Sortare Topologica

June 3, 2018

Student: Brinzan Florinel-Razvan

Calculatoare si tehnologia informatiei
(limba romana)

C.R 1.1 A

Anul 1

Introducere

In matematica, si mai precis in teoria grafurilor, un ***graf orientat*** (sau digraf) este un graf ale carui muchii au asociat un sens.
Grafurile aciclice orientate sunt grafuri orientate fara cicluri orientate.

Pentru un nod, numarul de capete adiacente unui nod este numit gradul interior al nodului si numarul de cozi adiacente unui nod este gradul exterior.

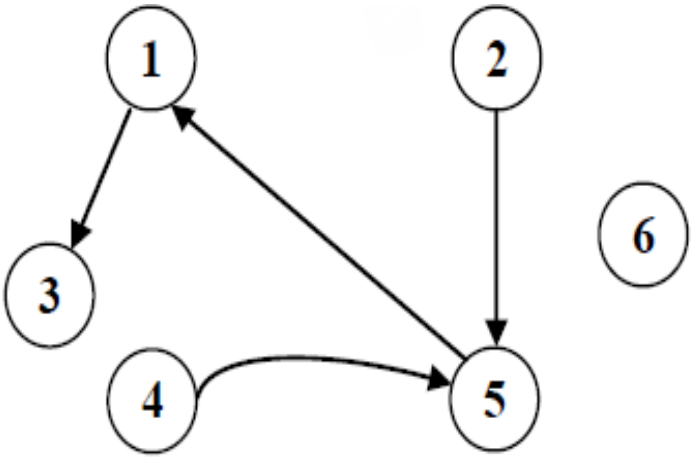
Sortarea Topologica

Dandu-se un graf orientat aciclic, sortarea topologica realizeaza o aranjare liniara a nodurilor in functie de muchiile dintre ele. Orientarea muchiilor corespunde unei relatii de ordine de la nodul sursa catre cel destinatie. Astfel, daca (u,v) este una dintre muchiile grafului, u trebuie sa apara inaintea lui v in insiruire. Daca graful ar fi ciclic, nu ar putea exista o astfel de insiruire (nu se poate stabili o ordine intre nodurile care alcatuiesc un ciclu).

Sortarea topologica poate fi vazuta si ca plasarea nodurilor de-a lungul unei linii orizontale astfel incat toate muchiile sa fie directionate de la stanga la dreapta.

Enuntul problemei

Alexandru, care este un copil mic, a primit cadou de ziua lui un joc Lego. Cu toate acestea, el nu a reuit sa construiasca jucaria Lego, deoarece exista multe piese. Alexandru a observat ca fiecare piesa este numerotata cu un numar de la 0 la 1000. Instruciunile jocului precizeaza piesele care trebuie asamblate inainte de fiecare piesa. Ajutati-l pe Alexandru sa-si construiasca jucaria Lego dezvoltand o aplicaie care determina ordinea corecta in care piesa Lego trebuie asamblata.



