key

// value, from the set of vertices

// not yet included in MST

static 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;

}

static 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++)

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

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

{

parent[v] = u;

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

}

}

}

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

int N = 5;

int SRC = 0;

Graph g = new Graph(4);

g.AddEdge(0, 1);

g.AddEdge(0, 2);

g.AddEdge(1, 2);

g.AddEdge(2, 0);

g.AddEdge(2, 3);

g.AddEdge(3, 3);

Console.Write("Following is Breadth First " +

"Traversal(starting from " +

"vertex 2)\n");

g.BFS(2);

class Graph

{

// No. of vertices

private int \_V;

//Adjacency Lists

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;

}

// Function to add an edge into the graph

public void AddEdge(int v, int w)

{

\_adj[v].AddLast(w);

}

// Prints BFS traversal from a given source s

public void BFS(int s)

{

// Mark all the vertices as not

// visited(By default set as false)

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

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

visited[i] = false;

// Create a queue for BFS

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

// Mark the current node as

// visited and enqueue it

visited[s] = true;

queue.AddLast(s);

while (queue.Any())

{

// Dequeue a vertex from queue

// and print it

s = queue.First();

Console.Write(s + " ");

queue.RemoveFirst();

// Get all adjacent vertices of the

// dequeued vertex s. If a adjacent

// has not been visited, then mark it

// visited and enqueue it

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

foreach (var val in list)

{

if (!visited[val])

{

visited[val] = true;

queue.AddLast(val);

}

}

}

}

}

}

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

## (E = number of edges, V = nubmer of vertices)

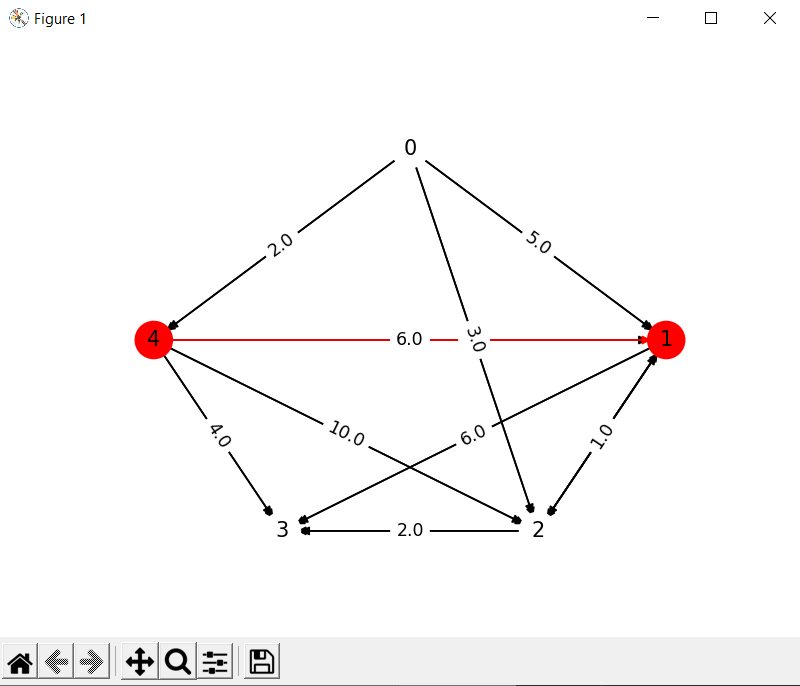
## tablo içeren bir resim Açıklama otomatik olarak oluşturuldu

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

G1=nx.DiGraph()  
  
list\_arcs = [(0, 1, 5.0), (0, 2, 3.0), (0, 4, 2.0), (1, 2, 2.0), (1, 3, 6.0),  
 (2, 1, 1.0), (2, 3, 2.0), (4, 1, 6.0), (4, 2, 10.0), (4, 3, 4.0)]  
  
G1.add\_weighted\_edges\_from(list\_arcs)  
G1.edges()

