

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

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

**2022–2023 FALL SEMESTER**

**PROJECT-4 REPORT**

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

**DELIVERY DATE**

05/01/2023

**PREPARED BY**

05210000281 - Salih Ali Kamacı

İçindekiler

[1) AVL ağacına istenilen değerlerin eklenerek ve silinerek oluşan yeni AVL ağaçlarının çizimleri 3](#_Toc121763930)

[2.a) B-Tree ekleme method (ya da AVL-Tree ekleme method/Red-Black tree/Huffman encoding tree) kodu 4](#_Toc121763931)

[2.b) B-Tree ekleme kodunun adım adım açıklanması 4](#_Toc121763932)

[3.a) Dijkstra kodu ve yapılan testler 5](#_Toc121763933)

[3.b) Prim MST kodu ve yapılan testler 5](#_Toc121763934)

[3.c) BFT ya da DFT kodu ve yapılan testler 8](#_Toc121763935)

[3.d) Verilen Big-O tablosunun doldurulmuş hali 9](#_Toc121763936)

[4.a Verilen çizgenin python ortamında grafiksel olarak oluşturan kod ve ekran görüntüsü 11](#_Toc121763937)

[4.b Verilen çizgede en kısa yol bulma sonuçları 11](#_Toc121763938)

[4.c Düğüm slime ve yeniden çizdirme 13](#_Toc121763939)

[5.a. Prim ve Kruskal algoritması karşılaştırması 13](#_Toc121763940)

[5.b Trie veri yapısı kodu ve açıklaması 14](#_Toc121763941)

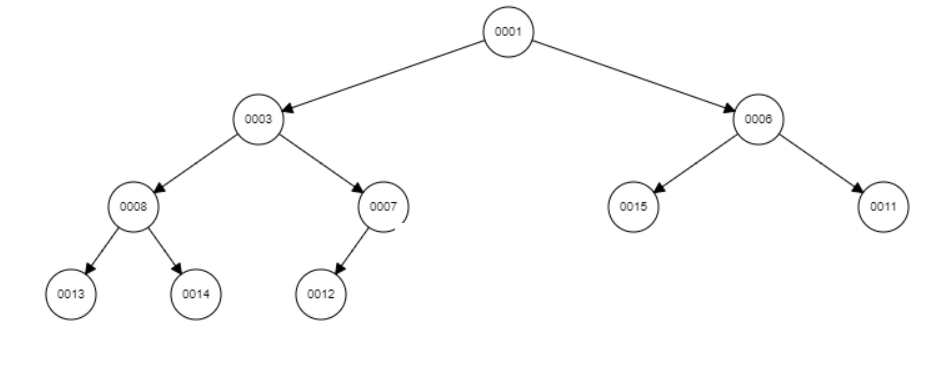
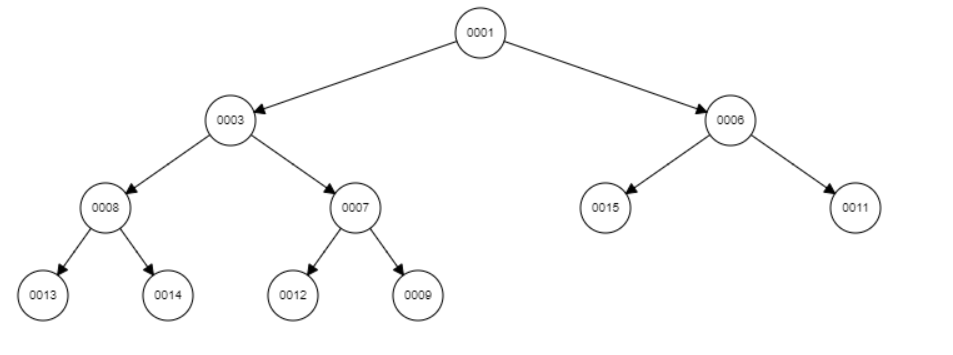
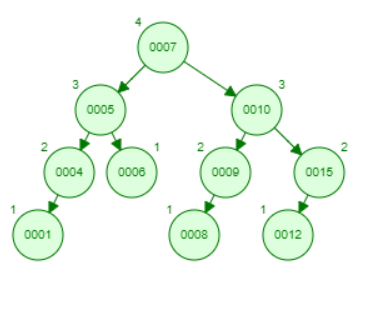
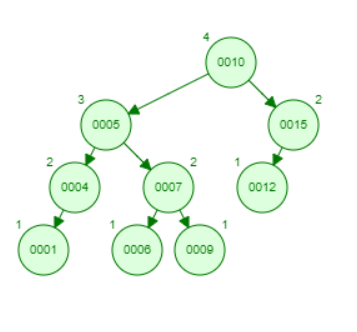
[5.c Kavramların açıklanması 14](#_Toc121763942)

