Structuri de date

Tema 3

Deadline soft: 12.05.2019

Deadline hard: 14.05.2019

Mihai Nan

mihai.nan.cti@gmail.com

Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2018 - 2019

1 Obiective

În urma realizării acestei teme, studentul va fi capabil:

- să implementeze şi să utilizeze grafuri în rezolvarea unor probleme;
- să implementeze algoritmi de prelucrare a grafurilor;
- să transpună o problemă din viaţa reală într-o problemă care uzitează teoria grafurilor;
- să rezolve o problemă de algoritmică, folosind noţiunile şi algoritmii studiaţi la curs pentru grafuri.

2 Problema 1

Filmul este o formă de artă care prezintă o poveste printr-o succesiune de imagini, lasând iluzia unei mişcări continue. Se spune despre film că este o formă importantă de divertisment, o foarte populară alegere pentru petrecerea timpului liber şi o puternică metodă de educaţie sau îndoctrinare. Trăind într-o lume a imaginii, suntem, fără îndoială, mari cinefili. Astfel, această temă are scopul de a vă introduce în vasta lume a filmului, îmbiâand, într-un mod plăcut, două pasiuni ale voastre: filmul şi grafurile.

Existând o gamă largă de filme şi, implicit, un număr foarte mare de actori care au contribuit la realizarea acestor filme, este greu de reţinut o colecţie vastă. În cadrul acestei teme, o să modelaţi o colecţie de filme, uzitând o structură de date avantajoasă, şi o să rezolvaţi o serie de cerinţe.

Pornind de la ideea unui joc celebru în lumea cinematografica: ***Cele şase grade ale lui Kevin Bacon***; o să definim algoritmul de construcţie a grafului ce va modela colecţia noastră. Actorul american Kevin Bacon este unul dintre cei mai apreciaţi actori de la Hollywood, având la activ zeci de filme celebre şi seriale. Graţie portofoliului său bogat şi carierei îndelungate, trei studenţi americani au elaborat în anul 1994 ipoteza potrivit căreia prolificul actor Kevin Bacon ar putea fi *înrudit*, din punct de vedere cinematografic, cu orice actor din lume, prin intermediul unui lanţ ce include cel mult şase filme.

2.1 Algoritmul de creare a grafului

Se dă un fişier text care conţine informaţii despre o colecţie de filme. Fiecare actor reprezintă un nod din graful care modelează colecţia, iar doi actori sunt conectaţi printr-o muchie dacă au fost implicaţi în realizarea unui film comun. Pentru o mai bună înţelegere a algoritmului de creare a grafului, se va oferi un exemplu detaliat.

Gradul de înrudire a doi actori este egal cu distanţa minimă dintre cele două noduri ale grafului, calculată ca număr de muchii parcurse. Dacă doi actori nu sunt conectaţi printr-o serie de filme, aceştia au ***gradul de înrudire*** egal cu 0.

Definim o ***producţie*** ca fiind o serie de filme în realizarea cărora au contribuit numai actori înrudiţi (cu gradul de înrudire mai mare ca 0).

2.2 Detalierea reprezentării

2.2.1 Informaţii

Se porneşte de la următoarea colecţie:

Film1
- Actor1
- Actor2
- Actor3

Film2
- Actor1
- Actor3
- Actor4

Film3
- Actor2
- Actor4
- Actor5

Film4
- Actor6
- Actor7
- Actor8

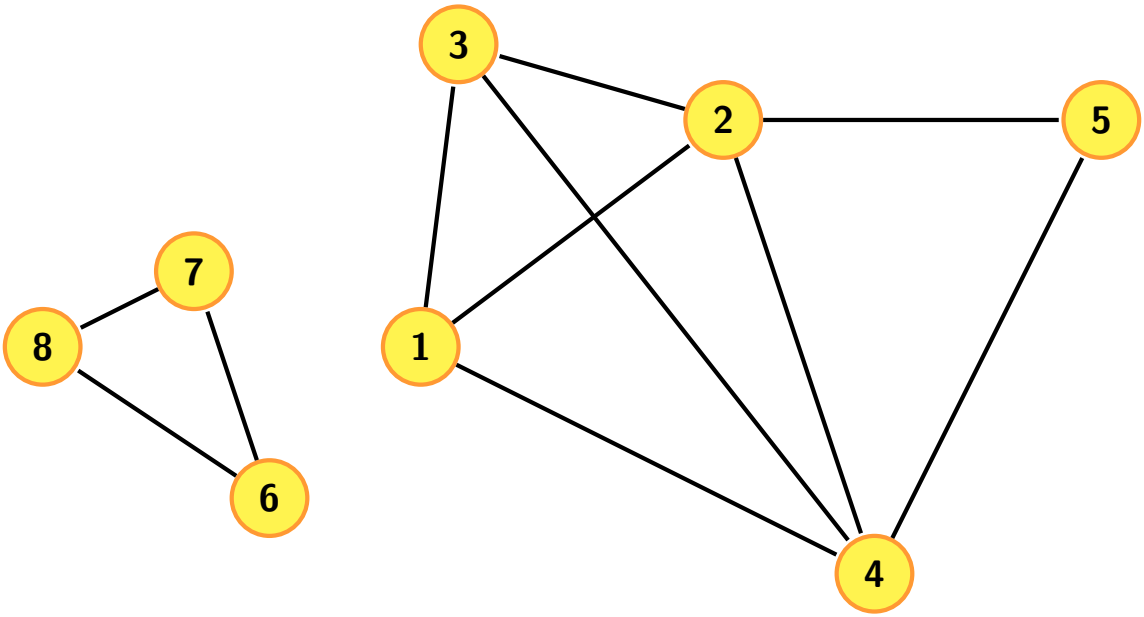
2.2.2 Explicaţii

Pentru simplitate şi o mai bună reprezentare, vom considera că fiecare nod este reprezentat printr-un nod ce are ca etichetă un număr (numărul existent şi în numele acestuia).

După cum se poate deduce, în exemplul expus există două producţii. Prima producţie conţine primele 3 filme (*Film1*, *Film2* şi *Film3*), iar a doua producţie conţine doar ultimul film (*Film4*).

Gradul de înrudire pentru actorii *Actor1* şi *Actor5* este 2, în timp ce pentru actorii *Actor1* şi *Actor7* este 0.

2.2.3 Reprezentare

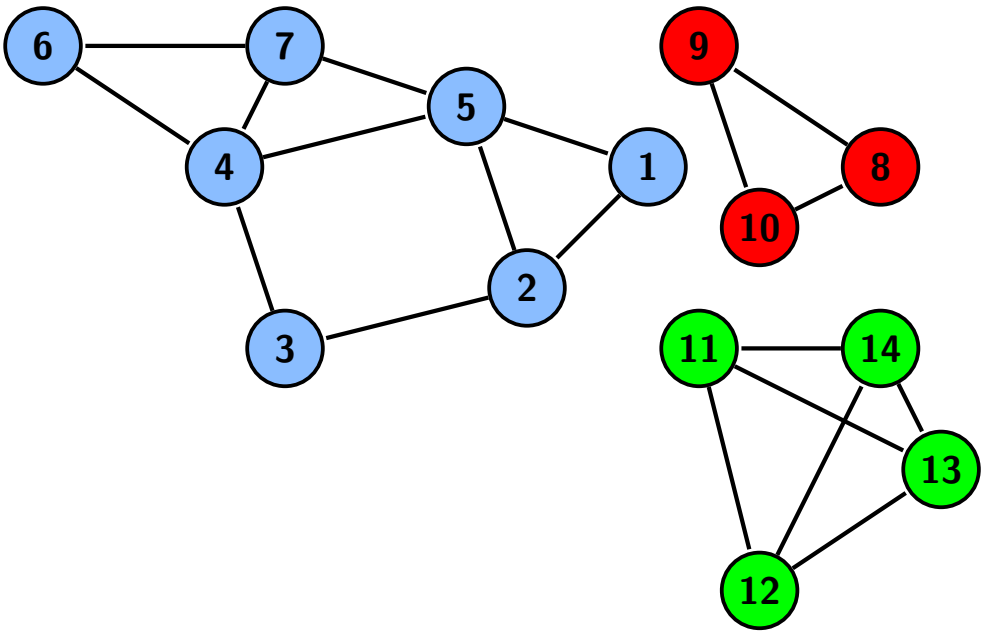


2.3 Cerinţe

2.3.1 Cerinţa 1

Se doreşte determinarea numărului de producţii independente existente în colecţia de filme pusă la dispoziţie.

Pentru a înţelege mai clar ce trebuie realizat la această cerinţă, pornim de la analizarea următorului exemplu care conţine 3 producţii, actorii din fiecare producţie fiind coloraţi cu una dintre culorile *albastru*, ***roşu***, ***verde***.



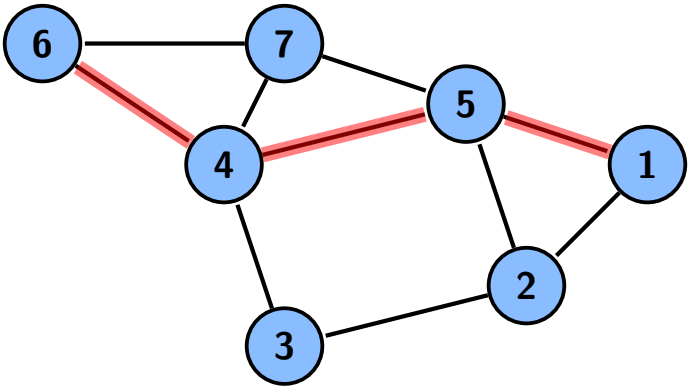
Important

Este **obligatoriu** ca rezolvarea acestei cerinţe să fie realizată folosind o structură de date de tip **graf** reprezentat uzitând liste de adiacenţă!

2.3.2 Cerinţa 2

Pentru această cerinţă, dorim să determinăm gradul de înrudire dintre doi actori, pe baza jocului ***Cele şase grade ale lui Kevin Bacon***. Dacă actorul X a jucat într-un film împreună cu actorul Y, atunci gradul de înrudire dintre X şi Y este 1. Dacă actorul X nu a jucat împreună cu actorul Y într-un film, dar actorul X a făcut parte din distribuţia unui film împreună cu Z, iar Y a jucat la rândul lui, într-un alt film, cu Z, atunci gradul de înrudire pentru X şi Y este 2.

Considerând drept exemplul graful de mai jos, gradul de înrudire dintre Actorul1 şi Actorul6 este 3.



Important

Este **obligatoriu** ca rezolvarea aceste cerinţe să fie realizată folosind o structură de date de tip **graf** reprezentat uzitând liste de adiacenţă!

2.3.3 Cerinţa 3

Definiţii

1. Se numeşte lanţ într-un graf neorientat, o secvenţă de vârfuri $[x_1, x_2, \ldots, x_k]$, cu proprietatea că oricare două vârfuri consecutive din secvenţă sunt adiacente.
2. Un graf se numeşte **conex** dacă pentru oricare două vârfuri x şi y diferite ale sale, există un lanţ care le leagă.
3. Un **punct de articulaţie** (***cut vertex***) este un nod al unui graf a cărui eliminare duce la creşterea numărului de componente conexe ale acelui graf.
4. Fie T un arbore de adâncime descoperit de parcurgerea grafului. Atunci, un nod v este punct de articulaţie dacă:
   - v este rădăcina lui T şi v are doi sau mai mulţi copii;
   - v nu este rădăcina lui T şi are un copil u în T, astfel încât nici un nod din subarborele dominat de u nu este conectat cu un strămoş al lui v printr-o muchie înapoi (copii lui nu pot ajunge pe altă cale pe un nivel superior în arborele de adâncime).

O informaţie care ar putea fi interesantă pentru o persoană pasionată de filme şi grafuri este aceea legată de numele actorilor care sunt puncte de articulaţie în graful construit pe baza raţionamentului prezentat anterior. Pentru determinarea punctelor de articulaţie într-un graf neorientat se foloseşte o parcurgere în adâncime modificată, reţinându-se informaţii suplimentare pentru fiecare nod. Acest algoritm a fost identificat tot de către **Tarjan**.

