# *GRAFU TEORIJAS ALGORITMISKIE JAUTĀJUMI*

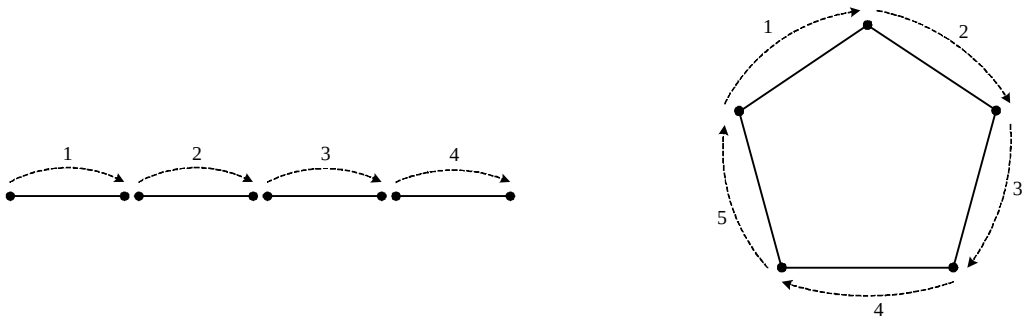
Attiecību-grafu plašā izmantošana praktiski visās zinātnēs un inženierpielietojumos ir radījusi nepieciešamību sistematizēt un attīstīt to algoritmu teorijas daļu, kas apkalpo grafu teorijas uzdevumus. Šajā nodaļā mēs īsi apskatīsim plašāk pielietotos grafu uzdevumus un algoritmus to risināšanai.

## *PLAŠUMMEKLĒŠANA UN DZIĻUMMEKLĒŠANA.*

Bieži risinot teorētiskus vai praktiskus uzdevumus, ir nepieciešams *apiet (pārmeklēt, pārlasīt, iezīmēt)* grafa virsotnes noteiktā, algoritmiskā kārtībā, izmantojot grafa šķautnes. Ir skaidrs, ka algoritms ir vajadzīgs, lai apiešanu varētu realizēt ar datora palīdzību. Šāda algoritma efektivitātes dēļ ir lietderīgi minimizēt katras virsotnes iezīmēšanas reižu skaitu, piemēram, pieprasīt, ka katra virsotne tiek iezīmēta ne vairāk kā vienu reizi.

Kāda varētu būt grafu virsotņu apiešanas dabiska kārtība?

Šo uzdevumu var viegli atrisināt, ja grafs ir, piemēram, ķēde vai cikls, tad ir skaidrs, kā tas ir jāapiet:

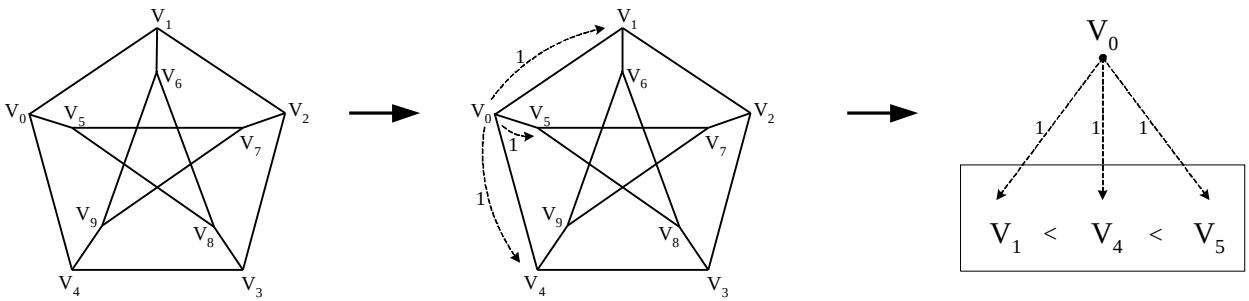



Zīm. 3.62. – ķēdes un cikla apiešanas veidi.

Ko darīt patvaļīga sakarīga grafa gadījumā?

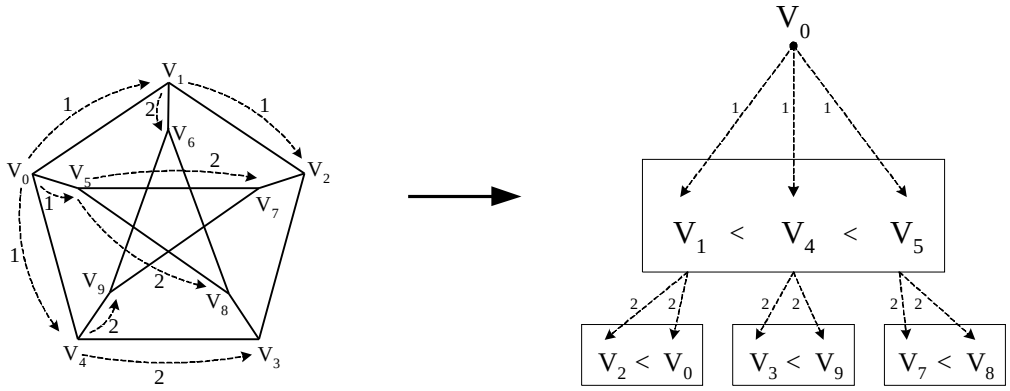
Ir vismaz divi dabiski grafu apiešanas veidi – *plašummeklēšana* un *dziļummeklēšana*. Abos algoritmos tiek pakāpeniski konstruēts orientēts koks, kuram atbilstošais neorientētais koks ir dotā grafa pārklājošais koks.

*Plašuma meklēšana (pārlase plašumā, breadth-first search):* sākam ar fiksētu virsotni $v_0$, atzīmējam visas ar to saistītās virsotnes $\{v_{01}^1, v_{02}^1, ..., v_{0n_1}^1\}$, pilnīgi sakārtojam tās veidā $v_{01}^1 < v_{02}^1 < ...$, konstruējam sakārtotu koku $B_1$:




Zīm. 3.63. – pirmais solis plašuma meklēšanas algoritmā.

Nākošajā solī konstruējam sakārtotu koku $B_2$ šādā veidā: apejam pilnīgi sakārtoto virkni $\{v_{01}^1, v_{02}^1, \ldots\}$ tās pieaugošajā kārtībā un katrai virsotnei $v_i^1$ atzīmējam un piekārtojam kā dēlu kopu visas virsotnes $\{v_{i1}^2, v_{i2}^2, \ldots\}$, kas ir saistītas ar $v_i^1$ un vēl nav atzīmētas:




Zīm. 3.64. – pirmie divi soļi plašuma meklēšanas algoritmā.

Turpinām šo procesu konstruējot sakārtotu koku virkni $B_1, B_2, \ldots$, kuras pēdējais loceklis tiek saukts par *plašuma meklēšanas koku*. Atzīmēsim, ka šis koks nav noteikts viennozīmīgi, tas ir atkarīgs no tā pirmās virsotnes un no katras iekšējās virsotnes dēlu sakārtojumu.

Novērtēsim operāciju skaitu, kas ir nepieciešams plašuma meklēšanai. Pieņemsim, ka ir dots grafs ar V virsotnēm un E šķautnēm, kas ir uzdots ar saistības sarakstu. Plašuma meklēšanas procesā katra virsotne tiek ievietota kokā vienu reizi, tātad summārais operāciju skaits ir O(V). Katras virsotnes saistības saraksts tiek apskatīts

vienu reizi, tātad summārais operāciju skaits šajā algoritma daļā ir O(E). Papildus operāciju skaits grafa inicializācijai ir O(V). Var redzēt, ka kopējais algoritma darba laiks ir O(V+E).

Izmantojot plašummeklēšanu var atrisināt šādus uzdevumus:

a) noteikt, vai grafs ir sakarīgs un atrast grafa komponenšu skaitu - to var izdarīt, veicot meklēšanu plašumā no jebkuras virsotnes, grafs ir saistīts tad un tikai tad, ja tiek apietas visas grafa virsotnes, komponenšu skaits ir vienāds ar nepieciešamo meklēšanu skaitu, kas ir nepieciešamas, lai apietu visas virsotnes,

b) noteikt attālumu starp divām virsotnēm - to var izdarīt veicot meklēšana no vienas virsotnes, attālums starp virsotnēm ir vienāds ar attālumu no sākotnējās virsotnes līdz otrai virsotnei meklēšanas kokā,

c) nokrāsot grafa virsotnes divās krāsās, ja ir zināms, ka tas ir divdaļīgs.