[Özdeğerlendirme Tablosu 15](#_Toc121763943)

GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS

## AVL ağacına ve Heap yapısına istenilen değerlerin eklenerek ve silinerek oluşan yeni AVL ağacı ve Heap yapısının çizimleri

1.soru 2.soru

3.soru

## 

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

Alıntı yapılan kaynaklar(AVL TREE İÇİN):

<https://www.geeksforgeeks.org/insertion-in-an-avl-tree/>

<https://simpledevcode.wordpress.com/2014/09/16/avl-tree-in-c/>

public class AVLTree

{

private Node root;

public void Insert(int key)

{

root = Insert(root, key);

}

//Node ekleme metodu

private Node Insert(Node root, int key)

{

if (root == null)

return new Node(key);

if (key < root.Value)

{

root.Left = Insert(root.Left, key);

}

else

{

root.Right = Insert(root.Right, key);

}

return Balance(root, key);

}

private Node RightRotate(Node root)

{

Node newRoot = root.Left;

root.Left = newRoot.Right;

newRoot.Right = root;

return newRoot;

}

private Node LeftRotate(Node root)

{

Node newRoot = root.Right;

root.Right = newRoot.Left;

newRoot.Left = root;

return newRoot;

}

// Düğümlerin dengesini kontrol eden (1 den fazla fark olmamasını) metod

private Node Balance(Node root, int key)

{

int balance = GetBalance(root);

// Left Left Case

if (balance > 1 && key < root.Left.Value)

{

return RightRotate(root);

}

// Right Right Case

if (balance < -1 && key > root.Right.Value)

{

return LeftRotate(root);

}

// Left Right Case

if (balance > 1 && key > root.Left.Value)

{

root.Left = LeftRotate(root.Left);

return RightRotate(root);

}

// Right Left Case

if (balance < -1 && key < root.Right.Value)

{

root.Right = RightRotate(root.Right);

return LeftRotate(root);

}

return root;

}

private int GetBalance(Node root)//Dengeyi döndürür

{

if (root == null)

return 0;

return Height(root.Left) - Height(root.Right);

}

private int Height(Node root)//Yüksekliği döndürür

{

if (root == null)

return 0;

return Math.Max(Height(root.Left), Height(root.Right)) + 1;

## }

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

Node oluşturuldu ardından küçüktse kökün soluna büyükse sağına insert metodu ile ekleme yapıldı.Daha sonra left ve right rotate ile sağ ve sol düğüm işlemleri yapıldı.Balance ile sol ve sağ derinliği farkı 1 den fazla olmaması için dengelenme amaçlı kullanıldı. Küçükten büyüğe sırayla yazdırıldı.

## 3.a) Dijkstra kodu ve yapılan testler

class DistPar // distance and parent

{ // items stored in sPath array

public int distance; // distance from start to this vertex

public int parentVert; // current parent of this vertex

public DistPar(int pv, int d) // constructor

{

distance = d;

parentVert = pv;

}

}

class Vertex

{

public string label; // label (e.g. ‘A’)

public bool isInTree;

public Vertex(string lab) // constructor

{

label = lab;

isInTree = false;

}

}

class Graph

{

private int MAX\_VERTS = 20;

private int INFINITY = 1000000;

private Vertex[] vertexList; // list of vertices

private int[,] adjMat; // adjacency matrix

private int nVerts; // current number of vertices

private int nTree; // number of verts in tree

private DistPar[] sPath; // array for shortest-path data

private int currentVert; // current vertex

private int startToCurrent; // distance to currentVert

public Graph() // constructor

{

vertexList = new Vertex[MAX\_VERTS];

// adjacency matrix

adjMat = new int[MAX\_VERTS,MAX\_VERTS];

nVerts = 0;

nTree = 0;

for (int j = 0; j < MAX\_VERTS; j++) // set adjacency

for (int k = 0; k < MAX\_VERTS; k++) // matrix

adjMat[j,k] = INFINITY; // to infinity

sPath = new DistPar[MAX\_VERTS]; // shortest paths

}

public void addVertex(string lab)

{

vertexList[nVerts++] = new Vertex(lab);

}

public void addEdge(int start, int end, int weight)

{

adjMat[start,end] = weight; // (directed)

}

public void path() // find all shortest paths

{

int startTree = 0; // start at vertex 0

vertexList[startTree].isInTree = true;

nTree = 1; // put it in tree

// transfer row of distances from adjMat to sPath

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

{

int tempDist = adjMat[startTree,j];

sPath[j] = new DistPar(startTree, tempDist);

}

// until all vertices are in the tree

while (nTree < nVerts)

{

int indexMin = getMin(); // get minimum from sPath

int minDist = sPath[indexMin].distance;

if (minDist == INFINITY) // if all infinite

{ // or in tree,

Console.WriteLine("There are unreachable vertices");

break; // sPath is complete

}

else

{ // reset currentVert

currentVert = indexMin; // to closest vert

startToCurrent = sPath[indexMin].distance;

// minimum distance from startTree is

// to currentVert, and is startToCurrent

}

// put current vertex in tree

vertexList[currentVert].isInTree = true;

nTree++;

adjust\_sPath(); // update sPath[] array

} // end while(nTree<nVerts)

displayPaths(); // display sPath[] contents

nTree = 0; // clear tree

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

vertexList[j].isInTree = false;

}

public int getMin() // get entry from sPath

{ // with minimum distance

int minDist = INFINITY; // assume minimum

int indexMin = 0;

for (int j = 1; j < nVerts; j++) // for each vertex,

{ // if it’s in tree and

if (!vertexList[j].isInTree && // smaller than old one

sPath[j].distance < minDist)

{

minDist = sPath[j].distance;

indexMin = j; // update minimum

}

} // end for

return indexMin; // return index of minimum

}

public void adjust\_sPath()

{

// adjust values in shortest-path array sPath

int column = 1; // skip starting vertex

while (column < nVerts) // go across columns

{

// if this column’s vertex already in tree, skip it

if (vertexList[column].isInTree)

{

column++;

continue;

}

// calculate distance for one sPath entry

// get edge from currentVert to column

int currentToFringe = adjMat[currentVert,column];

// add distance from start

int startToFringe = startToCurrent + currentToFringe;

// get distance of current sPath entry

int sPathDist = sPath[column].distance;

// compare distance from start with sPath entry

if (startToFringe < sPathDist) // if shorter,

{ // update sPath

sPath[column].parentVert = currentVert;

sPath[column].distance = startToFringe;

}

column++;

}

}

public void displayPaths()

{

for (int j = 0; j < nVerts; j++) // display contents of sPath[]

{

Console.Write(vertexList[j].label + "="); // B=

if (sPath[j].distance == INFINITY)

