T.C

KOCAELİ SAĞLIK VE TEKNOLOJİ ÜNİVERSİTESİ
MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

ÖDEV KONUSU
LİMAN OTOMASYONU

Hazırlayanlar

Nazlı Su KETÇİ 220501007

https://github.com/nzlktc02

Amine DERİN 220501004

https://github.com/aminederin

DERS SORUMLUSU

Prof.Dr.Hüseyin Tarık DURU

13.12.2023

```
import pandas as pd



class Gemi:
    Instances: dict = {}

    def __init__(self, gemi_adi, kapasite, gidecek_ulke):
        self.info = {
            "gemi_adi": gemi_adi,
            "kapasite": kapasite,
            "gidecek_ülke": gidecek_ulke
        }
        self.sira = gemi_adi
        self.guncel_kapasite = 0
```

Pandas, Python'da veri analizi ve manipülasyonu için sıkça kullanılan bir kütüphanedir.

class Gemi: Gemi adında bir sınıf tanımlanıyor. Sınıflar, nesne tabanlı programlamada (OOP) temel bir yapıdır. Gemi sınıfı, gemi nesnelerini temsil eder.

Instances: dict = {}: Instances adında bir sınıf değişkeni tanımlıyoruz. Bu, Gemi sınıfından oluşturulan gemi nesnelerini takip etmek için kullanılacak bir sözlüktür. (dictionary). Her gemi nesnesi, geliş zamanına göre bu sözlüğe ekleniyor.

def __init__(self, gemi_adi, kapasite, gidecek_ulke): Bu, Gemi sınıfının inşa (constructor) metodudur. Bir Gemi nesnesi oluşturulduğunda otomatik olarak çağrılıyor. Parametreler olarak gemi_adi, kapasite, ve gidecek_ulke parametresini alıyoruz.

self.info = {"gemi_adi": gemi_adi, "kapasite": kapasite, "gidecek_ülke": gidecek_ulke}: Bu satır, Gemi nesnesinin info adlı bir özelliğini oluşturur ve bu özelliğe gemi ile ilgili bilgileri içeren bir sözlük atar.

self.sira = gemi_adi: Bu satır, Gemi nesnesinin sira adlı bir özelliğini oluşturuyor ve bu özelliğe

gemi adını atıyor.

self.guncel_kapasite = 0: Bu satır, Gemi nesnesinin guncel_kapasite adlı bir özelliğini oluşturuyor ve bu özelliğe başlangıçta geminin taşıdığı yük miktarını sıfır olarak atıyor.

Bu koddaki Gemi sınıfı, gemilerin özelliklerini ve bu gemilerle ilgili işlem adımlarını gerçekleştirmek için kullanılacak metotları içeriyor. Bu özellikler ve metotlar, gemi objelerini oluşturmak ve bu objelerle çalışmak için kullanılıyor.

```
def yukle(self, yuk):
        self.guncel_kapasite += yuk


    def gitmeye_hazir_mi(self):
        return self.guncel_kapasite / self.info["kapasite"] >= 0.95


    @staticmethod
    def oku(path):
        df = pd.read_csv(path, encoding="cp1254")
        max_t = 0
        for i in range(len(df)):
            gemi = df.iloc[i]
            instanceGemi = Gemi(
                df.iloc[i]["gemi_adı"],
                df.iloc[i]["kapasite"],
                df.iloc[i]["gidecek_ülke"]
            )
            if Gemi.Instances.get(gemi["geliş zamanı"]) is None:
                Gemi.Instances[gemi["geliş_zamanı"]] = [instanceGemi]
            else:
                Gemi.Instances[gemi["geliş zamanı"]].append(instanceGemi)
            max_t = max(max_t, gemi["geliş_zamanı"])
        return max_t
```

def yukle(self, yuk): Bu metot, geminin mevcut kapasitesine yük eklemek için kullanılıyor. yuk parametresi, gemiye eklenen yük miktarını temsil eder. self.guncel_kapasite özelliği, geminin şu ana kadar taşıdığı toplam yük miktarını tutar. Bu metodun içinde, self.guncel_kapasite'ye yuk ekleniyor.

def gitmeye_hazir_mi(self): Bu metot, geminin belirli bir kriteri sağlayıp sağlamadığını kontrol etmeyi sağlar. Eğer geminin mevcut kapasitesi, geminin taşıma kapasitesinin en az %95'ine eşit veya daha büyükse, geminin gitmeye hazır olduğu kabul ediliyor. Bu kontrolün sağlanması için self.guncel_kapasite'yi geminin kapasitesine (self.info["kapasite"]) bölüp, sonucun %95'ten büyük olup olmadığını kontrol etmeliyiz.

@staticmethod: Bu bir dekoratördür ve aşağıdaki metodun bir sınıf metodunu (static method) temsil ettiğini belirtir. Sınıf metodu, sınıfa ait olup, bir örnek (instance) oluşturmadan sınıf üzerinden çağrılabiliyor.

def oku(path): Bu sınıf metodu, bir CSV dosyasındaki verileri okuyarak Gemi sınıfının örneklerini oluşturuyor. path parametresi, okunacak CSV dosyasının yolu olarak kullanılıyor. Metodun içinde pandas kütüphanesinin read_csv fonksiyonu kullanılarak CSV dosyası okunuyor ve bir DataFrame'e dönüştürülüyor.

df = pd.read_csv(path, encoding="cp1254"): CSV dosyasını okur ve bir DataFrame'e (df) dönüştürülüyor.

max_t = 0: En büyük zaman değerini (geliş zamanı) tutacak değişkeni başlangıçta sıfır olarak tanımlanıyor.

for i in range(len(df)): DataFrame'deki her satır için döngü başlatıyor.

gemi = df.iloc[i]: DataFrame'deki i. satırı temsil eden bir Seri (Series) nesnesi oluşturuluyor.

instanceGemi = Gemi(df.iloc[i]["gemi_adı"], df.iloc[i]["kapasite"], df.iloc[i]["gidecek_ülke"]): Oluşturulan seri üzerinden bir Gemi örneği (instanceGemi) oluşturuluyor.

if Gemi.Instances.get(gemi["geliş_zamanı"]) is None: Gemi geliş zamanına göre bir sözlük (Instances) içinde önceki gemi örnekleri aranıyor. Eğer bu geliş zamanına ait daha önce bir gemi örneği yoksa, yeni bir liste oluşturuluyor ve bu gemi örneği listenin ilk elemanı olarak ekleniyor.

else: Eğer geliş zamanına ait bir liste zaten varsa, bu listeye yeni gemi örneği eklenir.

max_t = max(max_t, gemi["geliş_zamanı"]): En büyük zaman değerini günceller.

return max_t: En büyük zaman değerini döndürmeyi sağlar.

Bu metot, verilen bir CSV dosyasındaki gemi bilgilerini okuyor, bu bilgileri kullanarak Gemi sınıfından örnekler oluşturuyoruz ve bu örnekleri geliş zamanına göre sınıflandırıyoruz.

```
class Tir:
    Instances: dict = {}

    def __init__(self, plaka: str, ulke: str, ton20: int, ton30: int, yuk_miktari: int, maliyet: int):
        self.info = {
```

```
            "plaka": plaka,
            "ülke": ulke,
            "20_ton_adet": ton20,
            "30_ton_adet": ton30,
            "yük_miktarı": yuk_miktari,
            "maliyet": maliyet
        }
        self.sira = plaka.split("_")[-1]

    @staticmethod
    def oku(path):
        df = pd.read_csv(path, encoding="cp1254")
        max_t = 0
        for i in range(len(df)):
            tir = df.iloc[i]
            instanceTir = Tir(
                df.iloc[i]["tır_plakası"],
                df.iloc[i]["ülke"],
                df.iloc[i]["20_ton_adet"],
                df.iloc[i]["30_ton_adet"],
                df.iloc[i]["yük_miktarı"],
                df.iloc[i]["maliyet"]
            )
            if Tir.Instances.get(tir["geliş_zamanı"]) is None:
                Tir.Instances[tir["geliş_zamanı"]] = [instanceTir]
            else:
                Tir.Instances[tir["geliş_zamanı"]].append(instanceTir)
            max_t = max(max_t, tir["geliş_zamanı"])
        return max_t
```

def __init__(self, plaka: str, ulke: str, ton20: int, ton30: int, yuk_miktari: int, maliyet: int): Bu metot, Tir sınıfının bir örneğini oluşturuyor (initialize eder) ve aşağıdaki özelliklere sahip bir info sözlüğü oluşturuyor:

plaka: Tırın plakası (str)

ülke: Tırın ait olduğu ülke (str)

20_ton_adet: 20 tonluk yük adeti (int)

30_ton_adet: 30 tonluk yük adeti (int)

yük_miktarı: Tırın taşıdığı toplam yük miktarı (int)

maliyet: Tırın taşıma maliyeti (int)

Ayrıca, sira özelliği, tır plakasından elde ediliyor.

@staticmethod: Bu bir dekoratördür ve aşağıdaki metodun bir sınıf metodunu (static method) temsil ettiğini belirtiyor. Sınıf metodu, sınıfa ait olup, bir örnek (instance) oluşturmadan sınıf üzerinden çağrılabiliyor.

def oku(path): Bu sınıf metodu, bir CSV dosyasındaki verileri okuyarak Tir sınıfının örneklerini oluşturuyor. path parametresi, okunacak CSV dosyasının yolu olarak kullanılıyor. Metodun içinde pandas kütüphanesinin read_csv fonksiyonu kullanılarak CSV dosyası okunur ve bir DataFrame'e dönüştürülüyor.

df = pd.read_csv(path, encoding="cp1254"): CSV dosyasını okur ve bir DataFrame'e (df) dönüştürür.

max_t = 0: En büyük zaman değerini (geliş zamanı) tutacak değişkeni başlangıçta sıfır olarak tanımlamayı sağlar.

for i in range(len(df)): DataFrame'deki her satır için döngü başlatmayı sağlar.

tir = df.iloc[i]: DataFrame'deki i. satırı temsil eden bir Seri (Series) nesnesi oluşturuyor.

instanceTir = Tir(df.iloc[i]["tır_plakası"], df.iloc[i]["ülke"], df.iloc[i]["20_ton_adet"], df.iloc[i]["30_ton_adet"], df.iloc[i]["yük_miktarı"], df.iloc[i]["maliyet"]): Oluşturulan Seri üzerinden bir Tir örneği (instanceTir) oluşturulur.

if Tir.Instances.get(tir["geliş_zamanı"]) is None: Tır geliş zamanına göre bir sözlük (Instances) içinde önceki tır örnekleri aranıyor. Eğer bu geliş zamanına ait daha önce bir tır örneği yoksa, yeni bir liste oluşturuluyor ve bu tır örneği listenin ilk elemanı olarak ekleniyor.

else: Eğer geliş zamanına ait bir liste zaten varsa, bu listeye yeni tır örneği ekleniyor.

max_t = max(max_t, tir["geliş_zamanı"]): En büyük zaman değerini güncellemeyi sağlar.

return max_t: En büyük zaman değeri döndürmeyi sağlar.

Bu metot, verilen bir CSV dosyasındaki tır bilgilerini okur, bu bilgileri kullanarak Tir sınıfından örnekler oluşturuyor ve bu örnekleri geliş zamanına göre sınıflandırıyor.

```
def istif_alani_yuku(istif_alani):
    return sum(map(lambda x: x[0], istif_alani))



if __name__ == "__main__":
    tir_max_t = Tir.oku("olaylar.csv")
    gemi_max_t = Gemi.oku("gemiler.csv")

    gemi_bekleme_alani = []
    tir_bekleme_alani = []
```

```
istif_alani = [[], []]


for t in range(1, max(tir_max_t, gemi_max_t) + 1):
    print(f"{t} tarihi için işlemler başladı")
    linc_kullanim_sayisi = 0
```

def istif_alani_yuku(istif_alani): Bu fonksiyon, bir istif alanındaki yük miktarını hesaplamak için kullanılıyor. istif_alani parametresi, iki boyutlu bir liste olarak düşünülmelidir. Bu liste, taşıma kapasitesi ve hedef ülke bilgisini içeriyor.

sum(map(lambda x: x[0], istif_alani)): istif_alani listesinin ilk elemanlarını (yani taşıma kapasitelerini) toplar ve bu toplamı döndürüyor. Yani, istif alanındaki toplam yük miktarını verir.

if __name__ == "__main__": Bu ifade, Python script'inin bir modül olarak başka bir script tarafından çağrılmadığını, yani doğrudan çalıştırıldığını kontrol etmeyi sağlar. Eğer bu script doğrudan çalıştırılıyorsa, aşağıdaki işlemleri gerçekleştiriyor.

tir_max_t = Tir.oku("olaylar.csv"): Tir sınıfının oku metodunu kullanarak "olaylar.csv" dosyasındaki tır verilerini okuyor ve en büyük geliş zamanını (tir_max_t) alıyor.