Vēsturiski abi meklēšanas algoritmi tika atklāti, risinot izklaidējošās matemātikas uzdevumus par ceļu meklēšanu labirintos.

Vēl viens efektīvs plašuma meklēšanas pielietošanas piemērs arī ir saistīts ar izklaidējošo matemātiku. Viens no pēdējo gadu aizraujošākajiem notikumiem matemātikā ir tā saucamās „Eternity Puzzle” spēles atrisināšana. 1999. gadā kāda britu rotaļlietu firma laida pārdošanā spēli, kuras uzdevums ir noteiktā veidā savietot 209

plakanas figūras. Par šī uzdevuma atrisināšanu gada laikā pirmā iesūtītā risinājuma autoram tika garantēta vienu miljonu britu mārciņu liela balva. Uzdevuma autori bija nepareizi novērtējuši šī uzdevuma grūtības pakāpi un acīmredzot bija pārliecināti, ka uzdevumu nav iespējams atrisināt gada laikā pat izmantojot jaudīgus datorus. Tas tomēr tika atrisināts septiņu mēnešu laikā veicot meklēšanu ar divu personālo datoru palīdzību un konkursa organizētājiem nācās izmaksāt apsolīto summu. Spriežot pēc risinājuma autoru publikācijām, uzdevums tika modelēts izmantojot grafus un meklēšana pamatojās uz bektrekingu, plašuma meklēšanas algoritmu un diskrēto varbūtību teoriju, kas tika izmantota nākošā elementa optimālai izvēlei. Pirmie uzdevumu atrisināja divi profesionāli matemātiķi.

Otrs grafu teorijā un grafu pielietojumos populārs meklēšanas veids ir *dziļuma meklēšana (pārlase dziļumā, depth-first search):*

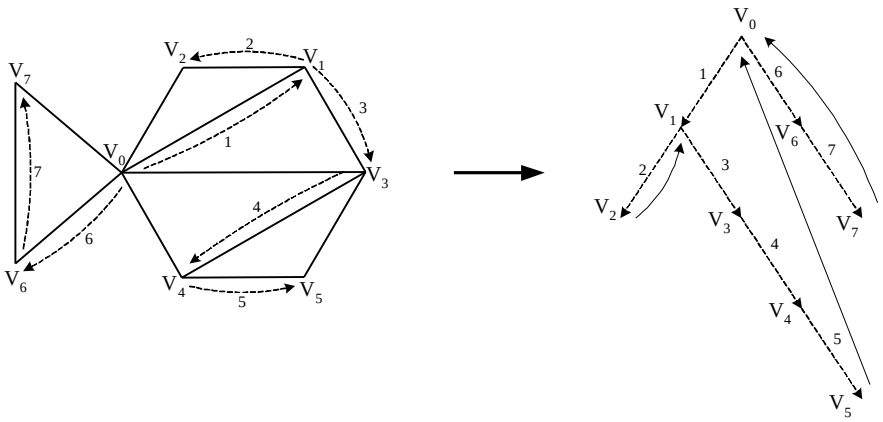
- sākam ar fiksētu virsotni $v_0$ kā topošā meklēšanas koka sakni, atzīmējam jebkuru vēl neatzīmētu virsotni $v_1$, meklēšanas kokā zīmējam šķautni $v_0 \to v_1$ un pārvietojamies uz virsotni $v_1$,
- vispārīgā gadījumā rīkojamies šādi: ja mēs atrodamies virsotnē v un eksistē vēl neatzīmēta virsotne w, kas ir saistīta ar v, tad
  1) atzīmējam virsotni w,
  2) meklēšanas kokā zīmējam šķautni $v \to w$ un

3) pārvietojamies uz virsotni w, pretējā gadījumā (ja visas virsotnes, kas ir saistītas ar v, jau ir atzīmētas) pārejam (atkāpjamies) uz virsotnes v tēvu,

- turpinām šo algoritmu tik ilgi, līdzko visas virsotnes ir atzīmētas un esam atgriezušies atpakaļ virsotnē $v_0$.

Rezultātā iegūsim *dziļummeklēšanas koku*. Bieži vien ir lietderīgi papildus atzīmēt arī „atkāpšanās" šķautnes.

## *PIEMĒRS*




Zīm. 3.65. – dziļuma meklēšanas algoritma realizācijas piemērs.

Orientētiem grafiem var izmantot abas meklēšanas metodes ar to divām atšķirībām:

1) virzīšanās pa šķautni ir atļauta tikai tās norādītajā virzienā un

2) dziļuma meklēšana var tikt veikta vairākas reizes, kamēr tiek atzīmētas visas virsotnes.

Šajā gadījumā dziļummeklēšanas algoritma darba rezultāts ir *dziļummeklēšanas mežs*.

Viens no neacīmredzamiem dziļuma meklēšanas pielietojumiem ir šarnīru noteikšana patērējot mazāk laika nekā tas ir iespējams ar kādu no naivajām metodēm.

Dziļuma meklēšanu tāpat kā plašuma meklēšanu pielieto grafa komponenšu un stipri sakarīgo komponenšu noteikšanā, metrisko invariantu (diametra, ekscentritātes, centra) noteikšanā, divdaļīga grafa sadalīšana daļās. Dziļuma meklēšanu pielieto arī algoritmos, kuros tiek risināti uzdevumi par augstāku kārtu sakarīgumu (šarnīru un bloki noteikšana, fiksētas kārtas sakarīgo komponenšu noteikšana) un planaritāti.

## *STINGRI SAKARĪGO KOMPONENŠU MEKLĒŠANA*

Praktiski svarīgs uzdevums orientētu grafu analīzē ir stingri sakarīgo komponenšu atrašana. Stingri sakarīgās komponentes parāda kā orientētais grafs ir sadalīts mazākās daļās, katrā no kurām jebkuras divas virsotnes ir savstarpēji sasniedzamas ar virzītu ķēžu palīdzību.

Naivs algoritms šī uzdevuma atrisināšanai varētu būt šāds: veiksim plašummeklēšanu no katras virsotnes un

aizpildīsim virsotņu savstarpējas sasniedzamības matricu, šai matricai atbilstošās ekvivalences attiecības klases ir stingri sakarīgās komponentes.

Var piedāvāt arī ātrāku algoritmu, kurā tikai divas reizes tiek veikta pilna dziļummeklēšana orientētajā grafā. Ja ir dots orientēts grafs $\Gamma = (V, E)$, tad definēsim jaunu grafu $\Gamma^{op} = (V, E^{op})$, kur $E^{op} = \{(u,v) \mid (v,u) \in E\}$, citiem vārdiem sakot, grafu $\Gamma^{op}$ iegūst no grafa $\Gamma$ mainot visām šķautnēm virzienu uz pretējo.

*Tardžana algoritms:*

1) veiksim dotajā orientētajā grafā $\Gamma$ pilnu dziļummeklēšanu, kamēr ir atzīmētas visas virsotnes, fiksēsim katras virsotnes v visu kaimiņu atzīmēšanas laiku f(v);

2) konstruēsim $\Gamma^{op}$;

3) veiksim grafā $\Gamma^{op}$ dziļummeklēšanu, izvēloties pirmās dziļummeklēšanas virsotnes parametra f samazināšanās kārtībā, katra dziļummeklēšanas koka virsotnes ir vienas stingri sakarīgas komponentes virsotnes.

## *BINĀRO KOKU ALGORITMI*

Binārie koki ir tā grafu klase, kas relatīvi visbiežāk tiek izmantota pielietojumos un praktiskajā programmēšanā.

Šajā nodaļā apskatīsim vienkāršākos bināro koku algoritmus un bināro koku pielietojumus.

Apskatīsim divus binārос kokus pielietošanas gadījumus – šķirošanas un meklēšanas uzdevumos. Viens no „ātrajiem” (O(nln n)) šķirošanas algoritmiem ir tā saucamais „heapsort” algoritms. Šis algoritms sastāv no diviem soļiem. Pirmajā solī nesakārtotā virkne tiek pārveidota noteikta binära koka – hīpa (heap) veidā. Hīps ir binārs koks ar virsotnēm piekārtotiem skaitliskiem indeksiem, kurā katras virsotnes indekss nav mazākas par tās dēlu indeksiem. Ja ir dota nesakārtota virkne, tad hīpu var konstruēt šādā veidā: pēctecīgi ņemam ārā no virknes elementus un definējam tās kā nākamās lapas kokā, ja tiek pārkāpts hīpa nosacījums, tad virsotnes tiek pārvietotas uz augšu, mainot virsotni vietām ar savu tēvu, tik ilgi, kamēr ir tiek apmierināts hīpa nosacījums. Šo procesu varētu salīdzināt ar dabiskās hierarhijas veidošanos cilvēku kolektīvā, kuram pakāpeniski pievienojas jauni locekļi – cilvēki ar līderu īpašībām laika gaitā paaugstina savu sociālo statusu, kamēr sasniedz tādu līmeni un sociālos kaimiņus, kas aptur viņu sociālo kustību. Otrajā algoritma solī hīps tiek pārveidots par sakārtotu virkni šādā veidā: pēctecīgi no hīpa izņem saknes, ievieto tās kā nākamos elementus sašķirotajā virknē un pārvieto nākamo lapu uz augšu līdz sakne mainot to vietā ar tās tēvu, kamēr nav apmierināt hīpa nosacījums, šo procedūru atkārto tik ilgi, kamēr hīps nav tukšs.

Populārs meklēšanas algoritms ir binārā koka meklēšanas algoritms. Šis algoritms tāpat kā hīpsorta algoritms arī sastāv no diviem soļiem: pirmajā solī kopa, kurā tiks veikta meklēšanas, tiek pārveidota noteikta binära koka *(binārā meklēšanas koka)* veidā, otrajā solī meklējamais objekts tiek meklēts binārajā kokā saskaņā ar noteiktu algoritmu. Par binārās meklēšanas koku sauksim binäru koku ar virsotnēm piekārtotiem skaitliskiem indeksiem, kurā katras virsotnes indekss nav mazākas par tās kreisā apakškoka virsotņu indeksiem un nav lielāks par tās labā apakškoka virsotņu indeksiem. Ja ir dots binārās meklēšanas koka, tad elementa meklēšanu var veikt salīdzinot meklējamo elementu no sākuma ar sakni un tālāk virzoties uz leju atkarībā no salīdzināšanas rezultātiem. Ja ir dots nesakārtots masīvs, tad no tā var inkrementāli izveidot binārās meklēšanas koku šādā veidā: kārtējais elements no masīva tiek meklēts jau konstruētajā kokā un pievienots kā lapa tajā vietā, kur tam ir jābūt, lai pēc pievienošanas atkal izveidotos binārās meklēšanas koks.

## *MINIMĀLĀ SVARA PĀRKLĀJOŠĀ KOKA MEKLĒŠANA.*

Pieņemsim, ka ir dots nosvērts (neorientēts) sakarīgs grafs $\Gamma = (V, E)$ , atgādināsim, ka katrai šķautnei e piekārtots pozitīvs skaitlis w(e) , jeb, citiem vārdiem sakot, ir dota funkcija $w : E \to R^+$. Apskatīsim šādu uzdevumu: konstruēt pārklājošo koku, kura šķautņu svaru summa ir

minimāla. Šādu pārklājošu koku sauksim par *minimāla svara skeletu*. Šī tipa uzdevumu bieži gadās pielietojumos, pārskaitīsim dažus no tiem:

1) inženiertehniska (ceļu tīkla), elektrotehniska vai datortehniska tīkla sakaru līniju kopējā garuma minimizēšana,
2) informācijas pārraides optimizēšana (piemēram, ir dots spiegu tīkls kādā valstī, rezidentam ir jānodod kāda ziņa visiem aģentiem tā, lai minimizētu kopējo risku, definēsim grafu, kura virsotnes ir aģentu, šķautnes ir aģentu pazīšanās un šķautnes svars ir sakaru līnijas riska pakāpe, minimāla svara skeletā ir iekodēts riska ziņā optimāls ziņas nodošanas algoritms)

Minimālā svara skeleta meklēšanai tiek izmantoti „rijīgie" algoritmi. Algoritmu sauksim par *rijīgu (alkatīgu, greedy)*, ja katrā solī tiek izdarīta lokāli (šajā laika momentā) optimālā izvēle. Nebūt nav acīmredzami, ka lokāli optimāla izvēli ir arī globāli optimāla, bet minimāla svara skeleta uzdevumā tas tā ir. Ir divi populāri algoritmi – Kraskala un Prima algoritms, kuros tiek inkrementāli būvēts mežs vai koks pievienojot šķautnes ar minimāli iespējamu svaru.

*Kruskala algoritms:*

*1)* Konstruējam grafu $T_1 = O_n + e_1$ - pievienojam tukšajam grafam ar virsotņu kopu V šķautni $e_1 \in E$, kuras svars ir minimāls.

*2)* Ja grafs $T_i$ jau ir konstruēts un i<n-1, tad konstruējam grafu $T_{i+1} = T_i + e_{i+1}$, kur $e_{i+1}$ ir grafa $\Gamma$, kurai ir minimāls svars starp visām tām šķautnēm, kas neieiet grafā $T_i$ un neveido ciklus ar grafa $T_i$ šķautnēm.

***TEORĒMA 3.40.*** Ja i<n-1, tad grafu $T_{i+1}$ ir iespējams konstruēt. Grafs $T_{n-1}$ ir minimāla svara pārklājošais koks.

***PIERĀDĪJUMS*** Grafā $T_i$ ir tieši i šķautnes, tāpēc tas ir nesakarīgs, ja i<n-1.Tā kā grafs $\Gamma$ ir sakarīgs, tad tajā ir vēl vismaz viena šķautne, kas nesastāda ciklus ar grafa $T_i$ šķautnēm. Tātad meklējamā šķautne $e_{i+1}$ eksistē. Apskatīsim grafu $T_{n-1}$. Tā kā grafs $T_{n-1}$ ir grafs ar n virsotnēm un n-1 šķautni bez cikliem, tad tas ir koks (saskaņā ar teorēmu par kokiem).

Atliek pierādīt, ka koka $T_{n-1}$ svars ir minimāls. Pieņemsim, ka koka $T_{n-1}$ svars nav minimāls un no visiem grafa $\Gamma$ skeletiem izvēlēsimies vienu skeletu T ar minimālu svaru, kuram kopēju šķautņu skaits ar $T_{n-1}$ ir maksimāls. Pieņemsim, ka šķautne $e_i = (a,b)$ ir šķautne ar minimālu numuru kokā $T_{n-1}$, kas nav kokā $T$. Kokā $T$ eksistē ķēde, kas savieno a un b. Pievienojot šai ķēdei šķautni $e_i$ iegūsim ciklu. Šajā ciklā ir vismaz viena šķautne e, kas neieiet kokā $T_{n-1}$. Aizvietojot kokā $T$

šķautni e ar $e_i$ iegūsim jaunu skeletu $T' = T - e + e_i$. Tā kā T ir minimāla svara skelets, tad

$$w(T') = w(T) + w(e_i) - w(e) \geq w(T)$$

un, tātad $w(e_i) \geq w(e)$. No otras puses, pievienojot šķautni e kokam $T_{i-1}$ mēs neiegūsim ciklu, jo šķautnes $e_1, e_2, ..., e_{i-1}$ ieiet kokā T. Ja $w(e_i) > w(e)$, tad koka $T_i$ konstruēšanai mēs nebūtu izvēlējušies šķautni $e_i$. Tātad $w(e_i) = w(e)$ un $w(T') = w(T)$. Esam ieguvuši, ka T' ir minimāla svara skelets. Kopēju šķautņu skaits kokos T' un $T_{n-1}$ ir lielāks nekā kokiem T un $T_{n-1}$, kas ir pretruna. *QED*.

Kraskala algoritma rezultātā tiek iegūts koks, ko sauksim par *Kraskala koku*. *Prima algoritms* atšķiras no Kraskala algoritma ar to, ka katrā solī tiek konstruēts, kam pievieno minimāla svara šķautnes, kas neveido ciklus ar jau esošo koku, koka pirmā virsotne var tikt izvēlēta patvaļīgi. Koks, kas tiek iegūts algoritma rezultāta, tiek saukts par *Prima koku*.