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

for i in range(5):  
 try:  
 sp = nx.dijkstra\_path(G1, source=4, target=i)  
 print(sp)  
 G1.\_node[0]['pos'] = (0, 3)  
 G1.\_node[1]['pos'] = (2, 0)  
 G1.\_node[2]['pos'] = (1, -3)  
 G1.\_node[3]['pos'] = (-1, -3)  
 G1.\_node[4]['pos'] = (-2, 0)  
 # The positions of each node are stored in a dictionary  
 node\_pos = nx.get\_node\_attributes(G1, 'pos')  
 # The edge weights of each arcs are stored in a dictionary  
 arc\_weight = nx.get\_edge\_attributes(G1, 'weight')  
 # Create a list of arcs in the shortest path using the zip command and store it in red edges  
 red\_edges = list(zip(sp, sp[1:]))  
 # If the node is in the shortest path, set it to red, else set it to white color  
 node\_col = ['white' if not node in sp else 'red' for node in G1.nodes()]  
 # If the edge is in the shortest path set it to red, else set it to white color  
 edge\_col = ['black' if not edge in red\_edges else 'red' for edge in G1.edges()]  
 # Draw the nodes  
 nx.draw\_networkx(G1, node\_pos, node\_color=node\_col, node\_size=450)  
 # Draw the node labels  
 # nx.draw\_networkx\_labels(G1, node\_pos,node\_color= node\_col)  
 # Draw the edges  
 nx.draw\_networkx\_edges(G1, node\_pos, edge\_color=edge\_col)  
 # Draw the edge labels  
 nx.draw\_networkx\_edge\_labels(G1, node\_pos, edge\_labels=arc\_weight)  
 # Remove the axis  
 plt.axis('off')  
 # Show the plot  
 plt.show()  
  
 except:  
 print("No path available")  
G1.remove\_node(1)  
for j in range(4):  
 try:  
 sp = nx.dijkstra\_path(G1, source=4, target=j)  
 print(sp)  
 G1.\_node[0]['pos'] = (0, 3)  
 G1.\_node[2]['pos'] = (1, -3)  
 G1.\_node[3]['pos'] = (-1, -3)  
 G1.\_node[4]['pos'] = (-2, 0)  
 # The positions of each node are stored in a dictionary  
 node\_pos = nx.get\_node\_attributes(G1, 'pos')  
 # The edge weights of each arcs are stored in a dictionary  
 arc\_weight = nx.get\_edge\_attributes(G1, 'weight')  
 # Create a list of arcs in the shortest path using the zip command and store it in red edges  
 red\_edges = list(zip(sp, sp[1:]))  
 # If the node is in the shortest path, set it to red, else set it to white color  
 node\_col = ['white' if not node in sp else 'red' for node in G1.nodes()]  
 # If the edge is in the shortest path set it to red, else set it to white color  
 edge\_col = ['black' if not edge in red\_edges else 'red' for edge in G1.edges()]  
 # Draw the nodes  
 nx.draw\_networkx(G1, node\_pos, node\_color=node\_col, node\_size=450)  
 # Draw the node labels  
 # nx.draw\_networkx\_labels(G1, node\_pos,node\_color= node\_col)  
 # Draw the edges  
 nx.draw\_networkx\_edges(G1, node\_pos, edge\_color=edge\_col)  
 # Draw the edge labels  
 nx.draw\_networkx\_edge\_labels(G1, node\_pos, edge\_labels=arc\_weight)  
 # Remove the axis  
 plt.axis('off')  
 # Show the plot  
 plt.show()  
 except (networkx.exception.NetworkXNoPath):  
 print("No path available")

Örneklerden biri

## 5.a. Prim ve Kruskal algoritması karşılaştırması

// karşılaştırmanızı buraya yazınız

Bilgisayar biliminde, Prim's ve Kruskal'ın algoritmaları, bağlı ağırlıklı bir yönsüz grafik için minimum yayılan ağacı bulan bir algoritmadır. Yayılan ağaç, grafiğin her bir düğümü bir ağaç olan bir yolla bağlı olacak şekilde bir grafiğin alt grafiğidir. Her yayılan ağacın bir ağırlığı vardır ve tüm yayılan ağaçların olası minimum ağırlıkları/maliyeti minimum yayılan ağaçtır.

Prim’in algoritması düğümle başlar iken Kruskal’ın algoritması bir kenarla başlar.

Primin’in algoritması O(n ²) iken Kruskal’ın algoritması O(logn) şeklindedir.

Prim algoritmasında, grafik bağlı bir grafik olmalı, Kruskal'lar ise bağlantısı kesilmiş grafiklerde de çalışabilir.

Prim'in algoritmaları bir düğümden diğerine yayılırken, Kruskal'ın algoritması kenarları kenarın konumu son adıma dayanmayacak şekilde seçer.

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

// verilen terimlerden seçtiğiniz 5 tanesini burada açıklayınız.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

## Özdeğerlendirme Tablosu

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