gemi_max_t = Gemi.oku("gemiler.csv"): Gemi sınıfının oku metodunu kullanarak "gemiler.csv" dosyasındaki gemi verilerini okuyor ve en büyük geliş zamanını (gemi_max_t) alıyor.

gemi_bekleme_alani = []: Gemilerin bekletildiği bir alanı oluşturur. Her zaman bir liste olarak düşünülmelidir.

tir_bekleme_alani = []: Tırların bekletildiği bir alan oluşturur. Her zaman bir liste olarak düşünülmelidir.

istif_alani = [[], []]: İki boyutlu bir liste oluşturuyor. Bu liste, iki ayrı istif alanını temsil eder. Her bir alt liste, taşıma kapasitesi ve hedef ülke bilgisini içeriyor.

for t in range(1, max(tir_max_t, gemi_max_t) + 1): Geliş zamanları arasında döngü başlatılıyor. Bu döngü, en erken geliş zamanından en geç geliş zamanına kadar devam eder.

linc_kullanim_sayisi = 0: Her bir geliş zamanında, bir sayaç başlatılıyor. Bu sayaç, bir liman kullanim sinirini belirlemekte kullanılacaktır.

if Tir.Instances.get(t) is not None: Eğer bu geliş zamanında tırlar varsa:

Her bir tırı bekletme alanına ekliyor ve bilgi mesajı yazdırıyor. Bekletme alanındaki tırları sıraya dizerek sıralıyor.

if Gemi.Instances.get(t) is not None: Eğer bu geliş zamanında gemiler varsa Her bir gemiyi bekletme alanına ekler ve bilgi mesajı yazdırıyor. Bekletme alanındaki gemileri sıraya dizerek sıralıyor.

Tırları İstif Alanına Boşaltma:

while istif_alani_yuku(istif_alani[0]) < 750 and linc_kullanim_sayisi < 20: İstif alanı 1'in yük miktarı 750 tonun altında ve liman kullanım sınırı 20'yi geçmediği sürece

Eğer bekletme alanındaki tırlar bitmemişse bekletme alanındaki ilk tırı alır.Tırın taşıdığı yük miktarını hesaplar.

Eğer istif alanı 1'in yük miktarına eklenen yük ile birlikte 750 tonu geçmeyecekse yükü istif alanı 1'e ekler.Liman kullanım sayısını bir arttırır.Bilgi mesajı yazdırır.

Eğer yük miktarı 750 tonu geçerse tırı bekletme alanına geri koyar ve döngüyü sonlandırır.

Eğer istif alanı 1'in yük miktarı 750 ton ve liman kullanım sınırı 20'den küçükse bilgi mesajı yazdırır.

while istif_alani_yuku(istif_alani[1]) < 750: İstif alanı 2'nin yük miktarı 750 tonun altındaysa

Eğer bekletme alanındaki tırlar bitmemişse:

Bekletme alanındaki ilk tırı alır.

Tırın taşıdığı yük miktarını hesaplar.

Eğer istif alanı 2'nin yük miktarına eklenen yük ile birlikte 750 tonu geçmeyecekse:

Yükü istif alanı 2'e ekler.

Liman kullanım sayısını bir arttırır.

Bilgi mesajı yazdırır.

Eğer istif alanı 2'nin yük miktarı 750 tonu geçmişse:

Bilgi mesajı yazdırır.

Gemi İstif Alanına Boşaltma:

index = 0: Bir indeks değişkeni başlatılır.

while istif_alani_yuku(istif_alani[0]) > 0 and index < len(gemi_bekleme_alani) and linc_kullanim_sayisi < 20:: İstif alanı 1'in yük miktarı sıfır değil, gemi bekletme alanındaki gemiler bitmemiş ve liman kullanım sınırı 20'yi geçmediği sürece:

Eğer bekletme alanındaki gemiler bitmemişse:

Bekletme alanındaki indeksteki gemiyi alır.

Geminin hedef ülkesini belirler.

İstif alanı 1'deki o ülkeye gidecek kargoları seçer.

Seçilen kargoları gemiye yükler.

Liman kullanım sayısını bir arttırır.

İstif alanından kargoları kaldırır.

Bilgi mesajı yazdırır.

Gemi gitmeye hazır mı diye kontrol eder.

Eğer gemi gitmeye hazırsa:

Gemi bekletme alanından çıkar.

İndeksi bir azaltır.

Döngüyü sonlandırır.

İndeksi bir arttırır.

Eğer istif alanı 1'in yük miktarı sıfır değilse:

Bilgi mesajı yazdırır.

index = 0: Bir indeks değişkeni başlatılır.

while istif_alani_yuku(istif_alani[1]) > 0 and index < len(gemi_bekleme_alani) and linc_kullanim_sayisi < 20:: İstif alanı 2'nin yük miktarı sıfır değil, gemi bekletme alanındaki gemiler bitmemiş ve liman kullanım sınırı 20'yi geçmediği sürece:

Eğer bekletme alanındaki gemiler bitmemişse:

Bekletme alanındaki indeksteki gemiyi alır.

Geminin hedef ülkesini belirler.

İstif alanı 2'deki kargoyu seçer.

Kargoyu gemiye yükler.

Liman kullanım sayısını bir arttırır.

İstif alanından kargoyu kaldırır.

Bilgi mesajı yazdırır.

Gemi gitmeye hazır mı diye kontrol eder.

Eğer gemi gitmeye hazırsa:

Gemi bekletme alanından çıkar.

İndeksi bir azaltır.

Döngüyü sonlandırır.

İndeksi bir arttırır.

Eğer istif alanı 2'nin yük miktarı sıfır değilse:

Bilgi mesajı yazdırır.

Bilgi mesajı: İlgili tarih için işlemlerin bittiğini belirten bir bilgi mesajı yazdırır.

```
# Tirlari beklet
    if Tir.Instances.get(t) is not None:
        for tir in Tir.Instances[t]:
            tir_bekleme_alani.append(tir)
            print(f"\t{t} tarihinde {tir.info['plaka']} plakalı tır bekletme alanına geldi")
        tir_bekleme_alani.sort(key=lambda tir: tir.sira)
```

if Tir.Instances.get(t) is not None: Eğer belirli bir geliş zamanında (t) tırlar varsa:

for tir in Tir.Instances[t]: Her bir tırı alır ve aşağıdaki işlemleri gerçekleştirir:

tir_bekleme_alani.append(tir): Tırı tir_bekleme_alani listesine ekler. Bu liste, bekleyen tırları temsil eder.

print(f"\t{t} tarihinde {tir.info['plaka']} plakalı tır bekletme alanına geldi"): Konsola bir bilgi mesajı yazdırır. Bu mesaj, hangi tarihte ve hangi plakalı tırın bekletme alanına geldiğini belirtir.

tir_bekleme_alani.sort(key=lambda tir: tir.sira): Bekletme alanındaki tırları, tir.sira özelliğine göre sıralar. Bu özellik, her tırın sırasını temsil eder ve tırları sıralayarak belirli bir düzen oluşturur. Bu

düzen, tırların hangi sırayla işleme alınacaklarını belirler.

Bu kısım, belirli bir geliş zamanında tırları bekletme alanına almayı ve sıraya dizmeyi sağlar. Bu sayede, tırların geliş sıralarına göre işleme alınmaları sağlanır.

```
# Gemileri beklet
        if Gemi.Instances.get(t) is not None:
            for gemi in Gemi.Instances[t]:
                gemi_bekleme_alani.append(gemi)
                print(f"\t{t} tarihinde {gemi.info['gemi_adi']} gemisi bekletme alanına geldi")
            gemi_bekleme_alani.sort(key=lambda gemi: gemi.sira)
```

if Gemi.Instances.get(t) is not None: Eğer belirli bir geliş zamanında (t) gemiler varsa:

for gemi in Gemi.Instances[t]: Her bir gemiyi alır ve aşağıdaki işlemleri gerçekleştirir:

gemi_bekleme_alani.append(gemi): Gemiyi gemi_bekleme_alani listesine ekler. Bu liste, bekleyen gemileri temsil eder.

print(f"\t{t} tarihinde {gemi.info['gemi_adi']} gemisi bekletme alanına geldi"): Konsola bir bilgi mesajı yazdırır. Bu mesaj, hangi tarihte ve hangi geminin bekletme alanına geldiğini belirtir.