Console.Write("inf"); // inf

else { Console.Write(sPath[j].distance); } // 50

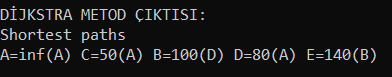
string parent = vertexList[sPath[j].parentVert].label;

Console.Write("(" + parent + ") "); // (A)

}

Console.WriteLine("");

## }



## 3.b) Prim MST kodu ve yapılan testler

class Prims //Bu sınıf, Prim algoritmasını kullanarak bir ağırlıklı yönlü grafikteki en küçük ağaç çubuğu(MST) bulmak için tasarlandı

{

public Prims(int[,] graph, int count)

{

Prim(graph, count);

}

private static int MinKey(int[] key, bool[] set, int verticesCount)//Bu fonksiyon, MST'ye daha önce dahil edilmemiş düğümler arasından en küçük anahtar değerli düğümü bulmak için bir yardımcı fonksiyondur.

{

int min = int.MaxValue, minIndex = 0;

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

{

if (set[v] == false && key[v] < min)

{

min = key[v];

minIndex = v;

}

}

return minIndex;

}

private static void Print(int[] parent, int[,] graph, int verticesCount)

{

Console.WriteLine("Edge Weight");

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

Console.WriteLine("{0} - {1} {2}", parent[i], i, graph[i, parent[i]]);

}

public static void Prim(int[,] graph, int verticesCount)//Bu fonksiyon, Prim algoritmasını uygular ve MST'yi hesaplar.

{

int[] parent = new int[verticesCount];

int[] key = new int[verticesCount];

bool[] mstSet = new bool[verticesCount];

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

{

key[i] = int.MaxValue;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

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

{

int u = MinKey(key, mstSet, verticesCount);

mstSet[u] = true;

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

{

if (Convert.ToBoolean(graph[u, v]) && mstSet[v] == false && graph[u, v] < key[v])

{

parent[v] = u;

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

}

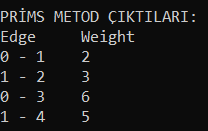
}

}

Print(parent, graph, verticesCount);

}

## }



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

class StackX

{

private int SIZE = 20;

private int[] st;

private int top;

public StackX() // constructor

{

st = new int[SIZE]; // make array

top = -1;

}

public void push(int j) // put item on stack

{ st[++top] = j; }

public int pop() // take item off stack

{ return st[top--]; }

public int peek() // peek at top of stack

{ return st[top]; }

public bool isEmpty() // true if nothing on stack-

{ return (top == -1); }

}

class VertexDFT

{

public string label; // label (e.g. ‘A’)

public bool wasVisited;

public VertexDFT(string lab) // constructor

{

label = lab;

wasVisited = false;

}

}

class GraphDFT

{

private int MAX\_VERTS = 20;

private VertexDFT[] vertexDFTList; // list of vertices

private int[,] adjMat; // adjacency matrix

private int nVerts; // current number of vertices

private StackX theStack;

public GraphDFT() // constructor

{

vertexDFTList = new VertexDFT[MAX\_VERTS];

// adjacency matrix

adjMat = new int[MAX\_VERTS,MAX\_VERTS];

nVerts = 0;

for (int j = 0; j < MAX\_VERTS; j++) // set adjacency

for (int k = 0; k < MAX\_VERTS; k++) // matrix to 0

adjMat[j,k] = 0;

theStack = new StackX();

}

public void addVertex(string lab)

{

vertexDFTList[nVerts++] = new VertexDFT(lab);

}

public void addEdge(int start, int end)

{

adjMat[start,end] = 1;

adjMat[end,start] = 1;

}

public void displayVertex(int v)

{

Console.WriteLine(vertexDFTList[v].label);

