DİJKSTRA ALGORİTMASININ GÖRSELLEŞTİRİLMESİ

Bu Python kodu, PyQt5 kütüphanesi kullanılarak yazılmış bir grafiksel kullanıcı arayüzü içinde Dijkstra algoritmasının görselleştirilmesini sağlar.

Hakan KARATEKE

Kullanılan Kütüphaneler

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton, QHBoxLayout, QComboBox, QGraphicsTextItem
from PyQt5.QtGui import QBrush, QColor, QPainter  # QPainter's ekledik
from PyQt5.QtWidgets import QDialog, QVBoxLayout, QLabel
from PyQt5.QtWidgets import QGraphicsScene, QGraphicsView, QGraphicsEllipseItem, QGraphicsLineItem
import heapq
```

1. **sys**: Sistemle ilgili işlemler için kullanılır.

2. **PyQt5.QtWidgets**: PyQt5 kütüphanesinin Widget sınıflarını içerir (*QApplication, QWidget, QPushButton, QHBoxLayout, QComboBox, QGraphicsTextItem, QDialog, QVBoxLayout, QLabel*).

3. **PyQt5.QtGui**: PyQt5 kütüphanesinin Grafik Arayüz (Graphics) sınıflarını içerir (*QBrush, QColor, QPainter*).

Kullanılan Class ve Metodlar

`Dijkstra Class`

```
1 usage
class Dijkstra:
    def __init__(self, graf):
        self.graf = graf
    2 usages
    def en_kisa_yollari_bul(self, baslangic):...
    3 usages
    def en_kisa_yol(self, baslangic, bitis):...
    def en_kisa_ikinci_yolu_bul(self, baslangic):...
```

→ *`__init__(self, graf)`*: Dijkstra sınıfının başlatıcı metodu, graf adlı bir parametre alır ve grafı bu sınıfa kaydeder.

→ `en_kisa_yollari_bul(self, baslangic):` Dijkstra algoritması kullanılarak en kısa yolları bulur.

```python
def en_kisa_yollari_bul(self, baslangic):
    mesafeler = {sehir: float('inf') for sehir in self.graf}
    onceki_sehirler = {sehir: None for sehir in self.graf}
    mesafeler[baslangic] = 0
    oncelikli_kuyruk = [(0, baslangic)]

    while oncelikli_kuyruk:
        suanki_mesafe, suanki_sehir = heapq.heappop(oncelikli_kuyruk)

        if suanki_mesafe > mesafeler[suanki_sehir]:
            continue

        for komsu, agirlik in self.graf[suanki_sehir]["komsular"].items():
            mesafe = suanki_mesafe + agirlik

            if mesafe < mesafeler[komsu]:
                mesafeler[komsu] = mesafe
                onceki_sehirler[komsu] = suanki_sehir
                heapq.heappush(oncelikli_kuyruk, _item: (mesafe, komsu))

    return mesafeler, onceki_sehirler
```

→ `en_kisa_yol(self, baslangic, bitis):` İki şehir arasındaki en kısa yolu bulur.

```python
def en_kisa_yol(self, baslangic, bitis):
    mesafeler, onceki_sehirler = self.en_kisa_yollari_bul(baslangic)
    yol = []
    while bitis is not None:
        yol.insert( _index: 0, bitis)
        bitis = onceki_sehirler[bitis]
    return yol
```

→ `en_kisa_ikinci_yolu_bul(self, baslangic):` Belirtilen başlangıç şehri için en kısa ikinci yolu bulur.

```python
def en_kisa_ikinci_yolu_bul(self, baslangic):
    mesafeler = {sehir: float('inf') for sehir in self.graf}
    onceki_sehirler = {sehir: None for sehir in self.graf}
    mesafeler[baslangic] = 0
    oncelikli_kuyruk = [(0, baslangic)]

    while oncelikli_kuyruk:
        suanki_mesafe, suanki_sehir = heapq.heappop(oncelikli_kuyruk)

        if suanki_mesafe > mesafeler[suanki_sehir]:
            continue

        for komsu, agirlik in self.graf[suanki_sehir]["komsular"].items():
            mesafe = suanki_mesafe + agirlik

            if mesafe < mesafeler[komsu]:
                mesafeler[komsu] = mesafe
                onceki_sehirler[komsu] = suanki_sehir
                heapq.heappush(oncelikli_kuyruk, _item: (mesafe, komsu))

    en_kisa_yol = self.en_kisa_yol(baslangic, min(mesafeler, key=mesafeler.get))

    # En kısa yolu ve mesafeyi kaldır
    mesafeler.pop(en_kisa_yol[-1])

    # En kısa ikinci yolu bul
    en_kisa_ikinci_yol = self.en_kisa_yol(baslangic, min(mesafeler, key=mesafeler.get))

    return en_kisa_ikinci_yol
```

YolPenceresi Class (QDialog)

→ `__init__(self, yol, mesafe, graf):` Başlangıç, bitiş şehirleri arasındaki en kısa yolu ve mesafeyi içeren bir pencere oluşturur.