gemi_bekleme_alani.sort(key=lambda gemi: gemi.sira): Bekletme alanındaki gemileri, gemi.sira özelliğine göre sıralar. Bu özellik, her geminin sırasını temsil eder ve gemileri sıralayarak belirli bir düzen oluşturur. Bu düzen, gemilerin hangi sırayla işleme alınacaklarını belirler.

```
# tirlari istif alanina bosalt
        while istif_alani_yuku(istif_alani[0]) < 750 and linc_kullanim_sayisi < 20:
            if len(tir_bekleme_alani) == 0:
                break
            tir = tir_bekleme_alani.pop(0)
            yuk = tir.info["20_ton_adet"] * 20 + tir.info["30_ton_adet"] * 30
            if istif_alani_yuku(istif_alani[0]) + yuk >
```

while istif_alani_yuku(istif_alani[0]) < 750 and linc_kullanim_sayisi < 20:: Bu while döngüsü, istif alanının yükü 750 tonu geçmediği ve linc (muhtemelen bir değişken) kullanım sayısı 20'yi geçmediği sürece çalışır.

if len(tir_bekleme_alani) == 0: Eğer bekleyen tır yoksa (tir_bekleme_alani boşsa), döngüyü sonlandırır.

tir = tir_bekleme_alani.pop(0): Bekleyen tırların bulunduğu listeden bir tır çıkarılır ve tir adlı değişkene

atanır.

yuk = tir.info["20_ton_adet"] * 20 + tir.info["30_ton_adet"] * 30: Tırın taşıdığı yük miktarı hesaplanır. Tırın bilgilerindeki 20_ton_adet ve 30_ton_adet özellikleri kullanılarak toplam yük miktarı hesaplanır

```
# tiri geri koy
            tir_bekleme_alani = [tir] + tir_bekleme_alani
            break
        else:
            istif_alani[0].append([yuk, tir.info["ülke"]])
            linc_kullanim_sayisi += 1
            print(
                f"\t{t} tarihinde {tir.info['plaka']} plakalı tır istif alani 1'e {tir.info['ülk
```

tir_bekleme_alani = [tir] + tir_bekleme_alani: Eğer tır istif alanına boşaltılamıyorsa, tır'ı geri koyar. Bu, tır'ı tir_bekleme_alani listesinin başına ekleyerek yapılır.

break: Döngüden çıkış yapılır, çünkü tır'ı istif alanına boşaltamıyoruz ve başka bir tır işleme alınmamalıdır.

else:Eğer tır başarıyla istif alanına boşaltılabiliyorsa, bu blok çalışır.

istif_alani[0].append([yuk, tir.info["ülke"]]): Tırın taşıdığı yük ve hedef ülke bilgilerini içeren bir liste, istif alanının birinci bölgesine eklenir.

linc_kullanim_sayisi += 1: Kullanım sayısı bir arttırılır.

print(f"\t{t} tarihinde {tir.info['plaka']} plakalı tır istif alani 1'e {tir.info['ülke']}'ye gidecek kargo yukledi"): İlgili bilgilerle birlikte bir print ifadesi ile ekrana bilgi mesajı yazdırılır

```
if istif_alani_yuku(istif_alani[0]) == 750 and linc_kullanim_sayisi < 20:
        print(f"\t{t} tarihinde 750 tonluk istif alanı 1 doldu")


    while istif_alani_yuku(istif_alani[1]) < 750:
        if len(tir_bekleme_alani) == 0:
            break
        tir = tir_bekleme_alani.pop(0)
        yuk = tir.info["20_ton_adet"] * 20 + tir.info["30_ton_adet"] * 30
        if istif_alani_yuku(istif_alani[1]) + yuk > 750:
```

```
            # tiri geri koy
            tir_bekleme_alani = [tir] + tir_bekleme_alani
            break
        else:
            istif_alani[1].append(yuk)
            linc_kullanim_sayisi += 1
            print(
                f"\t{t} tarihinde {tir.info['plaka']} plakalı tır istif alani 2'e {tir.info['ülke']}'ye gidecek kargo yukledi")

    if istif_alani_yuku(istif_alani[1]) == 750:
        print(f"\t{t} tarihinde 750 tonluk istif alanı 2 doldu")
```

if istif_alani_yuku(istif_alani[0]) == 750 and linc_kullanim_sayisi < 20: İlk istif alanının yükü 750 ton ve kullanım sayısı 20'den azsa, bu blok çalışır.

print(f"\t{t} tarihinde 750 tonluk istif alanı 1 doldu"): İlgili tarih ve bilgilerle bir mesaj ekrana yazdırılır

while istif_alani_yuku(istif_alani[1]) < 750: İkinci istif alanının yükü 750 ton olana kadar bu döngü devam eder.