}

public void dft() // depth-first search

{ // begin at vertex 0

vertexDFTList[0].wasVisited = true; // mark it

displayVertex(0); // display it

theStack.push(0); // push it

while (!theStack.isEmpty()) // until stack empty,

{

// get an unvisited vertex adjacent to stack top

int v = getAdjUnvisitedVertex(theStack.peek());

if (v == -1) // if no such vertex,

theStack.pop();

else // if it exists,

{

vertexDFTList[v].wasVisited = true; // mark it

displayVertex(v); // display it

theStack.push(v); // push it

}

}

for (int j = 0; j < nVerts; j++) // reset flags

vertexDFTList[j].wasVisited = false;

}

public int getAdjUnvisitedVertex(int v)

{

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

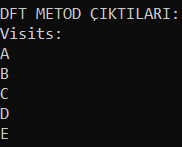
if (adjMat[v,j] == 1 && vertexDFTList[j].wasVisited == false)

return j;

return -1;

}

## }

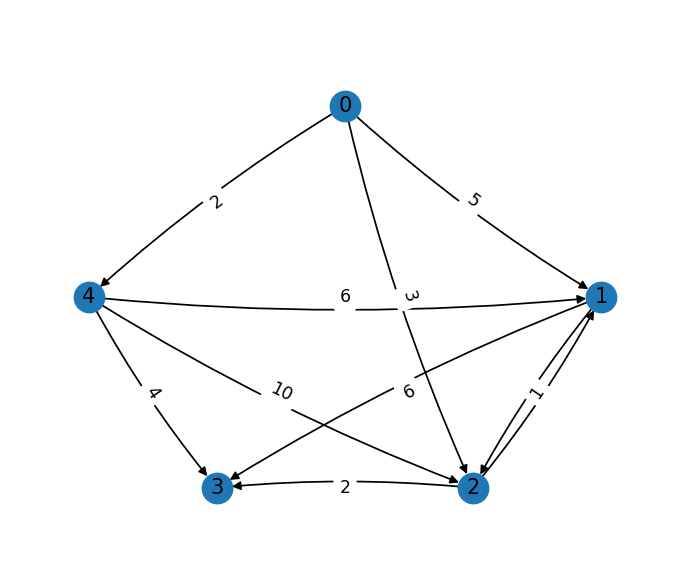


## 3.d) Verilen Big-O tablosunun doldurulmuş hali

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Dijkstra’s SP** | Prim’s MST | BFT | **Heap Deletion** |
| **Big-O** (Zaman Karmaşıklığı  Big-O Notasyonuna Göre) | O(V^2) | O(E log V) | O(N)  (DFT) | O(log n) |

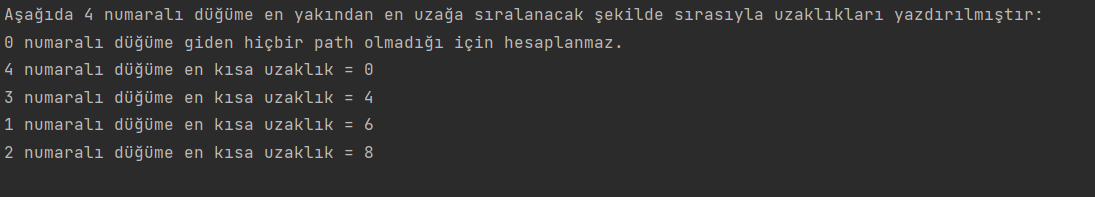
## 4.a) Verilen çizgenin python ortamında grafiksel olarak oluşturan kod ve ekran görüntüsü

import networkx as nx  
from matplotlib import pyplot as plt  
  
#Grafikteki düğümleri ve etiketleri çizen fonksiyon  
def PrintGraph(DGraph):  
 node\_pos=nx.get\_node\_attributes(DGraph,'pos')  
 arc\_weight=nx.get\_edge\_attributes(DGraph,'weight')  
  
 nx.draw\_networkx\_nodes(DGraph , pos=node\_pos)  
 nx.draw\_networkx\_labels(DGraph , pos=node\_pos)  
 nx.draw\_networkx\_edges(DGraph , node\_pos,connectionstyle='arc3, rad = 0.05')  
 nx.draw\_networkx\_edge\_labels(DGraph , node\_pos , arc\_weight)  
  
 plt.axis('off')  
 plt.show()  
  
