

# TOPOLOGINIO RŪŠIAVIMO UŽDAVINYS

Raimondas Čiegis

Matematinio modeliavimo katedra, e-paštas: rc@vgtu.lt

Gruodžio 8 d., 2022

Šioje paskaitoje nagrinėsime dar vieną svarbų duomenų rūšiavimo uždavinį.

Šioje paskaitoje nagrinėsime dar vieną svarbų duomenų rūšiavimo uždavinį.

Kodėl tik dabar?

Šioje paskaitoje nagrinėsime dar vieną svarbų duomenų rūšiavimo uždavinį.

Kodėl tik dabar?

Ar jo sprendimui reikia kurti naujus algoritmus, kai jau žinome keletą optimalių rūšiavimo algoritmų?

Šioje paskaitoje nagrinėsime dar vieną svarbų duomenų rūšiavimo uždavinį.

Kodėl tik dabar?

Ar jo sprendimui reikia kurti naujus algoritmus, kai jau žinome keletą optimalių rūšiavimo algoritmų?

Atsakymai:

1. Kaip matysime, naujasis rūšiamo uždavinys panašus į klasikinius rūšiavimo uždavinius, bet tas panašumas tėra paviršutiniškas.
2. Jo sprendimo algoritmus geriausia aprašyti [grafų teorijos](#) kalba. Ją naudojome jau ankstesnėse dviejose paskaitose.

Tokius uždavinius dažnai tenka spręsti sudarant tvarkaraščius.

Tokius uždavinius dažnai tenka spręsti sudarant tvarkaraščius.

1. Transporto tvarkaraščiai, kai siekiame derinti skirtingų maršrutų grafikus, kad keleiviai galėtų naudotis jungiamaisiais reisais ir persėdimų trukmė būtų kuo trumpesnė.

Tokius uždavinius dažnai tenka spręsti sudarant tvarkaraščius.

1. Transporto tvarkaraščiai, kai siekiame derinti skirtingų maršrutų grafikus, kad keleiviai galėtų naudotis jungiamaisiais reisais ir persėdimų trukmė būtų kuo trumpesnė.
2. Daugelis šiuolaikinės ekonomikos ir skaitmeninių technologijų procesų susideda iš didelio kiekio tarpinių operacijų ir reikia garantuoti šių veiksmų atlikimo eiliškumą.



Tokius uždavinius dažnai tenka spręsti sudarant tvarkaraščius.

1. Transporto tvarkaraščiai, kai siekiame derinti skirtingų maršrutų grafikus, kad keleiviai galėtų naudotis jungiamaisiais reisais ir persėdimų trukmė būtų kuo trumpesnė.
2. Daugelis šiuolaikinės ekonomikos ir skaitmeninių technologijų procesų susideda iš didelio kiekio tarpinių operacijų ir reikia garantuoti šių veiksmų atlikimo eiliškumą.
3. Lygiagrečiai skaičiavimai, kai skirtingi algoritmo veiksmai yra atliekami skirtinguose procesoriuose (branduoliuose). Vėl turime garantuoti skirtingų veiksmų eiliškumą (priklausomybę). Pvz. taip generuojami vaizdai Jūsų telefonuose, kai žiūrite filmuką ar žaidžiate jdomų internetinį žaidimą.

Tarkime, turime užduočių aibę

$$U = (u_1, u_2, \dots, u_N)$$

ir jų atlikimo eiliškumo sąryšių aibę

$$C = (u_{i_1} \prec u_{j_1}, u_{i_2} \prec u_{j_2}, \dots, u_{i_M} \prec u_{j_M}).$$

Tarkime, turime užduočių aibę

$$U = (u_1, u_2, \dots, u_N)$$

ir jų atlikimo eiliškumo sąryšių aibę

$$C = (u_{i_1} \prec u_{j_1}, u_{i_2} \prec u_{j_2}, \dots, u_{i_M} \prec u_{j_M}).$$

Čia sąryšys  $u_{i_k} \prec u_{j_k}$  reiškia, kad užduotį  $u_{j_k}$  galėsime pradėti vykdyti tik tada, kai bus baigta  $u_{i_k}$  užduotis.

Tarkime, turime užduočių aibę

$$U = (u_1, u_2, \dots, u_N)$$

ir jų atlikimo eiliškumo sąryšių aibę

$$C = (u_{i_1} \prec u_{j_1}, u_{i_2} \prec u_{j_2}, \dots, u_{i_M} \prec u_{j_M}).$$

Čia sąryšys  $u_{i_k} \prec u_{j_k}$  reiškia, kad užduotį  $u_{j_k}$  galėsime pradėti vykdyti tik tada, kai bus baigta  $u_{i_k}$  užduotis.

Reikia taip sutvarkyti aibę  $U' = (u'_1, u'_2, \dots, u'_N)$ , kad būtų įvykdyti visi aibės  $C$  sąryšiai, t.y. jei turime sąryšį  $u'_i \prec u'_j$ , tai  $i < j$ .

Aišku, kad šie reikalavimai dažnai neapibrėžia vienintelės surūšiuotos aibės ir galime rasti daug sprendinių.

Tokį uždavinį vadinsime **topologiniu rūšiovimu**.

## Medelių sodinimas

Reikia pasodinti tris medelius. Pažymėkime  $d_j$  duobės iškasimo,  $p_j$  medelio sodinimo  $j$  duobę,  $u_j$  duobės užkasimo užduotis, sodinant  $j$ -ąjį medelį.

## Medelių sodinimas

Reikia pasodinti tris medelius. Pažymėkime  $d_j$  duobės iškasimo,  $p_j$  medelio sodinimo  $j$  duobę,  $u_j$  duobės užkasimo užduotis, sodinant  $j$ -ąjį medelį.

Tada turime devynių užduočių aibę

$$U = (d_1, d_2, d_3, p_1, p_2, p_3, u_1, u_2, u_3).$$

## Medelių sodinimas

Reikia pasodinti tris medelius. Pažymėkime  $d_j$  duobės iškasimo,  $p_j$  medelio sodinimo  $j$  duobę,  $u_j$  duobės užkasimo užduotis, sodinant  $j$ -ąjį medelį.

Tada turime devynių užduočių aibę

$$U = ( d_1, d_2, d_3, p_1, p_2, p_3, u_1, u_2, u_3 ).$$

Darbų eiliškumą nusako šie akivaizdūs sąryšiai:

$$C = ( d_j \prec p_j, p_j \prec u_j, j = 1, 2, 3 ).$$

Egzistuoja kelios šio uždavinio topologiškai surūšiuotos aibės.  
Pavyzdžiui, galime sodinti kiekvieną medelį atskirai, tada gauname užduočių aibę

$$U' = (d_1, p_1, u_1, d_2, p_2, u_2, d_3, p_3, u_3).$$



Egzistuoja kelios šio uždavinio topologiškai surūšiuotos aibės.  
Pavyzdžiui, galime sodinti kiekvieną medelį atskirai, tada gauname užduočių aibę

$$U' = (d_1, p_1, u_1, d_2, p_2, u_2, d_3, p_3, u_3).$$

Darbus galime paskirstyti ir kitaip, pirmiausia turime iškasti visas duobes, į jas sodinti medelius, o paskui visas tris duobes užkasti (atskirų medelių eiliškumas vėl nesvarbus):

$$U'' = (d_1, d_2, d_3, p_3, p_2, p_1, u_2, u_3, u_1).$$

Egzistuoja kelios šio uždavinio topologiškai surūšiuotos aibės.  
Pavyzdžiui, galime sodinti kiekvieną medelį atskirai, tada gauname užduočių aibę

$$U' = (d_1, p_1, u_1, d_2, p_2, u_2, d_3, p_3, u_3).$$

Darbus galime paskirstyti ir kitaip, pirmiausia turime iškasti visas duobes, į jas sodinti medelius, o paskui visas tris duobes užkasti (atskirų medelių eiliškumas vėl nesvarbus):

$$U'' = (d_1, d_2, d_3, p_3, p_2, p_1, u_2, u_3, u_1).$$

Visai nevykęs, bet deja pasitaikantis variantas, kai darbai pradedami vykdyti tokia tvarka  $U^* = (d_1, d_2, d_3, u_2, u_3, u_1)$ . Dvi brigados džiaugiasi, kad darbą atliko, **bet rezultatas ...**

Dabar pateiksime tikslesnį topologinio rūšiavimo uždavinio formulavimą. Turime orientuotąjį grafą  $G = (V, E)$ , kuriame nėra ciklų.

