

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

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

**2023–2024 FALL SEMESTER**

**PROJECT-4 REPORT**

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

**DELIVERY DATE**

06/01/2024

**PREPARED BY**

05210000261, Bahrihan TORPİL

05210000292, Beyza ÇELİK

05210000264, Baran Koca

05210000257, Şevval Avşar

İçindekiler

[1.a Drawing of new AVL Trees after inserting requested values 3](#_Toc154511673)

[1.b Drawing of new Heaps after inserting and removing requested values 3](#_Toc154511674)

[2.a B-Tree insertion method (or AVL-Tree insertion method/Red-Black tree/Huffman encoding tree) code 3](#_Toc154511675)

[2.b Explanation of B-Tree insertion method steps 3](#_Toc154511676)

[2.c B-Tree insertion test 3](#_Toc154511677)

[3.a Dijkstra’s SP source code and tests 3](#_Toc154511678)

[3.b Prim’s MST source code and tests 3](#_Toc154511679)

[3.c BFT or DFT source code and tests 3](#_Toc154511680)

[3.d Filled version of the given Big-O table 3](#_Toc154511681)

[4.a Source code for drawing the given graph in python environment and screenshots 3](#_Toc154511682)

[4.b Computing distances of nodes 4](#_Toc154511683)

[4.c Traversing the graphs with DFS and BFS 4](#_Toc154511684)

[4.d Purposes of computing SP, BFS and MST 4](#_Toc154511685)

[4.e Method for accessing to the doctor 4](#_Toc154511686)

[5.a. Brief comparison of Prim’s and Kruskal’s algorithms 4](#_Toc154511687)

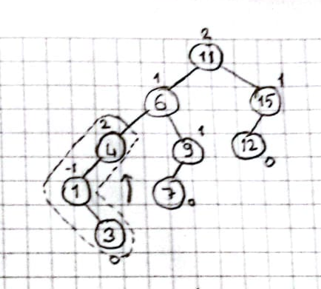
[5.b Implementation of Trie Data Structure and Insert Method 4](#_Toc154511688)

[5.c. Explanation of 4 Terms 4](#_Toc154511689)

[Self-assessment Table 5](#_Toc154511690)

GRAPHS, GRAPH ALGORITHMS, TREES and OTHER SUBJECTS

## 1.a Drawing of new AVL Trees after inserting requested values

 çizgi, diyagram, daire içeren bir resim

Açıklama otomatik olarak oluşturuldu

çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu çizgi, diyagram, daire içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 1.b Drawing of new Heaps after inserting and removing requested values

çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu çizgi içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 2.a B-Tree insertion method (or AVL-Tree insertion method/Red-Black tree/Huffman encoding tree) code

|  |
| --- |
| class AVLNode  {  public int Data;  public int Height;  public AVLNode Left, Right;  public AVLNode(int data)  {  Data = data;  Height = 1;  Left = Right = null;  }  }  class AVLTree  {  AVLNode root;  // Ağacın yüksekliğini döndür  int Height(AVLNode node)  {  if (node == null)  return 0;  return node.Height;  }  // Yükseklik farkını hesapla  int BalanceFactor(AVLNode node)  {  if (node == null)  return 0;  return Height(node.Left) - Height(node.Right);  }  // Yeni yükseklik değerini güncelle  void UpdateHeight(AVLNode node)  {  if (node != null)  node.Height = 1 + Math.Max(Height(node.Left), Height(node.Right));  }  // Sağa dön  AVLNode RightRotate(AVLNode y)  {  AVLNode x = y.Left;  AVLNode T2 = x.Right;  x.Right = y;  y.Left = T2;  UpdateHeight(y);  UpdateHeight(x);  return x;  }  // Sola dön  AVLNode LeftRotate(AVLNode x)  {  AVLNode y = x.Right;  AVLNode T2 = y.Left;  y.Left = x;  x.Right = T2;  UpdateHeight(x);  UpdateHeight(y);  return y;  }  // Dengele ve düzelt  AVLNode Balance(AVLNode node)  {  UpdateHeight(node);  int balance = BalanceFactor(node);  // Sol ağaç yüksekse  if (balance > 1)  {  if (BalanceFactor(node.Left) < 0)  node.Left = LeftRotate(node.Left);  return RightRotate(node);  }  // Sağ ağaç yüksekse  if (balance < -1)  {  if (BalanceFactor(node.Right) > 0)  node.Right = RightRotate(node.Right);  return LeftRotate(node);  }  return node;  }  // Yeni bir düğüm ekle  public void Insert(int data)  {  root = Insert(root, data);  }  // Yeni bir düğüm ekle  AVLNode Insert(AVLNode node, int data)  {  // Standart BST ekleme  if (node == null)  return new AVLNode(data);  if (data < node.Data)  node.Left = Insert(node.Left, data);  else if (data > node.Data)  node.Right = Insert(node.Right, data);  else  return node; // Duplicate veri eklenmez.  // Yükseklik güncelle  UpdateHeight(node);  // Dengele ve düzelt  return Balance(node);  }  // Ağacı preorder olarak gez  void InorderTraversal(AVLNode root)  {  if (root != null)  {  InorderTraversal(root.Left);  Console.Write(root.Data + " ");  InorderTraversal(root.Right);  }  }  void PreorderTraversal(AVLNode root)  {  if (root != null)  {  Console.Write(root.Data + " ");  PreorderTraversal(root.Left);  PreorderTraversal(root.Right);  }  }  // Ağacı Preorder olarak gezerek yazdır  public void PrintPreorder()  {  PreorderTraversal(root);  Console.WriteLine();  }  }  private static void AVLDemo()  {  AVLTree avlTree = new AVLTree();  avlTree.Insert(10);  avlTree.Insert(20);  avlTree.Insert(35);  avlTree.Insert(23);  avlTree.Insert(50);  avlTree.Insert(29);  avlTree.Insert(3);  Console.WriteLine("Preorder traversal of AVL tree:");  avlTree.PrintPreorder();  } |