Var pierādīt, ka Kraskala algoritma operāciju skaits ir O(E ln E) un Prima algoritma darba laiks ir O(E ln V), jo lielākā daļa no darba ir šķautņu šķirošana, kas var tikt veikta ar norādīto operāciju skaitu.

*ĪSĀKĀ CEĻA MEKLĒŠANA.*

Pieņemsim, ka ir dots nosvērts (neorientēts vai orientēts) sakarīgs grafs $\Gamma = (V, E)$ ar pozitīviem šķautņu svariem w, apzīmēsim šķautnes svaru ar w(e) un ķēdes summāro svaru ar *dist*.

Apskatīsim šādu uzdevumu: atrast minimāla svara ķēdi, kas savieno divas dotas virsotnes. Šādu ķēdi sauksim par īsāko ceļu starp divām dotajām virsotnēm. Par dotā orientētā grafa *īsāko ceļu koku* sākot no virsotnes v sauksim pārklājošu orientētu koku, kura sakne ir v un kurā maršruti līdz virsotnēm realizē īsākos ceļus.

Īsāko ceļu raksturīgas īpašības:

1) var redzēt, ka ja virsotņu virkne $(v, v_1, ..., v_{n-1}, t)$ ir īsākais ceļš no u līdz t, tad katrai $v_i$ virsotnei šajā virknē virsotņe virkne $(v, v_1, ..., v_i)$ ir īsākais ceļš no v līdz $v_i$;
2) var redzēt, ka īsāko ceļu garumi apmierina šādu nevienādību (optimalitātes nosacījumu):
$$dist(v,t) \le dist(v,u) + w(u,t),$$
kur w(u,t) ir šķautnes $u \to t$ svars.

Tipiski īsākā ceļa pielietošanas piemēri ir saistīti ar optimāla maršruta noteikšana transporta sistēmās.

Ja visu šķautņu svari ir vienādi, tad šis uzdevums var tikt atrisināts ar plašuma meklēšanas palīdzību.

Vispārīgā gadījumā var izmantot *Bellmana-Forda algoritmu* vai *Dijkstras algoritmu*, kas pēc būtības ir līdzvērtīgs Prima algoritmam ar nelielu modifikāciju.

Dijkstras algoritms balstās uz šādu teorēmu.

***TEORĒMA 3.41.*** Ja ir dotas īsākā (mazākā) ķēdes no virsotnes u līdz katrai virsotnei no kopas $\{v_0, v_1, ..., v_n\}$, kur $v_0 = u$, tad eksistē virsotne v, tāda, ka īsākā ķēde no u līdz v ir formā $(u, ..., w_k, v)$, kur ķēde $(u, ..., w_k)$ ir viena no dotajām ķēdēm vai kāda no nepārtrauktām apakšķēdēm, kas sākas ar u.

***PIERĀDĪJUMS*** Ievērosim, ka ja ķēde (u,...,w) ir īsākā ķēde, tad jebkura tās nepārtraukta apakšķēde, kas sākas ar u ir īsākā ķēde līdz tās galapunktam. Apskatīsim virsotni ***v*** tādu, kas minimizē lielumu $dist(u, w_k) + w(w_k, v)$, kur $w_k$ var būt jebkura no doto īsāko ķēžu virsotnēm. Var redzēt, ka virsotne v ir meklējamā virsotne. *QED*

Tātad, lai atrastu īsāko ceļu starp divām virsotnēm nosvērtā grafā pēc Dijkstras algoritma, mums ir jākonstruē Prima koks ar papildus nosacījumiem: ir jāsāk ar vienu no dotajām virsotnēm, katrā solī tiek pievienota šķautne $(w_i, v)$, kas neveido ciklus un minimizē lielumu $dist(u, v_i) + w(v_i, v)$. Pēc galīga skaita soļu tiks konstruēts koks, kas satur abas no dotajām virsotnēm un attālums starp tām ir vienāds ar attālumu konstruētajā kokā.

Oriģinālais Dijkstras algoritms tiek uzdots ar šādas pseidoprogrammas palīdzību:

*Klasiskais Dijkstras algoritms*

Uzdot nosvērtu grafu $\Gamma = (V, E)$ ar nenegatīvu šķautņu svaru funkciju w;
izvēlēties sākuma virsotni v;
definēt dist(v)=0, pred(v)=0, $dist(u) = \infty$, ja $u \neq v$;
definēt *LIST:=V, T – tukšais grafs*;

*while* $LIST \neq \varnothing$
atrast $s \in LIST$, kurai $dist(s) = \min_{t \in LIST} dist(t)$;
izmest s no LIST;
pievienot šķautni (pred(s),s) kokam T;

for $t \in \Gamma(s)$ do
if $dist(t) > dist(s) + w(s,t)$ then
dist(t)=dist(s)+w(s,t), pred(t)=s;
end if;
end for;
end while;
izvadīt T (īsāko ceļu koku sākot no v)

Var pierādīt, ka Dijkstras algoritma operāciju skaits ir $O(V^2)$.

Otrs klasisks īsākā ceļa meklēšanas algoritms ir Bellmana-Forda algoritms:

*Bellmana-Forda algoritms*

Uzdot nosvērtu grafu $\Gamma = (V, E)$ ar nenegatīvu šķautņu svaru funkciju w*;*
izvēlēties sākuma virsotni v;
definēt dist(v)=0, pred(v)=0, $dist(u) = \infty$, ja $u \neq v$;
definēt *LIST:={v}, T – tukšais grafs*;

*while* $LIST \neq \varnothing$
    izmest jebkuru virsotni s no LIST;
    for $t \in \Gamma(s)$ do
        if $dist(t) > dist(s) + w(s,t)$ then
            dist(t)=dist(s)+w(s,t), pred(t)=s;
            if $t \notin LIST$ then
                pievienot t kopai LIST;
            end if;
        end if;
    end for;
end while;
pievienot grafam T visas šķautnes formā (pred(s),s),
izvadīt T (īsāko ceļu koku sākot no v)

Būtu jāpierāda, ka Bellmana-Forda algoritms apstājas, tas ir patstāvīgs darbs lasītājam.

Izmantojot īsākā ceļa atrašanas uzdevumu, var aizpildīt grafa virsotņu *attālumu matricu*, kuras rūtiņās tiek ierakstīti grafu teorētiskie attālumi starp rindai un kolonnai atbilstošajām virsotnēm. Attālumu matricu izmanto grafa metrisko invariantu aprēķināšanai.

Attālumu matricu var atrast, veicot īsākā ceļa meklēšanu no katras virsotnes un apkopojot rezultātus matricā. Aprakstīsim arī divus ātrākus algoritmus attālumu matricas atrašanai – matricu reizināšanas algoritmu un Floida-Voršala (Floyd-Warshall) algoritmu.

Abi algoritmi sākas ar to, ka tiek uzdota orientētā nosvērtā grafa $\Gamma = (V, E)$ *šķautņu svaru matrica* W, kur

$$w_{ij} = \begin{cases} w(i \to j),\ ja\ i\ un\ j\ ir\ savienotas \\ \infty,\ ja\ i\ un\ j\ nav\ savienotas \\ 0,\ ja\ i = j \end{cases}.$$

Definēsim matricu *attālumu reizināšanas operāciju*: ja ir dotas divas nxn matricas A un B, tad definēsim $A \bullet B = C$, kur $c_{ij} = \min_{1 \le k \le n}(a_{ik} + b_{kj})$. Definēsim $A^{\bullet k} = \underbrace{A \bullet A \bullet \ldots \bullet A}_{k\ reizes}$.

*Matricu reizināšanas algoritms*

Uzdot nosvērtu grafu $\Gamma = (V, E)$ ar n virsotnēm ar nenegatīvu šķautņu svaru funkciju w;
konstruēt atbilstošo šķautņu svaru matricu W;
izvadīt $W^{\bullet(n-1)}$.

*Floida-Voršala algoritms*

Uzdot nosvērtu grafu $\Gamma = (V, E)$ ar n virsotnēm ar nenegatīvu šķautņu svaru funkciju w;
konstruēt atbilstošo šķautņu svaru matricu W;

for k:=1 to n do
for $(i, j) \in V \times V$ do
if $w_{ij} > w_{ik} + w_{kj}$ then
$w_{ij} = w_{ik} + w_{kj}$,
end if;
end for;
izvadīt W(īsāko ceļu matricu).

## *PLŪSMAS MAKSIMIZĀCIJA.*

Pielietojumos ir izplatīti uzdevumu, kas ir saistīti ar fizikālu vai cita veida plūsmu vadīšanu to transportēšanas tīklos. Piemēram, uzdevums varētu būt saistīts ar maksimālas caurlaides spējas noteikšanu autoceļu tīklā, naftas vadu tīklā, datortīklā, tehnoloģiskā līnijā, asinsvadu vai nervu tīklā. Viens no svarīgākajiem jautājumiem, kas var būt uzdots šādā situācijā, ir jautājums par maksimālo pieļaujamo plūsmu tīklā.

Dots tīkls $\Gamma = (V, E)$ ar avotu s un noteku t. Katrai šķautnei piekārtosim svaru, definēsim funkciju $c : E \to R^+$ - *šķautņu caurlaides funkciju*. Par virsotnes v *diverģenci (plūsmu caur virsotni)* sauksim lielumu $div(v) = \sum_{\{u|[u,v]\in E\}} c(u,v) - \sum_{\{w|[v,u]\in E\}} c(v,w)$. Funkciju $f : E \to R$ sauksim par *plūsmu* tīklā $\Gamma$, ja tā apmierina šādus nosacījumus:

1) katrai šķautnei (u,v) izpildās nosacījums

$$0 \le f(u,v) \le c(u,v)$$

*(plūsma pa jebkuru šķautni nepārsniedz šķautnes caurlaides spēju)*

3) katrai virsotnei v izņemot avotu un noteku izpildās

$$\mathrm{div}(v)=0$$

*(plūsma nekur neuzkrājas)*

Skaitli div(t) sauc par ***plūsmas lielumu,*** apzīmē ar w(f). Plūsmas maksimizācijas uzdevums ir šāds: *dotajam tīklam ar uzdotām caurlaides spējām katrai šķautnei atrast plūsmu ar maksimālu lielumu.*

Pieņemsim, ka f ir pieļaujama plūsma tīklā. Šķautni e sauksim par tukšu, ja plūsma pa to ir vienāda ar nulli: f(e)=0. Šķautni e sauksim par *piesātinātu*, ja plūsma pa to ir vienāda ar caurlaides spēju: f(e)=c(e). Plūsmu sauksim par *pilnu*, ja katrs maršruts no avota uz noteku satur vismaz vienu piesātinātu šķautni. Plūsmu sauc par *maksimālu*, ja tās lielums ir maksimāls starp visām pieļaujamām plūsmām.

Pirmajā tuvinājumā mēs varētu konstruēt pieļaujamu plūsmu, kas ir pilna.

*Pilnas plūsmas konstruēšanas algoritms:*

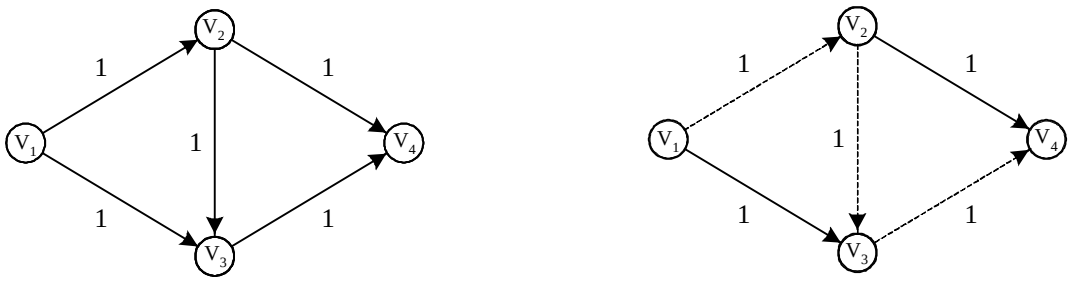
1) sākuma brīdī definēsim nulles plūsmu – f(e)=0 katrai šķautnei e, konstruējam palīggrafu $\Gamma' = \Gamma$.

2) iznīcinām grafā $\Gamma'$ visas piesātinātās šķautnes, konstruējam rezultātā jaunu grafu $\Gamma'$,

3) Meklējam grafā $\Gamma'$ orientētu ķēdi p no avota uz tīklu. Ja tāda ķēde neeksistē, tad ir iegūta pilna plūsma, ja ķēde eksistē, tad ejam uz soli 4);

4) palielinām plūsmu katrā orientētās ķēdes p posmā par lielumu $f_p$, kas ir vienāds ar minimālo caurlaides spēju ķēdē p (tā, lai vismaz viena šķautne būtu piesātināta). Kopējā plūsma palielināsies par $f_p$, plūsma paliks pieļaujama.. Pārejam uz soli 2).

Šāda tipa metodi sauksim par *uzlabojošo ķēžu metodi*. Diemžēl pilna plūsma var arī nebūt maksimāla, tāpēc uzdevums par maksimālo plūsmu vēl nav atrisināts.




*(a)* *(b)*

Zīm. 3.67. (a) - tīkla piemērs, (b) - pilnas un nemaksimālas plūsmas piemērs.

Pilnās plūsmas konstruēšanas algoritmā mēs apskatījām tikai orientētas ķēdes, kas ir virzītas no avota uz noteku.

Izrādās, ka ir lietderīgi apskatīt visas iespējamās, ne obligāti orientētās, ķēdes no avota uz noteku. Viegli redzēt, ka plūsmu var palielināt, ja eksistē ķēde no avota uz izteku ar šādu īpašību: katra šķautne, kas ir vērta no avota uz noteku, nav piesātināta un katra šķautne, kas ir vērsta no notekas uz avotu, nav tukša.(*"no full forward or empty backward edges").*

Var pierādīt, ka algoritms, kurā tiek pakāpeniski palielināta plūsma tik ilgi, kamēr nav ķēžu ar šo īpašību, tiešām atrod maksimālo plūsmu. Šāds algoritms balstās uz Forda-Falkersona jeb max-flow min-cut teorēmu: maksimālas plūsmas lielums ir vienāds ar minimālu griezuma kapacitāti.

Gan pilnas, gan arī maksimālas plūsmas meklēšana ar uzlabojošo ķēzu metodi tiek realizēta konstruējot mainīgu palīgtīklu, kura šķautņu caurlaides spējas tiek modificētas algoritma darba laikā, lai atspoguļotu konstruētās plūsmas ietekmi uz tīkla atlikušo kapacitāti. Virzītie maršruti šajā palīgtīklā tiek noteikti izmantojot īsākā ceļa meklēšanas algoritmus.

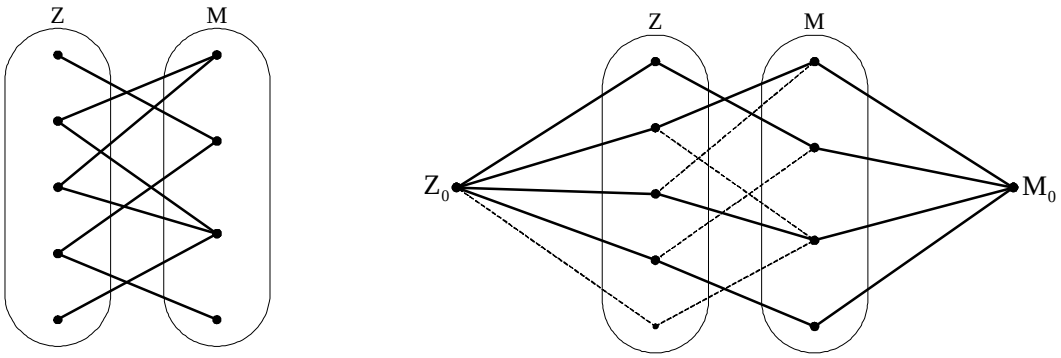
Var pierādīt, ka plūsmas maksimizāciju var realizēt ar algoritmu, kas patērē $O(VE^2)$ operācijas.