Algoritmul lui Tarjan

Vom porni de la următoarele notaţii:

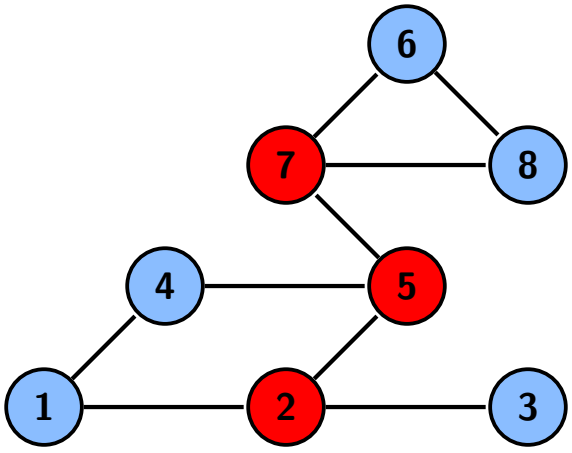
$$idx[u] = \text{timpul de descoperire a nodului} u$$

$$low[u] = \textbf{min}(\{idx[u]\} \cup \{idx[v] | (u, v) \text{ este o muchie înapoi }\} \cup \{low[vi] | vi \text{ copil al lui } u \text{ în arborele de adâncime }\})$$

Folosind aceste notaţii, condiţia pentru ca un nod v să fie punct de articulaţie în graf se poate scrie:

$$v \text{ este punct de articulaţie } \Leftrightarrow low[u] \geq idx[v], \text{ pentru cel puţin un copil } u \text{ al lui } v \text{ în T.}$$

În graful din figura de mai jos, nodurile colorate cu roşu reprezintă puncte critice.



Actorii care sunt **puncte de articulaţie** în graful producţiilor cinematografice vor fi afişaţi în ordine alfabetică.

Important

Este **obligatoriu** ca rezolvarea aceste cerinţe să fie realizată folosind o structură de date de tip **graf** reprezentat uzitând liste de adiacenţă!

Algorithm 1 PuncteArticulatie

```
1:  procedure PuncteArticulatie(G)
2:      (V, E) ← G
3:      timp ← 0
4:      for each v ∈ V do
5:          idx[v] ← −1
6:          low[v] ← ∞
7:      for each v ∈ V do
8:          if idx[v] == −1 then
9:              dfsCV(G, v, timp, idx, low)
10: procedure dfsCV(G, v, timp, idx, low)
11:     (V, E) ← G
12:     idx[v] ← timp
13:     low[v] ← timp
14:     timp ← timp + 1
15:     copii ← ∅
16:     for each (v, u) ∈ E do
17:         if idx[u] == −1 then                ▷ înseamnă că nodul u este nedescoperit, deci alb
18:             copii ← copii ∪ {u}
19:             dfsCV(G, u, timp, idx, low)
20:             low[v] ← min(low[v], low[u])
21:         else          ▷ înseamnă că nodul u este descoperit, deci gri, iar muchia v → u este muchie înapoi
22:             low[v] ← min(low[v], idx[u])
23:     if v este radacina arborelui then
24:         if |copii| ≥ 2 then
25:             print(v este punct de articulaţie)
26:     else
27:         if ∃u ∈ copii astfel încât low[u] ≥ idx[v] then
28:             print(v este punct de articulaţie)
```

2.3.4 Bonus

Acum că ştim gradul de înrudire pentru actori ne dorim să aflăm mulţimea maximală de actori care conţine doar persoane care se cunosc de pe platourile de filmare. Cu alte cuvinte, în această mulţime nu există o pereche de actori X şi Y astfel încât X să nu fi jucat în cel puţin un film împreună cu Y.

Definiţii

1. O **clică** C, într-un graf neorientat $G = (V, E)$, V – mulţimea vârfurilor grafului şi E – mulţimea muchiilor grafului, este o submulţime de noduri, $C \subseteq V$, astfel încât fiecare două noduri distincte sunt adiacente. Acest lucru este echivalent cu condiţia ca subgraful indus al lui G de către C să fie **graf complet**.
2. Un **graf complet** este un graf neorientat simplu în care fiecare pereche de noduri distincte este conectată printr-o muchie unică.
3. O **clică maximală** este o clică care nu poate fi extinsă prin includerea niciunui alt nod adiacent, adică o clică care nu este inclusă în mulţimea de noduri a unei clici mai mari.