Pseudocod

In primul rand, vom genera un graf aciclic orientat pe care il vom memora intr-o matrice de adiacenta urmand ca pe aceasta matrice sa aplimacam 2 sortari topologice (prima bazata pe gradul intern al nodurilor si cea de-a doua pe DFS).

```
randGraph(int adjMatrix[MAX][MAX])
1.      pentru i de la 0 la nr_nod - 1 executa
2.      pentru j de la 0 la nr_nod - 1 executa
3.          daca i = j
4.              continua
5.          daca adjMatrix[j][i] = 0
6.              adjMatrix[i][j] <- rand() %2
7.          daca verificareCiclu(adjMatrix) = 0
8.              adjMatrix[i][j] <- 0
```

```
topSort(int adjMatrix[MAX][MAX])
1.      pentru i de la 0 la nr_nod - 1 executa
2.      flag[i] <- 0
3.  cat timp count != 0
4.      pentru i de la 0 la nr_nod - 1 executa
5.          gradintern <- 0
6.          daca flag2[i] = 0
7.              pentru j de la 0 la nr_nod - 1 executa
8.                              gradintern <- gradintern + adjMatrix[i][j]
9.                              daca gradintern = 0
10.                 flag2[i] <- 1
11.                             afisare i
12.                 count --
13.                 pentru j de la 0 la nr_nod - 1 executa
14.                     adjMatrix[i][j] <- 0
```

```
dfs(int k, int pozitie)
1.  pentru i de la 0 la nr_nod - 1 executa
2.      daca adjMatrix[k][i] = 1 && visitedB[i] = 0
3.          dfs(i, pozitie)
4.  pentru j de la 0 la nr_nod - 1 executa
5.      daca adjMatrix[k][i] = 1 && visitedB[i] = 0
6.          nr++
7.  daca nr = 0
8.      daca Vector[pozitie] = -1
9.          Vector[pozitie] <- k
10.     altfel
11.         cat timp Vector[pozitie] != -1
12.             pozitie++
13.         Vector[pozitie] <- k
14.     visitedB[k] <- 1
```

```
TopSortDFS(int adjMatrix[MAX][MAX])
1.  pentru i de la 0 la nr_nod - 1 executa
2.       Vector[i] <- -1
3.  pozitie <- 0
4.  cat timp 1 > 0
5.      sw <- 0
6.      pentru i de la 0 la nr_nod - 1 executa
7.          gradintern <- 0
8.          daca visitedB[i] = 0
9.              pentru j de la 0 la nr_nod - 1 executa
10.                 gradintern <- gradintern + adjMatrix[j][i]
11.             daca gradintern = 0
12.                 dfs(i, pozitie)
13.                 sw <- 1
14.     daca sw = 0
15.         oprire
16. pentru j de la nr_nod - 1 la 0 executa
17.     scrie Vector[i]
```

Proiectarea Aplicatiei

Libraria contine un header ***functii.h*** care include toate functiile utilizate la rezolvarea problemei:
—void dfs(int iteratordfs, int pozitie)
—void TopSortDFS(int adjMatrix[MAX][MAX])
—void topSort(int adjMatrix[MAX][MAX])
—int verificareCiclu(int adjMatrix[MAX][MAX])
—void randGraph(int adjMatrix[MAX][MAX])
—void citirefisier(int adjMatrix[MAX][MAX])

Fisierul ***TopSort1.c*** contine sortarea topologica bazata pe gradul intern al nodurilor (void topSort(int adjMatrix[MAX][MAX])).
– Algoritmul initializeaza vectorul flag2 cu 0.
– Calculeaza gradul intern al fiecarui nod si in cazul in care gradul intern este 0 va afisa nodul.
– Pe pozitia corespunzatoare nodului va scrie in vectorul flag2 cifra 1, astfel memorand faptul ca nodul nu mai trebuie viziat(parcurs).
– Va decrementa variabila count care fusese initializata cu numarul nodurilor apoi va sterge legaturile nodului afisat din matricea de adiacenta.
– Algoritmul se opreste in momentul in care toate nodurile au fost afisate.

Fisierul ***TopSort2.c*** contine sortarea topologica bazata pe parcurgerea grafului utilizand DFS(Depth First Search).
— Algoritmul initializeaza vectorul Vector cu -1 pentru a diferentia pozitiile din vector in care s-a scris de spatiile ramase nescrise in vector(Spre exemplu, in algoritmul recursiv dfs exista sansa de a suprascrie anumite noduri peste altele deja memorate in Vector. Initializand vectorul cu -1 ii semnalam zonele ramase libere din Vector).
— Algoritmul va scrie nodurile in Vector in orinea inversa sortarii topologice.
— Vectorul visitedB este initializat cu 0. Pe pozitia corespunzatoare nodurilor vizitate va scrie cifra 1, astfel memorand faptul ca nodul nu mai trebuie viziat(parcurs).
— Algoritmul va iesi din bucla infinita (cat timp 1 mai mare decat 0) in momentul in care toate nodurile au fost vizitate si variabila sw va ramane 0.
— Afisarea vectorului in ordine inversa determina oridinea topologica din graf.

Fisierul ***main.c*** contine functiile: verificareCiclu(int adjMatrix[MAX][MAX]), randGraph(int adjMatrix[MAX][MAX]), citirefisier(int adjMatrix[MAX][MAX]) si functia main() in care se afla meniul de comenzi.

– Functia verificareCiclu(adjMatrix[MAX][MAX]) testeaza daca graful generat random prin intermediul functiei rand () este aciclic sau nu.

– Functia randGraph(adjMatrix[MAX][MAX]) genereaza un graf random si il memoreaza in matricea adjMatrix[MAX][MAX].

– Functia citirefisier(adjMatrix[MAX][MAX]) face citirea unui graf orientat aciclic din fisierul date.txt.

Source Code

```
//----------------------------functii.h------------------------------

#ifndef MAIN_H_INCLUDED
#define MAIN_H_INCLUDED

#include<stdio.h>
#include <stdlib.h>
#include<time.h>
#define MAX 1001
#define lenghtM 200

void topSort(int adjMatrix[MAX][MAX]);
void dfs(int iterator_dfs, int pozitie);
void TopSortDFS(int adjMatrix[MAX][MAX]);
int verificareCiclu(int adjMatrix[MAX][MAX]);
void randGraph(int adjMatrix[MAX][MAX]);
void citirefisier(int adjMatrix[MAX][MAX]);

#endif
```

```
//----------------------------TopSort1.c-----------------------------

#include "functii.h"

int adjMatrix[MAX][MAX];

void topSort(int adjMatrix[MAX][MAX]){
    int iterator1;
    int iterator2;
    int count = lenghtM;
    int gradintern;
    int flag2[MAX];
    for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
        flag2[iterator1] = 0;
    }
    while(count != 0){
        for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
            gradintern = 0;
            if(flag2[iterator1] == 0){
```

```
            for(iterator2 = 0; iterator2 < lenghtM; iterator2++){
               gradintern += adjMatrix[iterator2][iterator1];
            }
            if(gradintern == 0){
                flag2[iterator1] = 1;
                printf("%d ", iterator1);
                count--;
                for(iterator2 = 0; iterator2 < lenghtM; iterator2++)
                    adjMatrix[iterator1][iterator2] = 0;
            }
        }

    }

    printf("\n");
}

//--------------------------------TopSort2.c--------------------------------


#include "functii.h"

int visitedB[MAX];
int Vector[lenghtM+1];
int adjMatrix[MAX][MAX];

void dfs(int iterator_dfs, int pozitie){


    int iterator1;
    int nr = 0;
    int iterator2;
        for(iterator1 = 0;iterator1 < lenghtM; iterator1++){
            if(adjMatrix[iterator_dfs][iterator1] == 1 &&
                visitedB[iterator1] == 0){
                dfs(iterator1, pozitie);
            }
        }
        for(iterator2 = 0;iterator2 < lenghtM; iterator2++){
            if(adjMatrix[iterator_dfs][iterator1] == 1 &&
                visitedB[iterator1] == 0){
                nr++;
            }
        }
        if(nr == 0){
            if(Vector[pozitie] == -1){
                Vector[pozitie] = iterator_dfs;
            }else{
```

```
            while(Vector[pozitie] != -1){
                pozitie++;
            }
            Vector[pozitie] = iterator_dfs;
        }
        visitedB[iterator_dfs] = 1;
    }
}

void TopSortDFS(int adjMatrix[MAX][MAX]){
    int iterator1;
    int iterator2;
    int sw;
    int gradintern;
    for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
        Vector[iterator1] = -1;
     }
    int pozitie = 0;

    while(1>0){
        sw = 0;

        for(iterator1 = 0;iterator1 < lenghtM; iterator1++){
            gradintern = 0;
            if(visitedB[iterator1] == 0){
                for(iterator2 = 0; iterator2 < lenghtM; iterator2++){
                    gradintern += adjMatrix[iterator2][iterator1];
                }
                if(gradintern == 0){
                    dfs(iterator1, pozitie);
                    sw = 1;
                }