Papildus tiešajiem pielietojumiem transporta tīklos šim uzdevumam ir arī daži neacīmredzami pielietojumi. Izmantojot Mengera teorēmu, var piedāvāt algoritmu, kas atrod grafa sakarīgumu izmantojot plūsmas

maksimizāciju: maksimālais virsotņu šķirtu ķēžu skaits, kas savieno divas virsotnes ir vienāds ar maksimālas plūsmas lielumu, kas pieņem, ka katrai neorientētai šķautnei atbilst divas orientētas ar caurlaides spēju 1.

Bieži vien sākotnējais grafs ir jāmodificē tā, lai uzdevumu varētu atrisināt ar plūsmas maksimizācijas metodi, piemēram, grafam ir jāpievieno fiktīvas virsotnes un šķautnes.

Apskatīsim arī tā dēvēto *stabilo laulību* uzdevumu. Ir dota zēnu kopa Z un meiteņu kopa M. Starp dažiem zēniem un meitenēm eksistē savstarpējas simpātijas, kuras mēs vienkāršoti modelēsim ar šķautni, ja eksistē šķautne, tad uzskatīsim, ka atbilstošie cilvēki, var veidot stabilu laulību. Uzdevums ir šāds: noteikt kāds ir maksimālais stabilo laulību skaits, kas ir iespējams dotajam pārim (M, Z). Šo uzdevumu var atrisināt izmantojot plūsmas maksimizācijas pieeju. Pievienosim vēl dažas virsotnes $M_0$ un $Z_0$, savienosim virsotni $M_0$ ar visām kopas M virsotnēm un $Z_0$ ar visām kopas Z virsotnēm, uzskatīsim $M_0$ par avotu un $Z_0$ - par noteku, katru šķautni orientēsim virzienā no avota uz noteku un piešķirsim katrai šķautnei caurlaides spēju 1. Viegli redzēt, ka maksimāla plūsma šādā tīklā definē maksimālu pāru skaitu.




Zīm.3.68. – stabilo laulību uzdevuma risināšana ar plūsmas maksimizācijas metodi.

Apskatīsim uzdevumu par kopas apakškopu sistēmas pārstāvju izvēli. Ir dota kopa A un tās apakškopu kopa $\{A_i\}_{i\in I}$. Uzdevums ir atrast apakškopu sistēmas dažādu pārstāvju kopu – kopu $\{a_i\}_{i\in I} \subseteq A$, kas apmierina īpašību $a_i \in A_i$. Lai reducētu šo uzdevumu uz plūsmas maksimizācijas uzdevumu, definēsim tīklu šādā veidā: virsotņu kopa ir $\{A_i\}_{i\in I} \cup A \cup \{x, y\}$, kur x un y ir divas jaunas virsotnes, savienosim katru kopas A virsotni ar, savienosim katru kopas $\{A_i\}_{i\in I}$ virsotni ar y, uzskatīsim x par avotu un y par noteku, orientēsim šķautnes virzienā no avota uz noteku un piekārtosim katrai šķautnei caurlaides spēju 1. Viegli redzēt, ka apakškopu sistēmai eksistē pārstāvju kopa tad un tikai tad, ja maksimāla plūsma ir vienāda ar apakškopu skaitu.

Minēsim arī klasisku rezultātu, kas bieži tiek izmantots uzdevumos, kas ir saistīti ar apakškopu pārstāvju sistēmas meklēšanu. Ievērosim, ka iepriekšējā rindkopā

konstruētais grafs bez virsotnēm x un y ir divdaļīgs grafs. Virsotņu kopas apakškopai U definēsim $N(U) = \bigcup_{u \in U} N(u)$.

***TEORĒMA*** *(Holla teorēma)* Ir dots divdaļīgs grafs $\Gamma = (V, E)$ ar tādu virsotņu kopas sadalījumu (U,V), ka šķautnes savieno tikai virsotnes dažādās sadalījums daļās. Maksimāla neatkarīga šķautņu kopa satur |U| elementus tad un tikai tad, ja katrai $U' \subseteq U$ izpildās nosacījums $|U'| \le |N(U')|$.

***PIERĀDĪJUMS.*** Ja maksimāla neatkarīga šķautņu kopa satur |U| elementus, tad nosacījums $|U| \le |N(U)|$ izpildās.

Otrādi, pieņemsim, ka maksimāla neatkarīga šķautņu kopa S nav incidenta kādai kopas U virsotnei $u_0$. Konstruēsim maksimāli garu dažādu virsotņu virkni $M = (u_0, v_1, u_1, ...)$, kas apmierina šādu nosacījumu: $u_i \sim v_i \sim u_{j(i)}$, kur $0 \le j(i) < i$. Šāda virkne nevar beigties ar kopas U virsotni, jo apakškopai $\{u_0, ..., u_{i-1}\}$ apkārtne satur vismaz i elementus, un eksistē virsotne $v_i \notin \{v_1, ..., v_{i-1}\}$, kas apmierina nosacījumu $v_i \sim u_{j(i)}$. Pieņemsim, ka konstruētās virsotņu virknes pēdējais elements ir virsotne $v_n$. Apskatīsim virsotņu virkni $M' = (v_n, u_{j(n)}, v_{j(n)}, u_{j^2(n)}, v_{j^2(n)}, ..., u_0)$. Šai virsotņu virknei atbilstošās ķēdes pāra šķautnes (skaitot no $v_n$) pieder kopai S. Virsotne $v_n$ nav incidenta ne ar vienu kopas S šķautni, jo pretējā gadījumā mēs varētu pagarināt virkni M vai iegūtu pretrunu. Virknei M' atbilstošās ķēdes

nepāra šķautnes veido neatkarīgu šķautņu kopu, kurā ir par vienu vairāk elementu nekā kopā S, kas ir pretruna. Ir pierādīts, ka maksimāla neatkarīga virsotņu kopa satur |U| elementus. QED.

## *MAKSIMĀLU NEATKARĪGU KOPU MEKLĒŠANA.*

Dots nosvērts (neorientēts) grafs. Meklēsim neatkarīgu virsotņu vai šķautņu kopu ar maksimālu svaru. Šo uzdevumu ir iespējams atrisināt tikai apskatot visas iespējamās virsotņu vai šķautņu apakškopas, tātad ja grafam ir n virsotnes, tas ir jāapskata visas $2^n$ apakškopas un algoritma darba laiks ir $\Omega(2^n)$. To ir iespējams izdarīt, piemēram, ar bektrekinga algoritma palīdzību – tiek pēctecīgi ģenerētas visas apakškopas un uzglabāta apakškopa ar maksimālo svaru starp visām jau ģenerētajām apakškopām.

Var piedāvāt arī heiristisku algoritmu, kurš konstruē neatkarīgu virsotņu kopu šādā veidā:

1) izvēlēties grafā $\Gamma$ virsotni t ar minimālu pakāpi un ievietot to kopā U (sākumā kopa U ir tukša,
2) ja $\Gamma - (t \cup N_\Gamma(t))$ ir tukšs grafs, tad apstāties, ja $\Gamma - (t \cup N_\Gamma(t))$ nav tukšs, tad pārveidot grafu $\Gamma$ par $\Gamma - (t \cup N_\Gamma(t))$ (iznīcināt kopas $t \cup N_\Gamma(t)$ virsotnes) un iet uz soli 1).