if len(tir_bekleme_alani) == 0: Eğer tır bekleme alanında kalmamışsa, döngüden çıkılır.

tir = tir_bekleme_alani.pop(0): Bekleme alanındaki ilk tır çıkarılır.

yuk = tir.info["20_ton_adet"] * 20 + tir.info["30_ton_adet"] * 30: Tırın taşıdığı yük hesaplanır.

if istif_alani_yuku(istif_alani[1]) + yuk > 750: Eğer ikinci istif alanına yük eklenirse 750 tonu aşıyorsa, tır geri koyulur ve döngüden çıkılır.

else: Eğer yük 750 tonu aşmıyorsa, bu blok çalışır.

istif_alani[1].append(yuk): Yük, ikinci istif alanına eklenir.

linc_kullanim_sayisi += 1: Kullanım sayısı bir arttırılır.

print(f"\t{t} tarihinde {tir.info['plaka']} plakalı tır istif alani 2'e {tir.info['ülke']}'ye gidecek kargo yukledi"): İlgili bilgilerle birlikte bir mesaj ekrana yazdırılır.

Döngü, ikinci istif alanının yükünün 750 ton olması durumunda sona erer.

if istif_alani_yuku(istif_alani[1]) == 750: Eğer ikinci istif alanının yükü 750 ton olduysa, bu blok çalışır.

print(f"\t{t} tarihinde 750 tonluk istif alanı 2 doldu"): İlgili tarih ve bilgilerle bir mesaj ekrana yazdırılır.

```
# gemileri istif alanina bosalt
    index = 0
    while istif_alani_yuku(istif_alani[0]) > 0 and index < len(gemi_bekleme_alani) and linc_kullanim_sayisi < 20:
        if len(gemi_bekleme_alani) == 0:
            break
```

```
        gemi = gemi_bekleme_alani[index]
        hedef_ulke = gemi.info["gidecek_ülke"]
        kargolar = list(filter(lambda x: x[1] == hedef_ulke, istif_alani[0]))
        for kargo in kargolar:
            gemi.yukle(kargo[0])
            linc_kullanim_sayisi += 1
            istif_alani[0].remove(kargo)
            print(
                f"\t{t} tarihinde {kargo[0]} tonluk kargo istif alani 1'den gemi {gemi.info['gemi_adi']} yüklendi")
            if gemi.gitmeye_hazir_mi():
                print(f"\t{t} tarihinde {gemi.info['gemi_adi']} gemisi {hedef_ulke} ülkesine yola çıktı")
                gemi_bekleme_alani.pop(index)
                index -= 1
                break
        index += 1
```

index = 0: Birinci istif alanındaki gemilerin indeksini takip etmek için bir indis oluşturulur.

while istif_alani_yuku(istif_alani[0]) > 0 and index < len(gemi_bekleme_alani) and linc_kullanim_sayisi < 20: Birinci istif alanındaki yük olduğu, gemi bekleme alanında gemi olduğu ve kullanım sayısının 20'den az olduğu sürece bu döngü devam eder.

if len(gemi_bekleme_alani) == 0: Eğer gemi bekleme alanındaki gemi kalmamışsa, döngüden çıkılır.

gemi = gemi_bekleme_alani[index]: Bekleme alanındaki ilk gemi seçilir.

hedef_ulke = gemi.info["gidecek_ülke"]: Geminin gideceği ülke belirlenir.

kargolar = list(filter(lambda x: x[1] == hedef_ulke, istif_alani[0])): Hedef ülkede olan kargolar filtrelenir.

for kargo in kargolar: Her bir kargo için bu blok çalışır.

gemi.yukle(kargo[0]): Gemiye kargo yüklenir.

linc_kullanim_sayisi += 1: Kullanım sayısı bir arttırılır.

istif_alani[0].remove(kargo): Kargo birinci istif alanından kaldırılır.

print(f"\t{t} tarihinde {kargo[0]} tonluk kargo istif alani 1'den gemi {gemi.info['gemi_adi']} yüklendi"): İlgili bilgilerle bir mesaj ekrana yazdırılır.

if gemi.gitmeye_hazir_mi(): Eğer gemi yola çıkmaya hazırsa bu blok çalışır.

print(f"\t{t} tarihinde {gemi.info['gemi_adi']} gemisi {hedef_ulke} ülkesine yola çıktı"): İlgili bilgilerle bir mesaj ekrana yazdırılır.