            }
        }
        if( !sw ){
            break;
        }
    }

    for(iterator2 = lenghtM - 1; iterator2 >= 0 ;iterator2--){
        printf("%d ", Vector[iterator2]);
    }
}

//--------------------------------main.c--------------------------------

#include "functii.h"

int adjMatrix[MAX][MAX];
int aux[MAX][MAX];
```

```
int verificareCiclu(int adjMatrix[MAX][MAX]){

    int iterator1;
    int iterator2;
    int count=lenghtM;
    int gradintern;
    int sw;
    int flag[MAX];

    for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
        for(iterator2 = 0; iterator2 < lenghtM; iterator2++){
            aux[iterator1][iterator2] = adjMatrix[iterator1][iterator2];
        }
    }

    for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
        flag[iterator1]=0;
    }
    while(count != 0){
        sw = 0;
        for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
            gradintern = 0;
            if(flag[iterator1] == 0){
                for(iterator2 = 0; iterator2 < lenghtM; iterator2++){
                   gradintern += aux[iterator2][iterator1];
                }
                if(gradintern == 0){
                    sw = 1;
                    flag[iterator1] = 1;
                    count--;
                    for(iterator2 = 0; iterator2 < lenghtM; iterator2++){
                        aux[iterator1][iterator2] = 0;
                    }
                }
            }

        }
        if(sw == 0){
            return 0;
        }
    }

    return 1;
}
void randGraph(int adjMatrix[MAX][MAX]){

    int iterator1;
    int iterator2;
    time_t t;
```

```
    srand((unsigned)time(&t));

    for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
        for(iterator2 = 0; iterator2 < lenghtM; iterator2++){
            if(iterator1 == iterator2){
                continue;
            }
            if(adjMatrix[iterator2][iterator1] == 0){
                adjMatrix[iterator1][iterator2] = rand() % 2;
            }
            if(verificareCiclu(adjMatrix) == 0){
                adjMatrix[iterator1][iterator2] = 0;
            }

        }
    }
    for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
        for(iterator2 = 0; iterator2 < lenghtM; iterator2++){
            printf("%d ",adjMatrix[iterator1][iterator2]);
        }
        printf("\n");
    }

}


void citirefisier(int adjMatrix[MAX][MAX]){
    FILE *f;
    int iterator1;
    int iterator2;


    f = fopen("date.txt", "r");

    for(iterator1 = 0; iterator1 < lenghtM; iterator1++){
        for(iterator2 = 0; iterator2 < lenghtM; iterator2++){
            fscanf(f, "%d", &adjMatrix[iterator1][iterator2]);
        }
    }
}

int main()
{
    int alegere;
    printf("Alegeti metoda de citire:\n");
    printf("Apasati 1 pentru citirea din fisier.\n");
    printf("Apasati 2 pentru generarea random a unui graf.\n");
    scanf("%d", &alegere);
    if(alegere == 2){
        randGraph(adjMatrix);
```

```
    }
    else{
        citirefisier(adjMatrix);
    }
    printf("\n");
    printf("Alegeti metoda de aranjare a pieselor:\n");
    printf("Apasati 1 pentru Sortarea Topologica bazata pe eliminarea de
        varfuri.\n");
    printf("Apasati 2 pentru Sortarea Topologica bazata pe DFS.\n");
    scanf("%d", &alegere);
    if(alegere == 1){
        printf("\n========== Sortarea Topologica bazata pe eliminarea de
            varfuri ==========");
        printf("\nOrdinea in care Alexander trebuie sa aseze piesele
            este: ");
        topSort(adjMatrix);
    }
    else{
    printf("\n========== Sortarea Topologica bazata pe DFS ==========");
    printf("\nOrdinea in care Alexander trebuie sa aseze piesele este:
        ");
    TopSortDFS(adjMatrix);
    }

    return 0;
}
```