```python
class YolPenceresi(QDialog):
    def __init__(self, yol, mesafe, graf):
        super().__init__()

        self.setWindowTitle("En Kısa Yol")

        layout = QVBoxLayout()

        for sehir in yol:
            layout.addWidget(QLabel(sehir))

        layout.addWidget(QLabel("Toplam Mesafe: " + str(mesafe)))

        # Grafik sahne ve görüntü oluştur
        self.scene = QGraphicsScene()
        self.view = QGraphicsView(self.scene)
        self.view.setRenderHint(QPainter.Antialiasing, on: True)
        self.view.setRenderHint(QPainter.TextAntialiasing, on: True)
        self.view.setRenderHint(QPainter.SmoothPixmapTransform, on: True)
        self.view.setSceneRect(0, 0, 500, 500)

        layout.addWidget(self.view)

        self.setLayout(layout)

        # En kısa yolu çiz
        self.yolu_ciz(yol, graf)

    1 usage
    def yolu_ciz(self, yol, graf):...
```

→ `yolu_ciz(self, yol, graf):` Graf üzerindeki yolu çizer.

```python
def yolu_ciz(self, yol, graf):
    for i in range(len(yol) - 1):
        baslangic = yol[i]
        bitis = yol[i + 1]

        baslangic_koordinat = graf[baslangic]["koordinat"]
        bitis_koordinat = graf[bitis]["koordinat"]

        # Düğümleri çiz
        for koordinat in [baslangic_koordinat, bitis_koordinat]:
            x, y = koordinat
            ellipse = QGraphicsEllipseItem(x * 50 - 5, y * 50 - 5, 10, 10)
            ellipse.setBrush(QBrush(QColor(0, 0, 255)))
            self.scene.addItem(ellipse)

        # Kenarı çiz
        line = QGraphicsLineItem(baslangic_koordinat[0] * 50, baslangic_koordinat[1] * 50, bitis_koordinat[0] * 50, bitis_koordinat[1] * 50)
        self.scene.addItem(line)
```

DijkstraArayuz Class (QWidget)

→ `__init__(self, graf):` Dijkstra algoritması için bir kullanıcı arayüzü oluşturur.

```
class DijkstraArayuz(QWidget):
    def __init__(self, graf):
        super().__init__()

        self.graf = graf
        self.setWindowTitle("Dijkstra Algoritması Görselleştirme")

        # Grafik sahne ve görüntü oluştur
        self.scene = QGraphicsScene()
        self.view = QGraphicsView(self.scene)
        self.view.setRenderHint(QPainter.Antialiasing, on: True)
        self.view.setRenderHint(QPainter.TextAntialiasing, on: True)
        self.view.setRenderHint(QPainter.SmoothPixmapTransform, on: True)
        self.view.setSceneRect(0, 0, 500, 500)

        # Başlangıç ve bitiş şehirlerini seçmek için ComboBox'lar
        self.baslangic_combobox = QComboBox()
        self.baslangic_combobox.addItems(list(self.graf.keys()))
        self.bitis_combobox = QComboBox()
        self.bitis_combobox.addItems(list(self.graf.keys()))

        # En kısa yolu bulan buton
        self.bul_butonu = QPushButton("En Kısa Yolu Bul")
        self.bul_butonu.clicked.connect(self.en_kisa_yolu_bul_ve_gorsellestir)

        # Layout oluştur
        layout = QVBoxLayout()
        h_layout = QHBoxLayout()
        h_layout.addWidget(self.baslangic_combobox)
        h_layout.addWidget(self.bitis_combobox)
        h_layout.addWidget(self.bul_butonu)
        layout.addWidget(self.view)
        layout.addLayout(h_layout)

        self.setLayout(layout)

        # Grafik çizimini yap
        self.grafi_ciz()
```

→`en_kisa_yolu_bul_ve_gorsellestir(self):` Kullanıcının seçtiği başlangıç ve bitiş şehirleri arasındaki en kısa yolu bulur ve görselleştirir.

```
def en_kisa_yolu_bul_ve_gorsellestir(self):
    baslangic_sehri = self.baslangic_combobox.currentText()
    bitis_sehri = self.bitis_combobox.currentText()

    dijkstra = Dijkstra(self.graf)
    en_kisa_yol = dijkstra.en_kisa_yol(baslangic_sehri, bitis_sehri)
    mesafe = dijkstra.en_kisa_yollari_bul(baslangic_sehri)[0][bitis_sehri]

    print("En kısa yol:", en_kisa_yol)
    print("Toplam Mesafe:", mesafe)

    yol_penceresi = YolPenceresi(en_kisa_yol, mesafe,graf)
    yol_penceresi.exec_()
```

→ `grafi_ciz(self)`: Graf üzerindeki şehirleri ve kenarları çizer.

```
def grafi_ciz(self):
    for sehir, veri in self.graf.items():
        x, y = veri["koordinat"]

        # Düğümü çiz
        ellipse = QGraphicsEllipseItem(x * 50 - 5, y * 50 - 5, 10, 10)
        ellipse.setBrush(QBrush(QColor(0, 0, 255)))
        self.scene.addItem(ellipse)

        # Düğüm etiketi
        label = QGraphicsTextItem(sehir)
        label.setPos(x * 50, y * 50 - 15)
        self.scene.addItem(label)

        for komsu, agirlik in veri["komsular"].items():
            komsu_x, komsu_y = self.graf[komsu]["koordinat"]

            # Kenarı çiz
            line = QGraphicsLineItem(x * 50, y * 50, komsu_x * 50, komsu_y * 50)
            self.scene.addItem(line)

            # Kenar etiketi
            middle_x = (x + komsu_x) * 25
            middle_y = (y + komsu_y) * 25
            label = QGraphicsTextItem(str(agirlik))
            label.setPos(middle_x, middle_y)
            self.scene.addItem(label)

    # Görselleştirilen grafikte dolaşan bir etiket oluştur
    self.sehir_bilgisi_etiketi = QLabel()
    self.sehir_bilgisi_etiketi.setStyleSheet("background-color: white; border: 1px solid black;")
    self.scene.addWidget(self.sehir_bilgisi_etiketi)
    self.sehir_bilgisi_etiketi.hide()
```

→ `mouseMoveEvent(self, event)`: Fare hareket ettiğinde şehir bilgisi etiketini günceller.

```
def mouseMoveEvent(self, event):
    # Fare hareket ettikçe etiketi güncelle
    pos = self.view.mapToScene(event.pos())
    items = self.scene.items(pos)
    for item in items:
        if isinstance(item, QGraphicsEllipseItem):
            sehir = [sehir for sehir, veri in self.graf.items() if veri["koordinat"] == (pos.x() // 50, pos.y() // 50)]
            if sehir:
                self.sehir_bilgisi_etiketi.setText(f"Şehir: {sehir[0]}")
                self.sehir_bilgisi_etiketi.show()
            else:
                self.sehir_bilgisi_etiketi.hide()
```

→ `calistir(self)`: Kullanıcı arayüzünü gösterir.

```
def calistir(self):
    self.show()
```

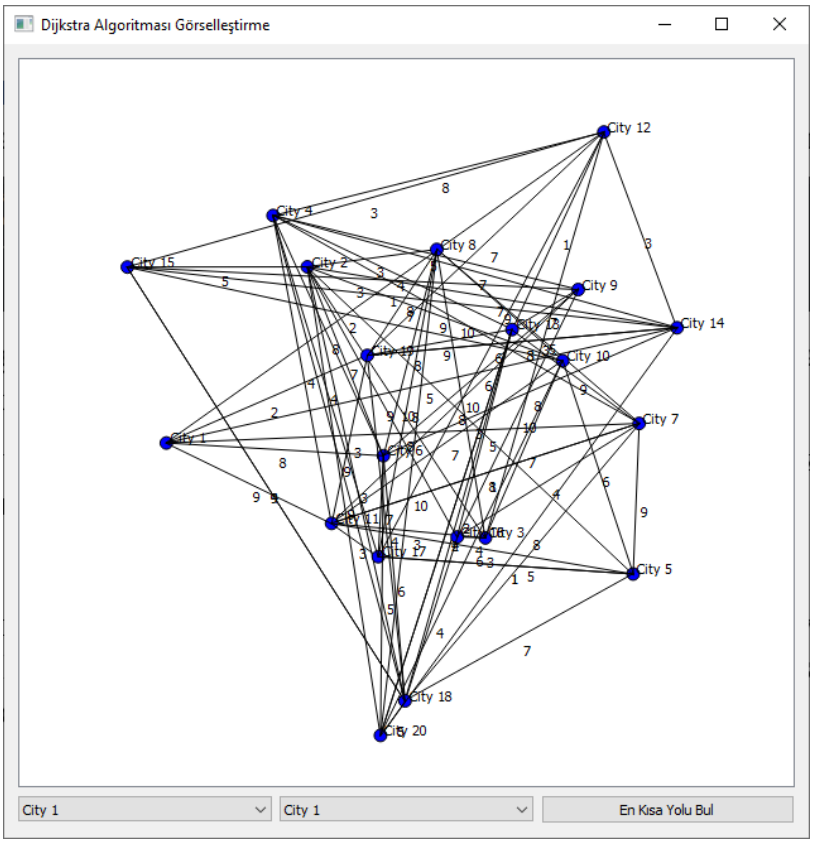
Ana Program

- Kullanıcıdan şehir sayısını ve her şehir için komşu sayısını alır.

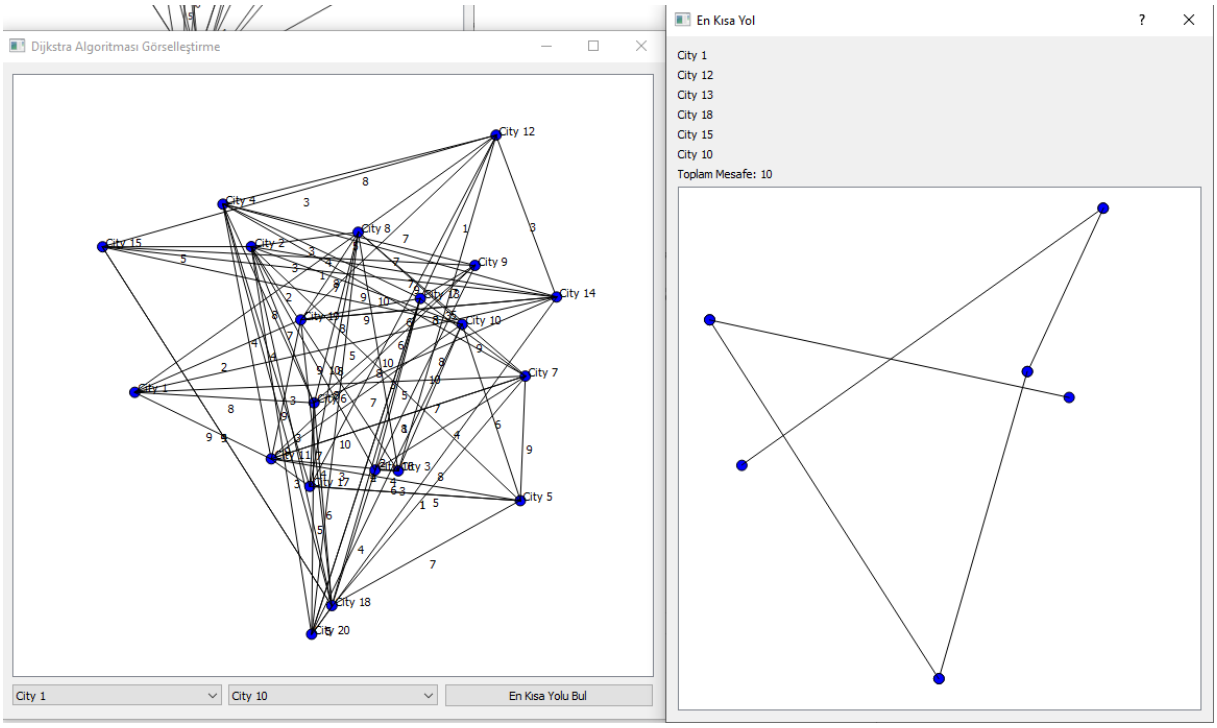
```
C:\Users\Hakan\Desktop\Pyqt\DijkstraProject\Scripts\python.exe C:\Users\Hakan\Desktop\Pyqt\main.py
Şehir sayısını girin: 20
Her şehir için komuşu sayısını girin: 4
```

- Rastgele koordinatlar ve komşuluk ilişkileri oluşturarak bir graf oluşturur.

- Oluşturulan grafı kullanarak PyQt5 tabanlı bir kullanıcı arayüzü başlatır.



Bu program, rastgele bir graf üzerinde Dijkstra algoritmasını uygular ve kullanıcıya en kısa yolu görsel olarak sunar. Ayrıca, fare ile şehirler arasında dolaşıldıkça bir etiket aracılığıyla şehir bilgisi sağlar.



Kaynakçalar

- https://youtu.be/aHDuxPmNDH0?si=CIFsbTqWE5dWZH-9
- https://chat.openai.com/
- https://bard.google.com/
- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/