## 2.b Explanation of B-Tree insertion method steps

Binary search tree mantığı ile aynı şekilde sola parent’tan küçük değerler, sağa büyük değerler yazdırılır.

Ekleme işlemi yapıldıktan sonra yükseklikler UpdateHeight() method ile kontrol edilir ve güncellenir.

UpdateMethod() : Math sınıfından max methodunu kullanarak düğüm yükseklikleri güncellenir.

Ekleme işlemi ve yükseklik güncelleme işleminden sonra Dengelemek için Balance() method çağırılır.

BalanceMethod() : balance değeri 1 ‘den büyük veya -1’den küçük ise dengesizdir ve 1’den fazlaysa ilgili düğümleri sola döndürme işlemi uygulanır, -1’den fazlaysa ilgili düğümleri sağa döndürme işlemi uygulanır.

LeftRotate ve RightRotate Methods : dengesiz bölgedeki 3 düğüm için değişkenler tutulur. Sağ ve sol kaydırma işlemi gereken yere uygulanır. Yeni dengeli ağaç Insert methodunda return edilir.

## 2.c B-Tree insertion test

## metin, yazı tipi, ekran görüntüsü, siyah içeren bir resim Açıklama otomatik olarak oluşturuldu

## 3.a Dijkstra’s SP source code and tests

|  |
| --- |
| public static void Dijkstra(int[,] graph, int startVertex)  {  int numVertices = graph.GetLength(0);  int[] distance = new int[numVertices];  bool[] shortestPathSet = new bool[numVertices];  // Başlangıç noktasına uzaklıkları initialize et  for (int i = 0; i < numVertices; i++)  {  distance[i] = int.MaxValue;  shortestPathSet[i] = false;  }  // Başlangıç noktasının kendisine uzaklığını 0 olarak ayarla  distance[startVertex] = 0;  for (int count = 0; count < numVertices - 1; count++)  {  int u = MinDistance(distance, shortestPathSet);  shortestPathSet[u] = true;  for (int v = 0; v < numVertices; v++)  {  if (!shortestPathSet[v] && graph[u, v] > 0 && distance[u] != int.MaxValue &&  distance[u] + graph[u, v] < distance[v])  {  distance[v] = distance[u] + graph[u, v];  }  }  }  PrintSolution(distance);  }  public static int MinDistance(int[] distance, bool[] shortestPathSet)  {  int min = int.MaxValue;  int minIndex = -1;  for (int v = 0; v < distance.Length; v++)  {  if (!shortestPathSet[v] && distance[v] <= min)  {  min = distance[v];  minIndex = v;  }  }  return minIndex;  }  public static void PrintSolution(int[] distance)  {  Console.WriteLine("En kısa mesafeler:");  for (int i = 0; i < distance.Length; i++)  {  Console.WriteLine($"0 --> {i}: {distance[i]} birim");  }  }  }  private static void DijkstraDemo()  {    int[,] directedWeightedMatrix = {  {0, 3, 0, 2, 0},  {0, 0, 1, 3, 5},  {0, 0, 0, 0, 2},  {0, 0, 0, 0, 3},  {0, 0, 0, 0, 0}  };  DijkstraAlgorithm.Dijkstra(directedWeightedMatrix, 0);  } |

## metin, yazı tipi, ekran görüntüsü, tipografi içeren bir resim Açıklama otomatik olarak oluşturuldu

## 3.b Prim’s MST source code and tests

|  |
| --- |
| static int V = 5;  public int MinKey(int[] key, bool[] mstSet)  {  int min = int.MaxValue, minIndex = -1;  for (int v = 0; v < V; v++)  {  if (mstSet[v] == false && key[v] < min)  {  min = key[v];  minIndex = v;  }  }  return minIndex;  }  public void PrintMST(int[] parent, int[,] graph)  {  Console.WriteLine("Minimum Spanning Tree:");  for (int i = 1; i < V; i++)  {  Console.WriteLine($"Edge: {parent[i]} - {i}, Weight: {graph[i, parent[i]]}");  }  }  public void PrimAlgorithm(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);  }  private static void PrimDemo()  {  PrimMST prim = new PrimMST();  int[,] graph = {  {0, 0, 0, 0, 3},  {0, 0, 4, 6, 4},  {0, 4, 5, 0, 0},  {0, 6, 5, 0, 2},  {3, 4, 0, 2, 0}  };  prim.PrimAlgorithm(graph);  } |

metin, yazı tipi, ekran görüntüsü, sayı, numara içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 3.c BFT or DFT source code and tests

|  |
| --- |
| public static void BreadthFirstSearch(int[,] graph, int startVertex)  {  int vertices = graph.GetLength(0);  bool[] visited = new bool[vertices];  Queue<int> queue = new Queue<int>();  visited[startVertex] = true;  queue.Enqueue(startVertex);  while (queue.Count != 0)  {  startVertex = queue.Dequeue();  Console.Write($"{startVertex} ");  for (int i = 0; i < vertices; i++)  {  if (graph[startVertex, i] != 0 && !visited[i])  {  visited[i] = true;  queue.Enqueue(i);  }  }  }  }  public static void DepthFirstSearch(int[,] graph, int startVertex, bool[] visited)  {  Console.Write($"{startVertex} ");  visited[startVertex] = true;  for (int i = 0; i < graph.GetLength(0); i++)  {  if (graph[startVertex, i] != 0 && !visited[i])  {  DepthFirstSearch(graph, i, visited);  }  }  } |

## metin, yazı tipi, ekran görüntüsü, beyaz içeren bir resim Açıklama otomatik olarak oluşturuldu

## 3.d Filled version of the given Big-O table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Dijkstra’s SP** | Prim’s MST | BFT | **Heap Deletion** |
| **Big-O** (Zaman Karmaşıklığı  Big-O Notasyonuna Göre) | O(Vertex+Edge \* log(Vertex)) | O(Vertex+Edge \* log(Vertex)) | O(Vertex + Edge) | O(logN)(N: eleman sayısı) |

## 4.a Source code for drawing the given graph in python environment and screenshots

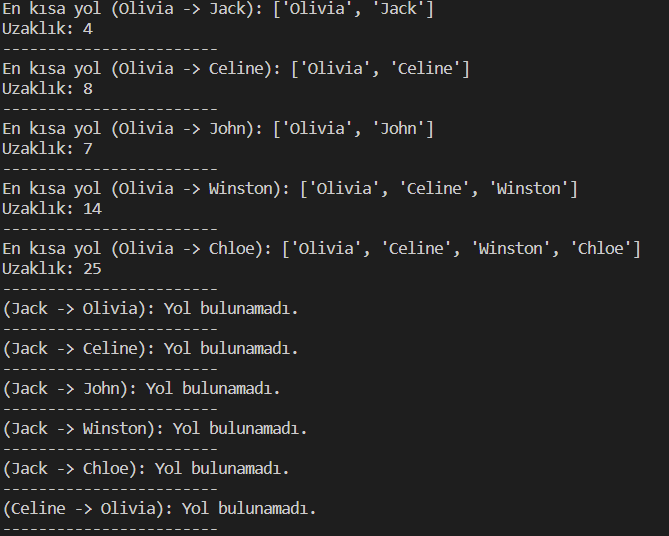
|  |
| --- |
| import networkx as nx  import matplotlib.pyplot as plt  graph = nx.DiGraph()  graph.add\_node("Olivia")  graph.add\_node("Jack")  graph.add\_node("Celine")  graph.add\_node("John")  graph.add\_node("Winston")  graph.add\_node("Olivia")  graph.add\_edge("Olivia", "Celine", weight = 8)  graph.add\_edge("Olivia", "Jack", weight = 4)  graph.add\_edge("Olivia", "John", weight = 7)  graph.add\_edge("Celine", "Jack", weight = 5)  graph.add\_edge("Celine", "Winston", weight = 6)  graph.add\_edge("Winston", "Jack", weight = 7)  graph.add\_edge("Winston", "Chloe", weight = 11)  graph.add\_edge("Chloe", "Jack", weight = 5)  graph.add\_edge("Chloe", "John", weight = 7)  graph.add\_edge("John", "Jack", weight = 9)  shell\_layout = [["Jack"], ["Olivia", "Celine", "Winston", "Chloe", "John"]]  pos = nx.shell\_layout(graph, nlist=shell\_layout)  # Grafik çizimi  labels = nx.get\_edge\_attributes(graph, 'weight')  nx.draw(graph, pos, with\_labels=True, node\_size=700, node\_color='skyblue', font\_size=10, font\_color='black')  nx.draw\_networkx\_edge\_labels(graph, pos, edge\_labels=labels)  plt.show() |

çizgi, ekran görüntüsü, diyagram, metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 4.b Computing distances of nodes

|  |
| --- |
| people = ["Olivia", "Jack", "Celine", "John", "Winston", "Chloe"]  for person1 in people:  for person2 in people:  if person1 != person2:  try:  shortest\_path = nx.dijkstra\_path(graph, source=person1, target=person2)  shortest\_distance = nx.shortest\_path\_length(graph, source=person1, target=person2, weight="weight")  print(f"En kısa yol ({person1} -> {person2}): {shortest\_path}")  print(f"Uzaklık: {shortest\_distance}")  except nx.NetworkXNoPath:  print(f"({person1} -> {person2}): Yol bulunamadı.")  print("------------------------") |

 metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 4.c Traversing the graphs with DFS and BFS

|  |
| --- |
| dfs\_edges = list(nx.dfs\_edges(graph, source="Olivia"))  print("DFS Yolu:", dfs\_edges)  #(BFS) ile gezme  bfs\_edges = list(nx.bfs\_edges(graph, source="Olivia"))  print("BFS Yolu:", bfs\_edges) |

## 4.d Purposes of computing SP, BFS and MST

SP için konuşursak, Yön/rota planlamalarında en kısa yol tercihi, ağ yönlendirmesi gibi uygulamalarda kullanılır. Yol maliyetlerini indirgemek için sıklıkla kullanılır.

BFS ise bağlı bileşenleri(components) bulma, topolojik sıralama gibi uygulamalarda kullanılır. Dolaşma ve bağlantı bulma problemlerinde kullanılabilir.

Son olarak MST, doğal olarak en küçük ağırlıkları bağlayarak çalışan bir algoritma olduğu için Maliyeti en düşüğe indirger. Verileri kümelendirme, min yol maliyeti veya ağı, elektrik şebekeleri, telekomünikasyon ağları oluşturmak bazı kullanım alanlarıdır.

## 4.e Method for accessing to the doctor

|  |
| --- |
| Öncelikle doctor olan olivia’ya her kişinin en kısa yolunu Dijkstra algoritması ile bulabiliriz. Bir kişinin ilişkili olduğu kişilerle olan sürelerinden en kısa olan seçilir ve doktora ulaşılana kadar bu tekrarlı olarak devam eder. Böylece en kısa sürede ulaşım sağlanır. |

Zaman karmaşıklığı O((V+E)\* logV)’dir. V : kişi sayısı, E : bağlantı sayısı

Bir kişinin telefonu açmama olasılığı da işin içine katılırsa her telefonu açma süresi açmama olasılığı ile çarpılarak yeni ağırlık değerleri bağlantılara yazılmalıdır.

## 

## 5.a. Brief comparison of Prim’s and Kruskal’s algorithms

// Insert your comparison here.

## 5.b Implementation of Trie Data Structure and Insert Method

|  |
| --- |
| public class TrieNode  {  public TrieNode[] Children { get; set; }  public bool IsEndOfWord { get; set; }  public TrieNode()  {  Children = new TrieNode[26]; // 26 harf için  IsEndOfWord = false;  }  }  internal class Trie  {  private TrieNode root;  public Trie()  {  root = new TrieNode();  }  private int GetIndex(char c)  {  return c - 'a'; // Küçük harfler için  }  public void Insert(string word)  {  TrieNode node = root;  foreach (char c in word)  {  int index = GetIndex(c);  if (node.Children[index] == null)  {  node.Children[index] = new TrieNode();  }  node = node.Children[index];  }  node.IsEndOfWord = true;  }  public void PrintTrie()  {  PrintTrie(root, "");  }  private void PrintTrie(TrieNode node, string currentWord)  {  if (node.IsEndOfWord)  {  Console.WriteLine(currentWord);  }  for (int i = 0; i < 26; i++)  {  if (node.Children[i] != null)  {  char ch = (char)('a' + i);  PrintTrie(node.Children[i], currentWord + ch);  }  }  } |

Preorder ile char türündeki köklerden ilerlenerek yazılan çıktı:

metin, yazı tipi, beyaz, tasarım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Yazılan kodda TrieNode türünde 26 elemanlı(ingilizce alfabedeki harf sayısı kadar) bir dizi oluşturulur.

Bir kelimeyi Trie’ye eklemek için her harfin alfabedeki yeri için indexi getIndex method ile bulunur.( Küçük harfler için ASCII değerinden 'a' çıkartılır.)

Trie’ye eklenmemiş bir harf ise yeni düğüm oluşturularak devam edilir. Yoksa bir sonraki düğüme geçilir.

IsEndOfWord Boolean değişkeniyle de kelime sonu(son harfteki düğüm) işaretlenir.

## 5.c. Explanation of 4 Terms

R-Tree, özellikle çok boyutlu veri (çoğunlukla geometrik nesneler) için etkili bir indeksleme yapısıdır. Her düğüm, belirli bir boyut aralığı içindeki nesnelerin sınırlarını içerir, böylece sorguların hızlı bir şekilde yanıtlanmasını sağlar.

Dinamik programlama, bir optimizasyon problemini alt problemlere bölerek çözen ve daha küçük alt problemlerin çözümlerini saklayarak tekrar hesaplamayı önleyen bir algoritma tasarım tekniğidir. Büyük bir problemi çözmek için daha küçük alt problemleri çözerek, çözümleri bir araya getirir. Çözülen alt problemlerin sonuçları genellikle bir tabloda saklanır ve tekrar kullanılabilir.

2-3-4 ağacı, her bir düğümde 2, 3 veya 4 çocuğa sahip bir denge ağacıdır.

Her düğüm, sıralı bir şekilde düzenlenmiş anahtarları içerir. 2 düğümden 4 düğüme kadar olan çeşitli durumları yönetir.

Warshall algoritması, bir grafın transitif kapanımını (transitive closure) bulan bir graf algoritmasıdır.

İki düğüm arasında doğrudan bir kenar varsa veya bir kenar zinciri üzerinden birbirine bağlılarsa, bu iki düğüm arasında bir yoldur.

Algoritma, grafın her bir düğüm çifti arasındaki erişilebilirlik ilişkilerini belirler ve bu sayede grafın kapalı olup olmadığını kontrol eder.

## 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. |
| **3 a) Dijkstra’s shortest path code + test** | **4** | 4 | Yapıldı. |
| **3 b) Prim’s MST code + test** | **4** | 4 | Yapıldı. |
| **3 c) BFT or DFT code + test** | **3** | 3 | Yapıldı. |
| **3 d) Filling Big-O Table** | **4** | 4 | Yapıldı. |
| **4 a) Graph Drawing** | **3** | 3 | Yapıldı. |
| **4 b) Finding Shortest Paths with Dijkstra’s** | **3** | 3 | Yapıldı. |
| **4 c) DFS and BFS** | **3** | 3 | Yapıldı. |
| **4 d) Thinking and Writing aims of given algorithms** | **3** | 3 | Yapıldı. |
| **4 e) Real Life Application** | **3** | 3 | Yapıldı. |
| **5 i) Comparison (Prim’s & Kruskal’s Algorithm)** | **5** | 5 | Yapıldı. |
| **5 ii) Trie Data Structure and Insertion Method** | **5** | 5 | Yapıldı. |
| **5 iii) Explanations of 4 terms** | **10** | 10 | Yapıldı. |
| **Demo Video for Source Codes and Tests of Q2 Q3a, Q3b, Q3c and Q5.ii .** | **10** | 6 | Demovideosu yapılmadı. Testler yapıldı. Foto şeklinde atıldı. |
| **Self-assessment Table** | **10** | 10 | Yapıldı. |
| **Total** | **100** | 96 |  |