## *KRĀSOŠANA.*

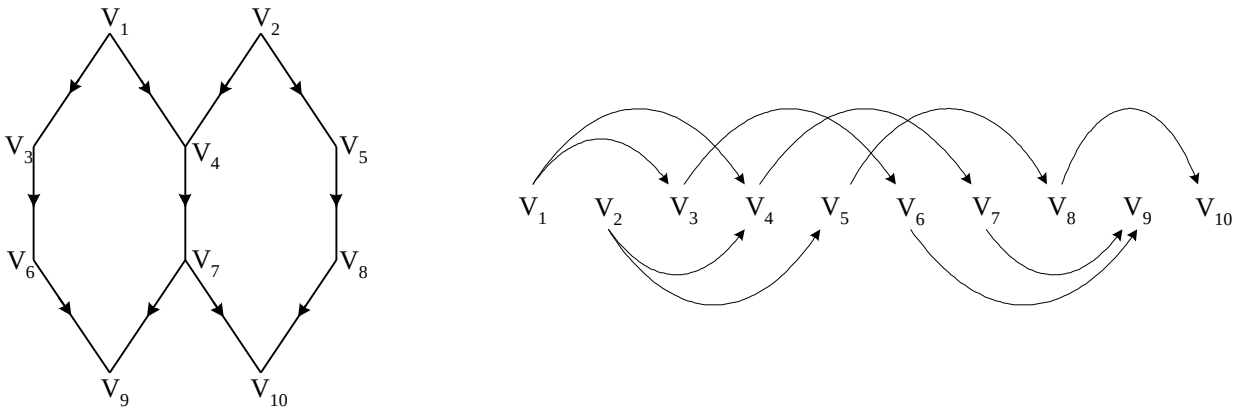
Ir dots grafs. Apskatīsim tā virsotņu krāsošanas uzdevumu, ja ir dots krāsu skaits. Tā kā virsotņu krāsošana ir tas pats, kas virsotņu kopas sadalījums neatkarīgu apakškopu apvienojumā, mēs iegūstam šādu krāsošanas algoritmu:

3) izvēlēties grafā $\Gamma$ maksimālu neatkarīgu virsotņu kopu S,
4) nokrāsot kopas S virsotnes kārtējā krāsā,
5) ja $\Gamma - S$ ir tukšs grafs, tad apstāties, ja $\Gamma - S$ nav tukšs, tad pārveidot grafu $\Gamma$ par $\Gamma - S$ (iznīcināt kopas S virsotnes) un iet uz soli 1.

Var pierādīt, ka šis algoritms atrod krāsojumu ar k krāsām, ja $c(\Gamma) \le k$ .

## *„TOPOLOĢISKĀ ŠĶIROŠANA”*

Ir dots AOG. Uzdevums ir sakārtot grafa virsotnes virknē tā, lai katras orientētas šķautnes beigu virsotne virknē būtu pa labi no sākuma virsotnes:




Zīm. 3.69. – topoloģiskās šķirošanas piemērs.

Šo uzdevumu sauksim par AOG *topoloģiskās šķirošanas uzdevumu*. Var redzēt, ka pēc virsotņu topoloģiskās šķirošanas AOG blakusattiecības matricai visi vieninieki ir zem galvenās diagonāles.

Viens no tā pielietojumiem ir šāds: pieņemsim, ka ir liels projekts, kas sastāv no vairākiem darbiem, darbi var būt atkarīgi viens no otra (piemēram, darbs B nevar būt iesākts pirms nav pabeigts darbs A), modelēsim šo projektu ar orientētu grafu, kurā virsotnes ir darbi un šķautnes savieno atkarīgus darbus to izpildes kārtībā. Topoloģiskās šķirošanas uzdevumu atrisinājumu šajā gadījumā var interpretēt kā darbu virkni, kas apmierina atkarības nosacījumus.

Ja AOG tiek interpretēts kā kādas daļēji sakārtotas kopas Hasses grafs, tad topoloģiskās šķirošanas uzdevums ir ekvivalents dotā daļējā sakārtojuma paplašināšanai līdz pilnam (lineāram) sakārtojumam. Daļēji sakārtoto kopu teorijā ir pierādīts, ka tas ir iespējams visos gadījumos.

Šis uzdevums var tikt atrisināts izmantojot šādu algoritmu:

1) dotajā grafā $\Gamma$ atradīsim visu avotu kopu S (pārskatot grafa matricu vai izmantojot dziļuma meklēšanu), ja kopa S ir tukša, tad darbs ir pabeigts, ja nē, tad pārejam uz soli 2),
2) pievienosim kopu S virknes labajā galā jebkurā kārtībā, pārveidosim $\Gamma$ par $\Gamma - S$ pāriesim uz soli 1).

## *DARBU PLĀNOŠANAS PROBLĒMA*

Plānojot darbu projektā, kas satur vairākus savstarpēji atkarīgus darbus, varbūt nepieciešams modelēt šādu projektu ar grafu palīdzību, atrisināt radušos grafu teorijas uzdevumus un izstrādāt atbilstošus algoritmus. Projekti var tikt modelēti ar grafiem vismaz šādos veidos:

1) grafa virsotnes ir projekta darbi, orientētas šķautnes savieno darbus, kas ir savstarpēji atkarīgi, šādu modeli angļu valodā apzīmē ar AoN („activity on node”);

2) grafa virsotnes ir projekta izpildes stāvokļī, orientētas nosvērtas šķautnes ir darbi, šķautnes svars nozīmē izpildes laiku, izmaksas vai citus lielumus, šādu modeli angļu valodā apzīmē ar AoA („activity on arc”);

3) grafa virsotnes ir projekta izpildes stāvokļi un darbi, orientētas šķautnes savieno savstarpēji atkarīgus stāvokļus vai stāvokļus un tiem sekojošos vai priekštecējošus darbus;

4) grafa virsotnes ir projekta darbi un izpildītāji, virsotnes savienu izpildītājus ar darbiem, kurus tie var izpildīt.

Vienkāršākie darbu plānošanas uzdevumi var būt šādi:

1) noteikt darbu izpildes secību, ievērojot darbu savstarpējo atkarību izmantojot topoloģisko šķirošanu;

2) noteikt vairāku darbu vienlaicīgas izpildes grafiku izmantojot virsotņu un šķautņu neatkarīgu kopu meklēšanu un grafu krāsošanu;

3) piekārtot maksimāli lielu skaitu darbu pieļaujamajiem izpildītājiem izmantojot tīkla plūsmas maksimizāciju;

4) minimizēt projekta izpildes laiku ar neierobežotu vai ierobežotu izpildītāju skaitu;

5) minimizēt projekta kopējās izmaksas.

## *EILERA CIKLA KONSTRUĒŠANA*

Ir dots sakarīgs grafs $\Gamma$, kuram visu virsotņu pakāpes ir pāra skaitļu. Uzdevums ir atrast Eilera ciklu. Grafu (ne obligāti sakarīgu), kura visu virsotņu pakāpes ir pāra skaitļi, sauksim par kvazi-Eilera grafu. Šī uzdevuma atrisinājums balstās uz to apstākli, ka izdzēšot no kvazi-Eilera grafa kāda cikla šķautņu kopu, atkal iegūsim kvazi-Eilera grafu. Uzdevuma risināšanai var piedāvāt šādu algoritmu:

1) izvēlēties grafā $\Gamma$ virsotni, kuras pakāpe nav 0 un konstruēt jebkuru ciklu Z sākot ar šo virsotni, ierakstīt Z ciklu masīvā;
2) ja $\Gamma - Z$ ir bezšķautņu grafs, tad iet uz soli 4), ja nē, tad uz soli 3);
3) definēt $\Gamma := \Gamma - Z$ un iet uz soli 1);
4) konstruēt iegūto ciklu sakarības grafu – virsotnes ir cikli, šķautne savieno divas virsotnes, ja atbilstošajiem cikliem ir kopīga virsotne;
5) ciklu sakarības grafā katras virsotnes z kaimiņus sakārtot tādā kārtībā, kādā šie ciklu krusto z;
6) ciklu sakarības grafā veikt dziļummeklēšanu, izvēloties virsotņu atzīmēšanas kārtību saskaņā ar solī 5) noteikto kārtību;
7) apiet ciklus tādā kārtībā, kādā tie ir atzīmēti veicot solī 6) aprakstīto dziļummeklēšanu.

Klasisks Eilera cikla konstruēšanas algoritms ir *Flerī algoritms*: sākot no jebkuras virsotnes izvēlamies incidentas šķautnes, kuras uzreiz pēc izvēlēšanas tiek

nodzēstas, tā lai kārtējā izvēlētā šķautne nav tilts atlikušajā grafā (ja nav citas izvēles).

Var pierādīt, ka šajā procesā tiek konstruēts cikls, kas satur visas škautnes, tātad Eilera cikls. Tiešām, tā kā sākotnēji katras virsotnes pakāpe ir pāra skaitlis, tad algoritms beigs darbu tajā virsotnē, kurā sāka, tātad algoritms konstruē kaut kādu ciklu. Jāpierāda, ka šis cikls satur visas šķautnes. Pieņemsim pretējo – algoritms konstruē ciklu Z, kas nav Eilera cikls. Izdzēsim no sākotnējā grafa cikla Z šķautnes un apskatīsim jebkuru atlikušā grafa sakarības komponenti C, kas nav triviāla. Apskatīsim cikla Z šķautnes, kas ir incidentas ar C virsotnē un sanumurēsim tās to apiešanas kārtībā. Šķautne e ar vislielāko numuru pirms apiešanas bija tilts un vienlaicīgi eksistēja komponentes C šķautnes, kas nebija tilti un kuras varēja izvēlēties šķautnes e vietā. Iegūtā pretruna pierāda to, ka algoritmā konstruētais cikls satur visas šķautnes.