  
# Bu kod, NetworkX kütüphanesinin DiGraph sınıfını kullanarak boş bir yönlü grafik oluşturur.  
DGraph = nx.DiGraph()  
  
# Bu kod, grafikteki düğümleri (veya noktaları) listeler ve 0 ile 4 arasında numaralandırılmıştır.  
vertex\_list = [0,1,2,3,4]  
  
# Bu kod, grafikteki kenarları listeler.Tuple sırasıyla kaynak düğüm, hedef düğüm ve kenar ağırlığıdır.  
edges\_list = [(0,4,2),(0,2,3),(0,1,5),  
(1,2,2),(1,3,6),  
(2,1,1),(2,3,2),  
(4,3,4),(4,2,10),(4,1,6),  
]  
  
# Düğümleri ve kenarları boş grapha ekler  
DGraph.add\_nodes\_from(vertex\_list)  
DGraph.add\_weighted\_edges\_from(edges\_list)  
  
# Bu satırlar, grafikteki düğümlere pozisyon nitelikleri ekler.  
DGraph.\_node[0]['pos'] = (0,8)  
DGraph.\_node[1]['pos'] = (4,2)  
DGraph.\_node[2]['pos'] = (2,-4)  
DGraph.\_node[3]['pos'] = (-2,-4)  
DGraph.\_node[4]['pos'] = (-4,2)  
  
# 4. düğümden diğer tüm düğümlere giden en kısa yol uzunluklarını bulduk.  
path\_lengths = nx.single\_source\_dijkstra\_path\_length(DGraph, 4)  
  
  
# En kısa yol uzunluklarını yazdırdık.  
print(f"Aşağıda 4 numaralı düğüme en yakından en uzağa sıralanacak şekilde sırasıyla uzaklıkları yazdırılmıştır:")  
print(f"0 numaralı düğüme giden hiçbir path olmadığı için hesaplanmaz.")  
for node, length in path\_lengths.items():  
 print(f"{node} numaralı düğüme en kısa uzaklık = {length}")  
  
  
# Fonskiyon çağırarak çizgeyi yazdırdık.  
PrintGraph(DGraph)  
  
# 2 numaralı düğümü kaldırdık.  
DGraph.remove\_node(2)  
  
# 2 Numaralı düğümü çıkardıktan sonra değişen çizgeyi yazdırdık.  
PrintGraph(DGraph)

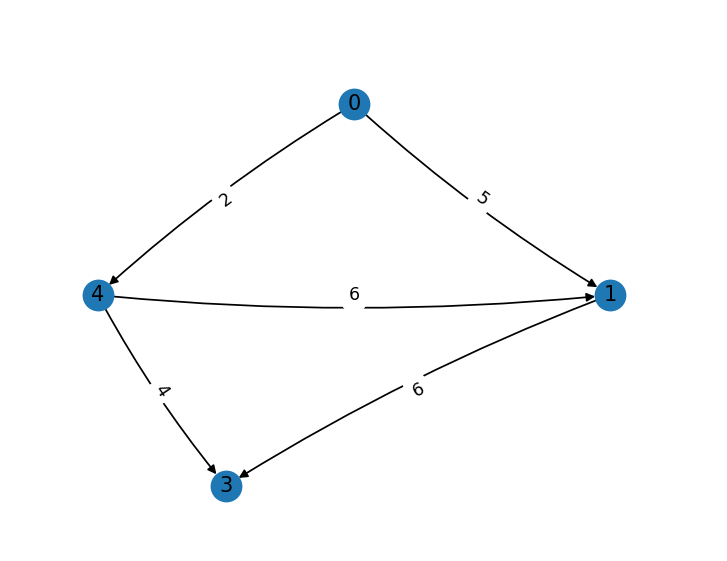


## 4.b) Verilen çizgede en kısa yol bulma sonuçları

## 4.c) Düğüm slime ve yeniden çizdirme



## 



## 5.a) Prim ve Kruskal algoritması karşılaştırması

## Prim algoritması, grafteki herhangi bir noktadan başlatılabilir. Kruskal algoritması ise, grafteki tüm noktaların bir araya geleceği bir ağaç oluşturmak için grafteki tüm kenarları sıralayarak başlar.