Pentru rezolvarea problemei determinării tuturor clicilor dintr-un graf putem folosi un algoritm de generare a acestora, folosind tehnica **Backtracking**. În cadrul acestui algoritm, vom folosi următoarele notaţii:

- CS – mulţimea vârfurilor din CS induce un subgraf complet în G (***C**omplete **S**ubgraph*);
- CA – mulţimea vârfurilor utilizate pentru a extinde CS astfel încât să devină clică (***CA**ndidates*);
- NOT – mulţimea vârfurilor care au fost utilizate până acum pentru extinderea CS şi sunt în continuare excluse.

Mulţimile CA şi NOT conţin toate vârfurile grafului, care nu aparţin mulţimii CS, dar sunt adiacente cu toate vârfurile din CS.

Pe parcursul algoritmului, vom nota NOT_i şi CA_i mulţimile NOT şi CA de la pasul i.

Algoritmul, prezentat în continuare, generează recursiv toate extensiile mulţimii CS, adăugând vârfuri din CA, nu din NOT. Vârfurile din NOT sunt excluse, deoarece într-o etapă anterioară au fost construite toate extensiile mulţimii CS care conţin acele vârfuri.

Algorithm 2 GenerareClici

```
1:  procedure GenerareClici(G, i, CS, NOT_i, CA_i)
2:      if CA_i ≠ ∅ then
3:          (V, E) ← G
4:          c ← extract_first(CA_i)
5:          CA_i ← CA_i \ {c}
6:          CS ← CS ∪ {c}
7:          CA_{i+1} ← CA_i \ {x|x ∈ CA_i, (x, c) ∉ E}
8:          NOT_{i+1} ← NOT_i \ {x|x ∈ NOT_i, (x, c) ∉ E}
9:          GenerareClici(G, i + 1, CS, NOT_{i+1}, CA_{i+1})
10:         CS ← CS \ {c}
11:         NOT_i ← NOT_i ∪ {c}
```

Pentru ca CS să fie clică maximală, trebuie ca atât CA, cât şi NOT să fie vide.

Condiţia de mărginire: Dacă NOT conţine un vârf (să-l notăm v) adiacent cu toate vârfurile din CA, nu se va obţine o nouă clică (pentru că în orice apel ulterior v va rămâne în NOT şi subgraful complet obţinut nu e maximal).

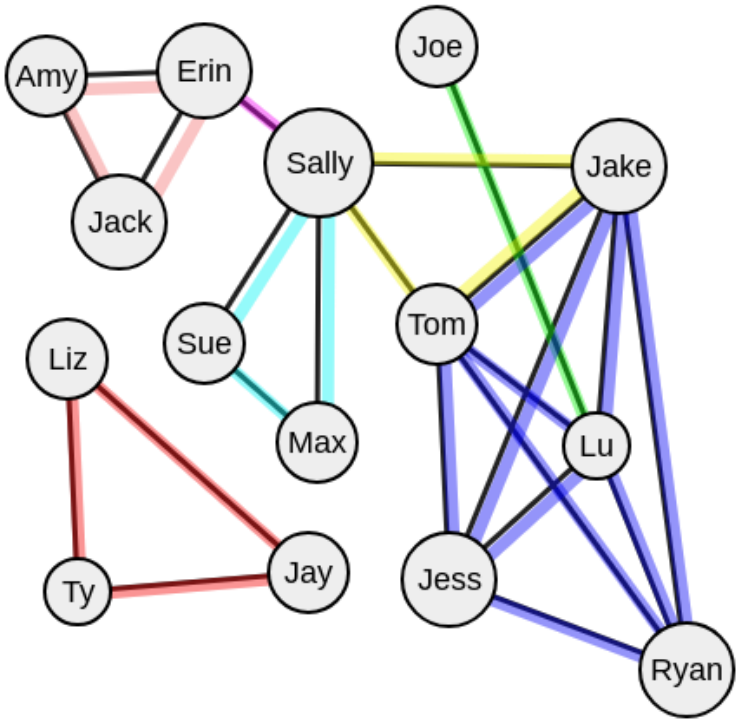
Algoritmul Bron-Kerbosch

Algoritmul precedent, bazat pe tehnica **Backtracking**, generează toate clicile, în ordine lexicografică.

Algoritmul **Bron-Kerbosch** selectează un nou vârf din mulţimea de candidaţi astfel încât condiţia de mărginire să devină adevărată cât mai repede cu putinţă. Pentru aceasta:

1. Se determină un vârf v din $NOT \cup CA$ cu număr maxim de vârfuri adiacente în CA.

2. Dacă $v \in CA$, v va fi noul candidat. La revenirea din apelul recursiv, v va fi plasat în NOT. În acest moment, $v \in NOT$ şi vom continua selectând mai întâi acei candidaţi care nu sunt adiacenţi cu v. Dacă nu mai există candidaţi în CA care nu sunt adiacenţi cu v, aplicăm condiţia de mărginire şi revenim.



Consecinţă

Algoritmul **Bron-Kerbosch** nu generează clicile în ordine lexicografică, ci are tendinţa de a genera mai întâi clicile de dimensiune mai mare.

Clicile identificate şi colorate, în graful de mai sus, sunt următoarele:

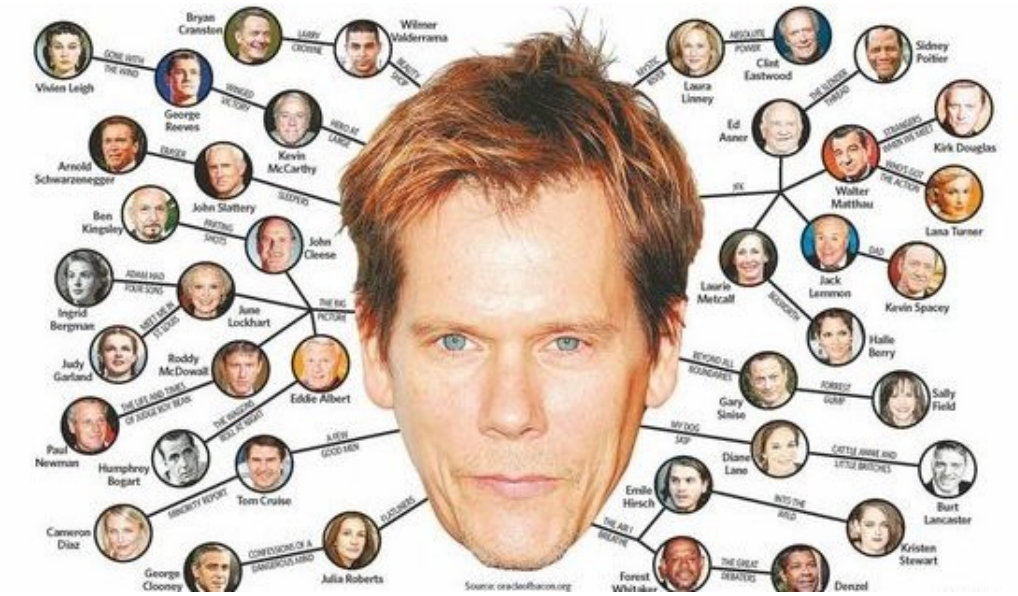
$[Amy, Jack, Erin]$; $[Erin, Sally]$; $[Sally, Sue, Max]$; $[Sally, Jake, Tom]$; $[Jake, Tom, Lu, Ryan, Jess]$;

$[Lu, Joe]$; $[Liz, Ty, Jay]$

Având în vedere că dorim să determinăm submulţimea maximală care induce o clică, răspunsul acestei probleme, pentru exemplul anterior, este $[Jake, Tom, Lu, Ryan, Jess]$.

Important

Pentru reprezentarea mulţimilor **este interzisă folosirea vectorilor**. O structură de date avantajoasă pentru menţinerea datelor în ordine, căutarea, inserarea şi eliminarea unei valori este **arborele binar de căutare**.



2.4 Formatul fişierelor

Fişierul de intrare va conţine, pe prima linie, un număr natural **Nr**, reprezentând numărul de filme din colecţie. Pe următoarele linii se vor găsi informaţiile despre cele **Nr** filme. Pentru fiecare film se va specifica numele filmului, numărul de actori implicaţi în realizarea acelui film şi o listă a actorilor (câte unul pe rând).

Pentru **cerinţa 1**, acest fişier nu va conţine informaţii suplimentare, iar fişierul de ieşire va conţine un singur număr, reprezentând numărul de producţii independente.

Pentru **cerinţa 2**, fişierul de intrare va conţine, la sfârşit două linii suplimentare. Pe penultima linie din fişier şi pe ultima se vor afla numele actorilor pentru care ne dorim să determinăm gradul de înrudire.

Pentru **cerinţa 3**, nu vom avea informaţii suplimentare în fişierul de intrare, iar în fişierul de ieşire se va afişa pe prima linie numărul de puncte de articulaţie din graful construit, iar pe următoarele linii se vor afişa numele actorilor a căror eliminare duce la creşterea numărului de componente conexe în graful producţiei. Numele actorilor se vor afişa în ordine alfabetică.

Pentru **cerinţa bonus**, fişierul de intrare va avea formatul standard, iar fişierul de ieşire va conţine pe prima linie numărul de actori care formează clica maximală, iar pe următoarele linii se vor afişa actorii din această clică, în ordine alfabetică, câte un actor pe linie.

3 Problema 2

Roboţelul Wall-e se află blocat într-un labirint, ce conţine N camere, din care doreşte să evadeze. Problema principală este că în acest labirint există o serie de paznici care îl patrulează, dar roboţelul a reuşit să realizeze o hartă în care sunt trecute pentru fiecare cameră momentele de timp în care se va afla un paznic care se asigură că nu pătrunde nimeni în camera respectivă. Camerele sunt conectate între ele printr-o serie de portaluri. De asemenea, în harta realizată de Wall-e sunt incluse şi informaţii despre aceste portaluri, iar fiecare portal este bidirecţional. Mai exact, roboţelul ştie pentru fiecare camera câte portaluri conţine, iar pentru fiecare portal ştie care este camera în care urmează să fie teleportat şi cât durează această călătorie enigmatică. Doar anumite camere conţin ieşiri, însă înţeleptul Wall-e şi-a completat această informaţie pe hartă.

Ştiind camera din care îşi începe misiunea iscusitul robot, dorim să determinăm timpul minim petrecut pentru a evada din labirint şi camerele prin care acesta trebuie să treacă. Vom considera că roboţelul porneşte în această mirifică misiune la momentul de timp 0. Dacă la acel moment de timp există un paznic în camera în care se află, el trebuie să aştepte până pleacă pentru a putea începe tentativa de evadare. Dacă roboţelul ajunge într-o camera la momentul de timp t, iar în harta sa are conspectat faptul că în camera respectivă ar trebui să se afle un paznic la momentul respectiv, atunci roboţelul rămâne captiv în camera respectivă până într-un moment de timp în care paznicul pleacă. Dacă există o ieşire din labirint într-o astfel de cameră, roboţelul reuşeşte să evadeze, nemaifiind necesar să aştepte până când paznicul pleacă din cameră pentru a-şi putea finaliza misiunea.

Sarcina voastră este de a-l ajuta pe Wall-e să evadeze din labirint, indicându-i camerele prin care trebuie să treacă pentru a ajunge la o ieşire şi timpul minim petrecut.

3.1 Formatul fişierelor

Fişierul de intrare

Numele fişierului de intrare este **labirint.in**. Pe prima linie se găsesc 3 numere naturale (N – numărul de camere din labirint, M – numărul de portaluri şi S – indicele camerei din care îşi începe aventura roboţelul) depărţite printr-un spaţiu. Pe următoarele M linii se vor găsi câte 3 numere naturale (u – prima extremitate a portalului, v – a doua extremitate a portalului, t – timpul necesar tranziţiei). În continuare, se vor găsi N linii care descriu informaţiile de pe hartă. Primul caracter de pe fiecare linie poate să fie Y – există o ieşire în camera respectivă; sau N – nu există o ieşire în camera respectivă. Apoi o să existe numărul de momente de timp în care se va afla un paznic în camera respectivă, după care se vor găsi valorile momentelor de timp.