Eilera cikla konstruēšanas uzdevums var būt svarīgs pilsētu ielu apkopes speciālistiem – ja pilsēta tiek modelēta kā grafs, kura virsotnes ir krustojumi un šķautnes ir ielas, tad Eilera cikls norāda minimāla garuma maršrutu, kas ir jāveic sniega tīrīšanas mašīnai, lai attīrītu no sniega visas ielas.

## *HAMILTONA CIKLA UN „CEĻOJOŠĀ TIRGOŅA" PROBLĒMA*

Ir dots nosvērts grafs. Uzdevums ir atrast šajā grafā Hamiltona ciklu, kura šķautņu svaru summa ir minimāla, šādu uzdevumu sauksim par „ceļojošā tirgoņa problēmu", jo tas ir cēlies no uzdevuma par tirgoni, kam ir jāapmeklē visas pilsētas valstī un jāatgriežas sākotnējā pilsētā ceļojot pēc iespējas īsāku maršrutu. Speciālgadījumā, ja visu šķautņu svari ir vienādu, iegūsim uzdevumu par Hamiltona cikla atrašanu.

Vispārīgā gadījumā šim uzdevumam nav atrasts algoritms, kas patērē mazāk laika nekā to, kas ir nepieciešamas visu virsotņu permutāciju apskatīšanai. Tātad, ja grafam ir n virsotnes, tad algoritma darbības laiks ir $\Omega(n!)$. Speciālu grafu klasēm šī uzdevuma risināšanai ir izstrādātas efektīvas heiristiskas metodes.

## *GRAFU IZOMORFISMS*

Ir doti divi viena tipa (neorientēti, orientēti u.c.) grafi $\Gamma_1 = (V, E_1)$ un $\Gamma_2 = (V, E_2)$. Uzdevums ir noteikt, vai šie grafi ir izomorfi.

Naivs veids, kā atrisināt šo uzdevumu ir šāds: ar bektrekinga palīdzību ģenerēt visas iespējamās virsotņu kopas V permutācijas $f : V \to V$ un pārbaudīt vai grafa $f(\Gamma_1)$ matrica ir vienāda ar grafa $\Gamma_2$ matricu. Redzam, ka šis algoritms patērē laiku $\Omega(n!)$, jo permutāciju skaits kopai ar n elementiem ir vienāds ar n!. Šādu bektrekingu

var būtiski pāatrināt, ja abu grafu virsotņu kopas sadala apakškopās ar vienādām īpašībām, piemēram, vienādām virsotņu pakāpēm.

## *GRAFU ZĪMĒŠANA*

Praktiski svarīgs uzdevums ir efektīva grafu attēlošana jeb zīmēšana plaknē. Parasti nepietiek ar to, ka dotais uzdevums tiek modelēts un risināts ar grafu teorijas metodēm, grafi vēl ir jāattēlo tā, lai to struktūra un īpašības ir viegli uztveramas. Zīmējot grafus, parasti ir liela izvēle, piemēram, virsotņu koordinātes var būt veseli skaitļi vai patvaļīgi skaitļi, šķautnes var zīmēt taisnas vai lokveida, šķautnes var būt vērstas tikai horizontālā vai vertikālā virzienā vai arī leņķi starp šķautnēm var būt patvaļīgi. Dažādos veidos var attēlot virsotņu un šķautņu indeksus u.t.t. Lai saprastu, kā grafs ir jāzīmē, ir jānosaka kādas grafa īpašības un invarianti ir svarīgi uzdevuma risināšanā. Pārskaitīsim dažus populārākos grafu zīmēšanas veidos un atbilstošos invariantus:

1) grafs tiek zīmēts tā, lai minimizētu šķautņu krustošanos skaitu, šajā gadījumā ir jāizmanto planāru grafu plakanizācijas algoritmi,

2) grafs tiek zīmēts tā, lai minimizētu tā laukumu vienlaicīgi nodrošinot noteiktu attāluma minimumu starp jebkurām divām virsotnēm,

3) grafs tiek zīmēts tā, lai minimizētu garākās šķautnes garumu,

4) grafs tiek zīmēts tā, lai tiktu skaidri parādītas tā augstāku kārtu sakarīguma komponentes, šajā gadījumā no sākuma ir jāatrod šīs komponentes, piemēram, ir jāatrod visi šarnīri, tilti, bloki u.t.t.,

5) orientēts grafs tiek zīmēts tā, lai tiktu skaidri parādītas tā stipri saistītās komponentes, šajā gadījumā no sākuma ir jāatrod šīs komponentes,

6) grafs tiek zīmēts tā, lai tiktu skaidri parādītas tā kliķes, šajā gadījumā no sākuma ir jāatrod visas grafa kliķes,

7) grafs tiek zīmēts tā, ka visas virsotnes tiek izvietotas par riņķi un iespējami vairāk blakusesošas virsotnes ir savienotas ar šķautnēm, šajā gadījumā grafā ir jāatrod cikls ar maksimālu škautņu skaitu, vislabākais gadījums ir tad, ja grafs ir Hamiltona grafs,

8) grafs tiek zīmēts tā, lai tiktu skaidri parādītas grafa orbītas (attiecībā uz automorfisma grupas darbību) un automorfisma grupas darbība, šajā gadījumā no sākuma ir jāatrod grafa automorfisma grupa un tās darbības orbītas,

9) grafs tiek zīmēts tā, lai tiktu skaidri parādītas tā metriskās īpašības (centrs, diametrs, rādiuss u.t.t.),

šajā gadījumā no sākuma tiek atrasti nepieciešamie metriskie invarianti, bieži šādā gadījumā grafa centra virsotnes zīmē zīmējuma centrā un pārējās virsotnes mēģina zīmēt tā, lai to attālums līdz centra virsotnēm aptuveni sakristu ar to grafu-teorētisko attālumu *(radiālais attēlojums),*

10) grafs tiek zīmēts tā, lai tiktu skaidri parādīts kāds virsotņu kopas sadalījums vai pārklājums (grafs tiek zīmēts kā m-daļīgs grafs, grafs tiek zīmēts ar noteiktu krāsojumu u.c),

11) grafs tiek zīmēts tā, lai akcentētu kādu tā noteiktu apakšgrafu, piemēram, tā skeletu,

12) grafs tiek zīmēts tā, lai tiktu pēc iespējas skaidrāk parādīts sākotnējais uzdevuma modelis pirms grafa definēšanas, šajā gadījumā grafu teorija nav jāpielieto,

13) orientēts koks tiek zīmēts ar sakni zīmējumā augšā un lejupejošām šķautnēm tā, lai virsotnes, kuru attālums līdz saknei ir vienāds atrodas vienā horizontālajā līmeni *(rangu attēlojums)* .

Grafu zīmēšanā var izmantot dažādas fizikālas izcelsmes heiristiskas metodes. Piemēram, var uzskatīt, ka katra virsotne ir elektrisks lādiņš un katra šķautne ir atspere, kas saista divas virsotnes, uzskatīsim, ka virsotnes atgrūžas viena no otras ar spēku, kas ir atkarīgs no to grafu-

teorētiskā attāluma un uzskatīsim, ka atspere darbojas uz virsotnēm saskaņā ar Huka likumu, zīmēsim grafu kā definētās fizikālās sistēmas līdzsvara stāvokli.

Visbiežāk grafu zīmēšanas algoritmi sevī iekļauj ģeometriskas vai grafu-teorētiskas optimizēšanas algoritmus, kas ir eksponenciāli vai faktoriāli atkarīgi no virsotņu skaita pat speciālām grafu klasēm, piemēram, kokiem.

Grafu zīmēšanas algoritmi tiek sevišķi plaši pielietoti elektrisko ķēžu projektēšanā, programminženierijā un algoritmu animācijā.