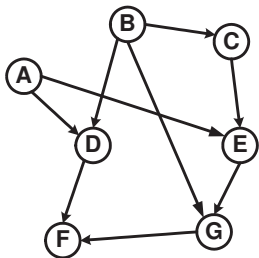
Dabar pateiksime tikslesnį topologinio rūšiavimo uždavinio formulavimą. Turime orientuotąjį grafą  $G = (V, E)$ , kuriame nėra ciklų.

Grafo viršūnes reikia sužymėti taip, kad kiekviena briauna jungtų mažesnio numerio viršūnę su didesnio numerio viršūne.

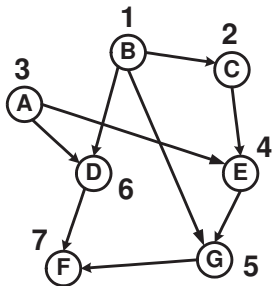
Dabar pateiksime tikslesnį topologinio rūšiavimo uždavinio formulavimą. Turime orientuotąjį grafą  $G = (V, E)$ , kuriame nėra ciklų.

Grafo viršūnes reikia sužymėti taip, kad kiekviena briauna jungtų mažesnio numerio viršūnę su didesnio numerio viršūne.

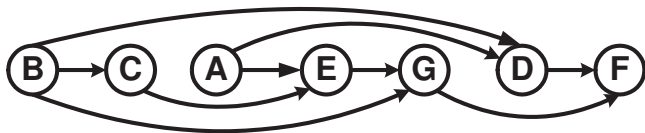
Topologinio rūšiavimo uždavinio sprendimo pavyzdys pateiktas paveiksle.



a) pradinis grafas



b) surūšiuotas grafas



c) viršūnių išdėstymas tiesėje.

## Paieškos gilyn metodas

Matysime, kaip šį sudėtingą rūšiavimo uždavinį išsprendžiame vien tik specialiai parinkdami grafo viršūnių apilankymo tvarką.

## Paieškos gilyn metodas

Matysime, kaip šį sudėtingą rūšiavimo uždavinį išsprendžiame vien tik specialiai parinkdami grafo viršūnių apšaukimo tvarką.

Prisiminkite aritmetinės išraiškos spausdinimą **prefix, infix ir postfix** forma, kai tinkamai parinkome grafo viršūnių apšaukimo rekursinį algoritmą.



## Paieškos gilyn metodas

Matysime, kaip šį sudėtingą rūšiavimo uždavinį išsprendžiame vien tik specialiai parinkdami grafo viršūnių aplankymo tvarką.

Prisiminkite aritmetinės išraiškos spausdinimą **prefix, infix ir postfix** forma, kai tinkamai parinkome grafo viršūnių aplankymo rekursinį algoritmą.

Šios paieškos strategija yra paprasta: iš duotosios viršūnės einame į jai gretimą, paieškos metu dar neaplankytą grafo viršūnę.

## Paieškos gilyn metodas

Matysime, kaip šį sudėtingą rūšiavimo uždavinį išsprendžiame vien tik specialiai parinkdami grafo viršūnių aplankymo tvarką.

Prisiminkite aritmetinės išraiškos spausdinimą **prefix**, **infix** ir **postfix** forma, kai tinkamai parinkome grafo viršūnių aplankymo rekursinį algoritmą.

Šios paieškos strategija yra paprasta: iš duotosios viršūnės einame į jai gretimą, paieškos metu dar neaplankytą grafo viršūnę.

Jei tokių viršūnių nėra, tai žengiame vieną žingsnį atgal ir ieškome naujo kelio iš tėvo viršūnės. Taip surandame visas viršūnes, kurias galima pasiekti iš pasirinktos pradinės viršūnės.

Jei grafas nėra jungusis, tai algoritmą kartojame, imdami naują dar neaplankytą pradinę viršūnę.

Jei grafas nėra jungusis, tai algoritmą kartojame, imdami naują dar neaplankytą pradinę viršūnę.

Kadangi paieškos metu pirmiausia aplankome labiausiai nutolusias viršūnes, tai metodą vadiname paieškos gilyn metodu (angl. *depth first search*).

Kiekviena grafo viršūnė gali būti vienoje iš trijų būsenų (būsenas žymėsime skirtingomis spalvomis). Pradžioje visos viršūnės yra *neaplankytos* ir dažomos **balta** spalva.

Kiekviena grafo viršūnė gali būti vienoje iš trijų būsenų (būsenas žymėsime skirtingomis spalvomis). Pradžioje visos viršūnės yra *neaplankytos* ir dažomos **balta** spalva.