Experimente si Rezultate

```
Exemplul 1

Numar de noduri: 15

Matricea de adiacenta:

0 0 0 1 1 1 0 0 1 0 1 0 1 0 1
0 0 1 1 1 1 1 1 1 1 0 0 0 0 0
1 0 0 1 1 1 1 1 0 0 0 0 1 1 0
0 0 0 0 0 1 0 0 0 0 1 1 0 0 1
0 0 0 0 0 1 0 0 0 1 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0 1 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0
1 0 0 1 0 1 0 0 1 1 1 0 0 1 0
0 0 0 0 0 1 0 0 0 1 0 0 1 1 0
0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0



Output:

========== Sortarea Topologica bazata pe eliminarea de varfuri ==========
Ordinea in care Alexander trebuie sa aseze piesele este:
1 2 7 0 4 8 9 3 5 12 6 13 10 11 14

========== Sortarea Topologica bazata pe DFS ==========
Ordinea in care Alexander trebuie sa aseze piesele este:
1 2 7 0 8 4 9 3 5 12 6 13 11 10 14
```

Exemplul 2

Numar de noduri: 33

Matricea de adiacenta:

```
0 0 1 0 1 1 1 1 1 1 0 0 0 0 1 1 0 1 1 1 0 0 0 1 0 1 1 1 0 0 1 1 0
0 0 0 1 1 1 0 0 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0
0 1 0 1 1 1 1 1 0 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 1 0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 1 0 1
0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 1 0 1 0 1 1 0 1 1 1 1 0 0 1 1
0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0
0 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 0 1 1 1 1 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 1 1 0 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 1 1 1 1 0 0 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 1 1 1 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 1 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 0 1 1 0 0 1 1 0 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 0 1 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 1 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

Output:

```
========== Sortarea Topologica bazata pe eliminarea de varfuri ==========
Ordinea in care Alexander trebuie sa aseze piesele este:
0 2 6 1 3 4 5 7 8 9 10 14 17 19 16 11 15 13 12 18 21 20 25 23 22 26 27
    28 24 29 30 32 31
```

```
========== Sortarea Topologica bazata pe DFS ==========
Ordinea in care Alexander trebuie sa aseze piesele este:
0 2 6 1 5 4 3 7 9 8 10 14 19 17 16 15 11 13 12 18 21 20 25 23 22 26 27
    28 24 29 32 31 30
```

Concluzii

Lucrand la acest proiect am reusit sa-mi imbunatatesc abilitatile de proiectare si implementare a algoritmilor bazati pe parcurgerea grafurilor orientate cat si pe afisarea nodurilor in functie de muchiile dintre ele (Sortare Topologica). De asemenea, lucrand un timp indelungat cu grafuri am inteles aplicabilitatea lor in viata de zi cu zi, spre exemplu: Cablurile de inalta tensiune care pornesc dintr-o centrala pot fi si ele reprezentate cu usurinta cu ajutorul unui graf orientat, indicand si directia de deplasare a curentului. In acest caz centrala este un nod sursa. La fel se poate reprezenta si un sistem de canalizare, de incalzire sau reteaua de apa curenta.

Referinte

1)https://www.cs.usfca.edu/ galles/visualization/TopoSortDFS.html

2)http://home.cse.ust.hk/faculty/golin/COMP271Sp03/Notes/MyL08.pdf

3)https://visualgo.net/en/dfsbfs

4)https://ocw.cs.pub.ro/courses/pa/laboratoare/laborator-07

5)http://www.sharelatex.com

6)https://infoarena.ro/problema/sortaret

7)https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/