# Proiect14

Gila Ionut Catalin

September 2018

## 1 Enuntul problemei:

Alexander, which is a toddler, has received as a birthday gift a Lego game. However, he was not able to construct the toy Lego, as there are many pieces. Alexander has observed that each piece is numbered with a number starting from 1 to 1000. The instructions of the game state the pieces that need to be assembled befoare each an every piece. Help Alexander to build his Lego toy by developing an application which determines de correct order in which the piece of Lego need to be assembled.

## 2 Concluzie enunt:

Observam ca este vorba de un joc Lego, adica de un puzzle. Prin urmare Alexander are nevoie de instructiunile jocului de Lego care sa ii spuna in ce ordine trebuie asamblata fiecare piesa, iar pentru a-l ajuta vom considera piesele de lego noduri ale unui graf aciclic. Odata ce o piesa a fost asezata, Alexander nu se va mai putea ajuta de ea. Sortarea topologica va gasi exact ordinea corecta in care vor trebui puse piesele.

## 3 Pasii prin care se rezolva problema:

Din enuntul problemei am constatat ca aceasta trebuie sa urmeze urmatorii pasi:
1) Generarea unui numar random de noduri cu ajutorul matricei de adiacenta;
2)Verificarea matricii daca este genarata aciclic, in caz negativ trebuiesc eliminate muchiile care formeaza ciclul;
3)Sortarea topologica a matricii;

# 4 Codul

```
1
2   #include <stdio.h>
3   #include <stdlib.h>
4   #define MAX 500
5   #define lenghtM 100
6   int adjMatrix[MAX][MAX];
7
8   int verificareCiclu(int adjMatrix[MAX][MAX]){
9
10      //param adjMatrix[MAX][MAX] - matricea de adiacenta
11
12      // Functia VerificareCiclu() verifica daca graful generat random
13
14      int iterator1;
15      int iterator2;
16      int count=lenghtM;
17      int gradintern;
18      int sw;
19      int aux[MAX][MAX];
20      int flag[MAX];
21      // initializarea matricei auxiliare
22      for(iterator1=0; iterator1 < lenghtM; iterator1++){
23          for(iterator2=0; iterator2 < lenghtM; iterator2++){
24              aux[iterator1][iterator2]=adjMatrix[iterator1][iterator2];
25          }
26      }
27      // initializarea vectorului flag cu 0
28      for(iterator1=0; iterator1 < lenghtM; iterator1++)
29          flag[iterator1]=0;
30      // structura repetitiva while() parcurge fiecare nod al grafului
31      // daca nodul respectiv formeaza sau nu ciclu cu un alt grup de no
32      while(count!=0){
33          sw=0;
```

```
25          }
26      }
27      // initializarea vectorului flag cu 0
28      for(iterator1=0; iterator1 < lenghtM; iterator1++)
29          flag[iterator1]=0;
30      // structura repetitiva while() parcurge fiecare nod al grafului si
31      // daca nodul respectiv formeaza sau nu ciclu cu un alt grup de nod
32      while(count!=0){
33          sw=0;
34          for(iterator1=0; iterator1 < lenghtM; iterator1++){
35              gradintern=0;
36              if(flag[iterator1]==0){
37                  for(iterator2=0; iterator2 < lenghtM; iterator2++){
38                     gradintern+=aux[iterator2][iterator1];
39                  }
40                  if(gradintern==0){
41                      sw=1;
42                      flag[iterator1]=1;
43                      count--;
44                      for(iterator2=0; iterator2 < lenghtM; iterator2++)
45                          aux[iterator1][iterator2]=0;
46                  }
47              }
48
49          }
50
51          if(sw==0) return 0;
52      }
53
54      return 1;
55
56  }
57  void randGraph(int adjMatrix[MAX][MAX]){
```

```
57  void randGraph(int adjMatrix[MAX][MAX]){
58
59      //\fn void randGraph(int adjMatrix[MAX][MAX])
60      //\param adjMatrix[MAX][MAX] - matricea de adiacenta
61      //\brief randGraph.
62      // Functia randGraph() genereaza un graf random.
63      int iterator1, iterator2;
64      time_t t;
65      srand((unsigned)time(&t));
66
67
68      for(iterator1=0; iterator1 < lenghtM; iterator1++){
69          for(iterator2=0; iterator2 < lenghtM; iterator2++){
70      // punem conditia ca nodul sa nu aiba drum catre el insusi.
71              if(iterator1==iterator2){
72                  continue;
73              }
74      // vom genera 1 sau 0 in matricea de diacenta.
75              if(adjMatrix[iterator2][iterator1]==0){
76                  adjMatrix[iterator1][iterator2]=rand() % 2;
77
78              }
79      // daca generarea anterioara de "1" a produs ciclu intre nodurile
80              if(verificareCiclu(adjMatrix)==0){
81                  adjMatrix[iterator1][iterator2]=0;
82              }
83
84          }
85      }
86      // Afisarea matricei de adiacenta in consola.
87      for(iterator1=0; iterator1 < lenghtM; iterator1++){
88          for(iterator2=0; iterator2 < lenghtM; iterator2++){
89              printf("%d ",adjMatrix[iterator1][iterator2]);
90          }
91          printf("\n");
92      }
93
94  }
95  void topSort(int adjMatrix[MAX][MAX]){
```

```
94  }
95  void topSort(int adjMatrix[MAX][MAX]){
96
97      //Sortarea topologica a matricei
98      int iterator1,iterator2,count=lenghtM,gradintern,flag2[MAX
99      for(iterator1=0; iterator1 < lenghtM; iterator1++)
100         flag2[iterator1]=0;
101     printf("Ordinea in care Alexanderr trebuie sa aseze piesel
102     while(count!=0){
103         for(iterator1=0; iterator1 < lenghtM; iterator1++){
104             gradintern=0;
105             if(flag2[iterator1]==0){
106                 for(iterator2=0; iterator2 < lenghtM; iterator
107                     gradintern+=adjMatrix[iterator2][iterator1]
108                 }
109                 if(gradintern==0){
110                     flag2[iterator1]=1;
111                     printf("%d ",iterator1+1);
112                     count--;
113                     for(iterator2=0; iterator2 < lenghtM; iter
114                         adjMatrix[iterator1][iterator2]=0;
115                 }
116             }
117
118         }
119
120     }
121
122     printf("\n");
123
124
125
126 }
127 int main()
128 {
129     randGraph(adjMatrix);
130     topSort(adjMatrix);
131     return 0;
132 }
```

## 5 Analiza cod:

1)Functia verificareCiclu— Scopul ei este acela de a descoperi daca exista cicluri in matricea pe care o generam.
-Se citeste un vector de tipul int, caruia ii atribuim pentru fiecare pozitie valoarea 0, astfel spunem ca toate nodurile din graf sunt ”nevizitate”.
-Se realizeaza o copie a matricii, procedeul trebuie repetat pentru toate nodurile.
-Se presupune ca graful este ciclic. Se cauta ”nodul de start”. ”Nodul de start”

este acela care nu primeste legaturi de la alte noduri, mai pe scurt cel care are pe intreaga coloana toate elementele egale cu 0. Cand acest nod este gasit, algoritmul elimina complet nodul din ecuatie si ii atribuie pe linia corespunzatoare lui valoarea 0.
-Dupa ce se gaseste "nodul de start" algoritmul il elimina complet, trecandu-l in vector ca fiind "vizitat" si repeta procedeul pentru urmatoarele noduri.
-In cazul in care programul nu gaseste un nod care sa indeplineasca conditiile, aciclitatea ramane 0 si functia returneaza valoarea 0, iar in cazul in care toate nodurile indeplinesc conditiile, aciclitatea devine 1 si functia returneaza valoarea 1.

2)Functia randGraph genereaza random o matricea de adiacenta, cu alte cuvinte codul genereaza un numar random de legaturi pentru fiecare nod.

3)Functia topSort are rolul de a sorta nodurile intr-o anumita ordine astfel incat la afisarea lor in consola acestea sa nu se repete, iar functia verificareCiclu face deja acest lucru.