gemi_bekleme_alani.pop(index): Gemi bekleme alanından çıkarılır.

index -= 1: İndis bir azaltılır.

break: Döngüden çıkılır.

index += 1: İndis bir arttırılır.

```
if istif_alani_yuku(istif_alani[0]) == 0:
        print(f"\t{t} tarihinde istif alanı 1 boşaldı")
```

if istif_alani_yuku(istif_alani[0]) == 0: Eğer birinci istif alanı boşalmışsa bu blok çalışır.

print(f"\t{t} tarihinde istif alanı 1 boşaldı"): İlgili tarih ve bilgilerle bir mesaj ekrana yazdırılır.

index = 0: İkinci istif alanındaki gemilerin indeksini takip etmek için bir indis oluşturulur.

```
index = 0
    while istif_alani_yuku(istif_alani[1]) > 0 and index < len(gemi_bekleme_alani) and linc_kullanim_sayisi < 20:
        if len(gemi_bekleme_alani) == 0:
            break
        gemi = gemi_bekleme_alani[index]
        hedef_ulke = gemi.info["gidecek_ülke"]
        kargolar = list(filter(lambda x: x[1] == hedef_ulke, istif_alani[1]))
        for kargo in kargolar:
            gemi.yukle(kargo[0])
            istif_alani[1].remove(kargo)
            linc_kullanim_sayisi += 1
            print(
                f"\t{t} tarihinde {gemi.info['gemi_adi']} gemisi istif alani 2'den {kargo[0]} tonluk kargo yükledi")
            if gemi.gitmeye_hazir_mi():
                print(f"\t{t} tarihinde {gemi.info['gemi_adi']} gemisi {hedef_ulke} ülkesine yola çıktı")
                gemi_bekleme_alani.pop(index)
                index -= 1
                break
        index += 1
```

while istif_alani_yuku(istif_alani[1]) > 0 and index < len(gemi_bekleme_alani) and linc_kullanim_sayisi < 20: İkinci istif alanındaki yük olduğu, gemi bekleme alanında gemi olduğu ve kullanım sayısının 20'den az olduğu sürece bu döngü devam eder.

if len(gemi_bekleme_alani) == 0: Eğer gemi bekleme alanındaki gemi kalmamışsa, döngüden çıkılır.

gemi = gemi_bekleme_alani[index]: Bekleme alanındaki ilk gemi seçilir.

hedef_ulke = gemi.info["gidecek_ülke"]: Geminin gideceği ülke belirlenir.

kargolar = list(filter(lambda x: x[1] == hedef_ulke, istif_alani[1])): Hedef ülkede olan kargolar filtrelenir.

for kargo in kargolar: Her bir kargo için bu blok çalışır.

gemi.yukle(kargo[0]): Gemiye kargo yüklenir.

istif_alani[1].remove(kargo): Kargo ikinci istif alanından kaldırılır.

linc_kullanim_sayisi += 1: Kullanım sayısı bir arttırılır.

print(f"\t{t} tarihinde {gemi.info['gemi_adi']} gemisi istif alani 2'den {kargo[0]} tonluk kargo yükledi"): İlgili bilgilerle bir mesaj ekrana yazdırılır.

if gemi.gitmeye_hazir_mi():Eğer gemi yola çıkmaya hazırsa bu blok çalışır.

print(f"\t{t} tarihinde {gemi.info['gemi_adi']} gemisi {hedef_ulke} ülkesine yola çıktı"): İlgili bilgilerle bir mesaj ekrana yazdırılır.

gemi_bekleme_alani.pop(index): Gemi bekleme alanından çıkarılır.

index -= 1: İndis bir azaltılır.

break: Döngüden çıkılır.

index += 1: İndis bir arttırılır.

```
if istif_alani_yuku(istif_alani[1]) == 0:
        print(f"\t{t} tarihinde istif alanı 2 boşaldı")


    print(f"{t} tarihi için işlemler bitti")
```

if istif_alani_yuku(istif_alani[1]) == 0: Eğer ikinci istif alanı boşalmışsa bu blok çalışır.

print(f"\t{t} tarihinde istif alanı 2 boşaldı"): İlgili tarih ve bilgilerle bir mesaj ekrana yazdırılır.

print(f"{t} tarihi için işlemler bitti"): İlgili tarih için işlemlerin bittiğini belirten bir mesaj ekrana yazdırılır.