## Prim algoritması daha çok özel durumlar için daha uygundur ve daha az kenar sayısına sahip graflarda daha hızlıdır. Kruskal algoritması ise, daha çok kenar sayısına sahip graflarda daha hızlıdır.

Prim algoritması, bir ağaç oluştururken bir adımda sadece bir kenar seçer. Kruskal algoritması ise, bir adımda iki noktayı birbirine bağlayacak olan en küçük ağırlıklı kenarı seçer.

## 5.b) Trie veri yapısı kodu ve açıklaması

// Trie class

public class Trie

{

private TrieNode root;

public Trie()

{

root = new TrieNode();

}

// Method to add an element to the Trie

public void Add(string word)

{

// Start at the root node

TrieNode current = root;

// Iterate through each character in the word

foreach (char c in word)

{

// If the current node doesn't have a child for the current character, add one

if (!current.Children.ContainsKey(c))

{

current.Children[c] = new TrieNode();

}

// Move to the child node for the current character

current = current.Children[c];

}

// Mark the last node as a word

current.IsWord = true;

}

public void TriePrint()

{

// Start at the root node

TrieNode current = root;

// Use a stack to keep track of the path through the Trie

Stack<char> path = new Stack<char>();

// Perform a pre-order traversal of the Trie

PreOrderTraversal(current, path);

}

private void PreOrderTraversal(TrieNode current, Stack<char> path)

{

// Base case: if the current node is null, return

if (current == null) return;

// Print the path if the current node is a word

if (current.IsWord)

{

Console.WriteLine(new string(path.Reverse().ToArray()));

}

// Visit the children

foreach (var child in current.Children)

{

// Push the current character onto the stack and visit the child node

path.Push(child.Key);

PreOrderTraversal(child.Value, path);

path.Pop();

}

## }

Trie sınıfı, Trie veri yapısında depolanan kelimelere erişmek için kullanılır.

Trie sınıfının fonksiyonları:

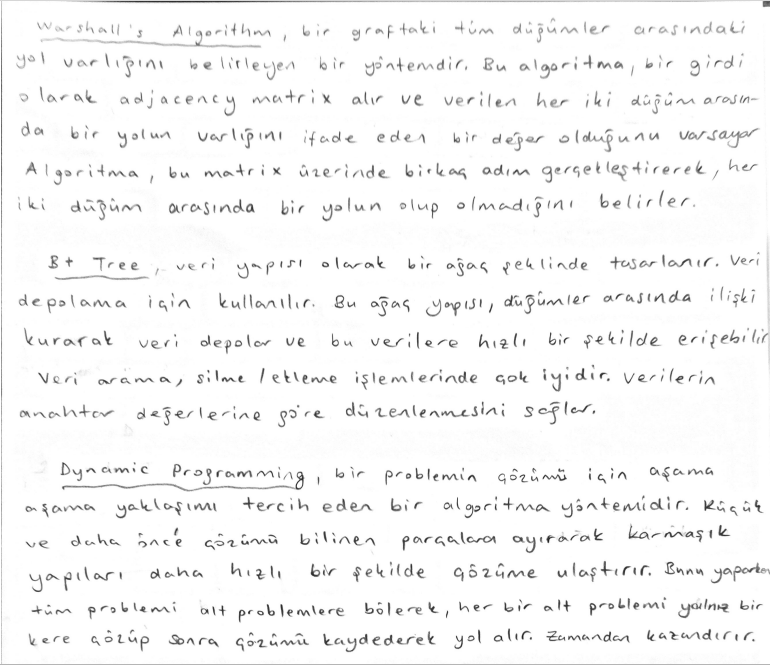
Add: Trie'ye bir kelime ekler.

