Algoritma Analizi
Ödev 3

GRAFLAR

Elif Yağmur Duran
18011071

Contents

Programın Çalışma Mantığı

1. Adım – Dosyadan Okuma

Program verilen dosyadaki bilgileri alarak flighttable struct tipinde bir arraye aktarır. Daha sonra kullanıcıya göstermek için bu struct tipine özel hazırlanmış fonksiyon ile ekrana bastırır.

2. Adım – Şehirlerin Tespiti ve Komşuluk Matrisinin Hazırlanması

Verilen problem veri yapısını graf şeklinde kullanıp graf algoritmalarından yararlanarak çözüleceği için, ilk önce grafın nodelarını temsil edecek olan tüm için şehirlerin bilinmesine ihtiyaç duyulur. Flighttable struct arrayi baştan sona okunurken, karşılaşılan her yeni şehirle birlikte city_matrix isimli matrise yeni bir satır eklenir. Daha sonra bulunan şehirlerde kullanıcıya gösterilir.

Grafın kenarlarını temsil edecek ve bunu saklayacak bir komşuluk matrisi oluşturulur ve matrisin her gözüne bir adjacency_matrix_struct tipinden struct koyulur. Böylece matrisin her satır ve sütun değeri o uçuşun gidiş dönüş şehirlerini

tutarken, matrisin gözünde o iki şehir arasında uçuş olup olmadığı değerinin yanı sıra, eğer varsa olan uçuşun süresi ve fiyat değeri de tutulabilir. Köşegendeki structlar, komşuluk matrisi oluşturma kuralı gereği her değerine 0 değeri alır. Bunun yanı sıra arasında uçuş olmayan şehirlerin structlarına da her değer 0 atanır. Bu adımın sonunda program, kullanıcıya göstermek için burada toplanan bilgileri de ekrana bastırır.

3. Adım – Kullanıcıdan Bilgilerin Alınması

Artık kullanıcı istediği sürece tekrar çalışmaya devam edecek bir döngüye girilmiştir. Bu kısımda uçuşlar için istenilen özellikler kullanıcıdan alınır. Artık program istenilen işi gerçekleştirebilir. Komşuluk matrisi ile DFS yönteminden faydalanarak gerekli aramaları yapması ve kullanıcıya göstermesi gerekir (bu kısma sürem yetmedi).

Değişkenler

Kullanılan Structurelar

struct flighttable {
int index;
char source[25];
int source_index;
char destination[25];
int destination_index;
int hour;
int minute;
int price;
};
	Uçuşları dosyadan alıp bir struct dizisinde tutmak için kullanılan tür. source, index, hour, minute ve price dosyada verilen bilgiler için, index uçuşun okunma sırasını saklamak için, source_index ve destination_index ise şehirler belirlenirken bu şehrin adjacency matrisinde hangi indexte olduğunu saklamak içindir.

struct adjacency_matrix_struct {
int adjacency;
int hour;
int minute;
int price;
};
	Adjacency matrisinin her gözüne yerleşecek struct. adjacency ile iki şehir uçuş olup olmadığını, varsa da hour, minute ve price ile bu uçuşun bilgilerini tutar.

File Okuma İşlemi Değişkenleri

int flight_count = 0; FILE *fptr; char filestr[75]; char temp_source[25]; char temp_destination[25];	Dosyanın okunacağı ve flights isimli struct dizisine aktarılacağı sırada kullanılırlar. Uçuş sayısı sıfırdan başlatılır.

int temp_hour; int temp_minute; int temp_price;	Temp stringleri ve integerları da bazı fonksiyonların doğru şekilde çalışabilmesi için kullanılmıştır.
struct flighttable *flights = (struct flighttable *)malloc(sizeof(struct flighttable) * 25);	Dinamik tanımlanan flighttable tipinden structların dizisi. Flight_count bilinmediği için 25 boyut tanımlanır ve daha sonra realloc yapılır. Uçuş bilgilerini düzenli ve kolay ulaşılır şekilde saklamak için kullanılır.
struct adjacency_matrix_struct **adj_matrix;	Adjacency matrisini uçuş bilgileri ile beraber tutmak için her göze bir struct yerleştirilir.

Diğer Değişkenler

int i, j, k; int c1, c2;	Sayı tutma değişkenleri. C1 ve C2 ileride şehir indexini tutmak için kullanıldı.
char user_source[25]; char user_destination[25]; int max_transfer; // 0<transfer<city	Kullanıcının tercihlerin almak için değişkenler. Bu satırlar kullanıcıdan zorunlu olarak alınacak uçuş bilgiler içindir.
int hour_constr; int minute_constr; int price_constr;	Kullanıcının isteğe göre ekleyebileceği kısıtlar içindir.
char selection_forextrainfo; char selection_tocontinue; char selection_duration; char selection_price;	y/n alabilen char değişkenleri. Kullanıcının devam edip etmek istemediği ya da bazı kısıtları ekleyip eklemek istemediği bilgisini saklar.
char **city_matrix = (char **)malloc(2 * sizeof(char *)); int city_count;	Bulunan şehirleri tutacak olan city_matrix. Dinamik tanımlanmış string dizisidir. city_count şehir sayısını bilmemiz içindir.

Kullanılan Fonksiyonlar

Print fonksiyonları kullandım.

void printadjmat(int city_count, struct adjacency_matrix_struct **table) { int i, j;	Adjacency matrisini gösteren fonksiyon.

printf("\nThe adjacency matrix is:\n");
for (i = 0; i < city_count; i++) {
for (j = 0; j < city_count; j++) {
printf(" | %d %d %d %d |\t", table[i][j].adjacency, table[i][j].hour, table[i][j].minute, table[i][j].price);
}
printf("\n");
}
}	

void printflights(struct flighttable table[], int size) {
int i;
i = 0;
while (i < size) {
printf("\nThe flight to %s (%d) from %s (%d) | %d hours %d minutes | Price: %d", table[i].source, table[i].destination, table[i].hour, table[i].minute, table[i].price);
i++;
}
}
	Flighttable ı gösteren fonksiyon.

Main()

File Okuma ve Struct Arrayi Oluşturulması

fptr = fopen("sample.txt", "r");
if (fptr == NULL) {
printf("sample.txt failed to open.");
exit(0);
} else {
	Dosyanın açıldığını kontrol etme

printf("\nThe file is now opened.\n"); while (fgets(filestr, 50, fptr) != NULL) {	Eğer açıldıysa file pointeri fptr NULL gösterene kadar devam eden döngü.
sscanf(filestr, "%s %s %d %d %d", temp_source, temp_destination, &temp_hour, &temp_minute, &temp_price);	Her satırda sscanf fonksiyonu sırasıyla string string integer integer integer değerlerini alır ve flighttable da o andaki indexin struct elemanlarına

flights[flight_count].index = flight_count;
 	
strcpy(flights[flight_count].source, temp_source);
 	
strcpy(flights[flight_count].destination, temp_destination);
flights[flight_count].hour = temp_hour;
flights[flight_count].minute = temp_minute;
flights[flight_count].price = temp_price;
flight_count++;
}
}
	koyar.
	Bu index sıfırdan başlatılmıştır ve işlem sonunda da toplam uçuş sayısını tutacaktır.

flights = realloc(flights, flight_count * sizeof(struct flighttable));
fclose(fptr);
printf("\n\nThe file is now closed.\n");
printflights(flights, flight_count);
	Flights struct dizisi artık biliniyor olan büyüklüğüne göre realloc edilir ve dosya kapatılır. Toplanan bilgiler kullanıcıya gösterilir.

Şehirlerin Belirlenmesi ve İndexlenmesi

city_matrix[0] = (char *)malloc(25 * sizeof(char));
sprintf(city_matrix[0], flights[0].source);
flights[0].source_index = 0;
city_matrix[1] = (char *)malloc(25 * sizeof(char));
sprintf(city_matrix[1], flights[0].destination);
flights[0].destination_index = 1;
city_count = 2;
	City_matrix yukarıda *char **city_matrix = (char **)malloc(2 * sizeof(char *));* şeklinde initialize edilmişti. Bu şu an için iki satırı olduğunu ifade eder. 0. Satır için string girebilecek kadar yer açılır ve için flights tablosundaki ilk şehir koyulur. Bu şehir 0. Şehir olduğu için flights tablosuna bu bilgi aktarılır. Aynı işlem 0. Uçuşun destination kısmı içinde yapılır. Şehir sayısı an itibariyle 2dir.

j = 1; while (j < flight_count) {	J Flighttable da gezecek indextir. Flight_count ta tutulan değeri geçmediği sürece döngü devam eder, ilk uçuş yani 0. Uçuş elle işlendiği için j 1 den başlatılır.
i = 0;	flights[j].source string i city matrix te var mı diye kontrol eden kısım.

while ((i < city_count) && (strcmp(city_matrix[i], flights[j].source) != 0)) {
i++;
}
if (i >= city_count) {
city_count++;
city_matrix[i] = (char *)malloc(25 * sizeof(char));
*city_matrix = realloc(*city_matrix, city_count * sizeof(char *));
sprintf(city_matrix[i], flights[j].source);
flights[j].source_index = i;
}
if (strcmp(city_matrix[i], flights[j].source) == 0) {
flights[j].source_index = i;
}
	City_matrix im 0. Gözünden başlayarak, şehir sayısını geçmediği sürece i yi artırır.
	Eğer flights[j].source deki şehir ile city matrix te karşılaşırsa döngüden çıkar. Döngüden çıkma sebebine göre iki farklı işlem yapılır.
	
	Eğer i şehir sayısını geçtiyse, şehir ilk defa karşılaşılan bir şehirdir. Bu yüzden city matrix i bir satır genişletilir ve o satır için dinamik olarak sütun açılır. Daha sonra bu şehir oraya aktarılır. Ayrıca i nin son değeri flights tablosunda bu şehrin yanına yazılır. Bu sayede ilerde komşuluk matrisi oluşturulurken şehir matrisinin defalarca okunmasından kaçınılabilir.
	
	Eğer döngüden flights[j].source deki şehir city matrix te bulunduğu için çıkıldıysa, yeni bir matris satırına gerek yoktur. Sadece Ayrıca i nin son değerinin flights tablosunda şehrin yanına yazılması yeterli olur.

// destination var mı
i = 0;
while ((i < city_count) && (strcmp(city_matrix[i], flights[j].destination) != 0)) {
i++;
}
if (i >= city_count) {
city_count++;
city_matrix[i] = (char *)malloc(25 * sizeof(char));
*city_matrix = realloc(*city_matrix, city_count * sizeof(char *));
sprintf(city_matrix[i], flights[j].destination);
flights[j].destination_index = i;
}
if (strcmp(city_matrix[i], flights[j].destination) == 0) {
flights[j].destination_index = i;
}
j++;
	flights[j].destination string i city matrix te var mı diye kontrol eden kısım. Aynı mantıkla ama destination değerleri için çalışır.

} 	

printflights(flights, flight_count);
printf("\nNumber of cities in the file: %d\n", city_count);
printf("City vs index:\n");
for (i = 0; i < city_count; i++) {
printf("\t%d - %s\n", i, city_matrix[i]);
}
 	
	Kullanıcıya göstermek için bazı printler.

Komşuluk Matrisinin Oluşturulması

adj_matrix = (struct adjacency_matrix_struct **)malloc(city_count * sizeof(struct adjacency_matrix_struct *));
for (i = 0; i < city_count; i++) {
adj_matrix[i] = (struct adjacency_matrix_struct *)malloc(city_count * sizeof(struct adjacency_matrix_struct));
}
 	
	Komşuluk matrisi dinamik olarak tanımlanır. Citiy_count x city_count boyutlarında olması gerektiği bilinmektedir.

for (i = 0; i < city_count; i++) {
for (j = 0; j < city_count; j++) {
adj_matrix[i][j].adjacency = 0;
adj_matrix[i][j].hour = 0;
adj_matrix[i][j].minute = 0;
adj_matrix[i][j].price = 0;
}
}
 	
	Önce her değer sıfırlanır, sonraki adımda bağlantısı olmayan hücrelere işlem yapılmayacaktır.

for (i = 0; i < flight_count; i++) {
c1 = flights[i].source_index;
c2 = flights[i].destination_index;
// first for the city 1 city 2 cell
adj_matrix[c1][c2].adjacency = 1;
adj_matrix[c1][c2].hour = flights[i].hour;
adj_matrix[c1][c2].minute = flights[i].minute;
adj_matrix[c1][c2].price = flights[i].price;
	Flight tablosunda tekrar gezilir. Her uçuş için daha önceden hazırlanmış indexler c1 ve c2 ye aktarılır. Flight tablosunun diğer değerleri [c1][c2] ve simetriği olan [c2][c1] hücrelerine aktarılır. Ayrıca adjacency değerleri de 1 olmalıdır.

// now for the city 2 city 2 cell
adj_matrix[c2][c1].adjacency = 1;
adj_matrix[c2][c1].hour = flights[i].hour;
adj_matrix[c2][c1].minute = flights[i].minute;
adj_matrix[c2][c1].price = flights[i].price;
}	

printadjmat(city_count, adj_matrix);	Kullanıcıyı göstermek için bastırılır.

Kullanıcıdan Bilgi Alınması

selection_tocontinue = 'y'; while (selection_tocontinue == 'y') {	Kullanıcı çıkmak istemediği sürece devam edecek program döngüsü.

printf("\nPlease select your flight preferences.");
printf("\nWhere would you like to lift off from?\t");
scanf("%s", &user_source);
printf("\nWhere would you like to land?\t");
scanf("%s", &user_destination);
	Kalkış ve iniş yerinin seçimi.

printf("\nMaximum transfers you would like to allow:\t");
scanf("%d", &max_transfer);
while (max_transfer > city_count) {
printf("\nThat is not allowed. Maximum allowed transfers are %d. Please pick again.\t", city_count);
scanf("%d", &max_transfer);
}
	Aktarma sınırı seçimi. Burada şehir sayısından yüksek seçilmemesine dair uyarılar verilir.

printf("\nWould you like to set additional constraints? y/n\t");
scanf("%c", &selection_forextrainfo);
scanf("%c", &selection_forextrainfo);
while ((selection_forextrainfo != 'y') && (selection_forextrainfo != 'n')) {
	Diğer kısıtlar kullanıcıya bağlı tercihtir. y/n ların doğru kullanılması içinde uyarılar verilir.

```
        printf("\nThat is not allowed.
Please select one of the two y/n:\t");
        scanf("%c",
&selection_forextrainfo);
        scanf("%c",
&selection_forextrainfo);
    }
    if (selection_forextrainfo == 'y') {
        printf("\nAdd a duration
constraint? y/n\t");
        scanf("%c",
&selection_duration);
        scanf("%c",
&selection_duration);
        while ((selection_duration != 'y')
&& (selection_duration != 'n')) {
            printf("\nThat is not allowed.
Please select one of the two y/n:\t");
            scanf("%c",
&selection_duration);
            scanf("%c",
&selection_duration);
        }
        if (selection_duration == 'y') {
            printf("How many hours?");
            scanf("%d", &hour_constr);
            printf("How many minutes?");
            scanf("%d", &minute_constr);
        }

        printf("\nAdd a price
constraint? y/n\t");
        scanf("%c", &selection_price);
        scanf("%c", &selection_price);
        while ((selection_price != 'y')
&& (selection_price != 'n')) {
            printf("\nThat is not allowed.
Please select one of the two y/n:\t");
            scanf("%c",
&selection_price);
            scanf("%c",
&selection_price);
        }
        if (selection_price == 'y') {
            printf("Maximum price?");
            scanf("%d", &price_constr);
        }
    }
```

printf("\nProgram ended. Would you like to make new choices? y/n\t");
scanf("%c", &selection_tocontinue);
scanf("%c", &selection_tocontinue);
while ((selection_tocontinue != 'y') && (selection_tocontinue != 'n')) {
printf("\nThat is not allowed. Please select one of the two y/n:\t");
scanf("%c", &selection_tocontinue);
scanf("%c", &selection_tocontinue);
}
}
	Program bittiğinde çıkış veya başa dönme sorusu.