

**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**

Student\_number, Name Surname

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

// draw the trees requested in i. and ii.

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

// draw the heaps requested in iii. and iv.

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

|  |
| --- |
| // source code |

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

// Explain your source code step by step

## 2.c B-Tree insertion test

//Insert your test screenshot or output here

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

|  |
| --- |
| public void dijkstra(int startVertex) {  int[] distance = new int[nVerts];  boolean[] visited = new boolean[nVerts];  // Initialize distances and visited array  for (int i = 0; i < nVerts; i++) {  distance[i] = INFINITY;  visited[i] = false;  }  distance[startVertex] = 0;  for (int count = 0; count < nVerts - 1; count++) {  int currentVertex = minDistance(distance, visited);  visited[currentVertex] = true;  for (int j = 0; j < nVerts; j++) {  if (!visited[j] && adjMat[currentVertex][j] != INFINITY  && distance[currentVertex] != INFINITY  && distance[currentVertex] + adjMat[currentVertex][j] < distance[j]) {  distance[j] = distance[currentVertex] + adjMat[currentVertex][j];  }  }  }  // Print the shortest distances  System.out.println("Dijkstra's Shortest Paths from vertex " + vertexList[startVertex].label + ":");  for (int i = 0; i < nVerts; i++) {  System.out.println("To " + vertexList[i].label + ": " + distance[i]);  }  } |

ekran görüntüsü, metin, grafik yazılımı, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, ekran görüntüsü, menü içeren bir resim

Açıklama otomatik olarak oluşturuldu

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

|  |
| --- |
| public void mstw() {  currentVert = 0;  int totalWeight = 0;  while (nTree < nVerts - 1) {  vertexList[currentVert].isInTree = true;  nTree++;  for (int j = 0; j < nVerts; j++) {  if (j == currentVert) { // skip if it’s us  continue;  }  if (vertexList[j].isInTree) { // skip if in the tree  continue;  }  int distance = adjMat[currentVert][j];  if (distance == INFINITY) {  continue;  }  putInPQ(j, distance);  }  if (thePQ.size() == 0) {  System.out.println("GRAPH NOT CONNECTED");  return;  }  Edge theEdge = thePQ.removeMin();  int sourceVert = theEdge.srcVert;  currentVert = theEdge.destVert;  System.out.print(vertexList[sourceVert].label);  System.out.print(vertexList[currentVert].label);  System.out.print(" ");  totalWeight += theEdge.distance;  }  System.out.println("\nToplam Agirlik: " + totalWeight);  for (int j = 0; j < nVerts; j++) {  vertexList[j].isInTree = false;  }  } |

ekran görüntüsü, metin, grafik yazılımı, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

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

|  |
| --- |
| public void bft(int start) {  boolean[] visited = new boolean[nVerts];  Queue<Integer> queue = new LinkedList<>();  visited[start] = true;  displayVertex(start);  queue.add(start);  while (!queue.isEmpty()) {  int currentVertex = queue.poll();  for (int j = 0; j < nVerts; j++) {  if (adjMat[currentVertex][j] != INFINITY && !visited[j]) {  visited[j] = true;  displayVertex(j);    queue.add(j);  }  }  }  System.out.println("");  } |

ekran görüntüsü, metin, grafik yazılımı, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldumetin, yazı tipi, ekran görüntüsü, siyah içeren bir resim

Açıklama otomatik olarak oluşturuldu

//Insert your test screenshot or output here

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

//Fill the table and insert here.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Dijkstra’s SP** | Prim’s MST | BFT | **Heap Deletion** |
| **Big-O** (Zaman Karmaşıklığı  Big-O Notasyonuna Göre) | O((E+V)logV) | O((E+V)logV) | O(V+E) | O(logn) |

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

|  |
| --- |
| def draw\_graph(graph):  pos = nx.spring\_layout(graph) # Düğümleri yerleştirme algoritması  nx.draw(graph, pos, with\_labels=True, font\_weight='bold', node\_size=700, node\_color='lightblue', font\_size=8)  labels = nx.get\_edge\_attributes(graph, 'weight')  nx.draw\_networkx\_edge\_labels(graph, pos, edge\_labels=labels)  plt.show() |

//Insert your test screenshot hereçizgi, ekran görüntüsü, diyagram içeren bir resim

Açıklama otomatik olarak oluşturuldu

## 4.b Computing distances of nodes

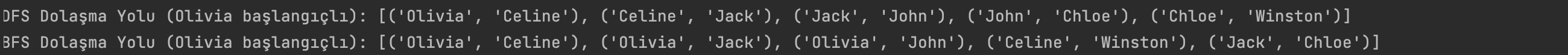
|  |
| --- |
| def dijkstra\_path(graph):  distances = {}  for person1 in graph.nodes:  for person2 in graph.nodes:  if person1 != person2:  distance = nx.dijkstra\_path\_length(graph, source=person1, target=person2)  distances[(person1, person2)] = distance  return distances |

## metin, ekran görüntüsü, tasarım içeren bir resim Açıklama otomatik olarak oluşturuldu

//Insert your results here

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

|  |
| --- |
| **def** dfs\_traversal(graph, start\_node):  **return** list(nx.dfs\_edges(graph, source=start\_node))   **def** bfs\_traversal(graph, start\_node):  **return** list(nx.bfs\_edges(graph, source=start\_node)) |

//Insert your results here

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

//Explain how computing SP, BFT and MST for this graph can be useful.

Shortest Path : İki kişi arasındaki en kısa yolları belirleyerek, iletişim için en efficient yolu bulur Acil durumlar için etkili olabilir

BFS : Belli bir kişiden başlayarak, o kişinin ulaşabileceği diğer kişileri belirlemek için kullanılır

MST : Cevap süreleri arasındaki temel bağlantıları belirleyerek, iletişim maliyetlerini minimize etmek için kullanılır

## 4.e Method for accessing to the doctor

|  |
| --- |
| Dijkstra Short Path |

//Explain the complexity of the method.

Düğüm sayısını (V) ve kenar sayısı (E) için O((V+E)⋅log(V))

//Explain how you should modify the method for the given requirement.

Graphteki ağırlıkları iki kişi için açılma olasılıklarıyla çarparak yeni bir ağırlıklı graph elde ederim ve buna göre bir Dijkstra Shortest Pathi bulurum

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

// Insert your comparison here.

Prim, MGA'yı tek bir düğümden büyütür; Kruskal, döngü oluşturmayan en küçük kenarları ekler.

Prim genellikle bir öncelik kuyruğu kullanır; Kruskal, döngü tespiti için bir bağlantısız küme veri yapısı kullanır.

Kruskal, seyrek grafikler için daha verimli; Prim, yoğun grafikler için daha verimli olabilir.

Prims Complexity: O((V + E) \* log(V)) Kruskal : Complexity: O(E \* log(E))

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

|  |
| --- |
| package dataStc4;  import java.util.HashMap;  import java.util.Map;  public class TrieNode {  Map<Character, TrieNode> cocuklar;  boolean kelimeninSonu;  public TrieNode() {  this.cocuklar = new HashMap<>();  this.kelimeninSonu = false;  }  }  class Trie {  private TrieNode kok;  public Trie() {  this.kok = new TrieNode();  }  public void ekle(String kelime) {  TrieNode current = kok;  for (char harf : kelime.toCharArray()) {  current = current.cocuklar.computeIfAbsent(harf, k -> new TrieNode());  }  current.kelimeninSonu = true;  }  // Arama, silme gibi ek metodlar buraya eklenebilir.  public static void main(String[] args) {  Trie trie = new Trie();  // Ekleme metodunu test et  trie.ekle("elma");  trie.ekle("armut");  trie.ekle("art");  trie.ekle("bardak");  // Trie'de kelimelerin olup olmadığını test et  System.out.println(" 'elma' kelimesi mevcut mu? " + trie.ara("elma"));  System.out.println(" 'armut' kelimesi mevcut mu? " + trie.ara("armut"));  }  private boolean ara(String kelime) {  TrieNode current = kok;  for (char harf : kelime.toCharArray()) {  current = current.cocuklar.get(harf);  if (current == null) {  return false; // Önek bulunamadı  }  }  return current.kelimeninSonu;  }  } |

//Insert your test screenshot or output here

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

//Explain your code briefly.

Kelimeyi harflerine ayırır ve her harfi bir düğüm olarak temsil eder düğümdeki harf çocukları için key olacak şekilde bir ağaç oluşturur kelimeninSonu attribute ile kelimenin sonuna geldiği durumu true yapar

## 5.c. Explanation of 4 Terms

// Insert your explanations here

## Self-assessment Table

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Points** | **Estimated Grade** | **Explanation** |
| **1 a) AVL Tree** | **10** |  |  |
| **1 b) Heap** | **10** |  |  |
| **2) B-Tree Insertion / AVL Tree Insertion / Red-Black Trees / Huffman Encoding Tree** | **10** |  |  |
| **3 a) Dijkstra’s shortest path code + test** | **4** | 4 | Doğru çalışıyor |
| **3 b) Prim’s MST code + test** | **4** | 4 | Doğru çalışıyor |
| **3 c) BFT or DFT code + test** | **3** | 3 | Doğru çalışıyor |
| **3 d) Filling Big-O Table** | **4** | 4 | Doğru hesaplandı |
| **4 a) Graph Drawing** | **3** | 3 | Doğru Çalışıyor |
| **4 b) Finding Shortest Paths with Dijkstra’s** | **3** | 3 | Doğru çalışıyor |
| **4 c) DFS and BFS** | **3** | 3 | Doğru çalışıyor |
| **4 d) Thinking and Writing aims of given algorithms** | **3** | 3 | Mantıklı bir açıklama |
| **4 e) Real Life Application** | **3** | 3 | Mantıklı bir açıklama |
| **5 i) Comparison (Prim’s & Kruskal’s Algorithm)** | **5** | 5 |  |
| **5 ii) Trie Data Structure and Insertion Method** | **5** | 5 |  |
| **5 iii) Explanations of 4 terms** | **10** |  |  |
| **Demo Video for Source Codes and Tests of Q2 Q3a, Q3b, Q3c and Q5.ii .** | **10** |  |  |
| **Self-assessment Table** | **10** |  |  |
| **Total** | **100** |  |  |