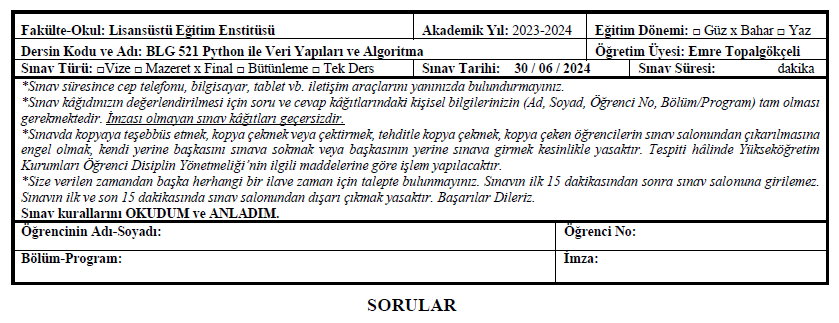
****

ABDULKADİR CESUROĞLU

20231556014

YAPAY ZEKA MÜHENDİSLİĞİ TEZSİZ YÜKSEK LİSANS

Sınavınız 3 sorudan oluşmaktadır. **Tüm soruları cevaplamanız beklenmektedir.** Cevaplarınızı Python programlama dilinde ve istediğiniz herhangi bir Geliştirme Ortamını (IDE) kullanarak verebilirsiniz. Çizim yapmanız istenen sorularda, çizimlerinizin yeterli açıklıkta olması şartıyla istediğiniz formatta (örneğin, dijital çizim araçları, elle çizim veya grafik yazılımları) yapabilirsiniz.

***Lütfen cevap kağıtlarınızı resmi cevap kağıdı şablonuna elle yazarak Enstitü tarafından belirlenen yerde ve saatte imza karşılığı teslim ediniz,*** ayrıca cevap kağıdınızın bir kopyasını en geç sınav saatine kadar emre.topalgokceli@nisantasi.edu.tr adresine gönderiniz. Lütfen unutmayın, e-posta ile iletmeniz fiziki teslimat yükümlülüğünüzün yerine geçmez.

**Hatırlatma I:** Sınavda ve derslerde öğrendiğiniz yöntem ve teknikleri kullanmanız önemle rica olunur. Başlangıç seviyesinde beklenenin ötesinde tekniklerin kullanılmasının uygun olmadığını lütfen unutmayınız. Soruları kendi bilgi ve becerilerinizle çözmeniz beklenmektedir. Gerek görülen öğrenciler, kodlarını çalıştırarak sınav kağıtlarını mülakat yoluyla açıklamaya davet edilebilir.

**Hatırlatma II:** Cevaplarınızın hangi soru ya da sorulara ait olduğunu belirtmezseniz, değerlendirmeye alınamayacağını unutmayın.

Başarılar dilerim.

**Soru 1** (30 puan): Aşağıdaki soruları İkili Arama Ağacı (Binary Search Tree - BST) kullanmaya karar verdiğinizi varsayarak cevaplayınız.

a)

Kök: 1300

[1300,1100,1500,1000,1150,1400,1700,800,1010,1140,1299,1395,1405,1695,1705,795,805,1005,1015,1135,1145,1294,1299,1300,1396,1400,1406,1694,1696,1704,1706]beyaz içeren bir resim

Açıklama otomatik olarak oluşturuldu

b)

class BST:

def \_\_init\_\_(self, value):

self.value = value

self.left = None

self.right = None

def add (self, value):

if value < self.value:

if self.left is None:

self.left = BST(value)

else:

self.left.add(value)

else:

if self.right is None:

self.right = BST(value)

else:

self.right.add(value)

tree = BST(794)

for i in [1300,1100,1500,1000,1150,1400,1700,800,1010,1140,1299,1395,1405,1695,1705,795,805,1005,1015,1135,1145,1294,1299,1300,1396,1400,1406,1694,1696,1704,1706]:

tree.add(i)

c) Yazdığınız ekleme metodunu kullanarak ağaca 30, 40, 50 ve 60 değerlerinde düğümler ekleyiniz.

for i in [30, 40, 50, 60]:

tree.add(i)



**Soru 2** (20 puan): Yukarıda yer alan grafı dikkate alarak aşağıda yer alan soruları yanıtlayınız.

1. a)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G |
| A | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| B | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| C | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| D | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| E | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| F | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| G | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

1. b)

A: B,C

B: A,D,G

C: E,D,A

D: F,E,C,B

E: D,C,G

F: G,D

G: B,E,F

**Soru 3** (50 puan):

Düğümler: A, B, C, D, E, F

Ağırlıklı Kenarlar:

* A - B: 4
* A - C: 2
* B - C: 5
* B - D: 10
* C - D: 3
* C - E: 4
* D - F: 11
* E - D: 1
  + E - F: 5

a)

def dijkstra(matrix, start):

n = len(matrix) # Matrisin boyutunu n olarak belirle

distances = [float('infinity')] \* n # Tüm düğümler için mesafeleri sonsuz olarak başlat

distances[start] = 0 # Başlangıç düğümünün mesafesini 0 olarak ayarla

visited = [False] \* n # Tüm düğümler için ziyaret durumunu False olarak başlat

previous\_nodes = [None] \* n # Tüm düğümler için önceki düğümleri None olarak başlat

for \_ in range(n): # n kere döngüye gir

min\_distance = float('infinity') # Minimum mesafeyi başta sonsuz olarak ayarla

current\_node = None # Şu anki düğümü başta None olarak ayarla

for node in range(n): # Tüm düğümleri kontrol et

if not visited[node] and distances[node] < min\_distance: # Eğer düğüm ziyaret edilmemişse ve mesafesi minimumdan küçükse

min\_distance = distances[node] # Minimum mesafeyi güncelle

current\_node = node # Şu anki düğümü güncelle

if current\_node is None: # Eğer şu anki düğüm None ise, döngüyü kır

break

visited[current\_node] = True # Şu anki düğümü ziyaret edilmiş olarak işaretle

for neighbor in range(n): # Tüm komşuları kontrol et

if matrix[current\_node][neighbor] != 0 and not visited[neighbor]: # Eğer komşu bağlantısı varsa ve ziyaret edilmemişse

distance = distances[current\_node] + matrix[current\_node][neighbor] # Yeni mesafeyi hesapla

if distance < distances[neighbor]: # Eğer yeni mesafe önceki mesafeden küçükse

distances[neighbor] = distance # Mesafeyi güncelle

previous\_nodes[neighbor] = current\_node # Önceki düğümü güncelle

return distances, previous\_nodes # Mesafeleri ve önceki düğümleri geri döndür

# Graf matrisi

graph\_matrix = [

[0, 4, 2, 0, 0, 0],

[0, 0, 5, 10, 0, 0],

[0, 0, 0, 3, 4, 0],

[0, 0, 0, 0, 0, 11],

[0, 0, 0, 1, 0, 5],

[0, 0, 0, 0, 0, 0],

]

# Başlangıç düğümü

start\_node = 0

# Bitiş düğümü

end\_node = 5

# Dijkstra algoritmasını çalıştır

distances, previous\_nodes = dijkstra(graph\_matrix, start\_node)

# Mesafeleri yazdır

print("Distances:", distances)

# Önceki düğümleri yazdır

print("Previous nodes:", previous\_nodes)

b)

çizgi, daire, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu