

În rezolvarea temei, am implementat un set de funcții și structuri de date pe care le-am folosit pentru a rezolva cerințele date.

Structuri:

CLIENT – definește un client prin nume(pointer către un șir de caractere), timpul până la stația de metrou(int) și suma pe care trebuie să o plătească(int)

GRAFCL – graful clienților de la o anumită stație. Conține numărul de străzi care există între clienți(int) și matricea de adiacență(pointer dublu la int)

STATIE – definește o stație prin nume(pointer către un șir de caractere), numărul de clienți de la acea stație(int), un tablou unidimensional cu informații despre clienți(pointer la CLIENT) și graful clienților de la acea stație(de tip GRAFCL)

GRAFM – graful stațiilor de metrou. Această structură conține numărul de rute dintre stațiile de metrou(int), numărul de stații de metrou(int), matricea de adiacență(pointer dublu la int) și un tablou unidimensional cu informații despre stații(pointer la STATIE)

Pentru început, am realizat funcția de deschidere a fișierelor de citire și scriere. Am folosit biblioteca "errno.h" pentru a realiza funcția "Eroare", care are rolul de a afișa pe ecran informații referitoare la erorile aparute în timpul executiei programului în diferite situații(deschidere nereușită a unui fișier, alocare dinamică eșuată). Funcția afișează pe ecran un text al erorii(ex: "Alocare dinamică eșuată!"), numele funcției în care a fost depistată eroarea, numărul erorii(errno) și explicația acesteia. De exemplu, pentru deschiderea nereușită a unui fișier, putem avea următorul output:

```
student@uso:~/Documents/T2$ ./curier nu exista.txt date.out
*****
[EROARE]: Deschiderea fisierului input a esuat!
**In functia DeschideFisiere
**No such file or directory(nr. eroare: 2)
*****
```

După depistarea unei erori și afișarea acesteia, programul este oprit din execuție.

În continuare, am realizat funcția "CitireGraf", care are rolul de a citi din fișierul input toate informațiile necesare și de a alocă dinamic memoria pentru structurile de date prezentate mai sus. Pentru matricele de adiacență(nxn), am alocat întâi memorie pentru un vector de n pointeri către int, după care am alocat memorie separat pentru n vectori de n elemente, evitând astfel ocuparea unui bloc foarte mare de memorie. După citirea grafului, am folosit funcția "CitireCerințe" pentru a citi operațiile ce trebuie să fie realizate și argumentele acestora, după care am apelat funcția corespunzătoare fiecărei operații, astfel:

Funcția Conexiune – se folosește de funcția "GăseșteClient", care are rolul de a căuta numele primului client și de a returna indicele stației la care este găsit și indicele lui în vectorul de clienți de la acea stație.

Apoi este folosita functia "IndiceClient", care primeste ca argument statia la care a fost gasit primul client si il cauta pe al doilea doar la statia respectiva pentru a obtine un timp de executie mai bun al programului. Avand acum indicii clientilor si statia la care se afla, putem verifica simplu daca exista o strada intre cei doi.

Functia Legatura – foloseste functia "IndiceStatie" pentru a cauta numele statiei si pentru a obtine indicele acesteia din vectorul de statii. Apoi se parcurge vectorul de statii si sunt afisate cele cu care exista o legatura.

Functia BlocheazaTunel – obtine indicii celor doua statii(folosind functiile de mai sus) si verifica daca exista o legatura intre acestea. Daca exista, distanta este setata la infinit(INF)

Functia BlocheazaStrada – obtine indicii clientilor si indicele statiei la care se afla clientii, verifica daca exista o legatura, iar daca exista, aceasta devine infinit(INF).

Functia AdaugaRuta – obtine indicii celor doua statii si seteaza distanta citita in matricea de adiacenta

Functia StergeRuta – functie similara cu "AdaugaRuta", aceasta seteaza valoarea 0 in matricea de adiacenta

Functia AdaugaStrada – se cauta clientii, se obtin indicii lor si indicele statiei, dupa care se seteaza valoarea citita in matricea de adiacenta de la acea statie

Functia StergeStrada – functie asemanatoare cu "AdaugaStrada", se seteaza valoarea 0 in matricea de adiacenta

Functia DrumMetrou – obtine indicii celor doua statii, dupa care apeleaza functia "DijkstraMetrou" pentru a gasi drumul minim dintre cele doua statii. Functia DijkstraMetrou, dupa cum o spune si numele, se foloseste de algoritmul lui Dijkstra adaptat pentru a gasi drumul minim dintre doua statii. Se alege prima statie ca sursa si se afla distantele minime pana la celelalte statii. Se alocă dinamic memorie pentru vectorul de distante, vectorul "cale" (cale[i] are valoarea 1 daca statia i este inclusa in drumul minim) si vectorul "prec" (prec[i] retine ultima statie prin care s-a trecut inainte de a ajunge la statia i). Initial distantele sunt setate la infinit si nicio statie nu face parte din drumul minim. Pentru statia de plecare, distanta este 0, iar statia precedenta are valoarea -1. Apoi se repeta pasii care urmeaza de nr_statii ori:

- se cauta statia care se afla la o distanta minima de sursa si nu face parte din drumul minim
- se adauga statia gasita la drumul minim
- se modifica distantele(daca distanta minima este diferita de infinit si noul drum este mai scurt) pana la statiile care au o legatura directa de distanta finita cu statia gasita si nu fac parte din drumul minim. De asemenea, se salveaza in vectorul prec indicele statiei precedente pentru a obtine astfel nu doar distanta minima, ci si drumul pana la statia de plecare

Dupa repetarea pasilor de mai sus, este apelata functia AfisareRutaMetrou, care primeste ca parametru vectorul prec, dar si statia de plecare si destinatia. Se pleaca de la destinatie si folosind vectorul prec, parcurgem drumul invers pana cand prec[i] = -1 (adica pana atunci cand ajungem la sursa), memorand statiile pe la care am trecut. Apoi se afiseaza drumul minim dintre cele doua statii si se elibereaza memoria alocata pentru vectorii folositi in algoritm.

Functia DrumStrada – similara cu functia “DrumMetrou”. Diferenta este ca algoritmul se aplica acum pe graful de la statia la care sunt gasiti cei doi clienti.

Functia TimpStatie – stabileste care este clientul cel mai apropiat de metrou si se foloseste un algoritm de tip Greedy pentru a calcula un drum optim. La fiecare pas, este ales cel mai apropiat client nevizitat si se adauga la suma distanta pana la acesta, pana cand toti clientii au fost vizitati. De asemenea, se adauga la suma distantele de la primul si de la ultimul client pana la metrou, iar la final este afisat rezultatul.


Functia ComandaStatie – parcurge vectorul de clienti de la statia citita, aduna sumele si afiseaza rezultatul.

Dupa realizarea tuturor operatiilor, este apelata functia “CitireMatrice” care are rolul de a citi matricea drumurilor(daca este cazul) si de a gasi drumul minim.

Se pleaca de la elementul din dreapta sus [0,0]. Intrucat exista doua strazi posibile(stanga si dreapta), inseamna ca exista doua directii posibile de deplasare prin matrice: in jos(0...n-1 pe linii) si la dreapta(0...n-1 pe coloane). Asadar, o rezolvare de complexitate $n \times n$ care conduce intotdeauna la drumul minim este parcurgerea matricei si realizarea unor sume parțiale la fiecare pas. Intai se completeaza linia si coloana elementului de start(adica linia 0 si coloana 0), dupa care se parcurge matricea incepand de la elementul [1,1]. Pentru orice element $a[i][j]$, se compara elementele de la care se poate ajunge direct in acel punct si se realizeaza operatia $a[i][j] = a[i][j] + \min(a[i-1][j], a[i][j-1])$.

Asadar, pentru matricea initiala din imaginea de mai jos se completeaza linia 0 si coloana 0:

10	12	44	5
23	40	55	33
11	60	35	87
99	2	22	42



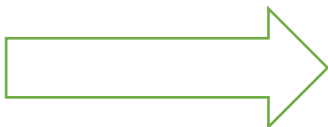
10	22	66	71
33	40	55	33
44	60	35	87
143	2	22	42

Observam faptul ca in fiecare punct evidentiata cu verde se poate ajunge mergand doar pe coloana sau doar pe linie. Astfel, sumele obtinute in elementele de pe linia 0 si coloana 0 reprezinta costul minim pana in punctele respective. In continuare se parcurge matricea, calculand la fiecare pas drumul minim pana la punctul respectiv.

10	22
33	62



10	22	66	71
33	40	55	33
44	60	35	87
143	2	22	42



10	22	66	71
33	62	117	104
44	104	139	191
143	106	128	170

Spre deosebire de o implementare cu un algoritm de tip greedy care ar fi fost mult mai buna ca timp de executie, aceasta solutie ofera intotdeauna cel mai scurt drum. Pe exemplul dat, un algoritm greedy ar fi functionat ca in imaginea de mai jos si ar fi dus la un drum mai lung decat cel minim.

Drum minim: $10+23+11+60+2+22+42 = 170$

Greedy: $10+12+40+55+33+87+42 = 279$

10	→ 12	44	5
23	↓	40	→ 55
11	60	35	↓ 87
99	2	22	↓ 42

10	12	44	5
↓ 23	40	55	33
↓ 11	→ 60	35	87
99	↓ 2	→ 22	→ 42

GREEDY vs DRUM MINIM

Dupa afisarea solutiei, sunt apelate functiile de eliberare a memoriei, respectiv de inchidere a fisierelor. Am verificat sursa folosind valgrind si nu au fost detectate erori/memory leaks.