Kai viršūnė  $v$  aplankoma pirmą kartą, ji tampa *nenauja* ir dažoma **pilka** spalva. Laiką, kada ji tapo nenauja, saugome masyvo elemente  $d(v)$  (angl. *discovered*).

Kiekviena grafo viršūnė gali būti vienoje iš trijų būsenų (būsenas žymėsime skirtingomis spalvomis). Pradžioje visos viršūnės yra *neaplankytos* ir dažomos **balta** spalva.

Kai viršūnė  $v$  aplankoma pirmą kartą, ji tampa *nenauja* ir dažoma **pilka** spalva. Laiką, kada ji tapo nauja, saugome masyvo elemente  $d(v)$  (angl. *discovered*).

Viršūnė nudažoma **juoda** spalva, kai išnagrinėjamos visos iš jos išeinančios briaunos, tokios viršūnės yra vadinamos *išsemtosiomis*. Laiko momentą, kada viršūnė tapo juoda, saugome masyvo elemente  $f(v)$  (angl. *finished*).

Kiekviena grafo viršūnė gali būti vienoje iš trijų būsenų (būsenas žymėsime skirtingomis spalvomis). Pradžioje visos viršūnės yra *neaplankytos* ir dažomos **balta** spalva.

Kai viršūnė  $v$  aplankoma pirmą kartą, ji tampa *nenauja* ir dažoma **pilka** spalva. Laiką, kada ji tapo nauja, saugome masyvo elemente  $d(v)$  (angl. *discovered*).

Viršūnė nudažoma **juoda** spalva, kai išnagrinėjamos visos iš jos išeinančios briaunos, tokios viršūnės yra vadinamos *išsemtosiomis*. Laiko momentą, kada viršūnė tapo juoda, saugome masyvo elemente  $f(v)$  (angl. *finished*).

Paieškos kelius įsimename masyve  $\pi$ , jo elemento  $\pi(v)$  reikšmė yra viršūnė  $u$ , iš kurios pirmą kartą aplankėme  $v$ , t. y.  $\pi(v) = u$ .



## Paieškos gilyn algoritmas

### DepthFirstSearch (G)

**begin**

(1) **for** (  $v \in V$  ) **do**

(2)      $spalva(v) = balta$ ;

(3)      $\pi(v) = \text{NULL}$ ;

**end do**

(4)  $t = 0$ ;

(5) **for** (  $u \in V$  ) **do**

(6)     **if** (  $spalva(u) == balta$  ) **then**

(7)         DFS\_Visit( $u$ );

**end if**

**end do**

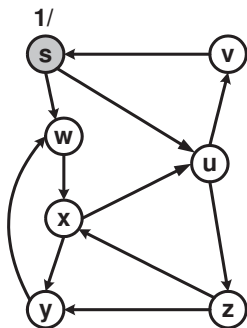
**end** DepthFirstSearch

## Rekursyvusis viršūnių aplankymo algoritmas

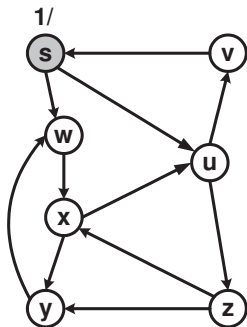
```
DFS_Visit (u)
begin
  (2) spalva(u) = pilka;
  (3) t = t + 1, d(u) = t;
  (4) for ( v ∈ N(u) ) do
  (5)   if ( spalva(v) == balta ) then
  (6)     π(v) = u;
  (7)     DFS_Visit(v);
        end if
      end do
  (8) spalva(u) = juoda;
  (9) t = t + 1, f(u) = t;
end DFS_Visit
```

Imkime tokį orientuotą grafą:

Imkime tokį orientuotą grafą:



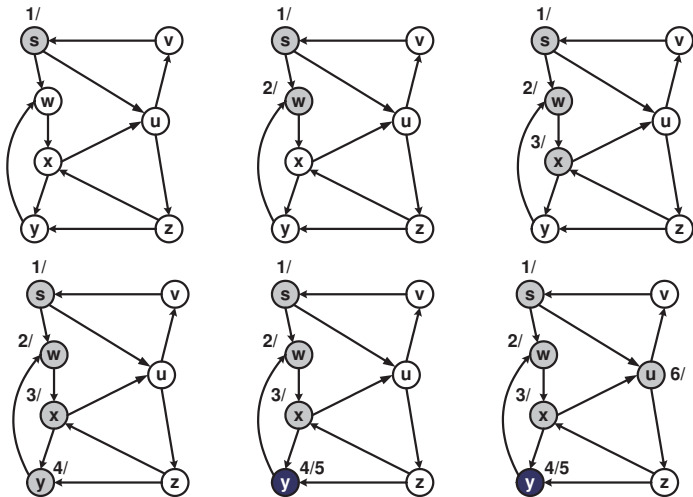
Imkime tokį orientuotą grafą:

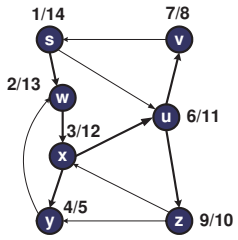
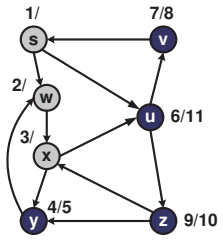
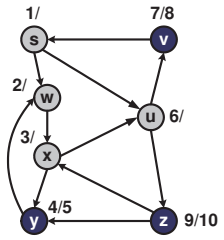
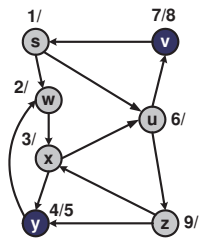
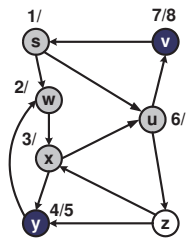
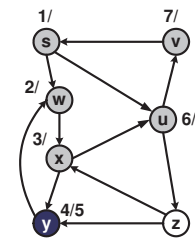


Ar jį galime topologiškai surūšiuoti?

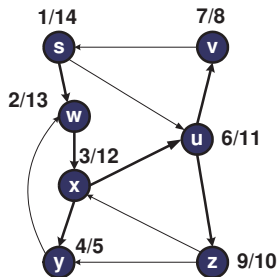
Grafo viršūnes aplankome taikydami paieškos gilyn metodą. Viršūnių lankymo eiga po kiekvieno kreipinio į *DFS\_Visit* funkciją pavaizduota paveiksle. Viršūnėse pateiktos  $(d(v), f(v))$  reikšmės.

Grafo viršūnes aplankome taikydami paieškos gilyn metodą. Viršūnių lankymo eiga po kiekvieno kreipinio į *DFS\_Visit* funkciją pavaizduota paveiksle. Viršūnėse pateiktos ( $d(v)$ ,  $f(v)$ ) reikšmės.

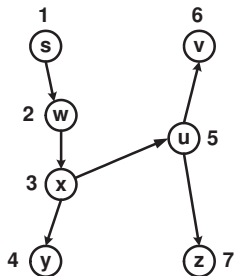








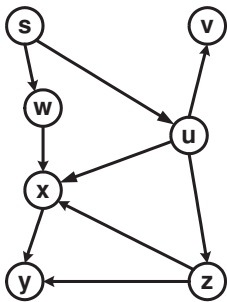
a) aplankytos viršūnės



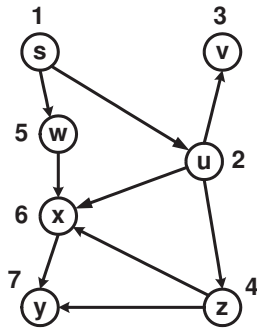
b) aplankymo eiliškumas

Norėdami gauti surūšiuotą viršūnių aibę, modifikuojame *DFS\_Visit* procedūrą, jos pabaigoje viršūnę  $u$  įterpiame į tiesinio sąrašo pradžią (užtenka naudoti *steką*) :

```
DFS_VisitSort ( $u$ )  
begin  
  (2)  $spalva(u) = pilka$ ;  
  (3)  $t = t + 1, d(u) = t$ ;  
  (4) for ( $v \in N(u)$ ) do  
  (5)   if ( $spalva(v) == balta$ ) then  
  (6)      $\pi(v) = u$ ;  
  (7)     DFS_VisitSort( $v$ );  
  end if  
  end do  
  (8)  $spalva(u) = juoda$ ;  
  (9)  $t = t + 1, f(u) = t$ ;  
  (10) List.InsertHead ( $u$ );  
end DFS_VisitSort
```



a)



b)