**Colegiul Naţional “Ion C. Brătianu” Pitesti**

**Atestat professional**

**Operații cu arbori**

***Disciplina: Informatică***

***Anul 2022***

**Elev: Drăguțoiu Vlad-Ioan Profesori coordonatori: Clasa: 12 B Șuțan Dana, Benghe Adrian**

Cuprins

1. [Introducere.](#Introducere)
2. [Generalități despre limbajul C++.](#Generalitati)
3. [Cum scriem un program in C++?](#Scriere)
4. [Structura și conținutul proiectului.](#Structura)
5. [Codul sursă.](#Sursa)
6. [Bibliografie.](#Bibliografie)
7. Introducere

Lucrarea de față are ca temă efectuarea anumitor operații pe arbori, fie având muchii cu costuri, fie fără, citiți de la tastatură sau transformarea unui graf cu costuri într-un arbore parțial de cost minim.

Proiectul a fost realizat integral în limbajul C++, în Code::Blocks 20.03, prin programare direct în cod sursă (scrierea liniilor de comandă).

1. Generalități despre limbajul C++

**Limbajele de programare** sunt limbaje asemănătoare cu limbajul uman. Conțin cuvinte (destul de puține), semne de punctuație, operații matematice și au reguli de scriere. Programele care rulează pe orice calculator au fost scrise într-un limbaj de programare. Există numeroase limbaje programare, precum C, C++, Pascal, Java, Python, PHP, Javascript, etc.

Programul scris într-un limbaj de programare se numește **program sursă** și trebuie traduse într-un limbaj pe care îl înțelege procesorul, numit **cod mașină**, sau **program executabil**. Pentru anumite limbaje de programare operația de traducere se numește **compilare** (cazul lui C, C++, Pascal, etc.), pentru alte limbaje (PHP, Python, Javascript, etc.) operația de traducere se numește **interpretare**. Traducerea este realizată de un program specializat numit **compilator** sau **interpretor**.

**C++** este un [limbaj de programare](https://ro.wikipedia.org/wiki/Limbaj_de_programare) general, [compilat](https://ro.wikipedia.org/wiki/Compilator). Este un [limbaj multi-paradigmă](https://ro.wikipedia.org/wiki/Limbaj_de_programare_multi-paradigm%C4%83), cu verificarea statică a tipului variabilelor ce suportă [programare procedurală](https://ro.wikipedia.org/wiki/Programare_procedural%C4%83), [abstractizare a datelor](https://ro.wikipedia.org/w/index.php?title=Abstractizare_(informatic%C4%83)&action=edit&redlink=1), [programare orientată pe obiecte](https://ro.wikipedia.org/wiki/Programare_orientat%C4%83_pe_obiecte). În [anii 1990](https://ro.wikipedia.org/wiki/Anii_1990), C++ a devenit unul dintre cele mai populare limbaje de programare comerciale, rămânând astfel până azi[[2]](https://ro.wikipedia.org/wiki/C%2B%2B" \l "cite_note-tiobe-2).

[Bjarne Stroustrup](https://ro.wikipedia.org/wiki/Bjarne_Stroustrup) de la [Bell Labs](https://ro.wikipedia.org/wiki/Bell_Labs) a dezvoltat C++ (inițial denumit *C cu clase*) în anii [1980](https://ro.wikipedia.org/wiki/1980), ca o serie de îmbunătățiri ale limbajului [C](https://ro.wikipedia.org/wiki/Limbajul_C). Acestea au început cu adăugarea noțiunii de [clase](https://ro.wikipedia.org/w/index.php?title=Clas%C4%83_(informatic%C4%83)&action=edit&redlink=1), apoi de [funcții virtuale](https://ro.wikipedia.org/wiki/Func%C8%9Bie_virtual%C4%83), [suprascrierea operatorilor](https://ro.wikipedia.org/w/index.php?title=Suprascrierea_operatorilor&action=edit&redlink=1), [moștenire multiplă](https://ro.wikipedia.org/w/index.php?title=Mo%C8%99tenire_multipl%C4%83&action=edit&redlink=1) ([engleză](https://ro.wikipedia.org/wiki/Limba_englez%C4%83) *multiple inheritance*), [șabloane](https://ro.wikipedia.org/wiki/%C8%98ablon_(programare)) ([engleză](https://ro.wikipedia.org/wiki/Limba_englez%C4%83) *template*) și [excepții](https://ro.wikipedia.org/w/index.php?title=Excep%C8%9Bie_(programare)&action=edit&redlink=1). Limbajul de programare C++ a fost standardizat în [1998](https://ro.wikipedia.org/wiki/1998) ca și [ISO](https://ro.wikipedia.org/wiki/ISO) 14882:1998, versiunea curentă fiind din 2003, ISO 14882:2003. Următoarea versiune standard, cunoscută informal ca [C++0x](https://ro.wikipedia.org/w/index.php?title=C%2B%2B0x&action=edit&redlink=1), este în lucru.

1. Cum scriem un program C++?

Avem nevoie cel puțin de un editor de text pentru scrierea sursei și de un compilator C++. Deși fișierul sursă poate fi realizat cu orice editor de text, de cel mai multe ori folosim un IDE. Un IDE pentru C/C++ foarte utilizat este Code::Blocks.

Etapele scrierii unui program în C++ sunt:

* editarea programului C++; se obține fișierul sursă, cu extensia .cpp
* compilarea fișierului sursă; aici se verifică corectitudinea sintactică a programului (corectitudinea cuvintelor folosite, prezența semnelor de punctuație, etc.); dacă programul este corect sintactic, se va obține fișierul obiect, cu extensia .o sau .obj
* editarea de legături; se stabilesc legături între fișierul obiect curent și alte fișiere obiect, ale programatorului sau incluse în compilator; în urma acestei etape se obține programul executabil. În Windows, fișierele executabile au extensia .exe;
* programul executabil poate fi lansat în execuție (rulat).

Exemplu explicat:

// primul program C++

#include <iostream>

int main()

{

/\*

primul program C++

il scriem in Code::Blocks

\*/

std :: cout << "Hello world";

return 0;

}

Dacă vom compila și rula acest program, pe ecran va apărea:

Hello world

Să analizăm acest program. El este alcătuit din mai multe linii:

* // primul program C++  
  Această linie reprezintă un **comentariu**. Comentariile sunt texte explicative care nu influențează comportamentul programului. Ele sunt pentru programatori, pentru a înțelege mai repede semnificația programului. Acest comentariu începe de la cele două caractere slash // și se termină la sfârșitul liniei.
* #include <iostream>  
  Liniile care încep cu # se numesc directive preprocesor. Ele sunt interpretate înainte de compilarea propriu-zisă, de către un program numit *preprocesor*. În cazul nostru, directiva #include cere preprocesorului să includă în sursă o secțiune a codului C++ standard, **header-ul** iostream, care permite realizarea operațiilor de citire și afișare – la noi afișarea mesajului Hello world pe ecran.
* int main()  
  Această linie reprezintă declararea unei funcții. În esență, o functie este un grup de instructiuni care un un nume dat; în acest caz, funcția se numește main și este alcătuită din toate instrucțiunile care urmează. Vom discuta pe larg despre functii mai târziu.  
  Funcția numită main este specială în toate programele C++; această funcție este apelată când se lansează în execuție programul și trebuie să apară în orice program C++.
* {  
  Parantezele acolade de la linia 4 și 10 delimitează instrucțiunile care fac parte din funcția main
* /\*
* primul program C++
* il scriem in Code::Blocks
* \*/

Și acesta este un comentariu. Textele cuprinse între /\* și \*/ nu influențează comportamentul programului. Ele pot să ocupe mai multe linii, sau pot să apară în interiorul unei linii.

* std :: cout << “Hello world”;  
  Aceasta este o instrucțiune C++. O instrucțiune este o construcție (expresie, comandă) care face ceva. Instrucțiunile sunt “miezul” programelor, ele stabilind comportamentul acestora. Instrucțiunile dintr-un program se execută în ordine, una după alta.  
  Această instrucțiune produce afișarea pe ecran a atextului Hello world. Ea este alcătuită din trei părți. std::cout semnifică dispozitivul standard de ieșire (**st**andar**d** **c**haracter **out**put) – de cele mai multe ori ecranul calculatorului. A doua parte este operatorul de inserție <<, care indică faptul că ceea ce urmează este inserat în std::cout (trimis spre ecran). A treia parte este textul, "Hello world", cuprins între ghilimele, care va fi inserat în std::cout.  
  Să observăm prezența caracterului ; la sfârșitul instrucțiunii. Orice instructiune C++ trebuie să se termine cu ;, la fel cum orice propoziție în limba română se termină cu caracterul . (punct).

Una dintre cele mai frecvente erori de sintaxă este să uităm să scriem ; la finalul unei instrucțiuni.

* return 0;  
  Această instrucțiune marchează finalul execuției funcției main și a programului nostru. Valoarea 0 semnifica faptul că programul s-a încheiat cu succes!  
  Dacă în programul nostru ar fi fost și alte instrucțiuni după instrucțiunea return 0;, acestea nu s-ar mai fi executat.
* }  
  Acolada închisă } reprezintă finalul funcției main.

Să reținem că nu toate liniile programului produc efecte la executarea programului. Unele linii (comentariile) sunt scrise numai pentru a ușura înțelegerea programului de către cel care îl citește/scrie. Mai mult, nu este obligatoriu ca fiecare instrucțiune să fie scrisă pe o singură linie. Următoarele trei exemple de funcție main au acelați efect:

int main()

{

cout << "Salut";

return 0;

}

int main() { cout << "Salut"; return 0; }

int main()

{

cout <<

"Salut";

return 0;

}

Să modificăm primul program, astfel încât să afișăm pe ecran două propoziții:

// al doilea program C++

#include <iostream>

int main()

{

std :: cout << "Hello world!";

std :: cout << "Primul program C++!";

return 0;

}

La rulare, programul va afișa:

Hello world!Primul program C++!

Să observăm că cele doua propoziții sunt scrie pe același rând al ecranului, chiar dacă au fost scrise cu două instrucțiuni distincte. Dacă dorim ca cele doua propoziții să fie afișate pe linii diferite ale ecranului, folosim std::endl.

// al doilea program C++

#include <iostream>

int main()

{

std :: cout << "Hello world!" << std::endl;

std :: cout << "Primul program C++!";

return 0;

}

Acum, programul va afișa:

Hello world!

Primul program C++!

Instrucțiunea using namespace std;

În C++, identificatorii sunt grupați în spații de nume – *namespaces*. Există un spațiu de nume predefinit, cu numele std, din care fac parte toți identificatorii din biblioteca C++ standard.

cout, ca și endl, este un identificator din spațiul de nume std și pentru a-l putea folosi trebuie folosită expresia std::cout. Pentru a ne referi mai simplu la identificatorii din spațiul de nume std se poate folosi instrucțiunea:

using namespace std;

Astfel, programul anterior poate fi rescris:

// al doilea program C++

#include <iostream>

using namespace std;

int main()

{

cout << "Hello world!" << endl;

cout << "Primul program C++!";

return 0;

}

1. Structura și conținutul proiectului

Aplicația a fost realizată integral in Code::Blocks 20.03, prin scrierea directă a liniilor de cod. Structura este relativ simplă, proiectul fiind constituit din 3 părți de baza, toate îmbinate in aceeași sursă. Pentru o înțelegere mai bună a programului, majoritatea elementelor sunt explicate direct în cod, prin comentarii, fără a afecta buna funcționare a acestuia. Aceste 3 părți sunt, în ordine:

1. Prima parte a programului conține directivele preprocesor, instrucțiunea “using namespace std;”, declararea a două constante, precum și a tuturor structurilor de date și variabilelor necesare programului.
2. Cea de-a doua parte conține toate funcțiile programului, fie de citire, fie de rezolvare a cerințelor, fie de precalculare sau funcții auxiliare.
3. Cea de-a treia parte este funcția “main”, “inima” programului, funcție ce este apelată când se lansează in execuție programul. Aceasta conține cele două meniuri de citire a datelor de la tastatură și de afișare a rezultatelor. Se încheie cu instrucțiunea “return 0” ce marchează sfârșitul programului și ne spune dacă s-a încheiat cu succes.

5. Codul sursă

#include <iostream>

#include <algorithm>

#include <vector>

#include <cmath>

#define nl '\n'

**using** **namespace** std**;**

const int NMAX **=** 200005**,** MMAX **=** 200005**;**

/// constante ce reprezinta dimensiunea maxima admisa a grafului

struct edge

**{**

int a**,**b**,**cost**;**

**};**

vector**<**edge**>** edges**;**

/// vector de structuri pentru a putea stoca muchiile cu costuri ale grafului

int root **=** 1**;**

/// radacina arborelui

int timp**;**

/// tinem minte la al catelea pas suntem in parcurgerea in adancime

int N**;**

/// numarul de noduri al grafului

int M**;**

/// numarul de muchii al grafului

vector**<** pair**<**int**,**int**>** **>** G**[**NMAX**],**newG**[**NMAX**];**

/// vector de perechi ( pentru a putea stoca si costul muchiilor,

/// daca este necesar ) alocat dinamic, din STL ce reprezinta ar-

/// borele sau graful in cazul in care citim graf

bool vis**[**NMAX**];**

/// tinem cont de nodurile vizitate

int par**[**NMAX**],**siz**[**NMAX**],**dep**[**NMAX**],**dist**[**NMAX**],**sub**[**NMAX**],**in**[**NMAX**],**out**[**NMAX**];**

/// parintii nodurilor, dimensiunea comp conexe din care fac parte, adancimea lor,

/// distanta pana la radacina, dimensiunea subarborelui, timpul de intrare in nod

///si timpul de iesire din nod( din parcurgerea in adancime a arborelui )

int par2**[**NMAX**][**20**];**

/// al 2^K ( K = 0,1,...,19 ) lea parinte al fiecarui nod

long long paths**[**NMAX**];**

/// numarul de muchii ce contin muchia i - par[i]

void citire1**()**

**{**

cout**<<**"Numarul de noduri: "**;**

cin**>>**N**;**

cout**<<**"\nNumarul de muchii: "**;**

cin**>>**M**;**

cout**<<**"\nIntroduceti cate o muchie, pe rand, sub forma nod1, nod2, cost: "**;**

**for(**int i**=**0**;**i**<**M**;**i**++)**

**{**

int a**,**b**,**cost**;**

cin**>>**a**>>**b**>>**cost**;**

G**[**a**].**push\_back**(**make\_pair**(**b**,**cost**));**

G**[**b**].**push\_back**(**make\_pair**(**a**,**cost**));**

**}**

system**(**"cls"**);**

**}**

void citire2**()**

**{**

cout**<<**"Numarul de noduri: "**;**

cin**>>**N**;**

cout**<<**"\nNumarul de muchii: "**;**

cin**>>**M**;**

cout**<<**"\nIntroduceti cate o muchie, pe rand, sub forma nod1, nod2: "**;**

**for(**int i**=**0**;**i**<**M**;**i**++)**

**{**

int a**,**b**;**

cin**>>**a**>>**b**;**

G**[**a**].**push\_back**(**make\_pair**(**b**,**1**));**

G**[**b**].**push\_back**(**make\_pair**(**a**,**1**));**

**}**

system**(**"cls"**);**

**}**

void citire3**()**

**{**

cout**<<**"Numarul de noduri: "**;**

cin**>>**N**;**

cout**<<**"\nPentru vector de tati tastati 1, pentru lista de adiacenta tastati 2: "**;**

int choice**;**

cin**>>**choice**;**

**if(**choice**==**1**)**

**{**

**for(**int i**=**1**;**i**<=**N**;**i**++)**

**{**

cout**<<**"\nIntroduceti tatal nodului "**<<**i**<<**", respectiv costul muchiei: "**;**

int cost**;**

cin**>>**par**[**i**]>>**cost**;**

G**[**par**[**i**]].**push\_back**({**i**,**cost**});**

G**[**i**].**push\_back**({**par**[**i**],**cost**});**

**}**

**}**

**else**

**{**

cout**<<**"\nIntroduceti cate o muchie, pe rand, sub forma nod1, nod2, cost: "**;**

**for(**int i**=**0**;**i**<**N**-**1**;**i**++)**

**{**

int a**,**b**,**cost**;**

cin**>>**a**>>**b**>>**cost**;**

G**[**a**].**push\_back**(**make\_pair**(**b**,**cost**));**

G**[**b**].**push\_back**(**make\_pair**(**a**,**cost**));**

**}**

**}**

system**(**"cls"**);**

**}**

void citire4**()**

**{**

cout**<<**"Numarul de noduri: "**;**

cin**>>**N**;**

cout**<<**"\nPentru vector de tati tastati 1, pentru lista de adiacenta tastati 2: "**;**

int choice**;**

cin**>>**choice**;**

**if(**choice**==**1**)**

**{**

**for(**int i**=**1**;**i**<=**N**;**i**++)**

**{**

cout**<<**"\nIntroduceti tatal nodului "**<<**i**<<**": "**;**

cin**>>**par**[**i**];**

G**[**par**[**i**]].**push\_back**({**i**,**1**});**

G**[**i**].**push\_back**({**par**[**i**],**1**});**

**}**

**}**

**else**

**{**

cout**<<**"\nIntroduceti cate o muchie, pe rand, sub forma nod1, nod2: "**;**

**for(**int i**=**0**;**i**<**N**-**1**;**i**++)**

**{**

int a**,**b**;**

cin**>>**a**>>**b**;**

G**[**a**].**push\_back**(**make\_pair**(**b**,**1**));**

G**[**b**].**push\_back**(**make\_pair**(**a**,**1**));**

**}**

**}**

system**(**"cls"**);**

**}**

void extragereMuchii**(**int node**)**

**{**

/// functie ce parcurge graful si extrage muchiile

vis**[**node**]=**1**;**

**for(**auto son **:** G**[**node**])**

**{**

**if(**vis**[**son**.**first**])**

**continue;**

edge e**={**node**,**son**.**first**,**son**.**second**};**

edges**.**push\_back**(**e**);**

extragereMuchii**(**son**.**first**);**

**}**

**}**

bool cmp**(**edge x**,** edge y**)**

**{**

/// functia cmp compara costurile a doua muchii si returneaza 0 daca ordinea

/// lor din vector NU este buna, respectiv 1 in caz contrar

**return** x**.**cost**<**y**.**cost**;**

**}**

int parent**(**int node**)**

**{**

/// functie care "compreseaza" lantul de parinti si returneaza parintele comp. conexe

/// Complexitate timp - O(log2(N))

**if(**par**[**node**]==**node**)return** node**;**

**return** par**[**node**]=**parent**(**par**[**node**]);**

**}**

void DSU**(**int a**,**int b**,**int cost**,**int **&**CostTotal**)**

**{**

/// functia DSU ( Disjoint Set Union ) uneste doua noduri si componentele conexe

/// disjuncte, aferente acestora, in complexitate timp ~ O(1)

int old\_a**=**a**,** old\_b**=**b**;**

a**=**parent**(**a**);**

b**=**parent**(**b**);**

**if(**a**!=**b**)** /// testam daca cele doua noduri apartin aceleiasi componente conexe

**{**

/// pastram in variabila "a" nodul cu comp. conexa cea mai mare

/// aceasta optimizare reduce complexitatea timpului de constructie a arborelui

/// de la O(N) la O(log2(N)), iar impreuna cu functia "parent", reduce complexitatea

/// timpului de "unire" a doua noduri pana la O(log\_star(N)) ~ O(1)

**if(**siz**[**b**]>**siz**[**a**])**

swap**(**a**,**b**);**

par**[**b**]=**a**;** /// nodul din "a" devine parintele nodului din "b"

siz**[**a**]+=**siz**[**b**];** /// actualizam marimea comp conexe a nodului din "a";

CostTotal**+=**cost**;** /// actualizam costul total

newG**[**old\_a**].**push\_back**({**old\_b**,**cost**});** /// construim APM-ul propriu-zis

newG**[**old\_b**].**push\_back**({**old\_a**,**cost**});** /// construim APM-ul propriu-zis

**}**

**}**

int APM**()**

**{**

///Algoritmul lui Kruskal

///Complexitate timp - O(N\*log2(N))

/// initializam vectorul par - fiecare nod este propriul sau parinte

/// initializam vectrorul siz - fiecare nod are dimensiunea comp. conexe egala cu 1

**for(**int i**=**1**;**i**<=**N**;**i**++)**

**{**

par**[**i**]=**i**;**

siz**[**i**]=**1**;**

**}**

/// extragem muchiile din graf

extragereMuchii**(**1**);**

/// sortam muchiile crescator dupa cost

sort**(**edges**.**begin**(),**edges**.**end**(),**cmp**);**

int CostTotal**=**0**;**

/// parcurgem muchiile si le adaugam pe cele care unesc doua comp. conexe distincte

**for(**auto e **:** edges**)**

**{**

DSU**(**e**.**a**,**e**.**b**,**e**.**cost**,**CostTotal**);**

**}**

**return** CostTotal**;**

**}**

void afisareArbore**(**int tip**,** int node**=**1**,**int prev**=**0**)**

**{**

/// functie pentru afisarea arborelui partial de cost minim

**for(**auto son **:** G**[**node**])**

**{**

**if(**son**.**first**==**prev**)**

**continue;**

**if(**tip**==**1**)**cout**<<**node**<<**' '**<<**son**.**first**<<**' '**<<**son**.**second**<<**nl**;**

**else** cout**<<**node**<<**' '**<<**son**.**first**<<**nl**;**

afisareArbore**(**tip**,**son**.**first**,**node**);**

**}**

**}**

void refresh**()**

**{**

/// functie pentru "curatarea" vectorilor

edges**.**clear**();**

**for(**int i**=**1**;**i**<=**N**;**i**++)**

**{**

G**[**i**].**clear**();**

newG**[**i**].**clear**();**

vis**[**i**]=**0**;**

par**[**i**]=**0**;**

siz**[**i**]=**0**;**

**}**

**}**

void dfs**(**int node**,**int prev**,**int cost**)**

**{**

dep**[**node**]=**dep**[**prev**]+**1**;** /// adancimea nodului

par2**[**node**][**0**]=**prev**;** /// primul parinte al nodului

dist**[**node**]=**dist**[**prev**]+**cost**;** /// distanta pana la radacina a nodului

in**[**node**]=++**timp**;** ///timpul la care am intrat in nod

/// calculam cel de-al 2^k-lea parinte al nodului fiind cel de-al 2^(k-1)-lea parinte

/// al cel de-al 2^(k-1)-lea parinte al nodului;

**for(**int k**=**1**;(**1**<<**k**)<=**dep**[**node**];**k**++)**

**{**

par2**[**node**][**k**]=**par2**[**par2**[**node**][**k**-**1**]][**k**-**1**];**

**}**

**for(**auto son **:** G**[**node**])**

**{**

**if(**son**.**first**==**prev**)**

**continue;**

dfs**(**son**.**first**,**node**,**son**.**second**);**

/// adaugam numarul de noduri din subarborele fiecarui fiu

sub**[**node**]+=**sub**[**son**.**first**];**

/// inmultim numarul de noduri din subarbore cu restul nodurilor

paths**[**son**.**first**]=**1ll**\***sub**[**son**.**first**]\*(**N**-**sub**[**son**.**first**]);**

**}**

out**[**node**]=++**timp**;** /// timpul la care am iesit din nod

sub**[**node**]++;** /// adaugam si nodul insusi la dimensiunea subarborelui sau

**}**

int LCA**(**int a**,**int b**)**

**{**

///Lowest Common Ancestor - cel mai adanc stramos comun a doua noduri

///Complexitate timp - O(log2(N)) per interogare

**if(**dep**[**a**]>**dep**[**b**])**swap**(**a**,**b**);** /// mereu b este nodul mai adanc

int diff**=**dep**[**b**]-**dep**[**a**];** /// diferenta initiala in adancime

/// parcurgem bitii de 1 din reprezentarea binara a lui diff si urcam cu nodul b

/// pana ajungem la adancimea nodului a

**for(**int k**=**0**;(**1**<<**k**)<=**diff**;**k**++)**

**{**

**if(**diff**&(**1**<<**k**))**b**=**par2**[**b**][**k**];**

**}**

**if(**a**==**b**)return** a**;** /// cazul in care b se afla in subarborele lui a

/// urcam "in sincron" cu a si b la al 2^k-lea stramosal fiecaruia, cat timp a diferit de b

**for(**int k**=**log2**(**dep**[**a**]);**k**>=**0**;**k**--)**

**{**

**if(**par2**[**a**][**k**]!=**par2**[**b**][**k**])**

**{**

a**=**par2**[**a**][**k**];**

b**=**par2**[**b**][**k**];**

**}**

**}**

/// acum parintele lui a este si parintele lui b

a**=**par2**[**a**][**0**];**

**return** a**;**

**}**

int distAB**(**int a**,**int b**)**

**{**

/// Complexitate timp ~ O(1)

int stramos**=**LCA**(**a**,**b**);**

/// principiul includerii si excluderii: adaugam distantele 1->a, 1->b si scadem

/// de doua ori distanta 1->stramos(a,b)

int distanta**=**dist**[**a**]+**dist**[**b**]-**2**\***dist**[**stramos**];**

**return** distanta**;**

**}**

int centroid**(**int node**,**int prev**)**

**{**

**for(**auto son **:** G**[**node**])**

**{**

**if(**son**.**first**==**prev**)**

**continue;**

**if(**sub**[**son**.**first**]\***2**>**N**)**

**{**

**return** centroid**(**son**.**first**,**node**);**

**}**

**}**

**return** node**;**

**}**

void maxDist**(**int node**,**int prev**,**int len**,** int **&**maxnode**,**int **&**maxlen**)**

**{**

**for(**auto son **:** G**[**node**])**

**{**

**if(**son**.**first**==**prev**)**

**continue;**

**if(**len**+**son**.**second**>**maxlen**)**

**{**

maxnode**=**son**.**first**;**

maxlen**=**len**+**son**.**second**;**

**}**

maxDist**(**son**.**first**,**node**,**len**+**son**.**second**,**maxnode**,**maxlen**);**

**}**

**}**

int treeDiameter**()**

**{**

int maxnode**=**1**,**maxlen**=**0**;**

/// parcurgem in adancime arborele plecand din nodul 1 si gasim cel mai departat nod

/// fata de nodul 1, dupa care parcurgem in adancime arborele plecand din nodul gasit

/// si aflam cel mai departat nod de acesta din urma

/// Complexitate timp - O(2\*N) ~ O(N)

maxDist**(**1**,**0**,**0**,**maxnode**,**maxlen**);**

maxlen**=**0**;**

maxDist**(**maxnode**,**0**,**0**,**maxnode**,**maxlen**);**

**return** maxlen**;**

**}**

int main**()**

**{**

int choice**,**a**,**b**,**aux\_choice**;**

**while(**1**)**

**{**

refresh**();**

cout**<<**"MENU - Principal: "**;**

cout**<<**"\n1. Graf conex cu costuri"

**<<**"\n2. Graf conex fara costuri"

**<<**"\n3. Arbore conex cu costuri"

**<<**"\n4. Arbore conex fara costuri"

**<<**"\n0. Exit"

**<<**"\n\nIntroduceti varianta: "**;**

cin**>>**choice**;**

system**(**"cls"**);**

**switch(**choice**)**

**{**

**case** 1**:**

citire1**();**

cout**<<**"\nGraful va fi transformat intr-un arbore partial de cost minim: "**;**

cout**<<**APM**()<<**nl**;**

swap**(**G**,**newG**);**

cout**<<**"\nMuchiile arborelui sunt: "**;**

afisareArbore**(**1**);**

cout**<<**"\nDaca doriti sa continuati, introduceti 1, altfel 0: "**;**

cin**>>**aux\_choice**;**

system**(**"cls"**);**

**if(**aux\_choice**==**0**)**

**continue;**

**break;**

**case** 2**:**

citire2**();**

APM**();**

swap**(**G**,**newG**);**

cout**<<**"\nGraful va fi transformat intr-un arbore cu muchiile: "**;**

afisareArbore**(**0**);**

cout**<<**"\nDaca doriti sa continuati, introduceti 1, altfel 0: "**;**

cin**>>**aux\_choice**;**

system**(**"cls"**);**

**if(**aux\_choice**==**0**)**

**continue;**

**break;**

**case** 3**:**

citire3**();**

**break;**

**case** 4**:**

citire4**();**

**break;**

**case** 0**:**

exit**(**0**);**

**break;**

**default:**

system**(**"cls"**);**

**continue;**

**}**

dfs**(**1**,**0**,**0**);**

bool noexit**=**1**;**

**while(**noexit**)**

**{**

cout**<<**"MENU - Operatii cu arbori: "

**<<**"\n\n!!! Vom considera nodul 1 drept radacina arborelui !!!\n"

**<<**"\n1. Distanta dintre doua noduri"

**<<**"\n2. Cel mai adanc stramos comun a doua noduri"

**<<**"\n3. Diametrul arborelui"

**<<**"\n4. Centroidul arborelui"

**<<**"\n5. Numarul de lanturi ce contin o muchie data"

**<<**"\n6. Se afla nodul a in subarborele nodului b? ( fara LCA )"

**<<**"\n0. Exit"

**<<**"\n\nIntroduceti varianta: "**;**

cin**>>**choice**;**

**switch(**choice**)**

**{**

**case** 1**:**

cout**<<**"\nIntroduceti cele doua noduri: "**;**

cin**>>**a**>>**b**;**

cout**<<**"\nDistanta dintre "**<<**a**<<**" si "**<<**b**<<**" este: "**<<**distAB**(**a**,**b**);**

**break;**

**case** 2**:**

cout**<<**"\nIntroduceti cele doua noduri: "**;**

cin**>>**a**>>**b**;**

cout**<<**"\nCel mai adanc stramos comun al lui "**<<**a**<<**" si "**<<**b**<<**" este: "**<<**LCA**(**a**,**b**);**

**break;**

**case** 3**:**

cout**<<**"\nDiametrul arborelui este: "**<<**treeDiameter**();**

**break;**

**case** 4**:**

cout**<<**"\nCentroidul arborelui este: "**<<**centroid**(**1**,**0**);**

**break;**

**case** 5**:**

cout**<<**"\nIntroduceti capetele muchiei: "**;**

cin**>>**a**>>**b**;**

**if(**dep**[**a**]<**dep**[**b**])**swap**(**a**,**b**);**

cout**<<**"\nNumarul de lanturi ce contin muchia "**<<**a**<<**"-"**<<**b**<<**" este: "**<<**paths**[**a**];**

**break;**

**case** 6**:**

cout**<<**"\nIntroduceti nodul a, respectiv nodul b: "**;**

cin**>>**a**>>**b**;**

**if(**in**[**a**]<=**in**[**b**]&&**out**[**b**]<=**out**[**a**])**

**{**

cout**<<**"\nNodul "**<<**b**<<**" se afla in subarborele nodului "**<<**a**;**

**}**

**else** cout**<<**"\nNodul "**<<**b**<<**" NU se afla in subarborele nodului "**<<**a**;**

**break;**

**case** 0**:**

noexit**=**0**;**

**break;**

**default:**

system**(**"cls"**);**

**continue;**

**}**

cout**<<**"\n\nContinuati cu arborele acesta? 1 pentru DA, 0 pentru NU: "**;**

int aux\_choice**;**

cin**>>**aux\_choice**;**

**if(**aux\_choice**==**0**)**noexit**=**0**;**

system**(**"cls"**);**

**}**

**}**

**return** 0**;**

**}**

6. Bibliografie

1. <https://en.wikipedia.org/wiki/Code::Blocks>

2. <https://www.pbinfo.ro/articole/59/introducere-in-cpp>

3. <https://cp-algorithms.com/>

4. <https://cses.fi/problemset/>