APLICAțIe GRAFURI

# TEMA SI MOTIVATIA ALEGERII

**FORMULAREA TEMEI**

Realizarea unei aplicații de tip meniu folosind limbajul de programare C++, cu scopul implementării a două tipuri de grafuri, cele neorientate și cele orientate.

**MOTIVAREA ALEGERII TEMEI**

Am ales să abordez această temă deoarece o consider mai potrivită pentru acest moment al pregătirii mele de student, din cel puțin trei motive:

1. Îmi permite să aprofundez în continuare limbajul *C++*;
2. Îmi îmbunătățesc cunoștintele despre grafuri și modul lor de implementare;
3. Îmi îmbunătățesc gândirea algoritmică;

Evident că existǎ și dezavantaje, cum ar fi complexitatea codului, dar acest dezavantaj va fi compensat cu avantajele enumerate mai sus.

# PREZENTAREA PROIECTULUI

**CE SUNT GRAFURILE?**

În matematică și mai specific în teoria grafurilor, un graf este o

structură care corespunde unui grup de obiecte, în care unele

perechi de obiecte sunt într-un anumit sens "legate" reciproc.

Obiectele corespund unor abstracții matematice numite într-un

graf noduri/vârfuri (numite și puncte) și fiecare legătură dintre

perechile de obiecte asociate se numește muchie (numită și arc

sau linie, prin care este și reprezentată). De obicei, un graf este

reprezentat în formă schematică ca un set/grup de puncte

pentru noduri, iar aceste sunt unite două câte două de linii sau

curbe pentru muchii.

Muchiile pot fi orientate/directe sau neorientate/nedirecte.

**SCHEMA UML**

|  |
| --- |
| **CLASA GRAF** |
| ***Public:***  **MOȘTENIRE PUBLICĂ**   * MatriceAdiacenta : \*\*Int * NrNoduri : Int |
| * GetNrNoduri() : virtual void * SetNrNoduri(int) : virtual Int * GetMatriceAdiacenta() : \*\*int * returnGradNod(int) : Int * citireMatriceFisier(ifstream&) : virtual void * CitireMatriceAdiacenta() : virtual void * AfisareMatriceAdiacenta() : virtual void * AfisareMatriceAdiacenta() : virtual void * ParcurgereAdancime(int) : virtual void * ParcurgereLatime(int) : virtual void * esteLant() : void |

|  |
| --- |
| **CLASA GRAF ORIENTAT** |
| ***Public:***   * returnGradNodOrientat(int) : int * AfisareMuchiiGraf() : void * CitireMatriceAdiacenta() : void |

Pentru reprezentarea grafurilor am folosit matricea de adiacență deoarece prezintă următoarele avantaje:

1. Simplu de implementat;
2. Este rapidă în căutarea și verificarea prezenței sau absenței unei anumite muchii între oricare două noduri.
3. Este ușoară adăugarea unei noi muchii;

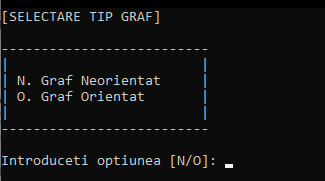
Dezavantaje:

1. Ocupă multă memorie;
2. Este lentă pentru iterarea tuturor muchiilor;

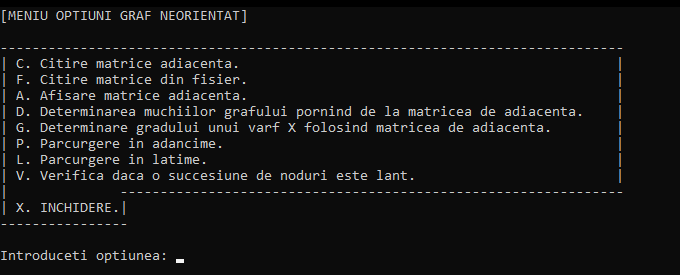
Dezavantajele apar cu cât numărul de noduri este mai mare, ceea ce nu e o problemă pentru această aplicație care are rol doar de a prezenta anumite operații ce se pot face cu grafurile.

**CUM FUNCȚIONEAZĂ APLICAȚIA?**

La rularea programului, apare un meniu cu opțiunea de a selecta între tipul grafului pe care dorim să l testăm, fie neorientat sau orientat.



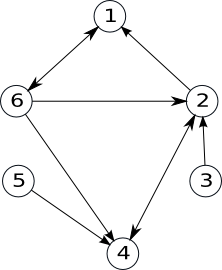
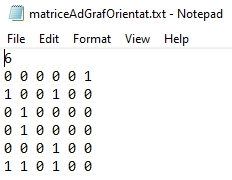
În funcție de optiunea selectată, ni se va afișa un nou meniu pentru operațiile pe care le putem efectua.



**EXEMPLU FUNCȚIONARE**

Selectăm opțiunea „O” pentru grafuri orientate iar apoi alegem să citim matricea de adiacență din fișier.

Fișierul este de forma:



* 6 reprezintă numărul de noduri, iar matricea reprezintă graful de mai sus;