TriePrint: Trie içindeki tüm kelimelere erişir ve bunları ekrana yazdırır.

TriePrint fonksiyonu, Trie veri yapısını derinlemesine tarayarak tüm kelimelere erişir. Bu işlem, PreOrderTraversal adlı bir yardımcı fonksiyon kullanılarak gerçekleştirilir. PreOrderTraversal, Trie veri yapısını derinlemesine tarayarak tüm kelimelere erişir ve bunları ekrana yazdırır.

## 5.c) Kavramların açıklanması

## Özdeğerlendirme Tablosu



|  |  |  |  |
| --- | --- | --- | --- |
|  | **Points** | **Estimated Grade** | **Explanation** |
| **1 a) AVL Tree** | **10** | **10** | **AVL Ağaacına sayılar eklendi.Dengesizleşince denge sağlandı.** |
| **1 b) Heap** | **10** | **10** | **En küçük değer sahip sayı köke taşındı.Sonrasında stackteki son değer silindi.** |
| **2) B-Tree Insertion / AVL Tree Insertion / Red-Black Trees / Huffman Encoding Tree** | **10** | **10** | **AVL Ağacı insert metodu eklendi denge sağlandı.** |
| **3 a) Dijkstra’s shortest path code + test** | **5** | **5** | **Köşeler arasındaki en kısa yol bulundu.** |
| **3 b) Prim’s MST code + test** | **5** | **5** | **MST vari bir anahtar eşleşmesi sağlandı.** |
| **3 c) BFT or DFT code + test** | **5** | **5** | **Derinlemesine ilk tarama yapıldı.Lineer bir sırayla inilip tüm düğümleri gezdi.** |
| **3 d) Filling Big-O Table** | **5** | **5** | **Yapıldı.** |
| **4 i) Graph Drawing** | **5** | **5** | **Pyhton ortamında hazır kodla yapıldı.** |
| **4 ii) Finding Shortest Paths with Dijkstra’s** | **5** | **5** | **Hesaplandı ve yazdırıldı.** |
| **4 iii) Node deletion and repeating i, ii.** | **5** | **5** | **Burda tekrara düşmemek için fonksiyon kullanıldı.** |
| **5 i) Comparison (Prim’s & Kruskal’s Algorithm)** | **5** | **5** | **Farkları açıklandı.Prims tek bir düğümden tamamına kruskal hepsini dolaşması gereken bir algoritma olduğu söylendi.** |
| **5 ii) Trie Data Structure and Insertion Method** | **5** | **5** | **Trie sözlük gibi hashe benzer ama daha etkin bir yapı olduğu anlaşıldı.** |
| **5 iii) Explanations of 3 terms** | **15** | **15** | **Dinamik programlama warshall ve b+tree açıklandı.** |
| **Demo Video for Source Codes and Tests of Q2 Q3a, Q3b, Q3c and Q5.ii .** | **5** | **4** | **Video 6 dakika olarak çekildi.** |
| **Self-assessment Table** | **5** | **5** | **Araştırarak iyi şeyler öğrendim ve yeni kazanımlar edindim.Kodları tek başıma yazdığım için uğraştığımı düşünüyorum.** |
| **Total** | **100** | **99** |  |

**Açıklama kısmında yapıldı, yapılmadı bilgisi ve hangi maddelerin nasıl yapıldığı veya neden yapılamadığı kısaca yazılmalıdır. Tahmini not kısmına da ilgili maddeden kaç almayı beklediğinizi yazmalısınız.**

**Not: Raporu teslim edilmeyen projeler değerlendirmeye alınmayacaktır.**