Fişierul de ieşire

Numele fişierului de ieşire este **labirint.out**. Pe prima linie se va afişa timpul minim de evadare. Pe următoarea linie se vor afişa camerele prin care trebui să treacă roboţelul pentru a evada în timpul indicat.

Dacă nu există o modalitate de evadare din labirint, o să afişaţi valoarea -1.

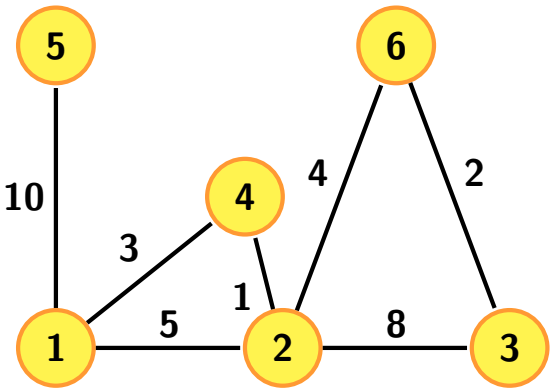
3.2 Exemplu

Intrare

```
6 7 1
1 2 5
1 4 3
2 4 1
2 3 8
2 6 4
3 6 2
1 5 10
N 0
N 4 2 3 4 7
Y 0
N 3 3 6 7
N 3 10 11 12
N 3 7 8 9
```

Ieşire

```
12
1 4 2 6 3
```



Explicaţie

Roboţelul porneşte din nodul 1 la momentul de timp $t_0 = 0$, iar pentru că atunci nu se află nicun paznic în camera 1 alege să meargă în camera 4 unde ajunge la momentul de timp $t_1 = t_0 + 3 = 3$. Când ajunge în camera 4 găseşte acolo un paznic şi este nevoit să aştepte până la momentul 4 când poate pleca din cameră. El alege să meargă în camera 2, unde ajunge la momentul de timp $t_2 = t_1 + 1 + 1 = 5$. Nu există paznic în camera 2 la acel moment de timp, deci poate trece portalul pentru a ajunge în camera 6 la momentul de timp $t_3 = t_2 + 4 = 5 + 4 = 9$, iar acolo găseşte un paznic. Abia la momentul de timp 10 poate pleca din camera 6, trecând prin portalul pentru camera 3. În camera 3 robotul ajunge la momentul $t_4 = 12$, iar această cameră conţine o ieşire şi poate evada din labirint.

4 Restricţii şi precizări

Temele trebuie să fie încărcate pe vmchecker. **NU** se acceptă teme trimise pe e-mail sau altfel decât prin intermediul vmchecker-ului.

O rezolvare constă într-o arhivă de tip zip care conţine toate fişierele sursă alături de un Makefile, ce va fi folosit pentru compilare, şi un fişier README, în care se vor preciza detaliile implementării.

Makefile-ul trebuie să aibă obligatoriu regulile pentru build şi clean. Regula build trebuie să aibă ca efect compilarea surselor şi crearea binarelor movies şi labirint.

Pentru prima problemă, programul vostru va primi, ca argumente în linia de comandă, numele fişierului de intrare şi a celui de ieşire, dar şi o opţiune, pentru fiecare cerinţă în parte, în felul următor:

./movies [-c1 | -c2 | -c3 | -b] [fisier_intrare] [fisier_iesire]

5 Punctaj

Cerinţa	Punctaj
Problema 1 – Cerinţa 1	15 puncte
Problema 1 – Cerinţa 2	15 puncte
Problema 1 – Cerinţa 3	25 de puncte
Problema 2	35 de puncte
Codying style, README, warning-uri	10 puncte
Problema 1 – Bonus	20 de puncte

Atentie!

Orice rezolvare care nu conţine structurile de date specificate nu este punctată.
Temele vor fi punctate doar pentru testele care sunt trecute pe vmchecker.
Nu lăsaţi warning-urile nerezolvate, deoarece veţi fi depunctaţi.
Dealocaţi toată memoria alocată pentru reţinerea informaţiilor, deoarece se vor depuncta pierderile de memorie.
Tema este individuală! Toate soluţiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului.